ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

GENERATIVE ADVERSARIAL NETWORKS BASED LEVEL GENERATION FOR ANGRY BIRDS

M.Sc. THESIS

Burkan Yılmaz

Department of Game and Interaction Technologies

Game and Interaction Technologies Programme

June 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

GENERATIVE ADVERSARIAL NETWORKS BASED LEVEL GENERATION FOR ANGRY BIRDS

M.Sc. THESIS

Burkan Yılmaz
(529171005)

Department of Game and Interaction Technologies

Game and Interaction Technologies Programme

Thesis Advisor: Assoc. Prof. Dr. Sanem Sarıel

June 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

ÇEKİŞMELİ ÜRETİCİ AĞLAR İLE ANGRY BIRDS BÖLÜMLERİNİN ÜRETİLMESİ

YüKSEK LİSANS TEZİ

Burkan Yılmaz
(529171005)

Oyun ve Etkileşim Teknolojileri Anabilim Dalı

Oyun ve Etkileşim Teknolojileri Programı

Tez Danışmanı: Assoc. Prof. Dr. Sanem Sarıel

Haziran 2022

Burkan Yılmaz, a M.Sc. student of ITU Graduate School student ID 529171005, successfully defended the thesis entitled "GENERATIVE ADVERSARIAL NETWORKS BASED LEVEL GENERATION FOR ANGRY BIRDS", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Dr. Sanem Sarıel** Istanbul Technical University	.....
Jury Members :	**Prof. Dr. Hüseyin AKCAN** Izmır University of Economics	.....
	Assoc. Prof. Dr. Gökhan İNCE Istanbul Technical University	.....

Date of Submission : 3 June 2022
Date of Defense : 30 June 2022

To my wife and son,

FOREWORD

First of all, I would like to thank my advisor, Assoc. Prof. Dr. Sanem Sarıel, for the mentorship, and perspective she gave me, and also for her patience throughout this process. I would like to express my gratitude to my lovely wife, Gamze Yılmaz. She has never stopped supporting me and is always there for me. I also would like to thank my dear little son Deniz Yılmaz, who has energized me in everything I wanted to achieve since he was born.

I am so appreciative, thank you all.

June 2022 Burkan Yılmaz

TABLE OF CONTENTS

Page

ABBREVIATIONS

ML : Machine Learning
DL : Deep Learning
CNN : Convolutional Neural Network
RNN : Recurrent Neural Network
DM : Discriminative Models
GM : Generative Models
AI : Artificial Intelligence
PCG : Procedural Content Generation
PLG : Procedural Level Generation
GAN : Generative Adversarial Network
VAE : Variational Autoencoders
DCGAN : Deep Convolutional Generative Adversarial Network
CycleGAN : Cycle-Consistent Adversarial Networks
PIL : Python Imaging Library
KL : Kullback–Leibler
ReLU : Rectified Linear Unit
LReLU : Leaky Rectified Linear Unit
CNN : Convolutional Neural Networks
WGAN : Wasserstein Generative Adversarial Network
EDA : Estimation of Distribution Algorithm
CMA-ES : Covariance Matrix Adaptation Evolution Strategy
RGM : Rube Goldberg Machine
CycleGAN : Cycle-Consistent Adversarial Networks
XML : Extensible Markup Language
BCE : Binary Cross Entropy
PCA : Principal Component Analysis
SVD : Singular Value Decomposition

SYMBOLS

θ	: Probability Distribution
θ^*	: Maximum Likelihood Estimator of θ
P_{model}	: Estimated Model Probability Distribution
$X^{(i)}$	: Sample in the Data Set
m	: Total Data Set Sample Count
X	: One Dimensional Image Data
$P_{model}(X)$	: Likelihood of Image
L	: Loss Function
KL	: Kullback–Leibler Divergence
$\mathbb{E}$	: Sum

LIST OF TABLES

Page

LIST OF FIGURES

Page

GENERATIVE ADVERSARIAL NETWORKS BASED LEVEL GENERATION FOR ANGRY BIRDS

SUMMARY

Recently, content generation has become one of the biggest challenges faced by creators. This process brings with it many conditions such as from the usefulness of the produced content to the environments to be used, to the production process, and to the originality of the content. Content generation has an important role in many fields from science to art, from art to technology.

In these days, content generation is customized and used in many areas procedurally. Procedural content generation methods are frequently used in many computer science fields such as computer graphics, artificial intelligence, human-computer interaction. As Togelius described that as a result, PCG can be described as algorithmic game content development with little or indirect user input [1]. In video games, content can correspond to a wide variety of products. There are lots of examples for video game contents such as sound effects, music tracks, textures, materials, object models, world terrain, enemy placement, character development, and more. The most important point in algorithms which are used for PCG, is the implementation of randomness. A wide variety of outputs can be produced through the given parameters and the random functions used in the algorithm. Since this diversity in every game, can provide many benefits from having a rich content of the game to increasing its replayability.

PCG methods were frequently used in the past because the resources and processing capabilities were limited. During this period, game manufacturers were also producing game content using PCG methods to run on devices with limited memory. As Joseph stated the game named Rogue, produced in 1980, was one of the remarkable turn-based games created using PCG technique [2]. Because of the memory constraints, Rogue was created using with generation algorithm. As Joel stated that Rogue was so successful, in fact, that it created a new subgenre of video games: the roguelike [3]. PCG was also used by game developers to make games different every time they are played. We can see the best example of this in the game named Diablo, which was developed by the Blizzard game company in 1996. In Diablo, players were procedurally encountering a completely randomly generated game map.

In today's games, PCG is used in two different ways, online and offline. In the online method, the game content is created while the game is being played. Games can be more difficult depending on player plays, or game contents can be created differently in online PCG methods. The best example of online PCG games is the video game Spore, published by Electronic Arts. In this game, players can create characters as they wish by combining their characters with other contents they find in the game. Rather than

having game designers create hundreds of 3-D models of game contents in preparation and program their behavior, he had his team create algorithms to animate the contents that are created by gamers using procedural generation [4]. In the offline method, the game content is created during the development process of the game and then included in the game. Content produced offline can be included directly in the game, or it can be added to the game after passing through the filters of the designers. No Man's Sky, developed by Hello Games, can be given as an example game whose content is created using the offline PCG method. In an unlimited randomly created galaxy, No Man's Sky is a videogame about discovery and surviving [5].

Different kinds of algorithms are preferred for PCG. As well as rule-based methods, evolutionary algorithms and methods based on machine learning can also be implemented in PCG. We can divide these algorithms into two groups as search-based approaches and constructive approaches [6]. The content produced by constructive algorithms is expected to be suitable for use, and the output is used as it is. Algorithms obtained by creating a set of rules are in this group called constructive. On the other hand, Search-based algorithms have a very different structure. As stated in the naming, they are used to find the most suitable generated content. In this method, the content is generated by the algorithm, tested, and then if it is accepted, it is used. Even if it is not suitable, new content is produced again and again until the most suitable one is found. Machine learning-based algorithms and evolutionary algorithms are good examples of search-based methods.

The widespread use of PCG methods has enabled video game contents to be converted into data sets and shared. The creation of game corpuses also helped to use machine learning methods for PCG. In this way, both supervised and unsupervised machine learning methods became available. Along with the developing technology, deep learning methods also started to be used as a PCG method. Today, deep learning methods like Variational Autoencoders and Generative Adversarial Networks, are used for generating content studies. In this thesis, Generative Adversarial Network methods are implemented and examined to generate game content procedurally.

Rovio Entertainment's Angry Birds is a popular physics-based game that was published in 2009 for mobile devices [7]. Angry Birds, like other physics games, was also used as a domain in PCG studies [8]. The physics constraints used in such games cause many problems for procedural level generation. These constraints also make it difficult to rate the quality of the generated levels of the game. As Shaker states, one of the most difficult aspects of creating game content is determining the quality of the created artifacts [9]. Due to a large number of situations and large movement areas in physics-based games, it is particularly difficult to verify the playability/solvability of the generated levels.

In this thesis, a procedural content generation method is developed for generating levels of the Angry Birds. Since Angry Birds is a commercial video game, source codes are not shared by the Rovio company. Thus, the generated levels were tested and visualized on the clone Angry Birds game named Science Birds, developed by Lucas Ferreira [10]. In order to generate levels, three different Generative Adversarial Network architectures are implemented. The first of these architectures is the main form of Generative Adversarial Networks (GAN) introduced by Goodfellow [11]. The second of the examined architectures is the Deep Convolutional Generative Adversarial Networks (DCGAN) which are derived from Convolutional Neural Networks and

were introduced by Alec Radford in 2015 [12]. The last of the architectures applied to generate levels is the Wasserstein Generative Adversarial Networks, which is the specialized version of the GAN [13].

Nowadays, deep learning methods have also become popular for procedural level generation, like as rule-based or evolutionary algorithms. Deep learning provides training using large data sets and provides an easier way to obtain features that are difficult to extract with numerical methods. With the ease of use of artificial neural networks, new Generative Adversarial Network architectures are proposed. In this thesis, the training data required to train GAN, were created using a rule-based Angry Birds level generator called Iratus Aves [14]. After the training levels were generated, optimizations like encoding were implemented on newly generated levels. These optimizations helped to improve the outputs of the GAN. Implementations made within the thesis have shown that GANs can also be used to generate levels procedurally in physics-based games such as Angry Birds. Since the generated levels are used in a physics-based game, the generated levels were evaluated according to performance criteria such as stability, playability, linearity, density, etc. As a consequence of the level expressivity analysis, it has been shown that Deep Convolutional Generative Adversarial Networks can be used successfully to generate levels of Angry Birds.

ÇEKİŞMELİ ÜRETİCİ AĞLAR İLE ANGRY BIRDS BÖLÜMLERİNİN ÜRETİLMESİ

ÖZET

İçerik oluşturma, üretici insanların karşılaştığı en büyük zorluklardan biri haline gelmiş durumdadır. Bu işlem üretilen içeriklerin kullanılacak olan ortamlara uygun olmasından, üretim sürecine ve içeriğin özgün olmasına kadar pek çok koşulu da bereberinde getirmektedir. İçerik üretimi bilimden sanata, sanattan teknolojiye pek çok alanda önemli bir role sahiptir.

Günümüzde içerik üretimi özelleştirilerek prosedürel haliyle pek çok alanda kullanılmaktadır. Bilgisayar grafikleri, yapay zeka, insan-bilgisayar etkileşimi gibi pek çok bilgisayar bilimi alanında prosedürel içerik oluşturma yöntemleri sıklıkla kullanılmaktadır. Genel olarak prosedürel içerik oluşturma, manüel içerik üretmek yerine algoritma ve parametrelerin kullanılarak içeriğin oluşturulması olarak tanımlanmaktadır. Video oyunlarında içerik, çok çeşitli ürünlere tekabül edebilir. Ses efektleri, müzik parçaları, dokular, materyaller, nesne modelleri, açık dünya haritası, düşman yerleşimi, karakter gelişimi ve daha fazlası video oyun içeriklerine örnek olarak verilebilir. Prosedürel İçerik Oluşturma (PİO) için kullanılan algoritmalardaki en önemli nokta, raslantıdan yararlanılmasıdır. Algoritma için kullanılan raslatı fonksiyonları sayesinde verilen parametrelerle çok çeşitli çıktılar üretilebilmektedir. Oluşturulabilecek bu çeşitlilk her oyunda farklı olacağından oyunun zengin bir içeriğe sahip olmasından tekrar oynanabilirliğini arttırmaya kadar pek çok fayda sağlayabilir.

Teknolojinin henüz çok ilerlemediği dönemlerde, kaynaklar ve işlem yetenekleri kısıtlı olduğu için PİO yöntemine sıklıkla başvuruluyordu. Bu dönemlerde oyun üreticileri de kısıtlı hafizaya sahip cihazlarda oyunların çalışabilmesi ve oyun içeriklerinin çeşitlendirilmesi için PİO yöntemlerini kullanarak oyun içeriklerini üretiyorlardı. 1980 yılında üretilen Rogue isimli oyun, PİO kullanarak oluşturulmuş dikkat çeken sıra tabanlı oyunlardan biriydi [2]. Rogue içindeki bölümler, hafıza kısıtları olduğu için algoritma kullanılarak oluşturulmuştur. Aslında bu oyun o kadar popüler oldu ki, video oyunlarının benzersiz bir alt kategorisini ortaya çıkardı: "Roguelike" [3]. Kaynakların çok sınırlı olduğu bu dönemlerde PİO, oyun geliştiricileri tarafından oyunların her oynanışta farklı olması için de kullanılıyordu. Bunun en güzel örneğini Blizzard oyun firması tarafından 1996 yılında geliştirilen Diablo oyununda görebiliyoruz. Diablo oyununda, oyuncular prosedürel olarak tamamen rastgele oluşturulmuş bir oyun haritasıyla karşılaşıyorlardı.

Günümüzdeki oyunlarda PİO, çevrimiçi ve çevrimdışı olarak iki farklı yöntemle kullanılmaktadır. Çevrimiçi yöntemde oyun içeriği, oyun oynanırken oluşturuluyor. Çevrimiçi olarak kullanılan PİO yöntemlerine, oyunun oyuncunun oynayış şekline

göre zorlaşması veya farklı içeriklerin oluşturulması örnek olarak verilebilir. Çevrimiçi olarak PİO oyunara en güzel örnek Electronic Arts firması tarafında yayınlanan Spore [4] adlı video oyunudur. Bu oyunda oyuncular karakterlerini oyun içerisinde buldukları diğer içeriklerle birleştirerek diledikleri gibi karakterler oluşturabilmektedir. Oyuncuların istedikleri şekilde tasarladıkları karakterlerin tüm animasyonları da PİO yöntemleri ile oluşturulmaktadır [4]. Çevrimdışı yöntemde ise oyun içeriği oyunun geliştirme sürecinde oluşturuluyor ve oyuna dahil ediliyor. Çevrimdışı olarak üretilen içerikler oyuna doğrudan dahil edilebildiği gibi tasarımcıların süzgecinden geçtikten sonra da değişiklik yapılarak oyuna eklenebiliyor. Hello Games tarafından geliştirilen No Man's Sky [5], çevrimdışı PİO yöntemi kullanılarak içeriği oluşturulan oyunlara örnek olarak verilebilir. No Man's Sky, prosedürel olarak oluşturulmuş sonsuz bir galakside keşif ve hayatta kalma oyunudur [5].

PİO için farklı çeşitlerde algoritmalar tercih edilmektedir. Kural tabanlı yöntemler olduğu gibi evrimsel algoritmaları kullanan yöntemler ve makine öğrenmesine dayalı yöntemler de kullanılabiliyor. Bu algoritmaları aramaya dayalı yaklaşımlar ve yapıcı yaklaşımlara olarak iki gruba ayırabiliriz [6]. Yapıcı algoritmalarda kullanılan işlemlerin üreteceği içeriğin, kullanıma uygun olması beklenir ve elde edilen çıktı bu haliyle kullanılır. Kurallar kümesi oluşturularak elde edilen algoritmalar bu yapıcı olarak isimlendirilen grupta yer almaktadır. Aramaya dayalı algoritmalar ise çok daha farklı bir yapıdadır. İsimlendirilme şeklinde de belirtildiği gibi üretilen içeriğin en iyi olanının aranılması üzerine kurulmuşlardır. Bu yöntemde içerik, algoritma tarafından üretilmekte, test edilmekte ve uygunsa kullanılmaktadır. Uygun değilse de yeni bir içerik tekrar üretilmekte ve bu işleme, en uygunu bulana kadar devam edilmektedir. En uygun içeriği belirlemek için de bir uygunluk fonksiyonu kullanılmaktadır. İçeriğin olması istenilen kriterlerine göre hazırlanan bu uygunluk fonksiyonu, oluşturulan içeriğin istenilen kriterlere uygun olup olmadığını kontrol etmektedir. Makine öğrenmesine dayalı algoritmalar ve evrimsel algoritmalar, aramaya dayalı yöntemler için örnek olarak verilmektedir.

PİO yöntemlerinin gelişmesi ve kullanımının yaygınlaşması, video oyun içeriklerinin veri setleri haline getirilmesine ve paylaşılmasına da ön ayak olmuştur. Mario'dan Diablo'ya kadar pek çok video oyunun veri setinin oluşturulması ve paylaşılması buna örnek olarak verilebilmektedir [15]. Veri setlerinin oluşturulması, PİO için makine öğrenmesi yöntemlerinin kullanılmasına da yardımcı olmuştur. Bu sayede hem denetimli hem de denetimsiz makine öğrenmesi yöntemleri PİO için de kullanılabilir hale gelmiştir. Gelişen teknolojiyle birlikte derin öğrenme yöntemleri de PİO yöntemi olarak kullanılmaya başlanmıştır. Yapay sinir ağları kullanılarak oluşturulan üretken modellerden Değişken Oto Kodlayıcılar [16] ve Çekişmeli Üretici Ağlar, [11] PİO için kullanılabilen yöntemlerdendir. Çekişmeli Üretici Ağlar, temelde iki farklı ağdan meydana gelmektedir. Bunlardan ilki üretici ağ, diğeri ise ayrıştırıcı ağdır. Üretici ağın görevi, ayrıştırıcı ağ'ın ayrıştırma kontrolünden geçebilmesi için gerçek verilerden ayırt edilemeyecek kadar gerçekçi sahte veriler üretmektir. Ayrıştırıcı ağın görevi ise, gerçek verilerle üretici ağ tarafından üretilen sahte verileri birbirinden ayırmaktır. Çekişmeli üretici ağlarda bulunan bu iki yapay sinir ağının birlikte eğitilmesi gerekmektedir. Üretici ağın ve ayrıştırıcı ağın eğitimlerinde geri bildirim yöntemi kullanılmaktadır. Birbirlerinden aldıkları geribildirimleri, dönüşümlü olarak tek bir yinelemede kullanarak eğitimlerini tamamlamaktadırlar. Üretken modeller genel olarak denetimsiz makine öğreniminin bir dalıdır. Fakat üretken modellerden

biri olan çekişmeli üretici ağların yapısını oluşturan mimarilerin eğitimi, denetimli makine öğreniminin bir alt dalı olarak kabul edilir. Bu tezde prosedürel olarak içerik oluşturmak için çekişmeli üretici ağ yöntemleri incelenmektedir.

Angry Birds, Rovio Entertainment tarafından geliştirilen ve mobil cihazlar için 2009 yılında piyasaya sürülen fizik tabanlı popüler bulmaca oyunudur [7]. Bu oyun kısa zamanda büyük bir başarı elde etti ve 2014'te 2 milyar indirmeye ulaşmıştır [17]. Angry Birds de diğer fizik oyunları gibi PİO çalışmalarında örnek olarak kullanılmıştır [8]. Angry Birds üzerinde çalışmaların artması ile Angry Birds için bölüm oluşturma, oyuncu ajan oluşturma gibi yarışmalar düzenlenmeye başlanmıştır [18]. Bu tür fizik tabanlı oyunlarda kullanılan fizik kısıtlamaları, prosedürel bölüm oluşturma için birçok sorun oluşturmaktadır. Bu durum oyun için oluşturulan bölümlerin kalitesini değerlendirmeyi de zorlaştırmaktadır. Fizik tabanlı oyunlardaki çok sayıdaki durum ve büyük hareket alanları nedeniyle, oluşturulan seviyelerin oynanabilirliğini/çözülebilirliğini doğrulamak özellikle zordur [9].

Bu tezde, Angry Birds oyunu için prosedürel olarak bölüm oluşturulmaya (PBO) çalışılmıştır. Angry Birds oyunu ticari bir oyun olduğu için Rovio şirketi tarafından kaynak kodları paylaşılmamaktadır. Bu sebeple üretilen bölümler, Lucas Ferreira tarafından geliştirilen Science Birds isimli Angry Birds klonu üzerinde denendi ve görselleştirilmiştir [10]. Tez kapsamında bölüm oluşturmak için üç farklı Çekişmeli Üretici Ağı mimarisi gerçeklenmiştir. Bu mimarilerden ilki Goodfellow tarafından tanıtılan Çekişmeli Üretici Ağların en yalın halidir [11]. İncelenen mimarilerden ikicisi, Derin Evrişimli Çekişmeli Üretici Ağlardır. Yalın Çekişmeli Üretici Ağ yapısının geliştirilmiş bir versiyonudur. Denetimli ve denetimsiz makine öğrenmesi olarak her iki şekilde de kullanılan evrişimli sinir ağlarından türetilerek oluşturulmuş Derin Evrişimli Çekişmeli Üretici Ağlar Alec Radford tarafından 2015 yılında tanıtılmıştır [12]. Bölüm oluşturmak için uygulanan mimarilerden sonuncusu da Çekişmeli Üretici Ağların özelleşmiş hali olan Wasserstein Çekişmeli Üretici Ağlarıdır [13].

Günümüzde derin öğrenme yöntemleri de kural tabanlı veya evrimsel algoritmalar gibi prosedürel içerik oluşturma için popüler hale gelmiş ve kullanımı artmıştır. Derin öğrenme, büyük veri kümeleri kullanarak eğitimin verilmesini ve sayısal yöntemlerle çıkarılması zor olan özniteliklerin daha kolay bir şekilde elde edilmesini sağlamaktadır. Yapay sinir ağlarının kullanımının kolaylaşmasıyla birlikte literatüre yeni eklenen Çekişmeli Üretici Ağ mimarileri de eklenmeye başlamıştır. Bu tezde çekişmeli üretici ağların eğitilebilmesi için gerekli olan eğitim bölümleri, Iratus Aves isimli kural tabanlı bir bölüm üreticisi kullanılarak oluşturulmuştur [14]. Tez kapsamında eğitim bölümü üretildikten sonra bu bölümler üzerinde belirli optimizasyonlar ve kodlama yapılarak gerçeklenen Çekişmeli Üretici Ağların çıktılarının iyileştirilmesi hedeflenmiştir. Tez kapsamında yapılan gerçeklemeler, Angry Birds gibi fizik tabanlı oyunlarda prosedürel bölüm oluşturabilmek için Çekişmeli Üretici Ağların da kullanılabileceğini göstermiştir. Oluşturulan bölümler, fizik tabanlı bir oyun içinde kullanılacağı için sağlamlık, oynanabilirlik, lineerlik vb. gibi kriterlere göre değerlendirilmiştir. Yapılan analizler sonucunda Derin Evrişimli Çekişmeli Üretici Ağların, Angry Birds gibi fizik tabanlı oyunların bölümlerini üretmek için başarılı bir şekilde kullanılabileceğini göstermiştir.

1. INTRODUCTION

This chapter explains the theoretical background of the thesis from general topics to specific topics, and briefly introduces the purpose of the thesis and literature research.

1.1 Purpose of Thesis

In recent years, there has been a growing interest in developing procedural content generators for various forms of game contents. Kerssemakers et al. stated that cellular automata, artificial evolution, subdivision of fractals and artificial agents are some of the base concepts used in these generators [19]. The aim of procedural content generation (PCG) is regularly to imitate a human creator as well as could be expected. For sure, there has been a lot of examination in how to create acceptable outputs in a wide range of game components, from little components like textures to levels and terrains to journeys and stories. With a few exceptions, PCG is employed in games to smoothly replace or augment human-generated material. Lots of modern games which incorporated with PCG are constructed as if all of the contents could have been generated by humans, given sufficient development time.

Procedural level generation (PLG) which is one of the most common variants of PCG, has been used in a wide range of digital games [20]. PLG is defined as the procedural development of game levels without human effort and it often needs numerous separate elements of a level to be produced in a dependent manner [19]. PLG allows us to create a huge number of levels in a short amount of time. This can drastically minimize the time it takes to develop a game and also the amount of memory resource it requires to run generated level [21]. Furthermore, PLG can also give unique and distinctive gaming experiences based on the user's playing style [22].

Popularity of implementing PCG's is increased rapidly during the 2000s when it was used to gaming visuals like textures [23]. Since that time, huge amount research on PCG have been published in both the academic and the industry [24]. In addition,

various competitions have been held at international conferences in recent years. The Platformer AI Competition, the Mario AI Level Generation Competition and the AI Birds Level Generation Competition are just a few examples. Deep learning and machine learning methods have been difficult to implement in video games due to the lack of an baseline standards for content generation. Nonetheless, recent efforts to build a game corpus, have encouraged several new initiatives [25], [15]. As a result, a new domain, Procedural Content Generation by Machine Learning (PCGML), has established [26]. PCGML may be utilized in a broad range of applications, including procedural generation, co-creation, mixed-autonomous design and analysis of contents. Several studies [27], [28] have proposed to PCG with machine learning and deep learning methods.

The goal of this research is to investigate different forms of Generative Adversarial Network (GAN) architectures [11] to develop new game levels that look like those produced by game designers. GANs are unsupervised deep neural networks that have demonstrated exceptional potential in replicating elements of images from a training corpus. In order to generate levels, three different generative adversarial network architectures are implemented. The first one of these architectures is the main form of GAN. The second one of the examined architectures is the DCGAN derived from convolutional neural networks, were introduced by Alec Radford in 2015 [12]. The last one of the architectures applied to generate levels, is the Wasserstein Generative Adversarial Networks, which is the specialized version of the GAN [13]. DCGANs and WGANs are also types of deep neural networks which can produce imitated data with the same characteristics as the training data. In this study, generation of new training data set is completed by using a rule- based level generator [14]. Newly generated training data set is used to demonstrate the usefulness of DCGANs, GANs and WGANs on images of game levels. We evaluate the generated levels based on a set of performance criteria such as; stability, linearity, density, difficulty and playability. To analyze expressivity of the generated levels, we also use an agent-based testing method.

1.2 Theoretical Background

1.2.1 Procedural Content Generation

Khaled et al. (2013) explain PCG as the production of game content using algorithms with little or no manual interaction [29]. A graphics researcher, an industry game designer, and an academic researchers are reluctant to commit on even what "content" is, much alone which generating techniques are attractive. Togelius et al. (2011) described difficulties of term in content as for this to happen, PCG has been attempted by far too many people with far too many diverse viewpoints [1].

1.2.1.1 Procedural

Procedural can be described as a series of events carried out in a specific order or a certain way. It is possible to argue that the procedural side of PCG expresses the desire to describe the way something is formed as a series of procedures that will produce the same outcome when fed the same set of conditions. To obtain such a set of instructions or processes, every designer or developer working on a specific portion of the game, should think on how they proceed on certain a task. A more exploratory alternative, in which the designer or developer comes up with a basic set of instructions and works their way towards the goal, would be an option.

1.2.1.2 Randomness in PCG

The term procedural implies that it has nothing to do with randomness. Although the use of randomly generated data to initiate the procedural side of PCG is essential for giving it a dynamic, unpredictable, and exploratory feel, randomization is not required for PCG to perform. No Man's Sky [5] is a great example of how deterministic and unpredictable behavior interact. It focuses to have its game world totally procedurally produced on the runtime, with no data kept on disk or in the cloud; everything is generated in real time. However procedural generation do not use randomness in this game, everything generated procedurally by rules of functions.

1.2.1.3 Content

Content is the particular interest of PCG. In video games, content can correspond to a wide variety of products. Sound effects, musics, textures, materials, items, terrains, enemies, characters, and more are examples of video game contents. However for different areas content can be wide range variety. There are no any boundaries on what describes as content or what does not. Limiting the definition of the term may limit what individuals think worthy of creation, and as a result, we may miss out on valuable information to be found in future studies, games, and commercial products.

1.2.1.4 Generation

The generative element of PCG is where the knowledge of how to generate something is put into action to create the desired content. That implies, need to set up a system that can be given a certain set of instructions, preferably parametrizable, and will return a specific outcome. This is also the section that explains how the process of generating anything begins and where the finished product is stored.

1.2.2 History of PCG in Games

In the past, Procedrual Content Generation (PCG) was commonly used to generate sections of the level. This might have been influenced by the memory constraints that game developers faced while game developing in the 1970s and 1980s. However, constraints were not only reasons certainly for PCG, creating variety contents and minimizing time consuming are the other reasons to use PCG in games.

1.2.2.1 Rogue

Rogue is one of the most significant game in terms of procedural dungeon generation as well as a host of other game elements. The game named Rogue, produced around 1980, was one of the remarkable turn-based games created using PCG technique [2]. It is string based game shown in the Figure 1.1.

Because of the memory constraints, Rogue was created using with generation algorithm. In fact, it was so successful that it gave rise to a new subgenre of video games known as "roguelikes." [3].

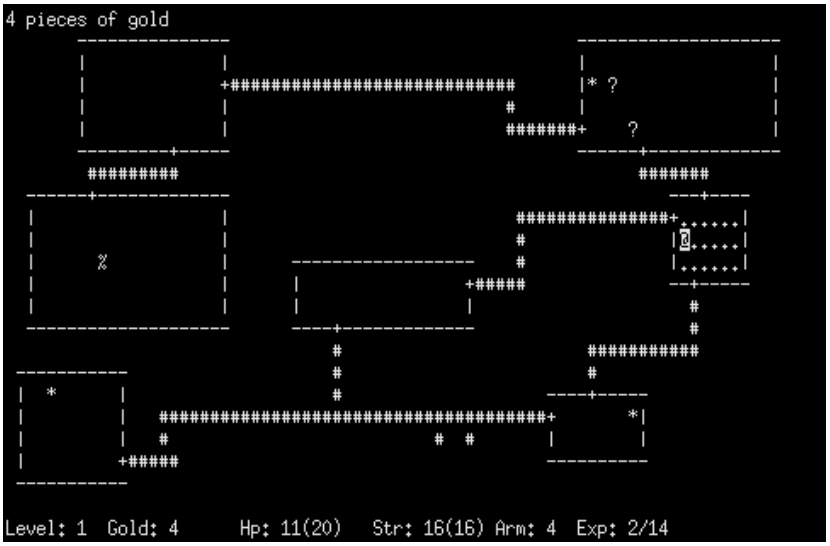


Figure 1.1 : Example level of Rogue [30].

1.2.2.2 Diablo

Diablo which was developed by the Blizzard game company in 1996, may be one of the most popular rogue-like games in recent history, aside from being praised for its dramatic atmosphere and sound design. The developers intended varied dungeons to provide each character a somewhat different experience. The Diablo map generating method turned out to be a combination of static and dynamic contents.

1.2.2.3 Spore

Spore published by Electronic Arts, which was also launched in 2008, is a game in which you govern and evolve a species from its earliest stages as a single cell organism to a space-faring society. In this game, players can create characters as they wish by combining their characters with other contents they find in the game. Rather of having game designers create hundreds of 3-D models of game contents in preparation and program their behavior, he had his team create algorithms to animate the contents that are created by gamers using procedural generation [4]. Not only is the creature's behavior and movement procedurally produced, but the background music is also based on the pieces your character in game generated procedurally .

1.2.3 Angry Birds

Rovio Entertainment's Angry Birds is a popular physics-based game that was published in 2009 for mobile devices [7]. Angry Birds, like other physics games, was

also used as an example in PCG studies [8]. The physics constraints used in such games causes many problems for procedural level generation. This constraints also makes evaluating the quality of the game's produced levels challenging. As Shaker states, one of the most difficult aspects of creating game material is determining the quality of the created artifacts [9]. Due to the large number of situations and large movement areas in physics-based games, it is particularly difficult to verify the playability/solvability of the generated levels.

Since Angry Birds is a commercial game, source codes are not shared by Rovio company. For the presentation and testing of levels, a clone of Angry Birds named Science Birds [10] is used in this thesis. In the procedural level generation study areas, Angry Birds is a very active game domain, with yearly level generating competitions [18].

In this game, the player shoots a limited quantity of birds using a slingshot in order to eliminate all of the pigs in a level as shown in the Figure 1.2. The game level includes a variety of items of various shapes and sizes. Platform, TNT blocks, pigs, and ordinary blocks are the four types of items in this game.

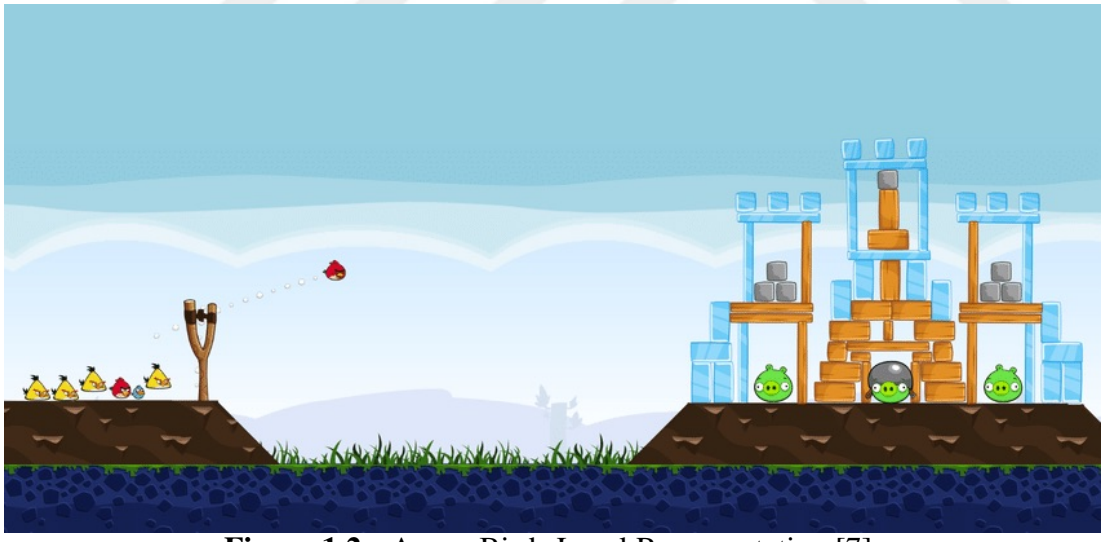


Figure 1.2 : Angry Birds Level Representation [7].

A list of things create the Angry Birds game level. The kind (e.g., little square block can be three different material form), vertical and horizontal coordinates, and angle are all included in each item. Gravity is included throughout the level, and gravity affects the majority of the blocks. Therefore, by taking into account the changes in size and shape of the blocks, they must be stacked properly under gravity.

The development of an Machine learning based level generating approach for Angry Birds faces two major challenges. Firstly, in terms of level design, it has a lot of freedom. Existing deep generative model based level generating algorithms have been used to tile-based gaming domains in which the levels are clearly split into squares, all blocks have the same size or unit size, and the blocks may be configured by providing the level array's index. Hence, current techniques like [31], encode the tile-based level as an image and apply deep generative models for image processing effectively. On the other hand, Angry Birds levels require the use of a huge number of polygons to show the precise stage. The various block sizes might easily result in overlapping blocks, making the level unplayable. Second challenge is that, due to the presence of gravity, the requirements for reaching a playable level are quite tough. A minor mistake might result in the entire level collapsing right after the game begins.

1.2.4 Machine Learning

Artificial intelligence (AI) and computer science collide in the field of machine learning (ML). It aims to enhance accuracy through time, researchers are using data and algorithms to simulate how people learn. ML is an important component of the rapidly growing field of data science.

Machine learning combines statistics together with computer science to enable robots to learn how to perform tasks without being programmed. Algorithms are taught for using statistical methods, to create classifications or predictions, obtaining important insights from data. With the side of computer science, statistical methods understands and processes this dataset in the most efficient manner. The key to achieving better results from ML is to use efficient algorithms and comprehensive datasets.

University of California divides the learning system of a ML algorithm into three parts. [32].

1. **Decision Process:** ML algorithms have been used to make accurate predictions or classifications generally. The program will generate a prediction about a relationship in the data on the basis of some labeled or unlabeled input data.

2. **Error Function:** This function is used to assess the accuracy of the model's prediction. If there are known examples, it may be used to measure the model's accuracy.

3. **Optimization Process:** When the model fits the datasets in the training set better, then to close the gap between the known example and the model's estimate, the weights are changed. This evaluation will be carried out by the algorithm together with tuning procedure again and again. This will update weights on its own until it reaches a specific degree of precision.

1.2.5 Machine learning methods

Machine learning classifiers can be separated into four primary categories.

1.2.5.1 Supervised machine learning

Supervised learning is described as well as the usage of labeled datasets to train algorithms which accurately categorize input or estimate outcomes. When additional data is added into the model, the weights are adjusted until the model is properly formed. During the cross validation phase, this indicates that the model is either very overfit or very underfit. Organizations may utilize supervised learning to address a different type of actual issues on a large scale, like as malicious classification in a operating systems and phishing attacks on emails.

1.2.5.2 Unsupervised machine learning

In unsupervised learning, machine learning algorithms are used to analyze and categorize unlabeled data. These algorithms find hidden correlations or dataset classifications even without manual interaction. It's perfect for data exploration, cross-selling tactics, buyer behavior, image recognition, and pattern recognition because of its ability to detect similarities and dissimilarities in data.

1.2.5.3 Semi-supervised learning

This learning is a good middle ground between supervised and unsupervised learning. It was trained on a small labeled data set for classification, but it was used to extract

features from a bigger unlabeled data set. When labeled data is not enough to train a supervised learning system, this learning method can provide successful results.

1.2.5.4 Reinforcement machine learning

Reinforcement learning is a sort of machine learning that depends on behavior. As similar as to supervised learning, however without the use of sample to train the algorithm. Reinforcement learning trains as it goes through iterations. A series of positive outcomes will be reinforced in order to determine the best suggestion or approach for a given circumstance. Reinforcement learning is similar to playing a game. Artificial intelligence receives a reward or a punishment depending on the decision it makes.

1.2.6 Machine learning models

- **Discriminative Models:** These models are supervised machine learning models. Discriminative models use the training dataset to discriminate between classes using a decision boundary. They're utilized for classification and regression.
- **Generative Models:** These models are useful for unsupervised machine learning models. Initially, from the training dataset, generative models attempt to learn the data distribution. Then, new examples can be created with utilizing this data distribution as an input. The objective may be to create new samples that are indistinguishable from those in the training dataset. Naive Bayes, Deep Belief Network, Variational Autoencoder, and the Generative Adversarial Network (GAN) are some good examples of generative models.

1.2.7 Deep learning

Machine learning includes deep learning as a subcategory. It can be called artificial neural network, is inspired from human brain structures called neurons. Deep learning algorithms cannot be written clearly, it enables learning from data sets. Deep learning algorithms need large data sets and processing power. In these days, with increasing in data resources, the growth of storage units and the developments in computing power, very large data sets are created and these data sets can be easily processed. These developments enable getting results by using deep learning models for many problems

that are tried to be solved with traditional methods in the field of procedural content generation. Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Autoencoders and GAN are the most common examples of the deep learning methods.

The neural network selects the characteristics without the need for human involvement. Neurons, the core asset of neural networks, are the nodes where information is processed. An input layer, hidden layers, and an output layer are main parts of neural networks. The information, which may be pixels from a picture, is sent to the neural network's input layer as an example. There are neurons that represent a digit in the last layer, which is the output layer. And also there are hidden layers between the input and output layers. The layers representation of the neural network can be shown in Figure 1.3.

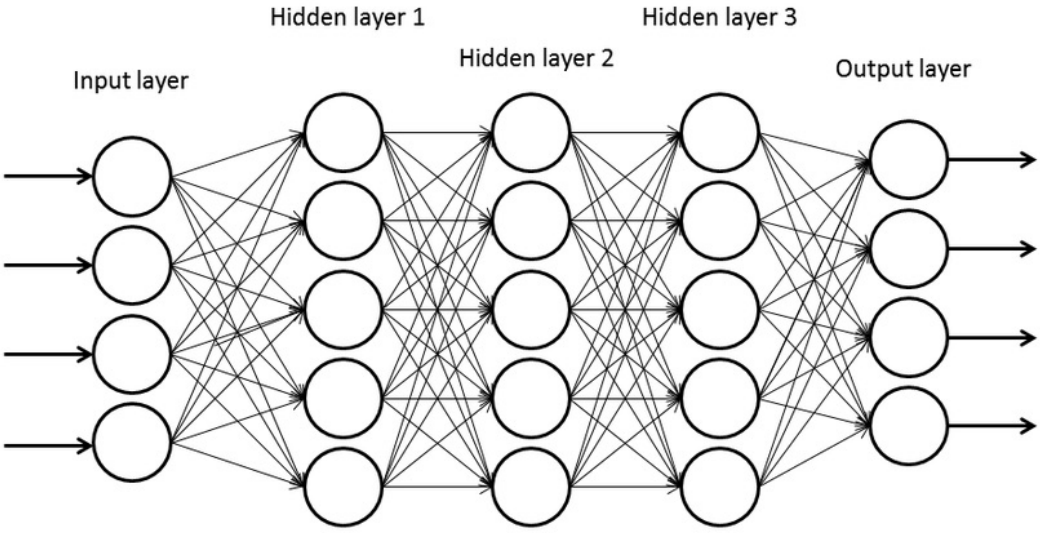


Figure 1.3 : Neural Network Layers [33].

The data is sent between the layers of the neural network, which are connected via connecting channels. Because each channel has a weight, they are called to as weighted channels. It's clear that each neuron has its own bias. This bias value is multiplied by the weighted total of inputs that reach the current neural network layer's neuron. Then this calculation used in activation function which determines whether or not a neuron will be activated by multiplying it by a positive or zero value. Information is sent by every activated neuron to the following layer. This process is repeated until the last hidden layer is reached. The activated neuron in the last layer corresponds to the inputs

that is given in input layer. During the neural network training process, the weights and biases are continually changed.

Deep learning methods are divided into two seperated models as Discriminative Models (DM) and Generative Models (GM). In Discriminative Models, the aim is to determine decision boundary that will allow to separate different classes in the data set. These boundaries are used in DMs to decide a sample is in which class after the training process. In Figure 1.4, a classification example with two different classes is shown.

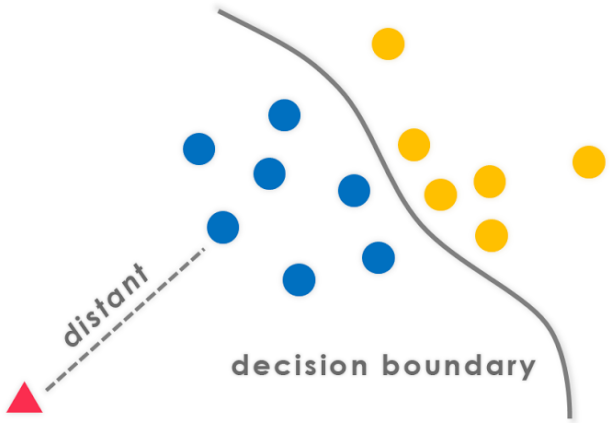


Figure 1.4 : Discriminative models [34].

The model learns to construct a decision boundary between two classes and classifies samples of unknown classes according to which side of the line they fall on. DMs learns the conditional probability distribution over these classes. DMs often used in supervised learning tasks. Supervised learning means, the class of each sample in the data set knowing before. In other words, labels of examples are added in data set. Conditional random fields, scalar vector machines, conventional neural networks, logistic regression, and closest neighbor are examples of discriminative models.

Generative models learn the probability distribution of each class in the data set. In Figure 1.5, an example is given for grouping two classes with unknown labels, taking into account the similarity of the samples. In GMs, after the training process, the class of any sample is determined by looking at the probability that this sample is produced from the distributions generated for each sample group. Since GMs learn probability distributions for the dataset, it enables to generate new samples from the dataset. GMs are very useful in unsupervised learning tasks. Unsupervised learning,

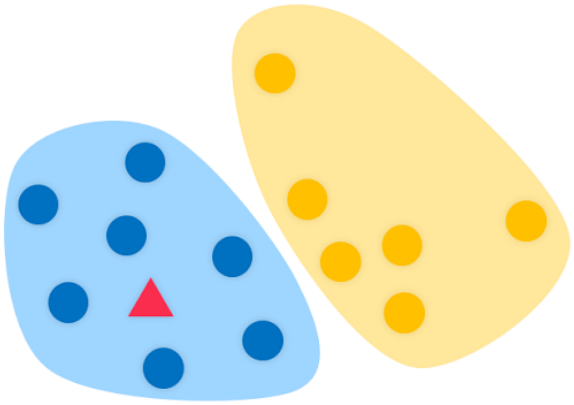


Figure 1.5 : Generative models [34].

where the examples in the data set are known; however, it refers to the situation where the label of the samples is not known. Until recently, it has been observed that discriminative models have more impact in deep learning studies and researchers have turned to generative models with the recommendation of GAN architecture. GANs are a generative model that aims to generate data similar to the existing data set. GAN architecture consists of two competing networks called generator and discriminator.

Generator network takes a randomly sampled latent vector from normal or uniform distribution as input and generates a data. On the other hand, the discriminator network, gives the probability that generated data it takes as input is real or not. In practice, discriminator and generator networks implemented with using deep neural networks. In model training, the parameters of the generator and discriminator networks are updated according to the output of the discriminator network. Finally, a balance is established between the two networks and the discriminator network produces almost the same probability value for both real and generated samples. This shows that the generator network produces samples similar to the real data samples.

Simple GAN architecture can be shown at Figure 1.6. In general, the discriminator is discarded after the training process is complete and the generator is used to generate samples similar to the data set. This approach allows to obtain successful results in sample generation similar to the data set, and also offers solutions to many problems.

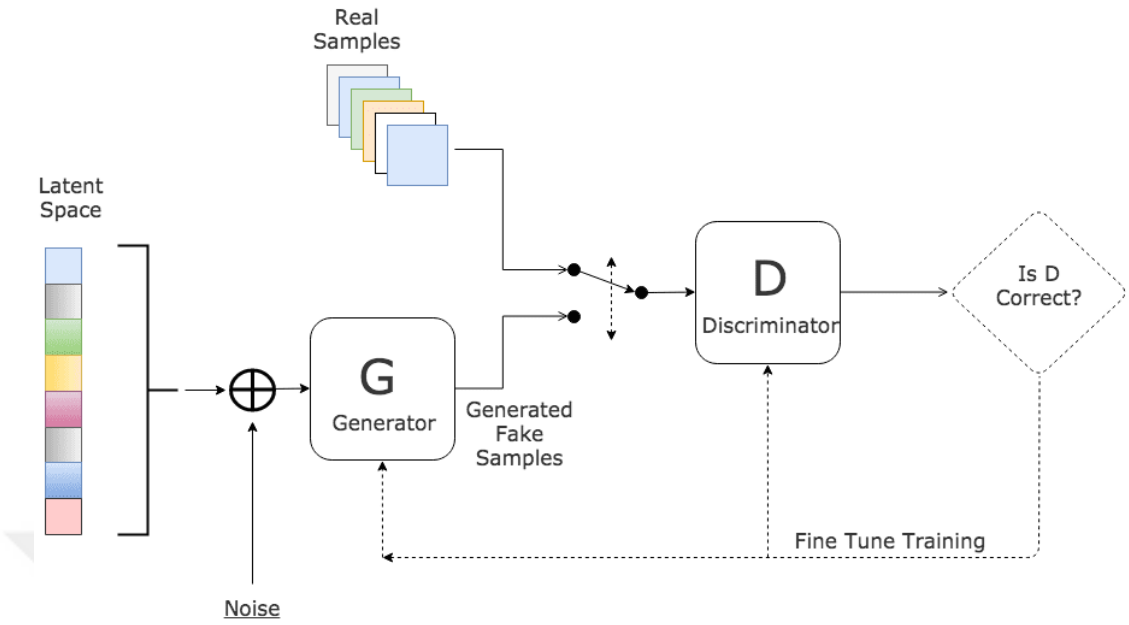


Figure 1.6 : Generative Adversarial Network Flow [35].

1.3 Procedural Content Generation with GANs

In these thesis, several generative adversarial network architectures and ways are studied to handle procedural content generation challenges. These GAN architectures are base generative adversarial networks [11], deep convolutional generative adversarial networks [12] and Wasserstein generative adversarial networks [13].

In general, tile based games are preferred for PCG to handle image encoding and train models easily. Lots of tile based games such as Zelda, Sokoban, Freeway, Mario are used in PCG with GANs studies [31], [27].

There are lots of studies and competitions for generating Angry Birds levels procedurally. In current studies, evolutionary algorithms, genetic algorithms and search based approaches are used to generate Angry Birds levels. Laura et al. introduced evolutionary algorithm to generate Angry Birds structures. [36]. Lucas Ferreira and Claudio Toledo (2014) presented a study using a genetic algorithm (GA) to generate Angry Birds levels. In this thesis, different GAN implementations are used to generate physic based game levels for Angry Birds. Mury et al. (2020) tried

Cycle-Consistent Adversarial Networks (CycleGAN) [37] to generate Angry Birds levels [38]. Takumi et al. preseneted a variational auto encoders and latent variable evolution method to generate Angry Birds levels [39]. However, GAN, DCGAN and WGAN are not implemented before to generate Angry Birds levels. Results indicates that GANs can be used effectively to generate physic based game levels.

Recent GAN structures with enhanced optimization approaches are simple to train, and the results are becoming increasingly accurate. The loss functions and optimizers are the main changes between the training, validation, and testing stages. The technique for preparing the dataset for training is same for GAN structures. For each problem, the technique for preparing the dataset to obtain training is different. Furthermore, the GAN part appears to be the same, with two separate networks, one serving as the generator and the other as the discriminator. Despite the fact that these two architecture accomplish the same task every time, with the generator attempting to generate fake images and the discriminator attempting to separate the generated fake images from the real images, they are changed from one implementation to the next.

1.4 Literature Review

In the past, Procedural Content Generation (PCG) was commonly used to generate sections of the level. This might have been influenced by the memory constraints that game developers faced while game developing in the 1970s and 1980s. However, constraints were not only reasons certainly for PCG, creating variety contents and minimizing time consuming are the other reasons to use PCG in games.

In 2013, Shaker et al. proposed a grammatical evolution based method for procedurally generated levels in the context of Cut the Rope, a physics-based puzzle game [40] [9]. In this study, algorithm uses grammatical evolution and a heuristic evaluation function to produce levels for this game. In addition, as Shaker et al. stated that they also add an agent for simulations of generated levels. Algorithms uses depth first search in these simulation space to find reachable components and possible actions in game. [9]

Ferreira and Toleado proposed an evolutionary level generator for Angry Birds in 2014 [8]. An estimation of distribution algorithm (EDA) is presented in this study to build levels for physics-based puzzle games using the Angry Birds mechanics. During

the evolutionary process, the proposed EDA retains three probability tables to sample new individuals that encode information about the quantity and positioning of game elements inside the level.

A rule based procedural level generator is proposed by Stephenson and Renz in 2016 [41]. The suggested method generates levels made up of a variety of self-contained structures located throughout a 2D region. Structure protection, occupancy estimate, and general dispersion are all used to determine where target objects should be placed. Several tests were carried out in order to find the best input parameters for creating appropriate levels. The proposed approach provides a broad range of interesting levels, according to the results.

In general, tile based games are preferred for PCG to handle image encoding and train models easily. Lots of tile based games like as Mario is used in PCG with GANs studies [27]. Volz et al. proposed using DCGAN and WGAN to generate Mario levels procedurally in 2018. DCGAN is trained with WGAN algorithm from a level corpus. As Volz et al. stated that after level generation completed, all levels improved with Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [27].

As stated before to generate levels, tile based games are usually preferred If GAN methods will use. Irfan et al. proposed a study which uses DCGAN to generate tile based games levels such as Zelda, Sokoban, Freeway. [31]. Their proposed method generate levels successfully. They also test generated levels with an agent to evaluate playabilities of levels.

In 2019, Abdullah et al. proposed a study for Angry Birds level generator which uses domino effect mechanism called Rube Goldberg Machine (RGM) [42], [43]. RGM generator produce levels which can be completed with one shot. The suggested rule based algorithm procedurally builds a variable structure on top of particular blocks in a preset segment to improve the diversity of created levels.

Recent efforts to create a game corpus, have encouraged several new initiatives like as machine learning methods in procedural level generation. In 2019 Zafar et al. published an Angry Birds game level corpus. As they stated that the total number of levels in corpus is 200, which includes 100 levels from the original Angry Birds

game and 100 levels created by the Baseline level generator [18], [25]. Tile-based encoding is used to create these levels in this corpus.

In 2020, Dewantoro et al. propesed a generator which uses drawn sketches and Cycle-Consistent Adversarial Networks (CycleGAN) together. As authors stated two datasets are used to train CycleGAN: sketch drawings and typical level architectures [28]. The suggested preprocessing strategy utilizing CycleGAN allows Sketch-to-Level Generator to generate levels from random sketch drawings more successfully, according to the proposed results. Nonetheless, the generated levels have the potential to become unstable, rendering them unplayable and not exactly replicating the original sketch.

Deep generative models have gained popularity as a method for automating level generation. In 2021, Tanabe et al. proposed a Angry Birds level generator which uses sequential VAE [39]. As authors stated that Angry Birds levels encoded sequentially and processed as text. After level generation is completed then latent variable evolution process is applied with an AI agent to keep levels stable and natural as possible as.

1.5 Tools and Technologies

- **Language:** This project was developed with Python version 3.7.13. Python is an object-oriented programming (OOP) language that converts code into machine code using an interpreter. It is one of the world's most widely used high-level languages. It is commonly used in web development, desktop application development, embedded software development, and also significantly data science applications.

- **Framework:**

    1. Tensorflow 2.8.0 has been used to develop this project. TensorFlow is a free, open-source machine learning software that may be used from beginning to end. It features a robust ecosystem of tools, libraries, and community resources that enable academics to improve the state-of-the-art in machine learning and developers to swiftly build and deploy ML applications. [44].
    2. Keras is an open sourced deep learning framework. In this project Keras layers packages is used and imported from Tensorflow framework. Keras is

not a machine-centric API, but more a human-centric one. Keras focuses on best practices for decreasing cognitive load, such as offering consistent and easy APIs, reducing the amount of user activities required for common use cases, and delivering clear and meaningful error signals. It includes extensive documentation as well as development instructions [45].

- **Environment:** In this project, all technologies such as the Python programming language, Tensorflow framework, machine learning libraries, and GPUs such as the NVIDIA Tesla T4 GPU were accessed through Google Colab. Google Colab provides a Google Drive-based environment that allows us to develop, execute, save, and share code. All training data set is used with Google Colab's Google Drive connection property. Google Colab is essentially a notebook made out of cells, similar to Jupyter Notebook. In those cells we can contain code, images, and text. Colab connects project to the cloud-based runtime environment. As a result, the Python code is run without the need for anything to be installed locally. As a running hardware accelerator, it initiate GPUs, CPUs, and tensor processing units (TPUs).

  A clone of Angry Birds, called Science Birds [10], is used for the presentation and testing of levels. Science Birds project runs on Unity [46] game engine. In addition, for agent tests Angry Birds competition [18] AI agents is used. This agent is console application and developed in Java [47] environment. Agent application runs with Unity application together. Generated levels can be played automatically by this AI agent.

- **Libraries:** Third-party libraries used in this project such as NumPy, Glob, Python Imaging Library (PIL), and Matplotlib. In data science applications, NumPy is one of the most widely used libraries for scientific computation. It's a third-party library that offers multidimensional arrays and matrices, as well as lots of mathematical, basic, and advanced programming operations. The function glob (shortly global) retrieves all file paths that fit a pattern. The glob module is included in the Python standard library and is quite useful. Matplotlib is a Python module for making static, animated, and dynamic graphics. PIL is also a third-party library for image processing that allows us to open, alter, and save pictures.

1.6 Thesis Overview

The chapters of the thesis are summarized in this section. The first chapter covers theoretical background as well as tools and technology. The theory underlying the project is presented in the theoretical background section, which goes from general to specific. The tools and technologies used in this project are discussed under the tools and technologies section.

Generative Adversarial Networks are explained with details in Chapter 2. GAN's structure and functioning mechanisms are described in Chapter 2. The training of GAN is explained with more detail in this chapter.

Chapter 3 is composed Angry Birds level generation. Related studies and approaches concerning the application are briefly detailed in the literature review section. The dataset used in training and testing for the application is described in the dataset section, as well as the pre-processing techniques utilized in the dataset. The model used in training and testing for the application is discussed in the neural network architecture section.

The results are presented in Chapter 4. The outcome of Chapters 3 is presented and described in this section. In addition, here is a comparison of the outcomes produced from various models and approaches.

The Chapter 5 is dedicated to the conclusion. The previous chapters findings are detailed here.

2. GENERATIVE ADVERSARIAL NETWORKS

This chapter explains general information about GANs. In this thesis, three different GANs implemented therefore these GANs are explained in details.

2.1 Generative Models

In generative models, the goal is to generate synthetic samples that are close to the existing dataset. In other words, it is to learn the distribution from which the samples are derived from a given data and to generate new samples from this distribution. In general, since the datasets contain a limited number of samples, it is not always possible to learn the dataset distribution efficiently. For this reason, it is tried to model a distribution as same as possible to the actual data distribution. In implementation, neural networks are used to provide the modeled distribution as similar as the real distribution.

2.1.1 Maximum Likelihood Estimation

This chapter focuses on generative models that work on the principle of maximum likelihood estimation; because recently, the most notable approaches refer to this principle. Maximum likelihood estimation; It aims to define the model parameters that best explain the samples in the data set, in other words, maximize the likelihood of the samples in the data set, and predict the probability distribution. If the actual probability distribution is represented by the parameters θ and the parameters θ^* that maximize the likelihood of the samples, then θ^* is determined using Equation 2.1 [48]. θ^* parameters is equal to maximum likelihood estimator of θ.

$$\theta^* = \arg\max_{\theta} \prod_{i=1}^{m} P_{model}(X^{(i)};\theta) \tag{2.1}$$

$$= \arg\max_{\theta} \log \prod_{i=1}^{m} P_{model}(X^{(i)};\theta) \tag{2.2}$$

$$= \arg\max_{\theta} \sum_{i=1}^{m} \log P_{model}(X^{(i)}; \theta) \quad \textbf{(2.3)}$$

In Equation 2.1, P_{model} represents the model estimating the probability distribution, $X^{(i)}$ represents each sample in the data set, and m represents the total number of samples in the data set. In practice, these operations are usually performed in logarithmic space for two reasons:

- It provides ease of operation by taking the logarithm of the model and converting it to the sum of the probabilities, rather than the product of the probabilities of the samples. In Equation 2.2 [48], exponential function is used to provide ease of calculation. This considerably reduces the computational load of models with exponential functions. In addition, in the digital environment, it avoids the problem of underflow caused by the multiplication of small likelihood values.
- Converting product functions to sum functions simplifies algebraic expressions for derivatives of likelihood.

Equation 2.3 [48] is obtained by taking the logarithm of the equation in Equation 2.1, which estimates the probability distribution for any data set.

Determining the distribution parameters from which complex datasets is a challenging problem. For this reason, in general, the approximated distribution parameters is tried find and neural networks are used to provide this parameters.

2.1.2 Taxonomy of Generative Models

Recently, fully visible belief networks [49] [50], variational autoencoders (VAE) [51] and GANs [11] are the most remarkable studies working with the maximum likelihood principle and making use of neural networks. Figure 2.1 shows the positions of generative models in the hierarchy when classification is made according to density estimation. Unlike other models the density function in GANs is not explicitly defined and the training process is carried out by sampling from the model.

Generative models which are using the principle of maximum likelihood differ in how they represent or approximate likelihood. Models on the left side of the hierarchy define an explicit density function. In addition, this models try to optimize the likelihood of this explicit function.

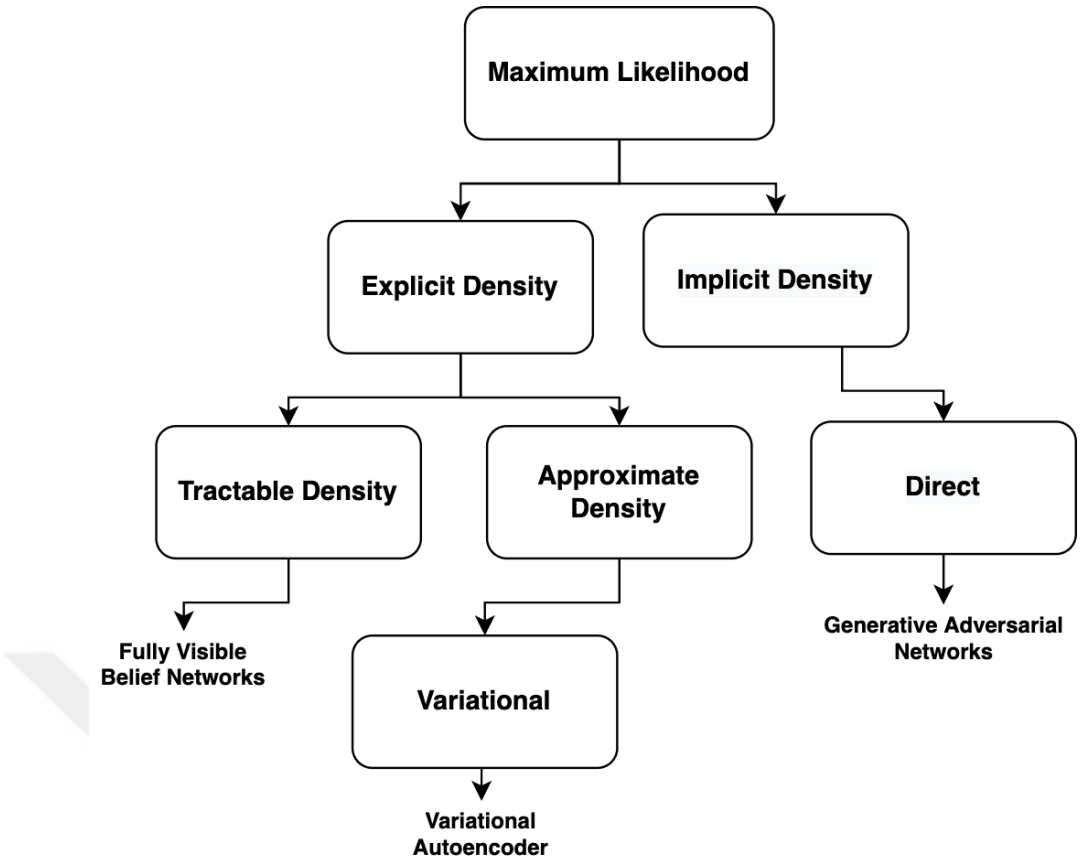


Figure 2.1 : Generative Models Classification [48].

Explicit density models are divided into two parts as the density can be tractable (problems with a polynomial lower limit, this can be solved in polynomial time using a polynomial time technique) or can be approximate (problems with an exponential lower limit, which cannot be solved in polynomial time technique). Models where density is approximated are also classified according to whether they use variational estimation or different approaches such as Markov Chain approach [52] [53] to maximize likelihood. In the implicit density models on the right side of the hierarchy, an explicit probability distribution is not represented for the model and a less direct interaction with the probability distribution is searched. Implicit density models try to obtain a function that can take a sample from the real distribution.

2.1.3 Explicit Density Models

The key challenge with explicit density models is coming up with a model that can represent all of the complexity of the data while being computationally tractable. To deal with this problem, two alternative approaches are used. First one is described in

section 2.1.3.1 creating models with a structure that ensures tractability and the second one as mentioned in section 2.1.3.2, approximations to the probability and its gradients that are manageable.

2.1.3.1 Tractable explicit models

Fully visible belief networks

Fully visible belief networks are probabilistic models that do not contain hidden vectors [49] [50]. These networks transform the joint probability distribution for an n dimensional x vector into the product of one-dimensional conditional probability distributions using the chain probability rule. Assuming the data set consists of images, using the chain rule to find the likelihood of an x image, the x image is transformed into one dimensional product of conditional probabilities as given in Equation 2.4.

$$P_{model}(X) = \prod_{i=1}^{n} P_{model}(X_i|X_1, X_2 \dots X_{i-1}) \tag{2.4}$$

In Equation 2.4, $P_{model}(X)$ is the likelihood of an x image, and $P_{model}(X_i|X_1, X_2 \dots X_{i-1})$ represents the probability of ith pixel given $X_i|X_1, X_2 \dots X_{i-1}$ pixels. Conditional probability distributions are expressed by neural networks in these models [48]. To sum up with, Fully visible belief networks explicitly calculating the likelihood of a sample. Explicitly calculating the likelihood of training data provides a good evaluation criterion and allows the generation of good examples. Since these models runs sequentially, they work slowly. Since these models does not contain hidden vectors in its structure, these models cannot provide some advantages of generative models.

2.1.3.2 Approximate explicit models

Variational autoencoders

Variational autoencoders (VAE); In order to avoid some of the drawbacks that come with designing models with traceable density functions, they are models that use approximate inference networks and can only be trained with gradient-based methods. VAEs include a probabilistic encoder network (approximate inference network) and a probabilistic decoder network. The encoder network $(q_\theta(z|x))$ takes a data sample x as input and embeds it in a discrete latent vector represented by an z. This model is

also called the latent vector generating model. The decoder network $(p_\theta(x|z))$ takes the latent vector z as input and reconstructs the x sample. The loss function used in VAEs is given below in Equation 2.5.

$$L(\theta, \phi; x_i) = E_z \, q_\phi(z|x_i) \log(p_\theta(x_i|z)) - KL(q_\phi(z|x_i)||p_\theta(z)) \tag{2.5}$$

In Equation 2.5, X_i represents ith example of real data distribution, θ and ϕ represent the parameters of the Encoder-Decoder models, respectively. The first part of the equation expresses the reconstruction loss, and the second part of the equation expresses the Kullback–Leibler (KL) [54] divergence, which tries to approximate the prior distribution (for example, a multivariate Gaussian with a diagonal covariance structure) and the Gaussian distribution (for example, the centered isotropic multivariate Gaussian). As seen in Figure 2.2, the parameters of the Encoder and Decoder models are updated by using the reconstruction loss in the image space and the KL divergence in the latent vector space. As an example, Kingma et al. (2016) [55] is one of the most notable recent study using this approach.

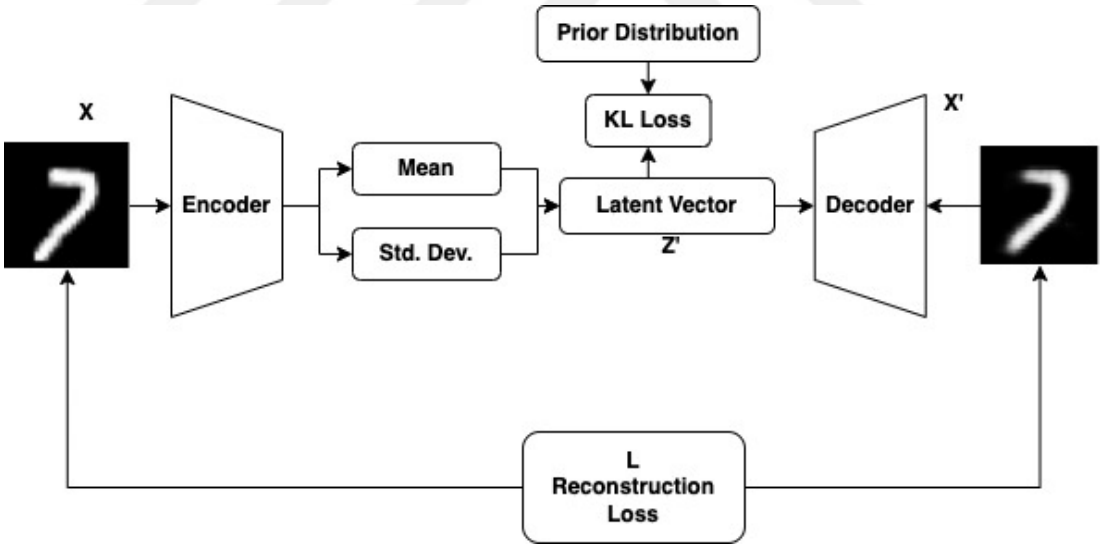


Figure 2.2 : Variational Autoencoder Architecture.

As shown in Figure 2.2, encoder takes the images as input and converts them to the mean and standard deviation vector. The decoder network takes the sampled latent vector as input and reconstructs the image. In architecture, KL loss is used in the latent vector space and reconstruction loss is used in the image space.

Compared to fully visible belief networks, VAEs are easier to implement in practice and faster image generation can be achieved in the testing phase since pixels are not produced sequentially. VAEs contain a latent vector in their structure and images similar to the data set can be produced by randomly sampling from this vector. In addition, since it contains the encoder network in its structure, conversion from image to hidden vector can also be done. The main disadvantage of VAEs is that the generated images are blurry in the models trained using the image dataset and it is not known what causes this situation [56].

Generative adversarial networks

GANs are an instances of generative modeling that uses deep learning techniques. GANs are often considered as the most effective method for training generative models. Generative adversarial networks [11] differ from other generative models in that they contain a meaningful latent vector in their structure and produce quality outputs. GANs are trained by sampling from the model without the need for a density function to be explicitly defined. In other words, the model learns the density function implicitly. In GANs, rather than maximizing an objective function, a two-player game is established between the generator and discriminator networks and the Nash equilibrium of this game [57] is tried to be found. A Nash equilibrium is a state in game theory when a player will stick to their chosen strategy after examining the opponent's approach and see no reason to modify it. However, this function can be more challenging than optimizing an objective function. Therefore, the model is often unstable during the training phase.

2.2 Architecture of GAN

GAN architecture flow is given in Figure 1.6. As mentioned earlier, the GAN architecture consists of two competing parameterized networks called generator and discriminator.

- Generator's purpose is to produce fake data that is indistinguishable from actual data for the discriminator. When creating this fake data, the generator takes into account the discriminator's response. In the discriminator's training phase, these fake data are used repeatedly.

- The Discriminator's job is to recognize the difference between real data and fake data created by the generator. The discriminator penalizes the generator if the created fake data can be separated from the real data. In this case, the discriminator can be considered a classifier. Any deep learning based network design can be selected by looking at the data that the discriminator is attempting to classify.

There is just one data sample for the generator training, which is a random input. This random input might be noise or a noisy image. Looking at the studies, we can see that the noise distribution isn't particularly essential if the random input is noise. Therefore, adopting an easy-to-sample distribution is desirable. For instance, a uniform distribution can be chosen as an easy-to-sample distribution.
The following is the generator training procedure:

1. A low-resolution image is acquired or a random input is sampled.
2. The generator network generates an output from fully randomized uniform distribution
3. After real and false data have been fed into the discriminator network, validity predictions for real and fake data have been generated.
4. The generator's loss function determines the loss based on the discriminator's classification. Generator network is penalized to generate data which can not mislead the discriminator.
5. All generator network is back-propagated by generator loss function. The model parameters are then updated again by the generator's optimizer. To sum up with, back-propagation and optimization are used to update the weights in each layer of the generator network.

There are two separate data samples for discriminator training. The first is known as real data, which in this study refers to levels of game. Other type is known as fake or generated data refers to data which has been generated by generator. Positive examples for discriminator training are real data. Negative examples for discriminator training are fake or generated data.
The following is the discriminator training procedure:

1. The discriminator network attempts to categorize both the real data sample and the generated data sample.

2. The discriminator's loss function calculates losses for both real and fake data samples. The loss function for the discriminator network also penalizes the discriminator network for false classification between real and generated data.

3. The whole discriminator network is back-propagated with regard to the discriminator's loss function, and then the model parameters are updated by the discriminator's optimizer. Back-propagation and optimization are also used to update the weights in each layer of the discriminator network.

Several loss functions and optimizers might be used for particular GAN problems not only in generator training but also in discriminator training.

During the first steps of training, the generator generates fake data, and the discriminator quickly learns to recognize that the data provided by the generator is fake. The generator begins to produce fake data that is indistinguishable to the discriminator as the training progresses. At the end, if the generator training fails, the discriminator will easily recognize real data from generated fake data. However, if the generator training goes well, the discriminator will have a hard time for understanding the difference between generated fake data and real data. As a result, instead of identifying the created fake data as fake data, the discriminator may label it as real data. The discriminator's accuracy will reduce as a result of this situation.

As a differentiable function G represents Generator network, the latent vector z sampled from a normal distribution is given as input to G and x sample $x = G(z)$ is produced. On the other hand, real or fake images are given as input to the discriminator network. The discriminator network gives the probability of a score being real or fake for the image given as input according to the loss function used.

There are two scenarios in the GAN training phase. In the first scenario, $x \in P_{Data}$ samples are taken from the real data set and randomly generated images $x' = G(z) and z \in P_z$, given as input to the discriminator network. In this scenario, the purpose of the discriminator network is to distinguish between real image x and fake

images x' as mentioned before. In other words, the discriminator network is trying to maximize Equation 2.6.

$$\max_{D} V(D) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[1 - \log D(G(z))] \tag{2.6}$$

In Equation 2.6, z represents latent vector randomly sampled from any normal distribution, D and G are represent discriminator and generator respectively. In this scenario, only the parameters of the discriminator network are optimized. In practice, the training of the discriminator network differs. As shown in Equation 2.6, real and fake images can be given as input to the network in a single batch, and also real and fake images can be given as input in different batches. In this case, the parameters of the generator network are optimized once while the parameters of the discriminator network are optimized more than once in each iteration of the training phase.

In the second scenario, random images are generated using the generator network and given as input to the discriminator network. In the basic GAN operation, the discriminator network ends with *sigmoid* therefore it produces an output between 0 and 1. Purpose of generator network is to enable the discriminator network to produce high scores for the generated images. In other words, the generator network tries to minimize Equation 2.7.

$$\min_{G} V(G) = \mathbb{E}_{z \sim p_z(z)}[1 - \log D(G(z))] \tag{2.7}$$

In summary, generator and discriminator networks play the two-player minimax game using the value function D and G given in Equation 2.8. [11]

$$\min_{G} \max_{D} \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[1 - \log D(G(z))] \tag{2.8}$$

Where everything have taken into consideration, as a result of this competition, a Nash equilibrium [57] is established between both networks and the discriminator network produces a probability value close to 0.5 for any image it receives as input when *sigmoid* function is used. This shows that the produced samples provided from the same distribution as the real data set [11].

2.3 Generative Adversarial Network Methods

In this section, the architectures that are used or mentioned during the thesis work are discussed and the strong and weak points of each architecture are stated in summary.

2.3.1 Generative Adversarial Network Architecture

Basic GAN architecture was first proposed by Goodfellow et al. (2014) In this architecture, which is also called the fully connected GAN architecture, fully connected layers are used for both generator and discriminator networks. MNIST named hand written digits dataset [58] was used in the related study for proof of concept. This dataset consist of images therefore these images are converted into one-dimensional pixel arrays in implementation. In Figure 2.3, the fully connected GAN architecture is given.

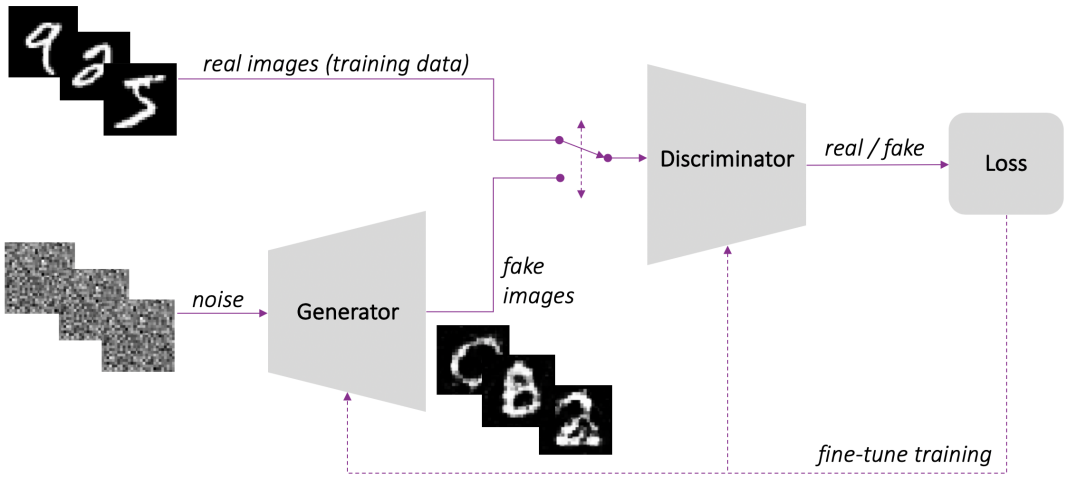


Figure 2.3 : GAN architecture with MNIST dataset [59].

The architecture is built in the structure of multi layer perceptrons. The hand written digit images are given as input after they are converted to pixel arrays. In the architecture, the sigmoid function is utilized in the last layer of the discriminator network and the maxout [60] activation function is used in the remaining layers. On the other hand in the generator network, Rectified Linear Unit(ReLU) [61] [62] and sigmoid activation functions are used mixed. During the training, the discriminator network is optimized several times, while the generator network is optimized once. In this way, when the generator network changes slowly, the discriminator network is kept close to the optimal solution.

This architecture is most significant approach because it is the first study in GANs. However, due to the use of fully connected layers, as the resolution of the image increases, the number of parameters increases and this complicates the training process. For this reason, low resolution (28x28 pixel resolution) image datasets were used in the this study.

2.3.2 Deep Convolutional Generative Adversarial Network

Deep Convolutional GAN (DCGAN) [12] architecture is suggested by Radford et al. (2016). As mentioned earlier, basic GAN architecture uses fully connected neural networks for both generator and discriminator networks. This structure causes several problems in the training process and furthermore increased number of parameters when the image resolution is large. Therefore, the basic GAN architecture does not show good generalization performance on high resolution image datasets. Before DCGAN study Convolutional Neural Networks (CNN), which have achieved great success in supervised learning studies, were used to ensure that they work on high resolution image data sets; however, these CNN studies have not been successful [12].

Radford et al. (2016) proposed to use the fractional strided convolution layer, similar to the deconvolution layer proposed by Zeiler and Fergus (2014), to visualize the properties of layers in CNNs in the generator network [12]. With this layer, the need to use a fully connected layer in the generator network is eliminated, in this way higher resolution image generation is provided. DCGAN study is a milestone in GANs, because of using fractional strided convolution, which was used for the first time in GANs (deconvolution layer is used more because there is not much difference between deconvolution layer and fractional strided convolution layer in recent studies) has become standard in almost all GAN studies.

Generator of the DCGAN architecture is shown as below in the Figure 2.4. In DCGAN architecture, convolutional layers are used in both generator and discriminator network. As Radford et al. (2016) stated that DCGAN was trained at 64x64 resolution using LSUN [63], ImageNet [64] and a customized face image dataset. [12] In DCGAN architecture, the image resolution is higher and the image quality is better when is compared to the basic GAN architecture. However, Radford et al. (2016)

stated that the stability problem continued during the training phase and the mode collapse decreased if the training phase was kept long. [12]

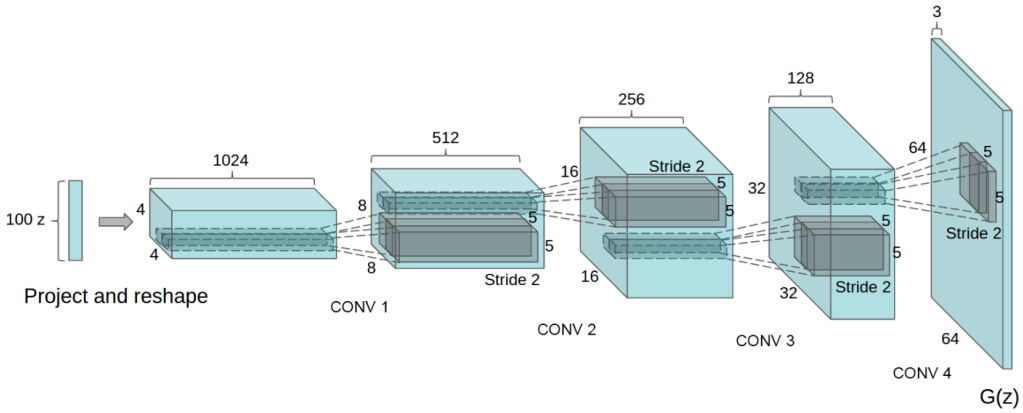


Figure 2.4 : Generator of the DCGAN Architecture [12].

In addition, some critical recommendations were made in the DCGAN study to ensure high resolution modeling and more stable training.

- Firstly, remove all pooling layers and replace them with strided deconvolutional and strided convolutional layers on the generator and discriminator, respectively.
- Secondly, batch normalization [65] layers are used in both generator and discriminator networks. This ensures that both generated and real samples are normalized to have zero mean and unit variance; in other words, it is possible to use similar statistics for produced and real samples.
- Thirdly, it is suggested to use global max-pooling layers instead of fully connected layers, which are usually used at the end of convolutional layers. It is stated that this layer increases stability in deeper models, but causes a slowdown in the convergence rate.
- Finally, using *tanh* function in the last layer of the generator network and ReLU activation function in all remaining layers are proposed. In addition, for each layer in the discriminator, using Leaky Rectified Linear Unit (LReLU) [66] as an activation function is also proposed.

2.3.3 Wasserstein Generative Adversarial Network

Generative Adversarial Networks (GANs) are based on the use of two basic probability distributions. The probability distribution of the generator (Pg), which refers to the distribution from the generator model's output, is one of the main entity. The probability distribution from the real photos is the other essential entity (Pr). The goal of Generative Adversarial Networks is to guarantee that both probability distributions are near to one other, resulting in highly realistic and high-quality output.

Mathematical statistics in machine learning presents three basic approaches for estimating the distance between these probability distributions: Kullback–Leibler divergence, Jensen–Shannon divergence, and Wasserstein distance. In simple GAN networks, the Jensen–Shannon divergence (also known as a typical GAN loss) is the more commonly used mechanism at first. Nevertheless, when working with gradients, this approach has difficulties that might lead to unstable training. As a result, the Wasserstein distance is employed to resolve such situations. The mathematical formula is represented in the Equation 2.9 below.

$$\max_{w \in \mathscr{W}} \mathbb{E}_{x \sim \mathbb{P}_r} [f_w(x)] - \mathbb{E}_{z \sim p(z)} [f_w (g_\theta(z)] \tag{2.9}$$

Arjovsky et al. stated Equation 2.9 as Wasserstein distance formula [13]. The maximum value represents the discriminator constraint in the Equation 2.9. The discriminator is referred to as the critic in the Wasserstein Generative Adversarial Network (WGAN) [13] architecture.

WGAN provides a novel cost function based on Wasserstein distance that has a smoother gradient everywhere instead of introducing noise. WGAN learns whether or not the generator is performing. In below Figure 2.5, there is a plot for both GAN and WGAN. It fills with locations with declining or vanishing gradients for GAN (the red line). Even when the generator is not creating good images, the gradient for WGAN (the blue line) is smoother everywhere and learns better. One of the reasons for this technique is the lack of a sigmoid activation function to limit the data to 0 or 1, indicating reals or fakes. WGAN discriminator networks provide a range of values rather than a single result, allowing it to operate as a critic with less rigidity.

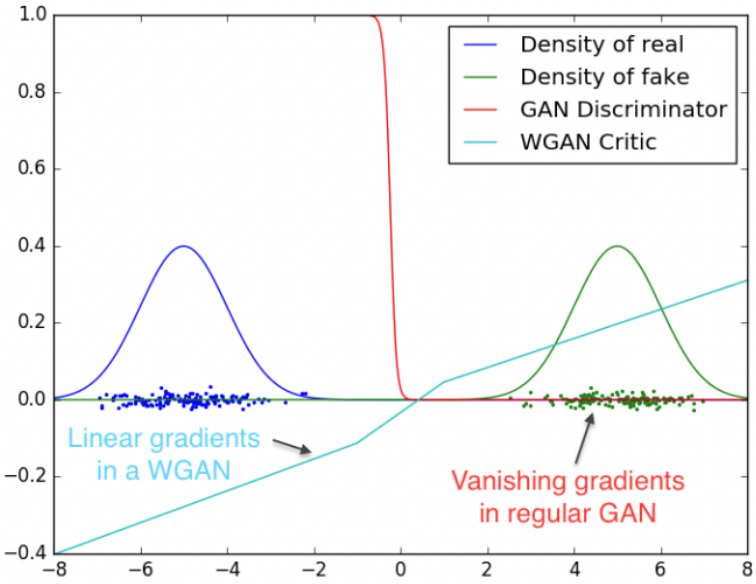


Figure 2.5 : Gradients for both GAN and WGAN [13].

The first half of the equation represents the real data, while the second half represents the generating data. Because discriminator (critic) aims to appropriately separate the data, it tries to increase the distance between the real data and the created data in the above equation. Because the generator wants the created data to be as real as possible, the generator network focuses on reducing the distance between genuine and generated data.

3. GENERATING LEVELS OF ANGRY BIRDS

3.1 Dataset

For Angry Birds level generation, two datasets have been used. The first dataset is Angry Birds corpus which is published by Zafar et al. [25]. As mentioned before corpus contains 100 real Angry Birds levels and 100 generated levels from the baseline generator of competition [18]. For the second dataset, the training levels are generated from the 2018 Angry Birds level generation competition winner IratusAves level generator [14]. To extend the training set, 9800 levels are generated with IratusAves. As a result, the dataset contains 10000 Angry Birds levels. Google Colab integrated Google Drive is used to process this dataset.

3.2 Level Representation

Angry Birds levels have different representations in both datasets. As mentioned before, the first dataset prepared from Angry Birds corpus [25]. In this corpus, as Zafar et al. stated levels are encoded with block colors, and the assigned values as shown in Figure 3.1.

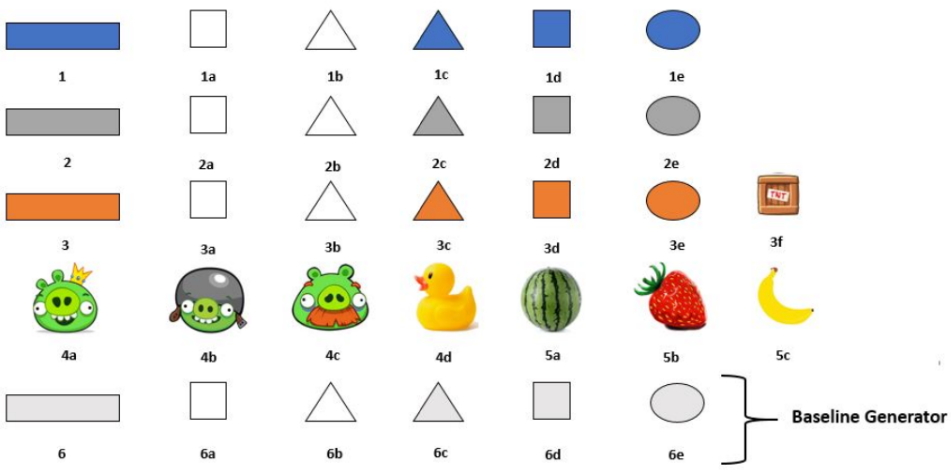


Figure 3.1 : Encoding scheme is proposed by Zafar et al. [25].

Every block is evaluated like a combination of a basic square in which sizes are predefined. After encoding objects in levels with preferred block sizes, matrices are created with these encoded values according to level size. Prepared matrices and sizes can be shown in Figure 3.2.

Data Collection Source	Matrix Size	Levels
Angry Birds Game 1	12 * 15	1-60
Angry Birds Game 1	12 * 30	61-74
Angry Birds Game 1	15 * 15	75-92
Angry Birds Game 1	12 * 30	93-100
Baseline Generator	16 * 32	1-10
Baseline Generator	15 * 20	11-80
Baseline Generator	16 * 32	81-100

Figure 3.2 : Matrices and sizes of the game level in the corpus [25].

Encoding and creation of matrices are shown as in Figure 3.3.

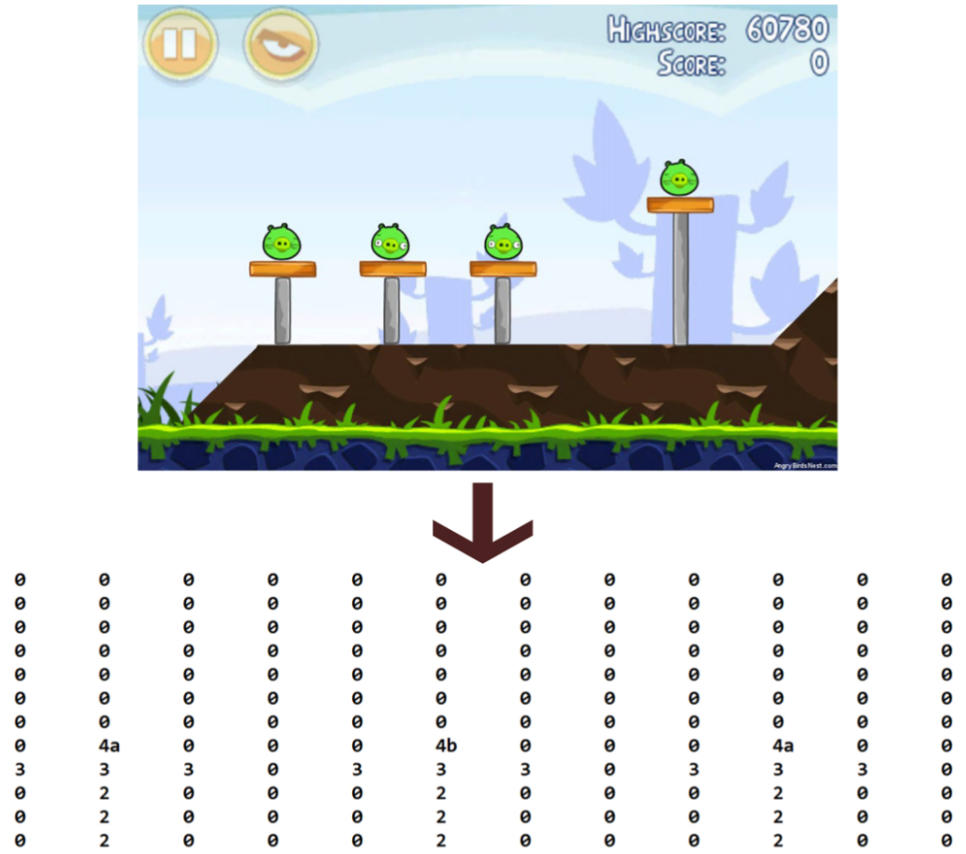


Figure 3.3 : Level is converted to the encoded matrix explained by Zafar et al. [25].

In this thesis, input levels are encoded as RGB values with material values which are listed in Table 3.1.

Table 3.1 : Relationship between object colors and object materials.

Color	Object Material	Object Value
Blue	Ice	1
Gray	Stone	2
Orange	Wood	3
Green	Pig	4
White	Gap	0

Every block is evaluated like a combination of a basic square. In Figure 3.4, example encoding is shown.

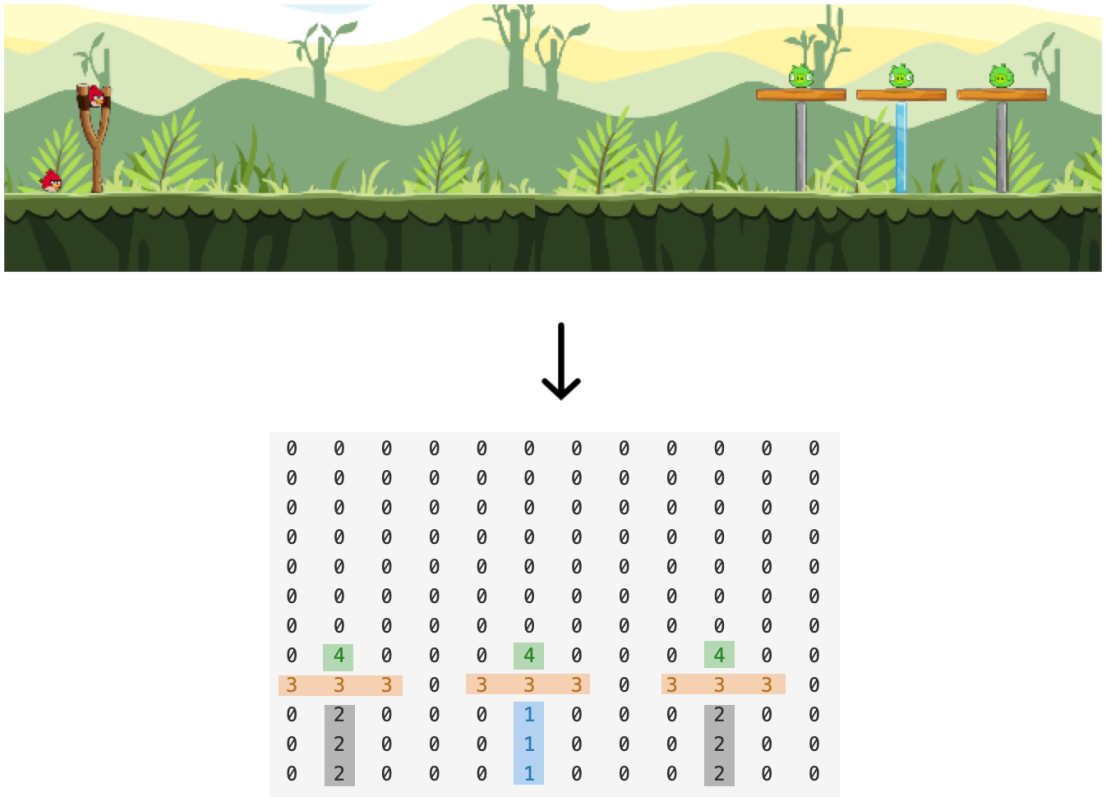


Figure 3.4 : Input level conversion example.

The second part of the dataset used in thesis, generated from IratusAves [14]. This rule base generator generates levels in XML file format. Because this generator is prepared for Angry Birds competition [18], and the competition uses Science Birds clone of

Angry Birds [10]. An example generated level of IratusAves output is shown in Figure 3.5.

```
<?xml version="1.0" encoding="utf-16"?>
<Level width ="2">
<Camera x="0" y="2" minWidth="20" maxWidth="30">
<Birds>
<Bird type="BirdRed"/>
<Bird type="BirdRed"/>
</Birds>
<Slingshot x="-8" y="-2.5">
<GameObjects>
<Block type="RectTiny" material="ice" x="3.1" y="-3.39" rotation="0" />
<Block type="RectTiny" material="stone" x="3.54" y="-3.39" rotation="0" />
<Block type="RectTiny" material="stone" x="3.9800000000000004" y="-3.39" rotation="0" />
<Block type="RectTiny" material="ice" x="4.42" y="-3.39" rotation="0" />
<Block type="RectTiny" material="ice" x="4.86" y="-3.39" rotation="0" />
<Block type="RectTiny" material="ice" x="5.300000000000001" y="-3.39" rotation="0" />
<Block type="SquareTiny" material="ice" x="2.9899999999999998" y="-3.17" rotation="0" />
<Block type="RectTiny" material="stone" x="3.3200000000000003" y="-3.17" rotation="0" />
<Block type="SquareTiny" material="ice" x="4.09" y="-3.17" rotation="0" />
<Block type="SquareTiny" material="ice" x="2.33" y="-2.95" rotation="0" />
<Block type="RectSmall" material="stone" x="2.88" y="-2.95" rotation="0" />
<Block type="SquareTiny" material="wood" x="4.09" y="-2.95" rotation="0" />
<Block type="RectTiny" material="wood" x="2.66" y="-2.73" rotation="0" />
<Block type="RectTiny" material="wood" x="2.66" y="-2.5100000000000002" rotation="0" />
<Block type="SquareTiny" material="wood" x="2.11" y="-2.29" rotation="0" />
</GameObjects>
</Level>
```

Figure 3.5 : Raw and in game representation of IratusAves generated level [14].

In order to make all levels identical in the dataset, a conversion procedure is applied. XML representation of IratusAves's levels converted to the encoded matrix similar to the Angry Birds corpus which is placed in the first part of the dataset. An example conversion from IratusAves raw representation to corpus representation is shown in Figure 3.6.

```
0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  3  0  3  0  3  0  0  0  0  0  0
0  0  1  1  1  1  1  0  0  0  0  0  0
0  0  1  1  0  1  1  0  0  3  0  3  0
0  0  1  1  0  1  1  0  0  1  1  1  1
0  0  1  1  0  1  1  0  0  1  1  0  1
0  0  1  1  4a 1  1  0  0  1  1  4a 1
0  0  2  2  2  2  2  0  0  2  2  2  2
0  0  2  2  2  2  2  0  0  2  2  2  2
0  0  2  2  2  2  2  0  0  2  2  2  2
```

```
<?xml version="1.0" encoding="utf-16"?>
<Level width ="2">
<Camera x="0" y="2" minWidth="20" maxWidth="30">
<Birds>
<Bird type="BirdRed"/>
<Bird type="BirdRed"/>
</Birds>
<Slingshot x="-8" y="-2.5">
<GameObjects>
<Block type="RectTiny" material="ice" x="3.1" y="-3.39" rotation="0" />
<Block type="RectTiny" material="stone" x="3.54" y="-3.39" rotation="0" />
<Block type="RectTiny" material="stone" x="3.9800000000000004" y="-3.39" rotation="0" />
<Block type="RectTiny" material="ice" x="4.42" y="-3.39" rotation="0" />
<Block type="RectTiny" material="ice" x="4.86" y="-3.39" rotation="0" />
<Block type="RectTiny" material="ice" x="5.300000000000001" y="-3.39" rotation="0" />
<Block type="SquareTiny" material="ice" x="2.9899999999999998" y="-3.17" rotation="0" />
<Block type="RectTiny" material="stone" x="3.3200000000000003" y="-3.17" rotation="0" />
<Block type="SquareTiny" material="ice" x="4.09" y="-3.17" rotation="0" />
<Block type="SquareTiny" material="ice" x="2.33" y="-2.95" rotation="0" />
<Block type="RectSmall" material="stone" x="2.88" y="-2.95" rotation="0" />
<Block type="SquareTiny" material="wood" x="4.09" y="-2.95" rotation="0" />
<Block type="RectTiny" material="wood" x="2.66" y="-2.73" rotation="0" />
<Block type="RectTiny" material="wood" x="2.66" y="-2.5100000000000002" rotation="0" />
<Block type="SquareTiny" material="wood" x="2.11" y="-2.29" rotation="0" />
</GameObjects>
</Level>
```

Figure 3.6 : Example conversion from IratusAves [14] raw representation to corpus representation.

After all encoding and conversion process is completed, some improvements are made on the corpus.

1. All objects which have the same material such as ice, wood, stone, pig and gap as shown in Figure 3.1 merged to create only one encoding value. With this manner, the representation complexity is reduced. As a result, five different values can be included for all levels. These values can be seen in Table 3.2.

Table 3.2 : Encoding Values according to the object material.

Material	Encoded Value
Ice	1
Stone	2
Wood	3
Pig	4
Gap	0

2. Levels in corpus were in different sizes. Size standardization was applied for all levels. Small levels enlarged, and filled with empty values like zeros. In this way, the training was not effected, and extra work is prevented before the training is started. A parametric structure was implemented in order to create the dataset levels in the desired size for different training tests.

3. Platforms on either floor or in the air are discarded in the generation process in order to generate flat structures and reduce complexity. Platforms are brown colored grounds as can be seen in Figure 3.3 and Figure 3.5.

4. Finally, the parameters of IratusAves generator are optimized to generate only specific structures. In order to reduce complexity, only rectangle and square objects are allowed to be included in generation. Oval and triangle objects are discarded.

3.3 Used Neural Network Architectures

In this study, three different GAN architectures are implemented to generate Angry Birds levels procedurally. All GAN architectures use the same dataset to train their generators. The first one is a base GAN architecture proposed by Goodfellow et al. [11]. The second one is a DCGAN architecture published by Radford et al. [12]. The last one is WGAN which is proposed by Arjovsky et al. [13].

3.3.1 Generative Adversarial Networks

In the basic GAN implementation, the dataset is preprocessed to train firstly. Level matrix values are associated with colors as in Table 3.1. As mentioned before, to gain performance and to get stable levels, the training levels generated as small as possible. With this manner, firstly the dataset prepared with 16x16 matrices, and the generator trained with 16x16 levels. Small sized levels in the training dataset causes

the generated levels to have dense structures. After an experimental analysis, the optimized level size was found as 28x28 like MNIST [58] for optimization and getting good results.

The neural network architecture parts are explained as follows:

- **Dense Layer:** It is sometimes referred to as the completely linked layer. The weight of the channel is multiplied by the values inside the neurons, and then the bias is added to the multiplication.
- **LReLU:** Leaky Relu has been used as a new activation function to solve the gradient descent problem. Leaky Relu outputs the negative values inside the neuron by multiplying them by a constant, while leaving the positive values untouched. Its formula is given in Equation 3.1. The constants can be small values like 0.2.

$$LReLU(x) = \max(ax, x), \text{ where } a > 0 \tag{3.1}$$

- **Batch Normalization:** It is a regularization function. It improves the training speed by lowering the number of training epochs. It keeps the neural network learning process stable.
- **Flatten:** It flats the input. Batch size is not affected.
- **Reshape:** Inputs are reshaped into the provided form by this layer.
- **Tanh:** The hyperbolic tangent activation function. The method takes any real value as input and returns a value between -1 and 1. The plot can be seen in Figure 3.7.

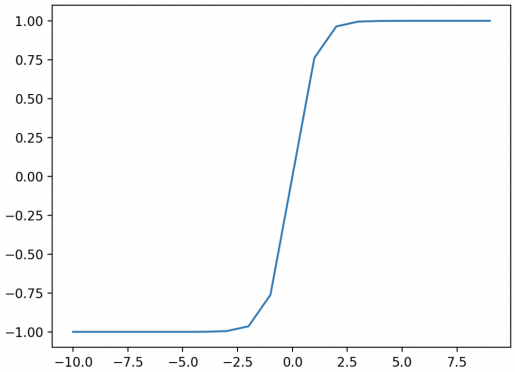


Figure 3.7 : The plot of inputs vs. outputs for the Tanh activation function.

- **Sigmoid:** The sigmoid activation function is also named as the logistic function. The method accepts any real value as input and returns a value between 0 and 1. The plot can be seen at Figure 3.8.

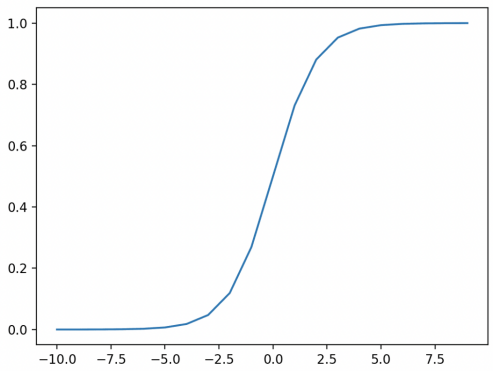


Figure 3.8 : The plot of inputs vs. outputs for the Sigmoid activation function.

Discriminator network runs to maximize the function given in Equation 2.6. On the other hand, generator network runs to minimize the function given in Equation 2.7. Binary Cross Entropy (BCE) is used as the loss function in the architecture, and it is calculated at the output of the discriminator network. For the optimization of the parameters of the two networks, the *Adam* [67] optimization method is used.

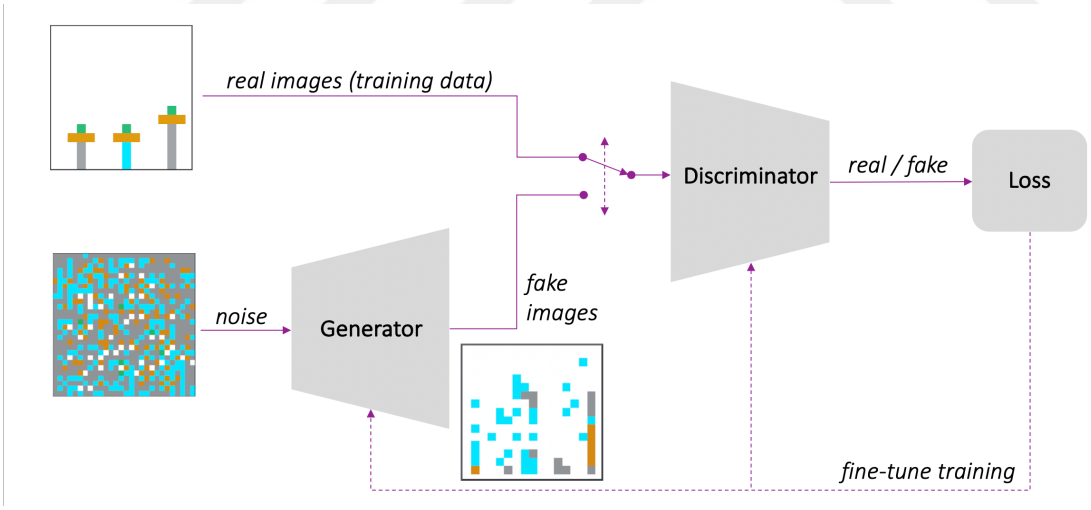


Figure 3.9 : GAN architecture to generate Angry Birds Levels

The created GAN architecture can be seen in Figure 3.9. In this architecture, the generator network contains 4 dense layers. At the last layer of the generative network, *tanh* activation function is used. On the other hand, the other layer uses *LReLU* activation function with 0.2 slope value. A randomly sampled 100 dimensional latent vector in the range [-1,1] from uniform distribution is given to the generator network

as input. An example output is produced from latent vector, can be seen below Figure 3.10.

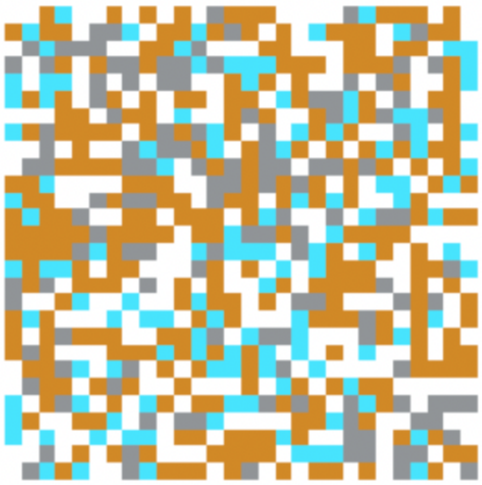


Figure 3.10 : Example 28x28 sized output of untrained generator network of GAN architecture.

Layer details of the generator network can be seen in Table 3.3.

Table 3.3 : Layer details of the generator network.

Layer Type	Output Shape
Dense	256
LReLU (0.2 Slope)	256
Batch Normalization	256
Dense	512
LReLU (0.2 Slope)	512
Batch Normalization	512
Dense	1024
LReLU (0.2 Slope)	1024
Batch Normalization	1024
Dense	784
Tanh	784
Reshape	(28,28,1)

By the way, the discriminator network contains 3 dense layers. At the last layer of the discriminator network, *sigmoid* activation function is used. On the other hand, the other layer uses *LReLU* activation function with 0.2 slope value similar to the generator network. The 3 dimensional (28, 28, 1) matrix is given to the discriminator as input. Layer details of the discriminator can be seen in Table 3.4.

Table 3.4 : Layer details of discriminator network.

Layer Type	Output Shape
Flatten	784
Dense	512
LReLU (0.2 Slope)	512
Dense	256
LReLU (0.2 Slope)	256
Dense	1
Sigmoid	1

3.3.2 Deep Convolutional Generative Adversarial Networks

This architecture is proposed in 2016 by Radford et al. [12]. Generator network contains one dense layer, three fractionally-strided convolution layer (upsampling). At the last layer of the generative network, tanh activation function is used. On the other hand, the other layer uses LReLU activation function. Randomly sampled 100 dimensional latent vector in the range [-1,1] from uniform distribution is given to the generator network as input. An example output is produced from latent vector, can be seen below Figure 3.11.

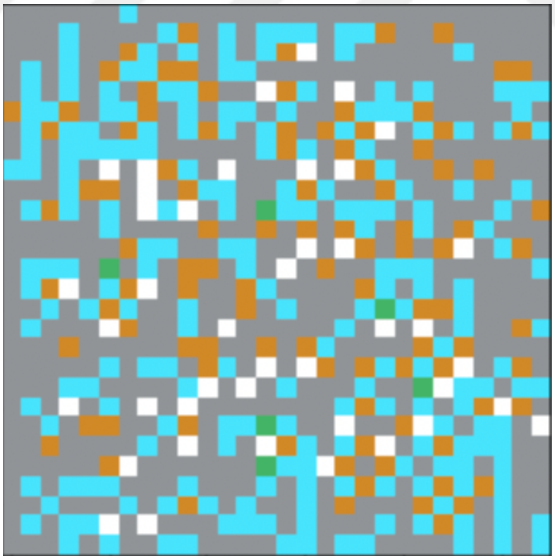


Figure 3.11 : Example 28x28 sized output of untrained generator network of DCGAN architecture.

Layer details of generator network and discriminator network can be seen in Table 3.5 and Table 3.6, respectively.

Table 3.5 : Layer details of DCGAN generator network.

Layer Type	Output Shape
Reshape	(7,7,256)
Dense	256
LReLU (0.2 Slope)	(7,7,256)
Batch Normalization	(7,7,256)
UpSampling	(7,7,256)
LReLU (0.2 Slope)	(7,7,256)
Batch Normalization	(7,7,256)
UpSampling	(14,14,64)
LReLU (0.2 Slope)	(14,14,64)
Batch Normalization	(14,14,64)
UpSampling	(28,28,1)
Tanh	(28,28,1)

Table 3.6 : Layer details of discriminator network.

Layer Type	Output Shape
DownSampling	(14,14,64)
LReLU (0.2 Slope)	(14,14,64)
Dropout (0.3)	(14,14,64)
DownSampling	(7,7,128)
LReLU (0.2 Slope)	(7,7,128)
Dropout (0.3)	(7,7,128)
Flatten	1
Dense	1

In figure 3.12, generator network layers are visualized according to the output sizes of layers showed in Table 3.5.

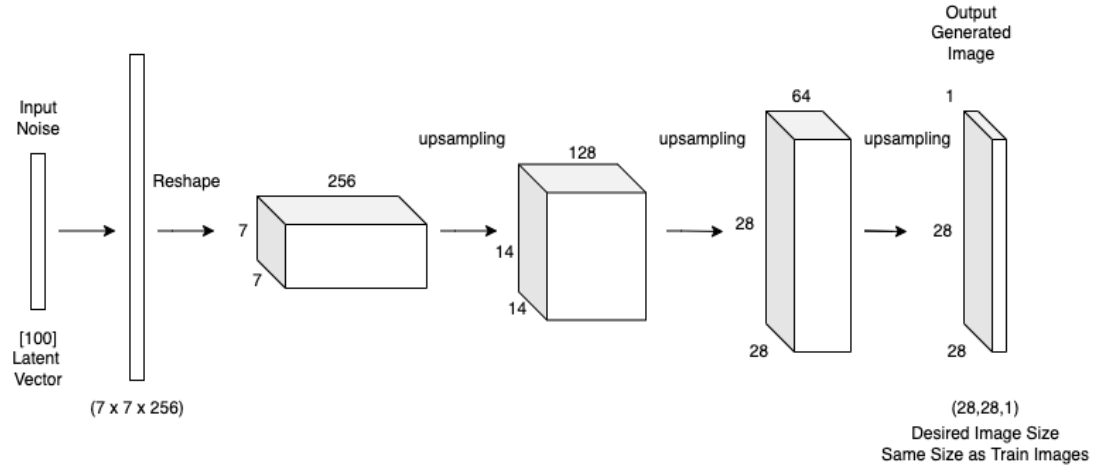


Figure 3.12 : DCGAN generator architecture.

In figure 3.13, discriminator network layers are also visualized according to output sizes of layers showed in Table 3.6.

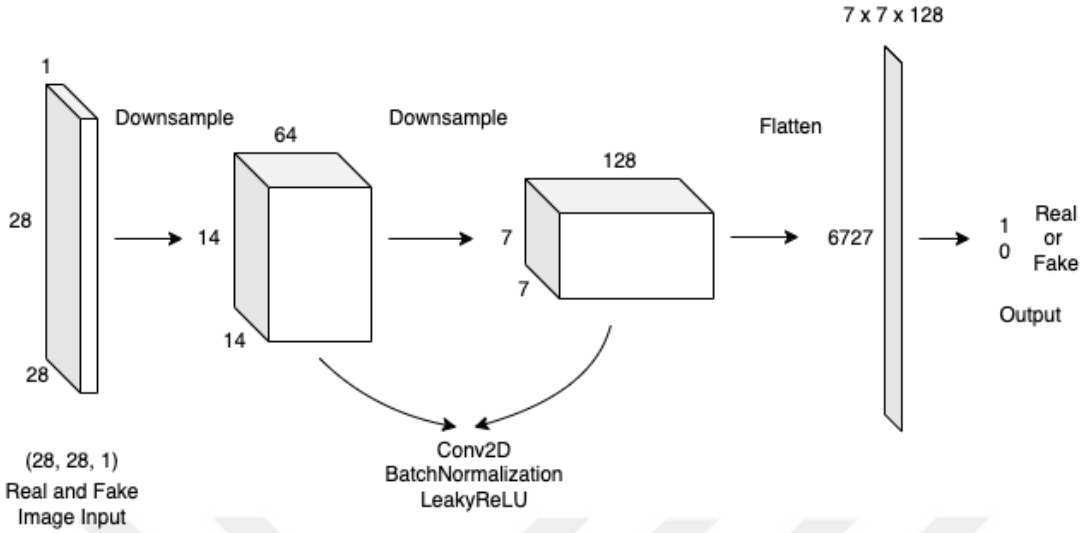


Figure 3.13 : DCGAN discriminator architecture.

As Radford et al. stated that for DCGAN, previous generative model designs were examined, and significant rule modifications were made in standard implementation [12]. Those rules are as follows:

- Remove all pooling layers and replace them with strided deconvolutional and strided convolutional layers on the generator and discriminator, respectively.
- Batch normalization [55] layers are used in both generator and discriminator networks.
- Delete all dense layers in hidden layers to make the neural network architecture deeper.
- Except for the last layer, use Relu as an activation function for each layer in the generator. Use Tanh as an activation function in the output layer.
- For each layer in the discriminator, use LeakyRelu as an activation function.

A sigmoid activation function was utilized in the discriminator's final layer. The sigmoid function is a neural network activation function as described at Section 3.3.1. The sigmoid activation function is used at discriminator network because it returns a number between 0 and 1. Because it predicts a probability as an output, it is beneficial

for the discriminator. In this way, the data is real if the output value is near to 1, and it is fake if the output value is close to 0. A Tanh activation function was used in the generator's final layer. It produces a value between -1 and 1, the negative inputs are also mapped. It has been demonstrated that adopting a limited activation function speeds up model learning.

As main GAN [11] architecture, Binary Cross Entropy (BCE) is used as loss function in architecture and it is calculated at the output of the discriminator network. The Adam optimizer [67] is used with a learning rate of 0.0002 for both the generator and discriminator, as indicated by the DCGAN study.

3.3.3 Wasserstein Generative Adversarial Networks

In basic GAN networks, the Jensen–Shannon divergence is the more commonly used mechanism for calculating loss. Nevertheless, when working with gradients, this approach has difficulties that might lead to unstable training. As a result, the Wasserstein distance is employed to resolve such situations. The mathematical formula is represented in the Equation 2.9.

The discriminator is referred to as the critic in the Wasserstein Generative Adversarial Network (WGAN) [13] architecture. The lack of a sigmoid activation function to restrict the values to 0 or 1 which means real or fake is one of the reasons behind this approach. WGAN discriminator networks return a range of values rather than a single result, allowing it to operate as a critic with less rigidity.

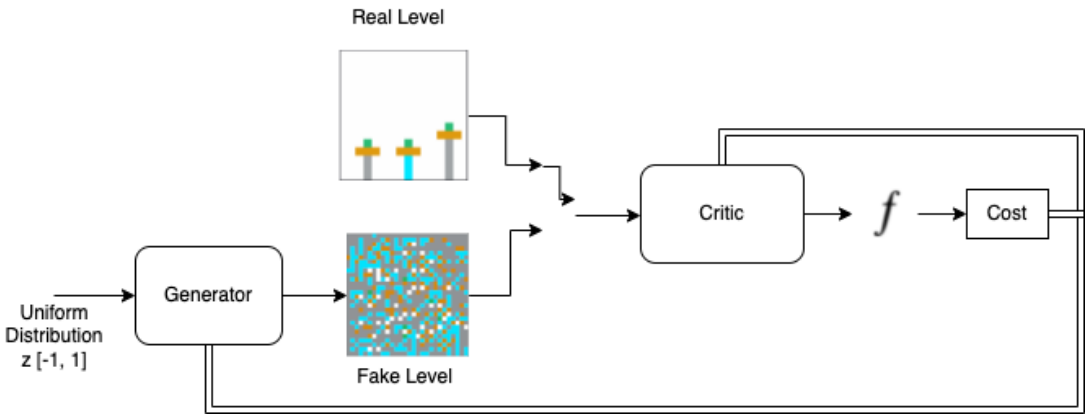


Figure 3.14 : WGAN architecture to generate Angry Birds Levels

The created WGAN architecture can be seen in Figure 3.14. In this architecture, generator network contains 1 dense layer and 3 convolution layers. At the last layer of

generative network, tanh activation function is used. On the other hand, the other layer uses ReLU activation function. A randomly sampled 100 dimensional latent vector in the range [-1,1] from uniform distribution is given to the generator network as input. Layer details of generator network can be seen in Table 3.7.

Table 3.7 : Layer details of WGAN generator network.

Layer Type	**Output Shape**
Dense	6272
Reshape	(7, 7, 128)
UpSampling	(14, 14, 128)
Convolution	(14, 14, 128)
ReLU	(14, 14, 128)
Batch Normalization	(14, 14, 128)
UpSampling	(28, 28, 128)
Convolution	(28, 28, 64)
ReLU	(28, 28, 64)
Batch Normalization	(28, 28, 64)
Convolution	(28, 28, 1)
Tanh	(28, 28, 1)

By the way, discriminator network contains 4 convolutional layers. All layers use *LReLU* activation function with 0.2 slope value. 3 dimensional (28, 28, 1) matrix is given to discriminator as input. Layer details of discriminator can be seen below Table 3.8.

In WGAN, there are minor changes against to original GAN. Compared to the original GAN algorithm, the following changes have been made to WGAN:

- After each gradient update in the critical function, the weights are compressed by a small interval [-0.01, 0.01].
- Wasserstein uses a new error function derived from distance. The Discriminator is now in the position of a helper for the implementation of the Wasserstein criterion rather than playing its critical role.
- In WGAN's article, the authors recommended using the RMSProp [68] optimization algorithm instead of momentum-based Adam in critical.

Table 3.8 : Layer details of discriminator network of WGAN.

Layer Type	Output Shape
Convolution	(14, 14, 16)
LReLU (0.2 Slope)	(14, 14, 16)
Dropout (0,25)	(14, 14, 16)
Convolution	(7, 7, 32)
Zero Padding	(8, 8, 32)
Batch Normalization	(8, 8, 32)
LReLU (0.2 Slope)	(8, 8, 32)
Dropout (0,25)	(8, 8, 32)
Convolution	(4, 4, 64)
Batch Normalization	(4, 4, 64)
LReLU (0.2 Slope)	(4, 4, 64)
Dropout (0,25)	(4, 4, 64)
Convolution	(4, 4, 128)
Batch Normalization	(4, 4, 128)
LReLU (0.2 Slope)	(4, 4, 128)
Dropout (0,25)	(4, 4, 128)
Flatten	2048
Dense	1

Randomly sampled 100 dimensional latent vector in the range [-1,1] from uniform distribution is given to the generator network as input. An example output is produced from latent vector, can be seen in Figure 3.15.

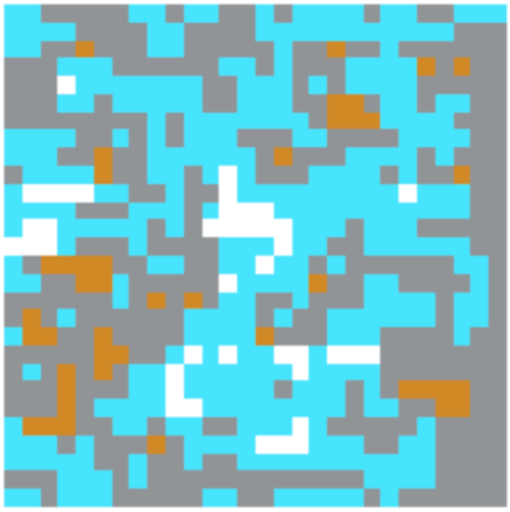


Figure 3.15 : Example 28x28 sized output of untrained generator network of WGAN architecture.

As Arjovsky et al. stated that WGAN is not perfect and weight clipping is a very bad way to constrain the Lipschitz constant [13]. When the weight clipping interval is

selected too large, WGAN goes through an unstable and slow training process. When the weight clipping window is selected too small, WGAN encounters the vanishing gradient problem as original GAN. Therefore clipping interval should be chosen carefully. In order to prevent vanishing problem, a small interval [-0.01, 0.01] is chosen in this thesis.

In WGAN architecture, Wasserstein distance is used as the loss function, and it is calculated at the output of the discriminator network. The RMSProp [68] optimizer is used with a learning rate of 0.00005 for both the generator and discriminator, as indicated by the WGAN study.

4. EXPERIMENTS

The findings for all of the GAN architectures for generating Angry Birds levels are explained in this chapter. These results include the dataset input levels, the produced output levels from the input levels and the model comparisons.

4.1 Input Levels

In one iteration, 256 input levels per batch were used in neural network training. Example training levels are shown in Figure 4.1. Both 16x16 and 28x28 matrix sized input levels are used for training of the stated architectures. 16x16 matrix sized levels are more cramped and structures are closer to each other however, in 28x28 matrix sized levels, structures are separated in level play area.

(a) 16x16 Matrix sized input level

(b) 28x28 Matrix sized input level

Figure 4.1 : Example training input levels demonstration.

4.2 Generative Adversarial Networks

Both 16x16 sized and 28x28 sized input levels are trained in the GAN architecture. After training is completed, similar lost graph were obtained. After 28x28 sized input

levels were used for training, both generator and discriminator network loss graphs is showed over 2000 epoch in Figure 4.2.

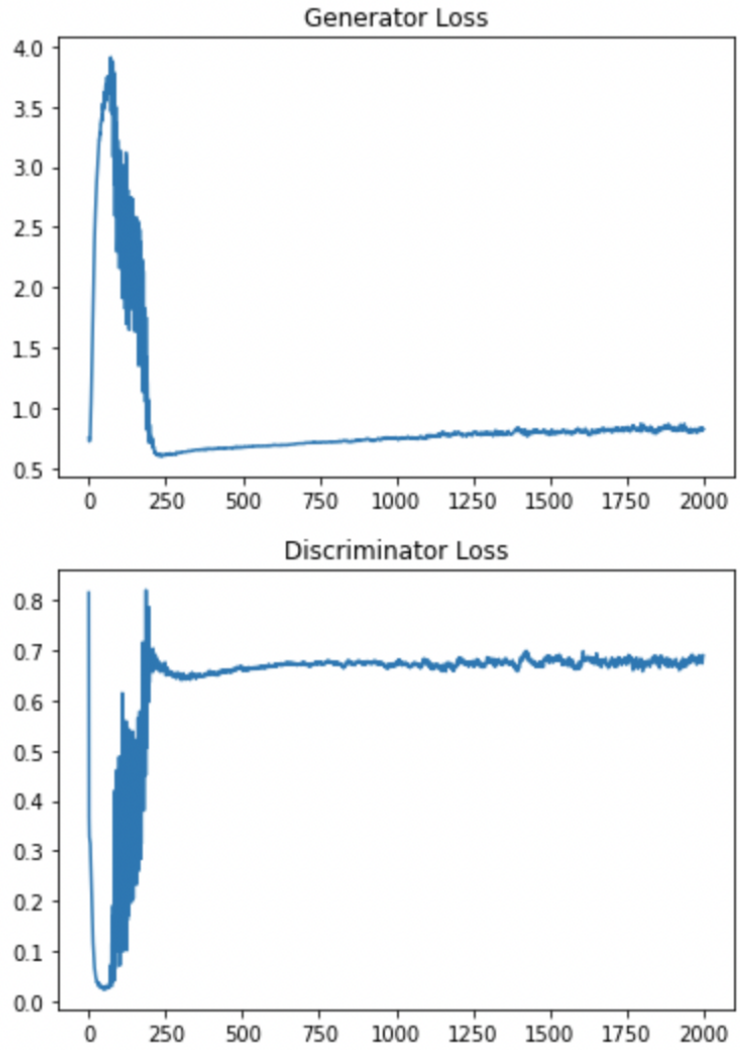


Figure 4.2 : Generator and Discriminator loss values over 2000 epoch.

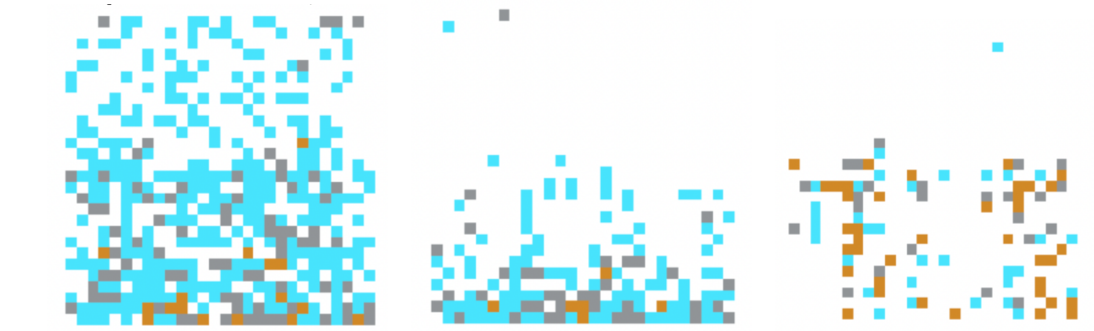
When Figure 4.2 is examined, the discriminator network can easily distinguish between real and fake images in the early stages of training. However, as the training progresses, the fake images generated by the generator approach the data distribution, and the discriminator network has difficulty in distinguishing the images. Finally, the discriminator network produces a loss value of about 0.67 for both real and fake images. Since the BCE loss function is used in the last layer of the discriminator network;

$$-\log q_i \cong 0.67 \tag{4.1}$$

from the above Equation 4.1, $q_i \cong 0.51$ can be found. Since the discriminator network ends with a *sigmoid* unit, it produces an output between 0 and 1. In other words, it is intended to produce 1 for images taken from the real dataset and 0 for images generated by the generator network. However, after the training phase, the discriminator network distinguishes any given image as real or fake with a probability of 0.51. As a consequence, this shows that the discriminator network cannot distinguish the generated images as real or fake, in other words, the generated images converge to the real data distribution.

After the training process is completed, the discriminator network is discarded and the generator network is used to generate images similar to the data set. Generated example levels with different training datasets and different epoch values can be seen in Figure 4.3.

(a) 16x16 input over 200 Epoch (b) 16x16 input over 1000 Epoch (c) 16x16 input over 2000 Epoch



(d) 28x28 input over 200 Epoch (e) 28x28 input over 1000 Epoch (f) 28x28 input over 2000 Epoch

Figure 4.3 : Example generated output levels from GAN generator. Color decoding is implemented.

As shown in this figure, colors represent materials as in Table 3.2. After generation is completed, image encodings are processed to generate encoded level matrix. Figure 4.4 shows the conversion process. When matrix conversion is completed, the encoded level can be represented in Science Birds clone of Angry Birds [10].

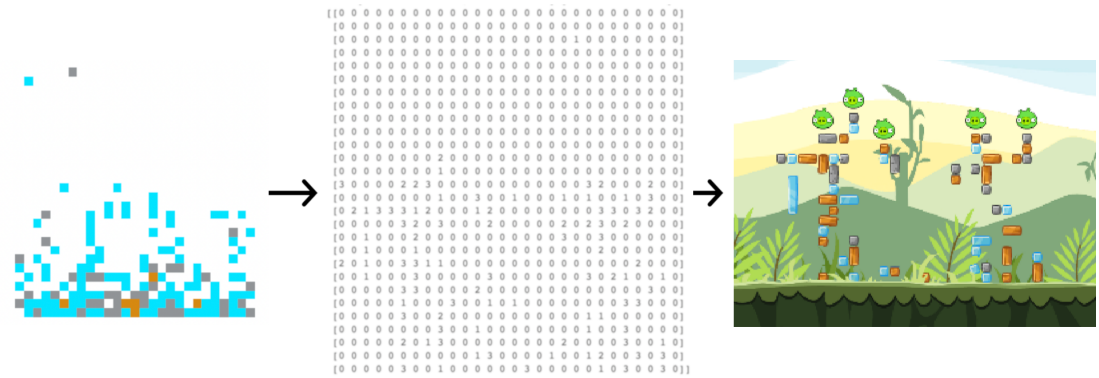


Figure 4.4 : Generated 28x28 sized image to level representation conversion.

In generated levels, pig density is very low due to the fact that pig counts in train levels are very low. Therefore, a rule base algorithm is developed to create playable levels. This algorithm scans generated levels and place pigs to suitable places. Number of pigs are determined with number of birds. Bird types are selected according to material density of generated structures. Blue birds have advantages against ice blocks. Therefore, if the level structure has more ice blocks other than materials, blue birds randomness factor is increased. Yellow birds has more advantages against wood block. Therefore randomness factor improvement is used for Yellow bird selections similar to the Blue birds selection. If material density for three different material is near each other than red, blue and yellow birds selected equally randomness factor. Number of birds are randomly selected. Number of placed pigs are twice the number of birds. Randomly selected generated levels which are trained with different input sets are seen in Figure 4.5 and Figure 4.6.

As seen in Figure 4.5 and Figure 4.6, training input levels affect results. 16x16 sized input levels cause smaller output levels. Pigs are even not included in some levels. Outputs that are generated from 28x28 sized input levels, include more structures and pigs. These levels have richer content than levels which are generated from 16x16 sized input levels. In addition, some objects are placed in the air as shown in Figure 4.5. When game is played, gravity is on, and these objects in the air, falls immediately. Therefore, some stability issues are occurred. In this manner, to calculate stability rates, all generated levels are evaluated by Angry Birds AI agent [18]. Level expressivity analyses such as stability, playability are explanined in details in the following sections.

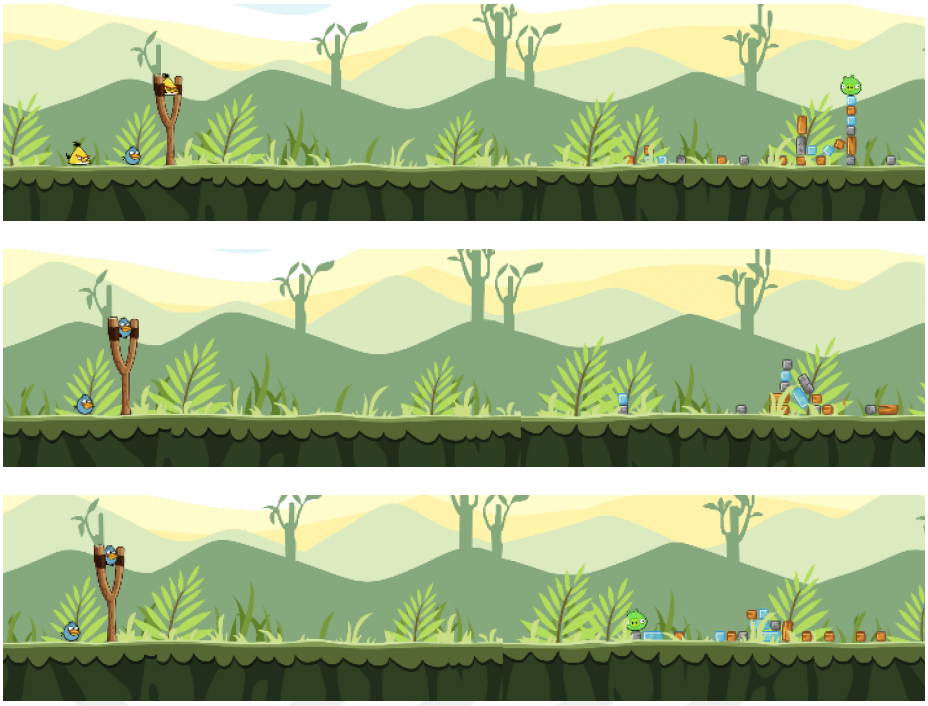


Figure 4.5 : Example generated levels of GAN architecture from 16x16 sized training input levels.

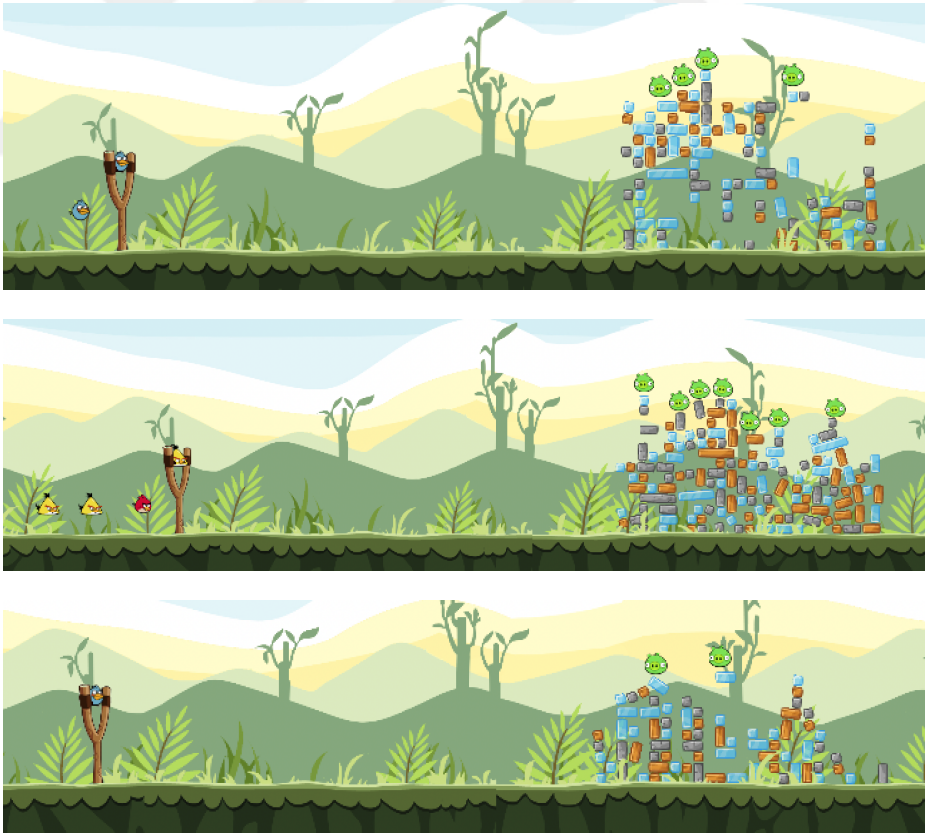


Figure 4.6 : Example generated levels of GAN architecture from 28x28 sized training input levels.

4.3 Deep Convolutional Generative Adversarial Networks

Both 16x16 sized and 28x28 sized input levels are trained by this architecture. When the training is completed, similar loss graphs were obtained. After generator was trained with 28x28 sized input levels, both generator and discriminator network loss graph is shown over 2000 epochs in Figure 4.7.

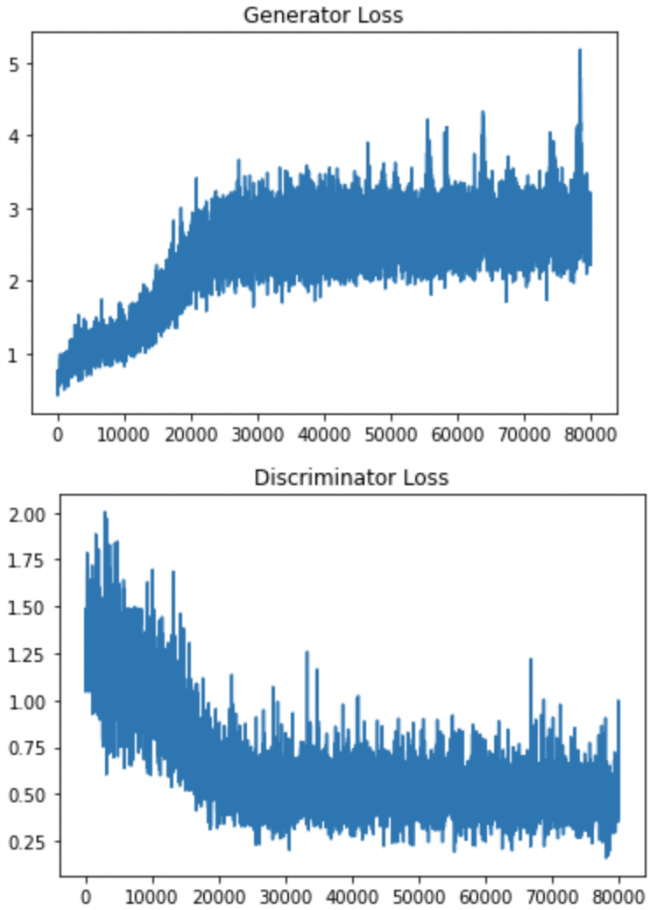
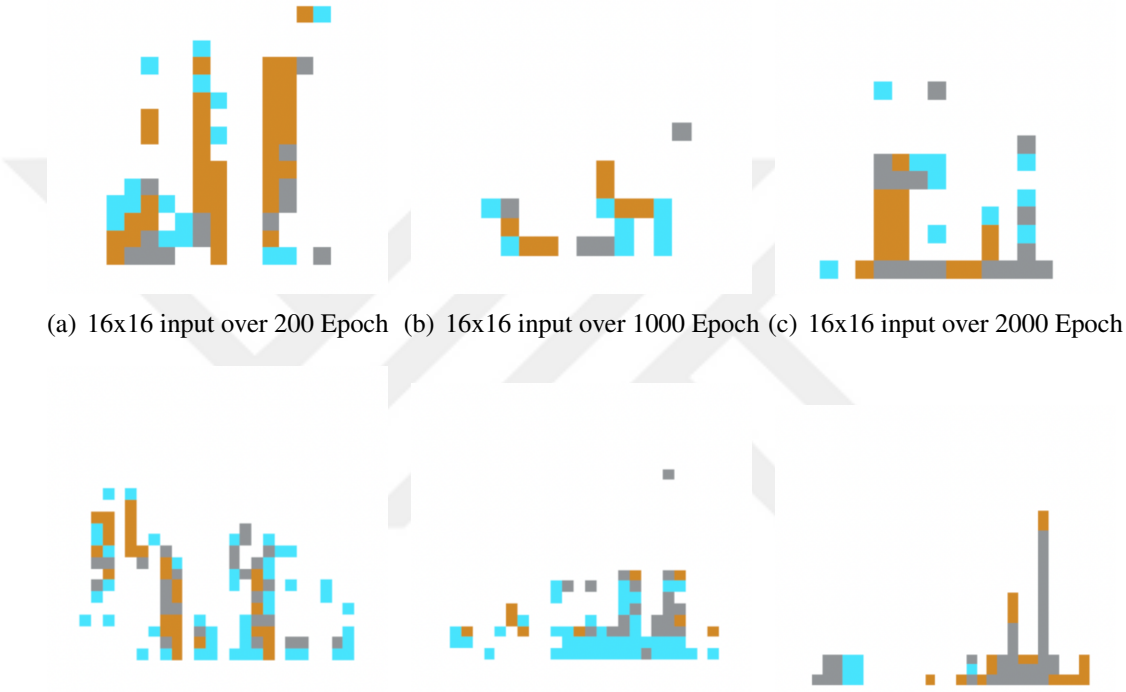


Figure 4.7 : DCGAN Generator and Discriminator networks loss values over 2000 epoch.

In the early phases of training, the discriminator network can readily discriminate between real and fake pictures, as shown in Figure 4.7. However, as the training proceeds, the generator's fake pictures approach the data distribution, making it impossible for the discriminator network to distinguish the images. Finally, for both real and fake pictures, the discriminator network generates a loss value of nearly 0.61.

$$-\log q_i \cong 0.54 \tag{4.2}$$

Because the BCE loss function is used in the discriminator network's last layer, from the above Equation 4.2, $q_i \cong 0.54$ can be found. In other words, the discriminator should produce 1 for pictures from the real dataset and 0 for images generated by the generator network. However, after the training phase, the discriminator network distinguishes any given image as real or fake with a probability of 0.51. After the training process is completed, generated example levels with different Epoch values can be seen in Figure 4.8.



(a) 16x16 input over 200 Epoch (b) 16x16 input over 1000 Epoch (c) 16x16 input over 2000 Epoch

(d) 28x28 input over 200 Epoch (e) 28x28 input over 1000 Epoch (f) 28x28 input over 2000 Epoch

Figure 4.8 : Example generated output levels from DCGAN generator. Color decoding is implemented.

After the generation is completed, image encodings are processed to generate encoded level matrix as in the basic GAN architecture. When Matrix conversion is completed, rule based pig placement and bird selection algorithm is used as in the GAN architecture then encoded levels can be represented in Science Bird [10]. Randomly selected generated levels can be seen in Figure 4.9 and Figure 4.10.

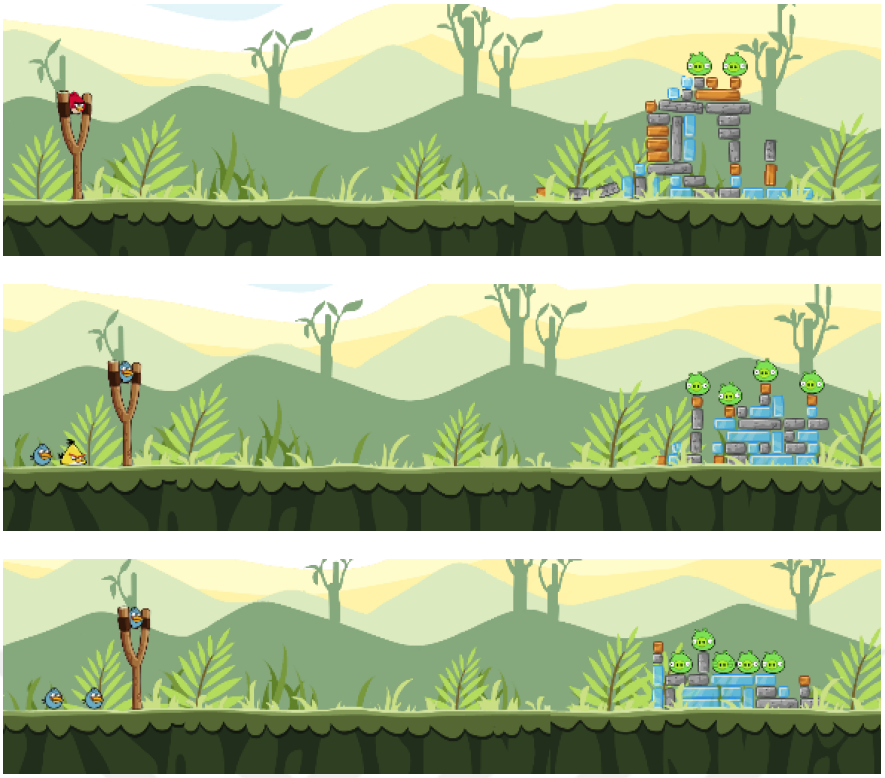


Figure 4.9 : Example generated levels of DCGAN architecture from 16x16 sized training input levels.

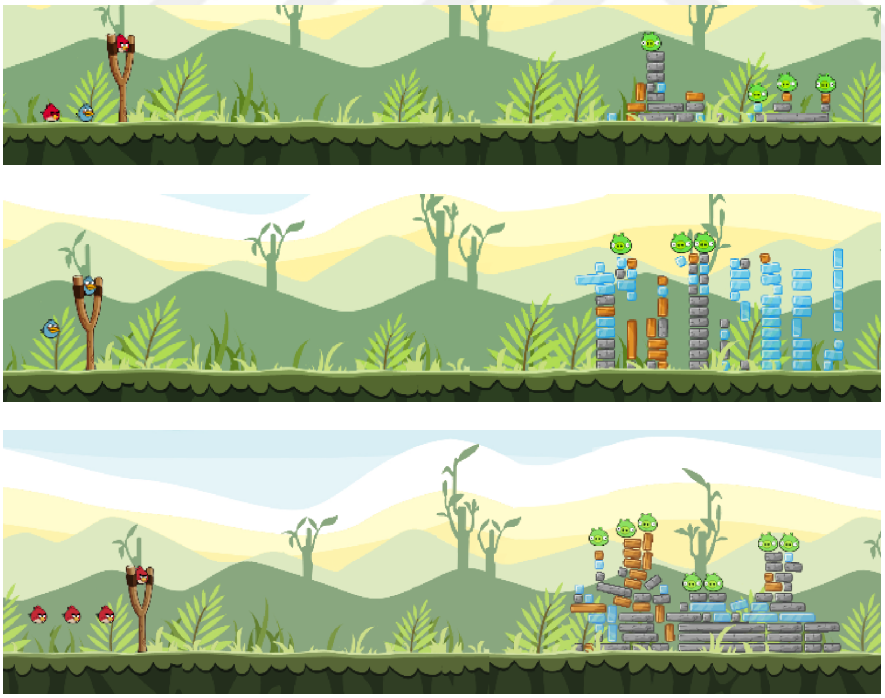


Figure 4.10 : Example generated levels of DCGAN architecture from 28x28 sized training input levels.

As seen in Figure 4.9 and Figure 4.10, training input levels also affect the results in DCGAN architecture. 16x16 sized input levels generates smaller and high density

levels. On the other hand, outputs that are generated from 28x28 sized input levels have richer content than levels which are generated from 16x16 sized input levels. As shown in Figure 4.10, stability issues are also occurred in generated DCGAN levels.

4.4 Wasserstein Generative Adversarial Networks

Both 16x16 sized and 28x28 sized input levels are trained by this architecture. Similar loss graphs were obtained once training is finished. Figure 4.11 shows the generator and discriminator network loss graphs over 2000 epochs after the generator was trained with 28x28 sized input levels.

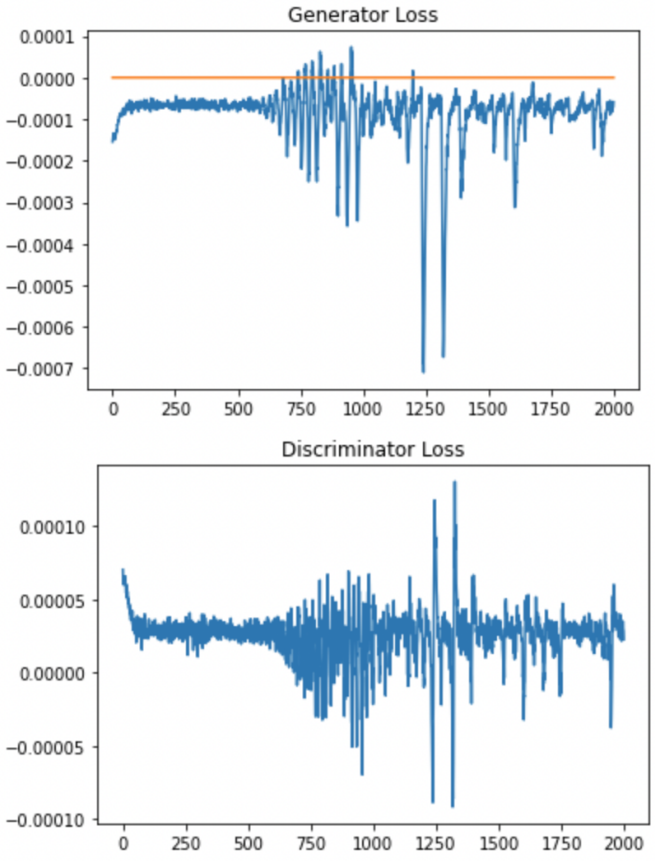
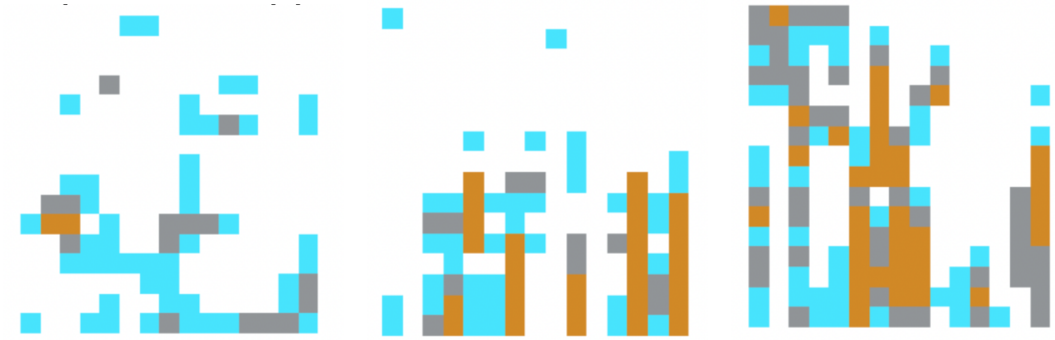


Figure 4.11 : WGAN Generator and Discriminator networks loss values over 2000 epoch.

As shown in Figure 4.11, discriminator generates loss value in range of clipping interval [-0.01, 0.01]. Instead of playing a primary role, the discriminator now provides as a facilitator for the Wasserstein criterion's implementation. After the training process is completed, the generated example levels with different epoch values can be seen in Figure 4.12.



(a) 16x16 input over 200 Epoch (b) 16x16 input over 1000 Epoch (c) 16x16 input over 2000 Epoch

(d) 28x28 input over 200 Epoch (e) 28x28 input over 1000 Epoch (f) 28x28 input over 2000 Epoch

Figure 4.12 : Example generated output levels from WGAN generator. Color decoding is implemented.

After the generation is completed, image encodings are processed to generate encoded level matrix as with the architectures. When Matrix conversion is completed,rule based pig placement and bird selection algorithm is used as in the GAN and DCGAN architectures than the encoded levels can be represented in Science Bird [14]. Randomly selected generated levels can be seen in Figure 4.13 and Figure 4.14.

As seen in Figure 4.13 and Figure 4.14, the size of the training input levels affects the density of the output levels in WGAN architecture. The generator that is trained with 16x16 sized input levels, produce minimalist levels. On the other hand, the generator trained with 28x28 sized input levels produce huge levels.

As a consequences, stability issues are occurred for each implementation of GAN architectures. Some generated levels can not resist gravity in Angry Birds.

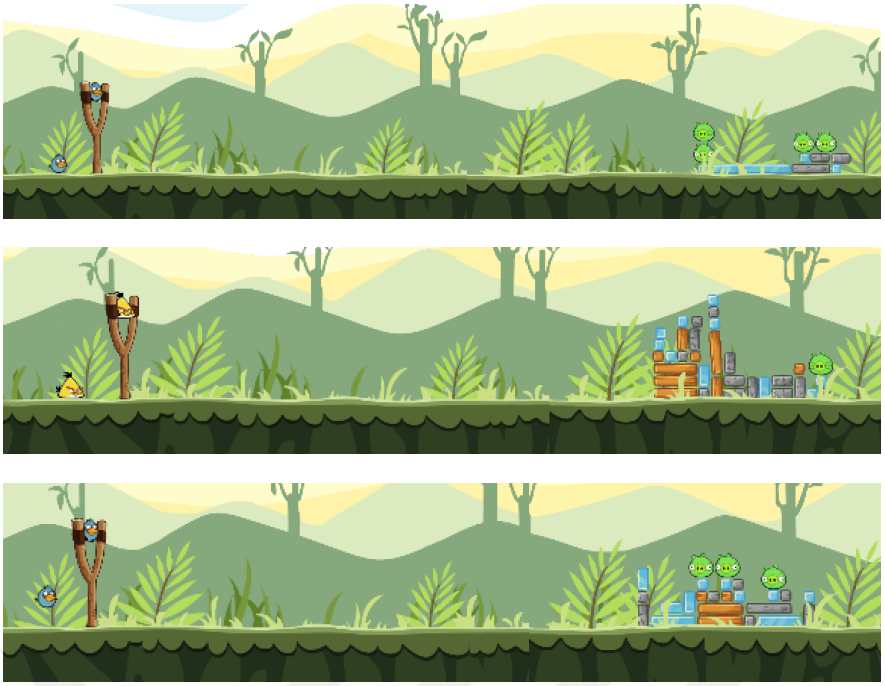


Figure 4.13 : Example generated levels of WGAN architecture from 16x16 sized training input levels.

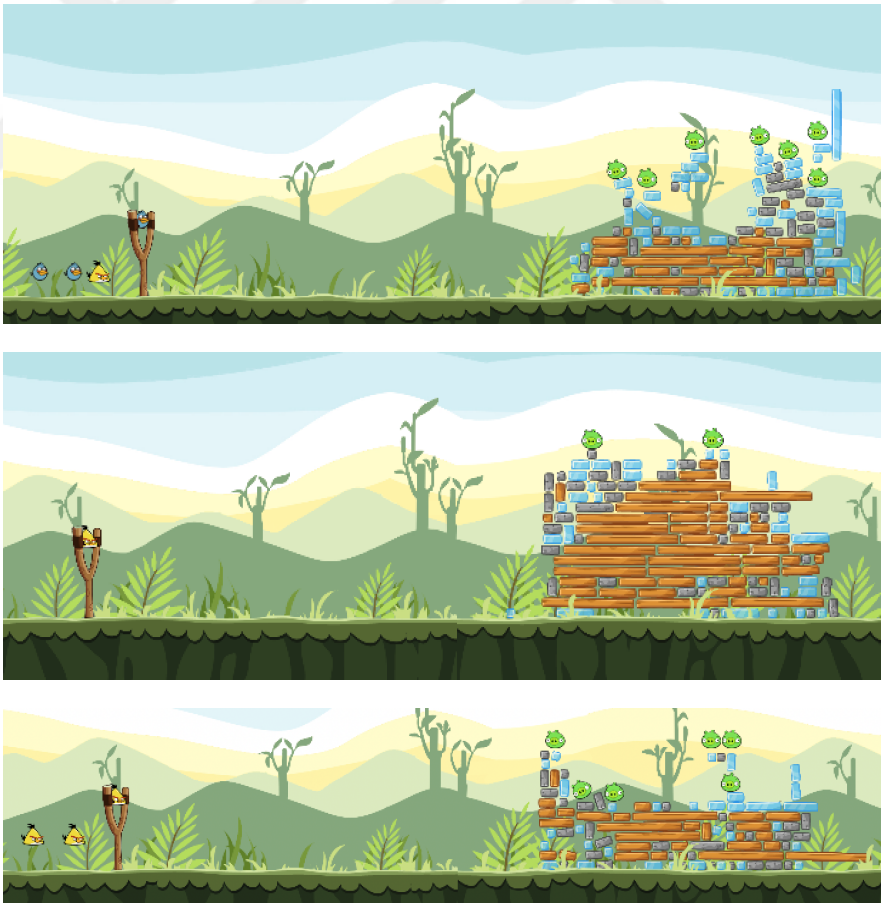


Figure 4.14 : Example generated levels of WGAN architecture from 28x28 sized training input levels.

4.5 Level Expressivity Analysis

The space of all levels that a level generator may produce is known as expressivity, and it is calculated by analyzing many features of a level to determine its advantages and disadvantages.

Several studies were conducted to evaluate the expressivity of the generated levels by GAN, DCGAN and WGAN methods. As a state-of-the-art method, VAE-based generative model [39], is also involved in this analysis. Furthermore, to provide a comparative analysis, the outputs of the IratusAves [14] are also used. To analyze all five studies, 100 Angry Birds levels are generated. Analysis is completed with these 100 levels in Science Birds clone of Angry Birds [10].

A number of metrics have been put up to evaluate the expressivity of a generator such as content richness, stability, linearity, density, playability and difficulty. Content richness is used for variety of levels. Stability is evaluated with number structures collapsed or not. Linearity is used to measure structure heights and level profile. Density is used to define compactness of structures in level. Playability is used to decide level can be completed or not. Difficulty is used to describe level is hard or easy.

4.5.1 Content Richness

Input data levels have significant role in the generated level's content richness. As shown in the generated example levels for each architecture, large input sets helps to generate more structures in levels. The investigated architectures are trained with both 16x16 and 28x28 input levels. The number of the generated structures are registered in this analysis for different training sets. To determine the final score, every separated set of blocks was counted. In Table 4.1, the average number of structures generated are presented.

As shown in Table 4.1, the generated structure sizes increase when the training input sizes increase. In DCGAN architecture, the structure count is dramatically increased. As it gives the best results about content richness rather than other architectures used. The generated structure count almost reaches to that of the rule base generator IratusAves [14].

Table 4.1 : Average structure counts in generated levels.

Method	Input Level Size	Average Structure Count	Standard Deviation
GAN	16x16	1.03	0.23
GAN	28x28	1.06	0.24
DCGAN	16x16	1.34	0.58
DCGAN	28x28	2.12	1.07
WGAN	16x16	1.12	0.56
WGAN	28x28	1.29	0.59
Sequential VAE [39]	-	1.29	0.80
IratusAves [14]	-	2.49	0.90

4.5.2 Stability

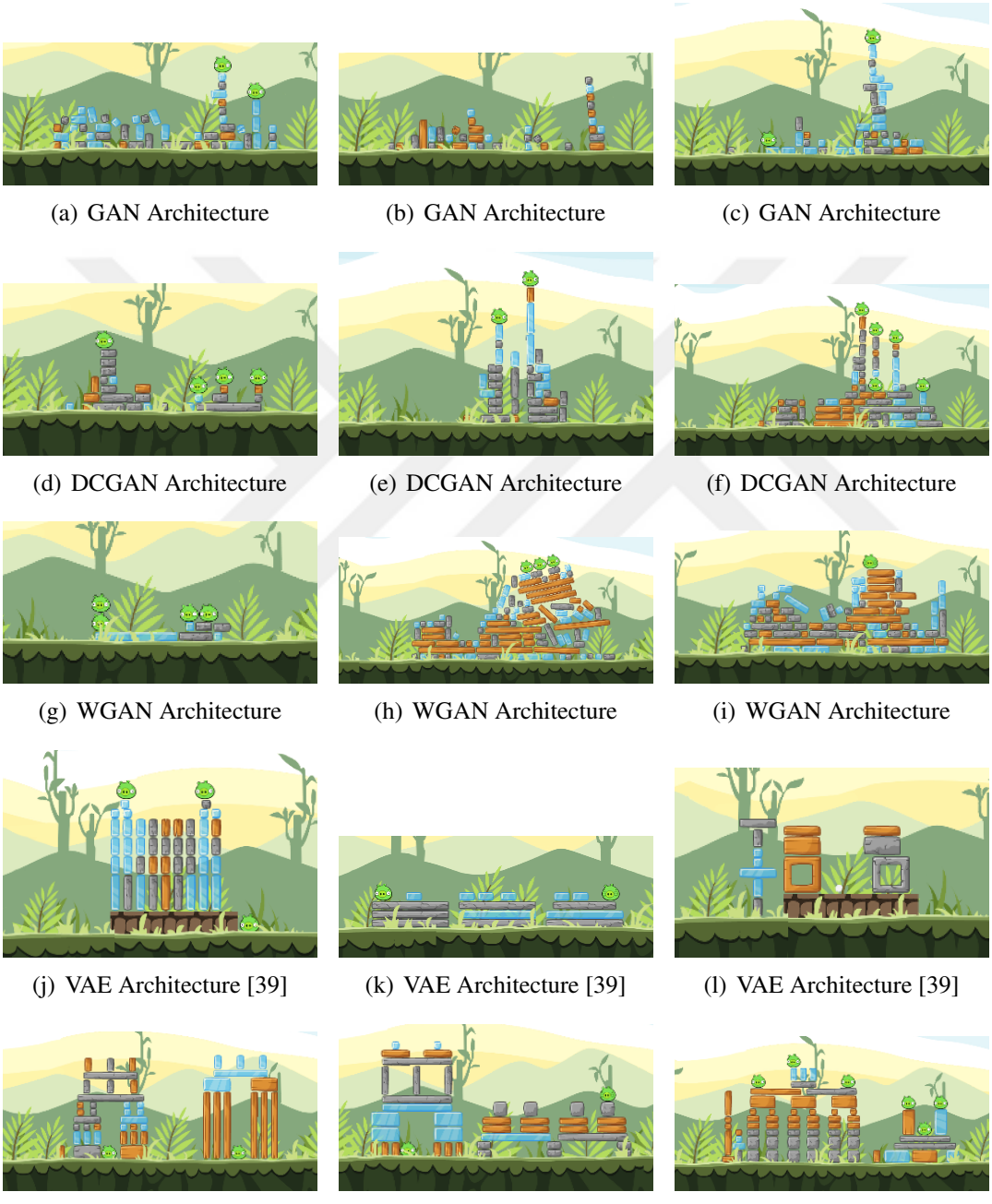
The quality of the levels generated is influenced by the stability of the structures formed by presented generators. When the generated levels opened in the playground, once the level is initialized, structures that cannot maintain themselves will collapse due to gravity in the game, reducing the level's overall playability. Therefore stability is the most significant measurement for the generated levels. To analyze the stability of the generated levels, 100 levels are generated, and imported to Science Birds [10]. The probability of generating a level that does not collapse without action is used to determine stability. As analyzed before, the former one generates small structures as shown in Table 4.1. In order to evaluate the architectures with the other methods namely Sequential VAE [39] and IratusAves [14], 28x28 sized training sets are selected for stability analysis. The results can be seen in Table 4.2.

Table 4.2 : Approaches and average stability percentages of their generated levels.

Method	Average Stability	Standard Deviation
GAN	41.14%	2.50
DCGAN	70.41%	2.54
WGAN	68.12%	3.21
Sequential VAE [39]	70.7%	3.27
IratusAves [14]	100%	0

As expected and shown in Table 4.2, IratusAves [14] rule base Angry Birds level generator, generates more stable levels than other approaches. This is because block and pig placement in the generated levels is hand crafted and more precise than neural network-based models. Sequantial VAE method [39] generates more stable

levels than GAN and WGAN architecures. The stability percentage of the generated levels by DCGAN architecure is almost the same with the Sequential VAE method and drastically better than GAN and WGAN architectures. Figure 4.15 presents example levels by these approaches. As shown in the Figure 4.15, Sequantial VAE [39] method and rule base algorithm IratusAves [14] generate more stable and well organized levels. Organized structures provide more stable levels.



(a) GAN Architecture (b) GAN Architecture (c) GAN Architecture

(d) DCGAN Architecture (e) DCGAN Architecture (f) DCGAN Architecture

(g) WGAN Architecture (h) WGAN Architecture (i) WGAN Architecture

(j) VAE Architecture [39] (k) VAE Architecture [39] (l) VAE Architecture [39]

(m) IratusAves [14] (n) IratusAves [14] (o) IratusAves [14]

Figure 4.15 : Example generated levels of different approaches.

4.5.3 Linearity

The profile of the produced levels is measured by linearity. The linearity of the levels containing items placed at various heights throughout the level space are scored low, the linearity of the levels when the items follow a straight path is scored high. The center positions of all blocks and pigs are used as data points to determine linearity. The generated 100 levels are analyzed, and the results are presented in Table 4.3.

Table 4.3 : Approaches and linearity percentages of their generated levels.

Method	Linearity
GAN	4.17%
DCGAN	32.29%
WGAN	44.79%
Sequential VAE [39]	91%
IratusAves [14]	44%

As shown in Table 4.3, the highest linearity is observed at Sequenatial VAE [39] generator. The generated levels are more linear than the other approaches, and more similar levels are generated. However Similarity is not a desirable situation in games where diversity is desired. WGAN architecture creates more linear levels than the other generative adversarial network architectures. The rule base generator IratusAves also gives higher linearity percentage. As a consequence, the implemented architectures generate non-linear levels. In this way, the generated levels are more unique, and blocks are distributed throughout the entire playground.

4.5.4 Density

The compactness of the blocks arranged inside a level provides density analysis. The total area of all structures, platforms, and pigs inside the level space is used to calculate density. This is then divided by the overall size of the level space to get a percentage that shows how much of the level was taken up by blocks. Low density levels are sparse, while high density levels are more crowded. To determine density percentage of levels, 100 levels were generated. In Table 4.4, density percentages of approaches are shown.

Table 4.4 : Approaches and density percentages of their generated levels.

Method	Density
GAN	61.46%
DCGAN	32.29%
WGAN	39.58%
Sequential VAE [39]	13%
IratusAves [14]	24.3%

When the density levels increase structures become more congested. The used GAN architecture generates more congested levels than the other approaches as shown in Table 4.4. However congestion can cause collisions and stability issues. Structures are more stable when they are sparse.

4.5.5 Playability

The term "playability" is used to describe whether or not a level can be completed. It's difficult to tell if a level can be completed due to the extremely large state and action space. There are lots of AI agents which can play Angry Birds levels however, the game playability results are different. An AI agent can confirm that level is solvable, but it is hard to confirm whether the level is unsolvable. In spite of the fact that every generated level should be solvable with an unlimited number of birds, it is uncertain if a level can be solved with the birds remained. Angry Birds AI agent [18] is used also for analysis of playability to provide solvability information. If level is stable and pigs are not destroyed by collapsing when the game is started then AI agent plays the level, and try to complete the level with given the birds in max of 3 attempts. Playability percentages of the generated 100 levels can be seen in Table 4.6.

Table 4.5 : Approaches and playability percentages of levels.

Method	Playability
GAN	42.71%
DCGAN	87.50%
WGAN	69.79%
Sequential VAE [39]	71.0%
IratusAves [14]	98%

Percentages of the playability analysis in Table 4.6 show that the generated stable levels which do not collapse at the game start, are playable in DCGAN architecture

other than GAN and WGAN architectures. Playability is defined not only with level is stable or not, but also level can be completed with the given birds and finite number of attempts. In Sequantial VAE [39], playability percentage is greater than GAN and WGAN architectures, however, the number of birds in their generated levels are fixed to 7. This number is very high and with 7 birds lots of levels even the hardest real levels of Angry Birds can be completed easily. Therefore, comparison between the generator of used the GAN architecures and Sequential VAE [39], is not fair and expressive. Rule-based generator IratusAves [14] generates more playable levels than GAN architectures as shown in Table 4.6. The generator which uses the DCGAN architecture gets closer results in playability to the rule-base generator.

4.5.6 Difficulty

Difficulty is used to describe that how hard a level is to complete successfully. In Angry Birds levels, killing all pigs with a provided number of birds achieves the completion of a level. The difficulty of a level is estimated with the number of attempts for a level to complete successfully. The number of pigs and the number of structures affect the difficulty of a level. To provide this analysis, an AI bot, Angry Birds AI agent [18], is used to play all the generated levels. The average number of attempts of all levels by this agent can be seen in Table 4.5.

Table 4.6 : Approaches and number of attempts required to complete successfully the generated levels.

Method	Attempts
GAN	1.04
DCGAN	1.07
WGAN	1.10
Sequential VAE [39]	1.0
IratusAves [14]	1.02

Difficulty can be estimated only for solvable levels. However hard to confirm whether levels are solvable or unsolvable. Therefore attempt counts are calculated for the levels which are stable and are not collapsed at the game start.

As a result, this analysis shows that generative models can get close to rule base algorithms in Angry Birds level generation. Especially with DCGAN architecture, generated levels are as stable as Sequential VAE [39] method and close to the

rule-based generator. Providing stability also brings playable levels. In addition, content richness is also obtained in the DCGAN architecture's generator as well as the rule base generator. With bigger-sized train input levels, more structures can be generated rather than in other generative models.

5. CONCLUSION

This thesis investigates generative adversarial network architectures to solve the content generation problem. Particularly, the Angry Birds game is selected as the case study, and the levels are generated procedurally. Three different generative adversarial network architectures are analyzed: GAN [11], DCGAN [12] and WGAN [13] for this case study.

Different GAN architectures are implemented to find best solution for Angry Birds level generation problem. The loss functions and optimizers are the primary variations between the training, validation, and testing phases in generative adversarial network architectures. Even though network parts of the GAN architectures are almost the same, generator network tries to generate fake levels and discriminator network tries to separate fake levels from generated levels.

The results showed that in general, GAN architectures can be used effectively for content generation problems. In implemented GAN methods, especially DCGAN provides better performance in level generation problem. The main factor of this performance is using convolutional layers rather than using dense layers. In general, convolutional networks focuses on finding areas of correlation in the input data. In this way, looking for correlation between blocks of Angry Birds levels helps to generate more stable output levels. Furthermore, in GAN and WGAN, the generated levels have more noises which causes stability issues. DCGAN has better performance than not only the other implemented GAN architectures but also the proposed Sequential VAE method [39] in Angry Birds level generation problem.

In conclusion, the experience gained in thesis can be generalized to other physic based games as well. This research has a wide range of potentials to enhance generation method in the future. One example could be to use real level image for training. Another future research direction is adding different block types in inputs.

REFERENCES

[1] **Togelius, J.**, **Kastbjerg, E.**, **Schedl, D. and Yannakakis, G.** (2011). What is Procedural Content Generation? Mario on the borderline.

[2] **Brown, J.** (2018). Pitching Diablo: On the Development of Marketing, Procedural Content, and Narrative.

[3] **LEE, J.**, (2014), How Procedural Generation Took Over The Gaming Industry, https://www.makeuseof.com/tag/procedural-generation-took-gaming-industry/.

[4] **Naonearchive, E.**, (2008), Creating Creatures, https://www.technologyreview.com/2008/06/17/127401/creating-creatures/.

[5] **Games, H.**, (2016), About No Man's Sky, https://www.nomanssky.com/about/.

[6] **Togelius, J.**, **Yannakakis, G.**, **Stanley, K. and Browne, C.** (2010). Search-Based Procedural Content Generation, pp.141–150.

[7] **Entertainment, R.**, (2022), Angry since 2009, https://www.rovio.com/games/#angrybirds.

[8] **Ferreira, L. and Toledo, C.** (2014). Generating Levels for Physics-Based Puzzle Games with Estimation of Distribution Algorithms, *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*, ACE '14, Association for Computing Machinery, New York, NY, USA, https://doi.org/10.1145/2663806.2663847.

[9] **Shaker, M.**, **Shaker, N. and Togelius, J.** (2013). Evolving Playable Content for Cut the Rope through a Simulation-Based Approach, *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'13, AAAI Press, p.72–78.

[10] **Ferreira, L. and Toledo, C.** (2014). A Search-based Approach for Generating Angry Birds Levels, *Proceedings of the 9th IEEE International Conference on Computational Intelligence in Games*, CIG'14.

[11] **Goodfellow, I.J.**, **Pouget-Abadie, J.**, **Mirza, M.**, **Xu, B.**, **Warde-Farley, D.**, **Ozair, S.**, **Courville, A. and Bengio, Y.**, (2014), Generative Adversarial Networks, 1406.2661.

[12] **Radford, A.**, **Metz, L. and Chintala, S.** (2016). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, *CoRR*, *abs/1511.06434*.

[13] **Arjovsky, M.**, **Chintala, S. and Bottou, L.**, (2017), Wasserstein GAN, 1701.07875.

[14] **Stephenson, M.**, (2018), Angry Birds 2018 Level Generation Competition, http://aibirds.org/level-generation-competition/2018-results.html.

[15] **Summerville, A.**, **Snodgrass, S.**, **Mateas, M. and Ontanon, S.** (2016). The VGLC: The Video Game Level Corpus.

[16] **Kingma, D.P. and Welling, M.** (2019). An Introduction to Variational Autoencoders, *Foundations and Trends® in Machine Learning*, *12*(4), 307–392, http://dx.doi.org/10.1561/2200000056.

[17] **Hillier, B.**, (2014), Angry Birds downloaded 2 billion times, has as many MAUs as Twitter, https://www.vg247.com/angry-birds-downloaded-2-billion-times.

[18] **Matthew, C. and Lucas**, (2021), Angry Birds AI Competition, http://aibirds.org/.

[19] **Kerssemakers, M.**, **Tuxen, J.**, **Togelius, J. and Yannakakis, G.** (2012). A procedural procedural level generator generator, pp.335–341.

[20] **Hendrikx, M.**, **Meijer, S.**, **Van Der Velden, J. and Iosup, A.** (2013). Procedural Content Generation for Games: A Survey, *ACM Trans. Multimedia Comput. Commun. Appl.*, *9*(1), https://doi.org/10.1145/2422956.2422957.

[21] **Dahlskog, S. and Togelius, J.** (2012). Patterns and Procedural Content Generation: Revisiting Mario in World 1 Level 1.

[22] **Yannakakis, G. and Togelius, J.** (2011). Experience-Driven Procedural Content Generation, *Affective Computing, IEEE Transactions on*, *2*, 147–161.

[23] **Ebert, D.**, **Musgrave, F.**, **Peachey, D.**, **Perlin, K.**, **Worley, S.**, **Mark, W. and Hart, J.** (2002). *Texturing and Modeling: A Procedural Approach: Third Edition*.

[24] **Shaker, N.**, **Togelius, J. and Nelson, M.** (2016). *Procedural Content Generation in Games*.

[25] **Zafar, A. and Hassan, S.** (2019). Corpus for Angry Birds Level Generation.

[26] **Summerville, A.**, **Snodgrass, S.**, **Guzdial, M.**, **Holmgård, C.**, **Hoover, A.K.**, **Isaksen, A.**, **Nealen, A. and Togelius, J.**, (2017), Procedural Content Generation via Machine Learning (PCGML), https://arxiv.org/abs/1702.00539.

[27] **Volz, V.**, **Schrum, J.**, **Liu, J.**, **Lucas, S.**, **Smith, A. and Risi, S.** (2018). Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network.

[28] **Dewantoro, M.F.**, **Abdullah, F.**, **Paliyawan, P.**, **Thawonmas, R. and Bachtiar, F.A.** (2020). Enhancement of Angry Birds Level Generation from Sketches Using Cycle-Consistent Adversarial Networks, *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, pp.564–565.

[29] **Khaled, R.**, **Nelson, M. and Barr, P.** (2013). Design metaphors for procedural content generation in games, pp.1509–1518.

[30] **Wikipedia**, (2022), Roguelike, `https://www.wikiwand.com/en/Roguelike`.

[31] **Irfan, A.**, **Zafar, A. and Hassan, S.** (2019). Evolving Levels for General Games Using Deep Convolutional Generative Adversarial Networks, pp.96–101.

[32] **Tamir, D.M.**, (2020), What Is Machine Learning (ML)?, `https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/`.

[33] **Miralles, L.**, **Rosso, D.**, **Jiménez, F. and García, J.** (2017). A methodology based on Deep Learning for advert value calculation in CPM, CPC and CPA networks, *Soft Computing*, *21*, 1–15.

[34] **Rodriguez, J.**, (2020), Microsoft Research Unveils Three Efforts to Advance Deep Generative Models, `https://jrodthoughts.medium.com/`.

[35] **Yokoyama, N.**, (2022), Generative Adversarial Networks, `http://naoki.io/dlt/gan.html`.

[36] **Calle, L.**, **Guervós, J.**, **Garcia-Valdez, M. and Mora, A.** (2019). Speeding Up Evaluation of Structures for the Angry Birds Game, pp.237–244.

[37] **Zhu, J.Y.**, **Park, T.**, **Isola, P. and Efros, A.** (2017). Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks, pp.2242–2251.

[38] **Dewantoro, M.F.**, **Abdullah, F.**, **Paliyawan, P.**, **Thawonmas, R. and Bachtiar, F.A.** (2020). Enhancement of Angry Birds Level Generation from Sketches Using Cycle-Consistent Adversarial Networks, *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, pp.564–565.

[39] **Tanabe, T.**, **Fukuchi, K.**, **Sakuma, J. and Akimoto, Y.** (2021). Level generation for angry birds with sequential VAE and latent variable evolution, *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, `https://doi.org/10.1145%2F3449639.3459290`.

[40] **Limited, Z.U.**, (2022), Cut the Rope, `https://cuttherope.net/`.

[41] **Stephenson, M. and Renz, J.** (2016). Procedural Generation of Levels for Angry Birds Style Physics Games, *Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'16, AAAI Press.

[42] **Wikipedia**, (2022), Rube Goldberg machine, https://www.wikiwand.com/en/Rube_Goldberg_machine.

[43] **Abdullah, F.**, **Paliyawan, P.**, **Thawonmas, R.**, **Harada, T. and Bachtiar, F.** (2019). Angry Birds Level Generator with Rube Goldberg Machine Mechanism.

[44] **Tensorflow.org**, (2022), Why TensorFlow, https://www.tensorflow.org/.

[45] **Keras.io**, (2022), Deep learning for humans., https://keras.io/.

[46] **Unity**, (2022), Unlock your creativity, https://unity.com/.

[47] **Oracle**, (2022), Java+You Download Today, https://www.java.com/en/.

[48] **Goodfellow, I.**, (2017), NIPS 2016 Tutorial: Generative Adversarial Networks, https://arxiv.org/abs/1701.00160.

[49] **Frey, B.J.**, **Hinton, G.E. and Dayan, P.** (1995). Does the Wake-Sleep Algorithm Produce Good Density Estimators?, *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, MIT Press, Cambridge, MA, USA, p.661–667.

[50] **Frey, B.J.** (1998). Graphical Models for Machine Learning and Digital Communication.

[51] **Rezende, D.J.**, **Mohamed, S. and Wierstra, D.**, (2014), Stochastic Backpropagation and Approximate Inference in Deep Generative Models, https://arxiv.org/abs/1401.4082.

[52] **Hinton, G.E.**, **Sejnowski, T.J. and Ackley, D.H.** (1984). Boltzmann Machines: Constraint Satisfaction Networks That Learn, **Technical ReportCMU-CS-84-119**, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

[53] **Hinton, G.** (2007). Learning multiple layers of representation, *Trends in cognitive sciences*, *11*, 428–34.

[54] **Kullback, S. and Leibler, R.A.** (1951). On Information and Sufficiency, *The Annals of Mathematical Statistics*, *22*(1), 79 – 86, https://doi.org/10.1214/aoms/1177729694.

[55] **Chen, X.**, **Kingma, D.P.**, **Salimans, T.**, **Duan, Y.**, **Dhariwal, P.**, **Schulman, J.**, **Sutskever, I. and Abbeel, P.**, (2016), Variational Lossy Autoencoder, https://arxiv.org/abs/1611.02731.

[56] **Goodfellow, I.**, **Bengio, Y. and Courville, A.** (2016). *Deep Learning*, MIT Press, http://www.deeplearningbook.org.

[57] **Kreps, D.M.**, (1989). Nash Equilibrium, Palgrave Macmillan UK, London, pp.167–177, https://doi.org/10.1007/978-1-349-20181-5_19.

[58] **LeCun, Y. and Cortes, C.** (2010). MNIST handwritten digit database, http://yann.lecun.com/exdb/mnist/.

[59] **Kouridi, C.**, (2019), Vanilla GAN with Numpy, https://christinakouridi.blog/2019/07/09/vanilla-gan-numpy/.

[60] **Goodfellow, I.J.**, **Warde-Farley, D.**, **Mirza, M.**, **Courville, A. and Bengio, Y.** (2013). Maxout Networks, https://arxiv.org/abs/1302.4389.

[61] **Nair, V. and Hinton, G.** (2010). Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair, volume 27, pp.807–814.

[62] **Xu, B.**, **Wang, N.**, **Chen, T. and Li, M.** (2015). Empirical Evaluation of Rectified Activations in Convolutional Network.

[63] **Yu, F.**, **Seff, A.**, **Zhang, Y.**, **Song, S.**, **Funkhouser, T. and Xiao, J.**, (2015), LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop, https://arxiv.org/abs/1506.03365.

[64] **Deng, J.**, **Dong, W.**, **Socher, R.**, **Li, L.J.**, **Li, K. and Fei-Fei, L.** (2009). ImageNet: A large-scale hierarchical image database, *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp.248–255.

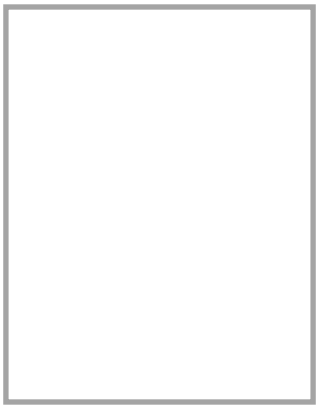
[65] **Ioffe, S. and Szegedy, C.** (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ***F. Bach and D. Blei**, editors, Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, PMLR, Lille, France, pp.448–456, https://proceedings.mlr.press/v37/ioffe15.html.

[66] **Maas, A.L.**, **Hannun, A.Y. and Ng, A.Y.** (2013). Rectifier nonlinearities improve neural network acoustic models, *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

[67] **Kingma, D.P. and Ba, J.**, (2014), Adam: A Method for Stochastic Optimization, https://arxiv.org/abs/1412.6980.

[68] **Tieleman, T. and Hinton., G.**, (2012), Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude., COURSERA: Neural Networks for Machine Learning.

CURRICULUM VITAE



Name Surname **:** Burkan YILMAZ

EDUCATION **:**

- **B.Sc.** **:** 2014, Istanbul Technical University, Faculty of Computer and Informatics, Department of Computer Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2014-2021 at Yapı Kredi A.Ş., Software Developer.
- 2021-2022 at Akbank A.Ş., Software Developer.
- 2022- ongoing at Garanti BBVA A.Ş., Software Developer.

B. YILMAZ	GENERATIVE ADVERSARIAL NETWORKS BASED LEVEL GENERATION FOR ANGRY BIRDS	2022