



UNIVERSITY OF
THESSALY

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

**ΕΦΑΡΜΟΓΗ ΕΡΓΑΛΕΙΩΝ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ
ΓΙΑ ΑΝΑΠΤΥΞΗ ΠΑΙΓΝΙΟΥ ΚΑΙ ΧΡΗΣΗ ΤΗΣ
ΜΗΧΑΝΗΣ UNREAL ΩΣ ΠΕΡΙΒΑΛΛΟΝ
ΠΡΟΣΟΜΟΙΩΣΗΣ ΒΑΘΙΑΣ ΕΝΙΣΧΥΤΙΚΗΣ ΜΑΘΗΣΗΣ**

ΚΟΥΡΚΟΥΤΑΣ ΔΗΜΗΤΡΙΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ
ΟΙΚΟΝΟΜΟΥ ΠΑΝΑΓΙΩΤΗΣ
ΒΑΘΜΙΔΑ
Adjunct Lecturer

ΛΑΜΙΑ, 2024



UNIVERSITY OF
THESSALY

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

**ΕΦΑΡΜΟΓΗ ΕΡΓΑΛΕΙΩΝ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ
ΓΙΑ ΑΝΑΠΤΥΞΗ ΠΑΙΓΝΙΟΥ ΚΑΙ ΧΡΗΣΗ ΤΗΣ
ΜΗΧΑΝΗΣ UNREAL ΩΣ ΠΕΡΙΒΑΛΛΟΝ
ΠΡΟΣΟΜΟΙΩΣΗΣ ΒΑΘΙΑΣ ΕΝΙΣΧΥΤΙΚΗΣ ΜΑΘΗΣΗΣ**

ΚΟΥΡΚΟΥΤΑΣ ΔΗΜΗΤΡΙΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ
ΟΙΚΟΝΟΜΟΥ ΠΑΝΑΓΙΩΤΗΣ
ΒΑΘΜΙΔΑ
Adjunct Lecturer

ΛΑΜΙΑ, 2024



UNIVERSITY OF
THESSALY

SCHOOL OF APPLIED SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

**EMPLOYING ARTIFICIAL INTELLIGENCE TOOLS IN
GAME DEVELOPMENT AND USE OF UNREAL ENGINE
AS DEEP REINFORCEMENT LEARNING SIMULATION
ENVIRONMENT**

KOYRKOYTAS DIMITRIOS

FINAL THESIS

ADVISOR
OIKONOMOY PANAGIOTIS
POSITION
Adjunct Lecturer

Lamia, 2024

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις (1), που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί χωρίς να τα περικλείω σε εισαγωγικά και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάστηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.
2. Δέχομαι ότι η αυτολεξεί παράθεση χωρίς εισαγωγικά, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.
3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια
4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: ...23.../10.../20.24...
Ο – Η Δηλ.

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ABSTRACT

As artificial intelligence(AI) technology advances, more and more industries integrate it into their workflow. In this thesis the focus will be on the game development industry, analyzing all aspects involved in the process of developing modern video game titles and how cutting-edge AI technology can be utilized to optimize different parts of the workflow. From creating behavior, to generating digital content, a deep analysis of the up-to-date proposed literature will be conducted, and an array of widely available online AI tools will be featured in detail. Continuing with the development of a demo game using the proposed AI tools and unreal engine, combined with an attempt to leverage the simulation capabilities of the engine for the purpose of designing, implementing and training a Deep Reinforcement Learning agent inside the demo game. Ending with a discussion of how to adapt the game environment for simulating real-world scenarios as well as the potential benefits of employing agents trained in simulated environments for real-world application.

ΠΕΡΙΛΗΨΗ

Όσο η τεχνολογία τεχνητής νοημοσύνης εξελίσσεται, όλο και περισσότερες βιομηχανίες την εντάσσουν στη ροή εργασίας τους. Σε αυτή τη διπλωματική εργασία, θα δοθεί έμφαση στη βιομηχανία ανάπτυξης παιχνιδιών, αναλύοντας όλες οι πτυχές που εμπλέκονται στη διαδικασία ανάπτυξης σύγχρονων τίτλων βιντεοπαιχνιδιών και πώς AI τεχνολογίες αιχμής μπορούν να χρησιμοποιηθούν για τη βελτιστοποίηση διαφορετικών τμημάτων της ροής εργασίας. Από τη δημιουργία συμπεριφοράς, ως τη δημιουργία ψηφιακού περιεχομένου, θα πραγματοποιηθεί μια εκτενής βιβλιογραφική ανασκόπηση και θα παρουσιαστεί λεπτομερώς μια σειρά από ευρέως διαθέσιμα διαδικτυακά AI εργαλεία. Συνεχίζοντας με την ανάπτυξη ενός demo βιντεοπαιχνιδιού χρησιμοποιώντας τα προτεινόμενα εργαλεία και την μηχανή Unreal, σε συνδυασμό με μια προσπάθεια αξιοποίησης των δυνατοτήτων προσομοίωσης περιβάλλοντος της Unreal Engine με σκοπό τον σχεδιασμό, την υλοποίηση και εκπαίδευση ενός πράκτορα Βαθιάς Ενισχυτικής Μάθησης μέσα στο demo παιχνίδι. Τελειώνοντας με μια αναφορά στο πως μπορεί να προσαρμοστεί το περιβάλλον προσομοίωσης πάνω σε πραγματικά σενάρια και τα πιθανά οφέλη που μπορεί να προσφέρει η εκπαίδευση πρακτόρων σε εικονικό περιβάλλον με σκοπό την μετέπειτα χρήση τους σε πρακτικές εφαρμογές.

Contents

ABSTRACT	1
ΠΕΡΙΛΗΨΗ	1
CHAPTER 1 Introduction	1
(Sub-chapter 1.1) Origins of Gaming	1
(Section 1.1.a) Brief History of Games	1
(Section 1.1.b) Game Development	1
(Sub-chapter 1.2) Artificial Intelligence-AI	2
(Section 1.2.a) Introduction to the Concept of AI and connection with the Game Development Process	2
(Sub-chapter 1.3) Thesis Motivation and Objective	2
CHAPTER 2 Literature Review	4
(Sub-chapter 2.1) Behavioral AI	4
(Section 2.1.a) Finite State Machines	5
(Section 2.1.b) Behavior Trees	6
(Section 2.1.c) Neural Networks &Deep learning	10
(Sub-chapter 2.2) Generative AI	15
(Section 2.2.a) Variational Auto-Encoders – VAE	16
(Section 2.2.b) Generative Adversarial Network - GAN	19
(Section 2.2.c) Latent Diffusion Models	39
(Sub-chapter 2.3) Chapter Conclusions	53
Chapter 3 Modern Video Game Development and Analysis of a Video Game’s Components	54
(Sub-chapter 3.1) Definition of a Video Game	54
(Sub-chapter 3.2) Breakdown of Video Game Development	55
(Section 3.2.a) Evolution of the Game Development Industry	55
(Section 3.2.b) Roles of a Development Team	56
(Sub-chapter 3.3) Components of a Video Game	60
(Section 3.3.a) Theoretical Components	60
(Section 3.3.b) Technical Components	64
(Sub-chapter 3.4) Chapter Conclusions	73
CHAPTER 4 Challenges of Video Game Development and Benefits of Employing AI	74
(Sub-chapter 4.1) Challenges Depending on the Scale	74
(Section 4.1.a) Large Scale game development	74
(Section 4.1.b) Small Scale game	76
(Sub-chapter 4.2) Benefits of Employing AI in Game Development	77
(Section 4.2.a) Prototyping	77
(Section 4.2.b) Productivity	77

(Section 4.2.c) Individual or Small teams	78
(Section 4.2.d) Realism, Immersion and Replay-ability	78
(Section 4.2.e) Post-Release Maintenance	78
(Sub-chapter 4.3) Negative Effects of AI	79
(Section 4.3.a)Reduction of jobs	79
(Section 4.3.b)Sentiment of uncertainty	79
(Section 4.3.c) Over-reliance on AI	79
(Sub-chapter 4.4) Chapter Conclusions	80
CHAPTER 5 Software used in Game Development and Replacement AI tools	81
(Sub-chapter 5.1) Software	81
(Section 5.1.a) Modeler	81
(Section 5.1.b) Painter	82
(Section 5.1.c) Audio Software	82
(Section 5.1.d) Game Engine	83
(Sub-chapter 5.2) AI Tools	84
(Section 5.2.a) 3D Models – Sloyd	84
(Section 5.2.b) Textures – Polycam and Real-ESRGAN	86
(Section 5.2.c) Audio – SOUNDRAW, Stable Audio and EvokeMusic	90
(Section 5.2.d) AI assistant - ChatGPT	95
(Section 5.2.d) Concept Art – Stable Diffusion/Easy Diffusion	96
(Sub-chapter 5.3) Chapter Conclusions	98
CHAPTER 6 Deep Reinforcement Learning inside Unreal Engine	99
(Sub-chapter 6.1) Agent Training inside Unreal Engine	99
(Section 6.1.a) Introduction to the MIND MAKER plug-in	99
(Section 6.1.b) Custom Environment for Agent Training	100
(Section 6.1.c) Designing an Agent	103
(Sub-chapter 6.2) Deep RL Algorithm Chosen for Agent Training	112
(Sub-chapter 6.3) Experiments, Training Results and Observations	113
(Section 6.3.a) Experiment 1- Movement and camera control	114
(Section 6.3.b) Experiment 2- Improved Movement and camera control	116
(Section 6.3.c) Experiment 3- Stationary Aiming	117
(Section 6.3.d) Experiment 4- Stationary Aiming and Shooting	119
(Section 6.3.e) Training Observations and Scalar Curves	121
(Sub-chapter 6.4) Adapting Unreal Engine Environment for Real-World Applications Simulation	133
CHAPTER 7 CONCLUSIONS	135
BIBLIOGRAPHY	136

List of Figures

1	Life magazine photo of comedian Danny Kaye standing in front of Bertie the Brain at the Canadian National Exhibition in 1950 Source:[8]	4
2	Example of Finite State Machine Behavior Control	5
3	Hierarchical FSM example. Same colored arrows represent interactions between states of the same level of hierarchy. B3 is zoomed out view of inner state of B.	6
4	A behavior tree that will retrieve and then try to enter every single door into a building and return success if the character succeeded in getting in any of the doors, and it will return failure if they did not. Source:[22]	7
5	Overview of the Robertson-Watson approach. Source:[24]	8
6	Reducing the BT using GLAM2. Source:[24]	8
7	Simple Unreal Engine 4 "Zombie" Behavior Tree. Source:[28]	10
8	Simple neural network graph representation. Source:[30]	10
9	Deep Neural Network Architecture Source:[33]	11
10	Path finding comparison Blue is on-policy and Red is off-policy Source:[34]	12
11	Value-Based and Policy-Based Algorithms. [Image by Owen Louis, Reproduced from David Silver's RL Course]	13
12	Online and offline learning processes comparison.	14
13	Overview of the OpenAI Five model. Source:[38]	15
14	Simplified latent space representation. Source:[41]	16
15	VAE Architecture Source:[42]	17
16	Mak,Han,Yin VAE model. Source:[44]	17
17	The zoom-in version of an output game map of Arknights after the mixing of several processed images in the VAE model. Source:[44]	18
18	Two sample output images obtained from the VAE model based on the anime-girl dataset. Source:[44]	18
19	Overview of C-GAN architecture. Source:[47]	19
20	BiGAN Architecture. In addition to the Generator G, an encoder maps data to latent representations. The BiGAN discriminator discriminate jointly in data space and latent space, where the latent component is either an encoder output or a generator input. Source:[49]	20
21	StyleGAN Generator. Source:[51]	20
22	StyleGAN Face mixing example. Source:[51]	21
23	CycleGAN architecture Horse to Zebra to Horse example. Source:[52]	21
24	DCGAN architecture. Source:[54]	22
25	GameGAN architecture. Source:[57]	23
26	GameGAN Element Swapping. Source:[55]	23
27	MC-GAN Architecture. Source:[58]	24
28	Context Encoder Architecture with example. Source:[59]	25
29	a) A Wang tile with all combinations of marked corners. b)Tiling from the expanded tile set that shows various combinations of marked/unmarked tile corners. c) two source images d) tiling that contains density information. e) a tiled plane (12x12) constructed from a tile set with inhomogeneities. Source:[60]	25

30	SeamlessGAN Architecture. Source:[61]	26
31	Comparison of SeamlessGAN with other technics. SeamlessGAN method generally captures better the overall structure of the texture, while providing seamless and semantically coherent borders, for enhancing tileability. Source:[61]	27
32	Overview of EST-GAN Architecture. Source:[63]	29
33	Example of Sampled G-Buffer Frames. Left from Unity Domain X. Right from RDR2 Domain Y. Source:[63]	29
34	Comparison of a) Sampled Albedo b) CycleGAN generated Albedo c) CycleGAN+Semantic Loss Albedo. Source:[63]	30
35	a) Domain X sharp normal edges b) Domain Y smoother edges c)Domain Y after Clustering. Source:[63]	30
36	Left Baseline Unity Frame. Right EST-GAN final model produced Frame. Source:[63]	31
37	Different 3D representations of a rabbit. Source:[67]	31
38	Ben Hamu et al model results. Source:[70]	32
39	Spick et al. GAN model architecture. Source:[76]	33
40	Spick et al. GAN model Results. Source:[76]	33
41	Spick et al. GAN model Results after Marching Cubes. Source:[76]	34
42	IG-GAN architecture. Source:[77]	35
43	Normal maps of objects generated by IG-GAN using multiple images of the same object from different points of view. Source:[77]	36
44	Terrain rendered from height-map using TerreSculptor.	37
45	Eric Guérin et al. terrain model architecture. Source:[79]	38
46	Left original image generated by PolycamAI 512x512. Right 4x upscaled image 2048x2048 using Real-ESRGAN (a free to use online implementation of the ESRGAN model).	39
47	Simplified example of a Hierarchy of Denoising Autoencoders. Forward process starts from top left and Backward process ends at top right.	40
48	Diffusion process example. Source:[86]	40
49	Latent Diffusion Model Architecture with notes. Source:[89]	41
50	3D point cloud diffusion. Source:[94]	43
51	Partial point clouds from depth map and PVD completed shape. Source:[99]	44
52	a) partial point cloud b) Possible variations. Source:[99]	44
53	LION same voxel input resulting in different final generations. Source:[101]	45
54	LION Architecture. Source:[101]	46
55	Example of a) Waveform b) MIDI representation.	47
56	MIDI latent diffusion model Architecture.	47
57	AudioLDM Architecture. Source:[110]	48
58	TANGO Architecture. Source:[117]	50
59	AudioLDM2 Architecture. Source:[121]	51
60	Stable Audio timing variables. Source:[123]	52
61	Difficulty curve examples a) Steady increase b) Easy to learn hard to master c) Teaching new mechanics d) Sudden difficulty spike, bad design. Source:[138]	63
62	Overview of the Left 4 Dead 2 director's influence in the game.	63

63	a) Crash Bandicoot b) Half-life 2 - Alyx c) Skyrim Daedric Armor d) Diablo IV Barbarian.	65
64	Example of LODs. Left-most is closest to camera Right-most the furthest away.	65
65	a) Animated Materia example from unreal engine 5. b) List of all supported maps used in unreal engine's PBR workflow. In this case (Opaque) only the supported ones are enabled for use.	66
66	Collision example from unreal engine 5. The GREEN lines represent the simple and the WHITE lines the complex collision.	67
67	Armature and mesh(displaced) from front, side and top view.	70
68	SLOYD AI playground interface.	84
69	Polycam Texture Generator interface. Prompt is used to generate 4 different variations of the described texture.	86
70	Real-ESRGAN playground interface. Source:[147]	88
71	a) Original generated texture b) Real-ESRGAN upscaled version.	90
72	Soundraw Interface.	91
73	Soundraw Track Editing Interface.	93
74	Stable Audio Interface.	94
75	Easy Diffusion Interface.	96
76	Prompt: "Sci fi space station, very dark environment, few emissive green red and blue lights, reflective surfaces"	97
77	Prompt: "Sci fi space station, very dark environment, few emissive green red and blue lights, reflective surfaces"	97
78	Prompt: "Sci fi space station, dark environment, emissive lights"	98
79	Unlit top-view of the level to be used as a simulation environment.	101
80	Screenshot 1 taken from the UE5 Simulation Environment.	101
81	Screenshot 2 taken from the UE5 Simulation Environment.	102
82	Screenshot 3 taken from the UE5 Simulation Environment.	102
83	Screenshot 4 taken from the UE5 Simulation Environment.	103
84	Agent training loop within the unreal engine environment.	103
85	Demonstration of the movement-camera linked system.	114
86	The symmetric distance problem. A and B are the same distance away from the goal but in different locations	116
87	The camera Z rotation problem. A and B are in the same location but have different camera Z rotation	116
88	Agent aiming diagram. The red dot represents the predicted hit location and yellow X the target location.	118
89	Agent aiming on target, despite the red dot not being on the yellow X.	120

List of Tables

2	Comparing On-policy and Off-policy	12
3	List of common Deep Reinforcement learning Algorithms	13
4	Tom Moore's Pix2Pix model generated PBR maps. Source:[62]	28
5	Comparison of discussed 3D GAN models	36
6	List of most important features of the three discussed models. Stable diffusion is clearly more advanced in terms of features due to the open-source nature of the project.	42
7	Comparison of PVD and LION models.	46
8	Quantitative results on MusicCaps.	52
9	Quantitative results on AudioCaps.	52
10	Comparison of the discussed audio generative models.	53
11	Bethesda Studio's TES V:Skyrim Development team(Excluding Voice actors, musicians and production staff).Source:[136]	60
12	Prompt: "Cyberpunk, metal, wall panel, sci fi, technology"	87
13	Prompt: "Mossy Runic Bricks, stone, moss"	88
14	Left the generated 512x512 textures. Right their upscaled versions of Real-ESRGAN	89
15	The action space and low and high bounds of each variable.	98
16	The action space and low and high bounds of each variable.	105
17	The observation space as well as low and high bounds of each variable. Some variables have no actual upper bound, arbitrary 999999 value is set.	107
18	Custom hyperparameters used for the PPO algorithm.	114
19	Experiment 1 action space and low and high bounds of each variable.	115
20	Experiment 1 observation space/ State with low and high bounds.	115
21	Experiment 2 observation space/ State with example of low and high bounds.	117
22	Experiment 3 action space and low and high bounds of each variable.	118
23	Experiment 3 observation space/ State with low and high bounds.	118
24	Experiment 4 action space and low and high bounds of each variable.	119
25	Experiment 4 observation space/ State with low and high bounds.	120
26	Total timesteps of continuous training per experiment. sessions.	121
27	Movement experiment 1 scalar curves. Three runs of the first experiment with different Observation spaces. Orange is a run with distance only, Blue is a run with distance + xy location, Dark Orange is a run with distance + xy location + camera rotation.	125
28	Movement experiment 2 scalar curves. Light Blue is session 1, Pink is session 6, Dark Orange is session 11, Orange is session 16, Dark Green is session 21, Blue is session 31, White is session 36	127
29	Aiming experiment 3 scalar curves. Orange is session 1, Pink is session 5, Dark Orange is session 10, Blue is session 16	130

- 30 Aiming+shooting experiment 4 scalar curves. Orange is session 1, Dark Orange is session 3, White is session 7, Blue is session 11 133

List of Abbreviations

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor Critic
AI	Artificial Intelligence
API	Application Programming Interface
BG	Background (ambience) - noise from the environment
BPM	Beats Per Minute
BT	Behavior Tree
CAD	Computer Aided Design
cGAN	Conditional Generative Adversarial Network
CLAP	Contrastive Language Audio Pretraining
CPU	Central Processing Unit
CRT Monitor	Cathode Ray Tube Monitor
DCGAN	Deep Convolutional Generative Adversarial Network
DDM	Denoising Diffusion Model
DDPG	Deep Deterministic Policy Gradient
DM	Diffusion Model
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
DX	Dialogue - any verbal speech in the game
ELBO	Evidence Lower Bound
ESR-GAN	Enhanced Super Resolution Generative Adversarial Network
EST-GAN	Enhancing Style Transfer Generative Adversarial Network
FOL	Foley - any sound effect that the player makes
FSM	Finite State Machine
GAN	Generative Adversarial Network
GLAM2	Gapped Local Alignment of Motifs
GPS	Global Positioning System
GPU	Graphics Processing Unit
HFSM	Hierarchical Finite State Machine
HTS-AT	Hierarchical Token Semantic Audio Transformer
IG-GAN	Inverse Graphics Generative Adversarial Network
LDM	Latent Diffusion Model
LiDAR	Light Detection and Ranging
LION	Latent Point Diffusion
LOA	Language Of Audio
LOD	Level Of Detail
LSTM	Long Short Term Memory

MARL	Multi Agent Reinforcement Learning
MC-GAN	Multi-Class GameGAN
MIDI	Musical Instrument Digital Interface
MMO	Massive Multiplayer Online
MOBA	Multiplayer Online Battle Arena
MX	Music - any non-diegetic music
NPC	Non Player Character
PBR	Physics Based Rendering
PC-GAN	Point Cloud Generative Adversarial Network
PC	Personal Computer
PGC	Procedurally Generated Content
PPO	Proximal Policy Optimization
PVD	Point Voxel Diffusion
RL	Reinforcement Learning
RoBERTa	Robustly Optimized Bidirectional Encoder Representations from Transformers Approach
SAC	Soft Actor Critic
SAP	Shape AS Points
SARSA	State-Action-Reward-State-Action
SFX	Sound Effects (Hard Effects) - any sound from a real-life object
TANGO	Text to Audio using instruction-Guided diffusion
TD3	Twin Delayed Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimization
TTA	Text To Audio
VAE	Variational Auto Encoder
VFX	Visual effects
VGG	Visual Geometry Group

CHAPTER 1 Introduction

(Sub-chapter 1.1) Origins of Gaming

(Section 1.1.a) Brief History of Games

Since the dawn of humankind, games have been intertwined with human life. Everyone has heard and used the word Game instinctively but if you ask anyone for a definition, they stare with a troubled look on their face. As per the Britannica Dictionary definition of GAME «a physical or mental activity or contest that has rules and that people do for pleasure».[1]

Humans most likely engage in such games, in their simplest form, even before learning to speak or write and as the species evolved so did the games. Bigger brain meant more complex games and the continuous evolution of civilization carried along the games which became more and more intricate.

In his 1938 book, *Homo Ludens*[2], Dutch cultural historian Johan Huizinga argued that games were a primary condition of the generation of human cultures. Huizinga saw the playing of games as something that "is older than culture, for culture, however inadequately defined, always presupposes human society, and animals have not waited for man to teach them their playing"

Language, law, war, philosophy, and art are all concepts, Huizinga, believed to be derived from innate human compulsion to play games. As society matured and science advanced through the centuries, so did the games.

(Section 1.1.b) Game Development

Games in general don't follow a single linear evolutionary path. Since antiquity games of similar traits appear in different parts of the world who did not take part in sociocultural exchanges. The term "convergent evolution" is used in biology to explain "the process by which unrelated or distantly related organisms evolve similar body forms, coloration, organs, and adaptations. Natural selection can result in evolutionary convergence under several different circumstances." [3]

Games and thus video games take part in a natural selection process based on popularity and resilience. From the almost infinite volume of games created only a handful can endure the test of time. So, the question is presented on how someone can create a game that can satisfy the criteria of a successful game.

The process of game development is strenuous and needs to take account of many aspects of human society and culture. It can involve one or many people, who through mutual collaboration and skill sharing, create a game.

From conception to completion, it may take an unfathomable amount of effort. This is not only referring to the technical difficulties one may encounter but in the process of designing a game based on an abstract idea.

It takes considerable skill to transform an idea into a game. From the playing medium to rules to target audience, the game designer must make decisions that will shape the game. The choice of elements to use, board, dice, cards or inventing an entirely new one, to the construction of a

comprehensive set of rules in order to provide the players with an entertaining yet challenging experience.

Video games share a lot of the characteristics of traditional games, as they are an evolution of them, played in a different medium.

(Sub-chapter 1.2) Artificial Intelligence-AI

(Section 1.2.a) Introduction to the Concept of AI and connection with the Game Development Process

Artificial intelligence is not a new concept within the scientific community. Mathematicians tried to simulate human intelligence inside computer processes since the early years of their invention. Allan Turing published "COMPUTING MACHINERY AND INTELLIGENCE" in 1950 [4], asking the question "Can machines think?" and sowed the seeds of an entire computer science field but the term Artificial intelligence was not coined till 1956, when John McCarthy used the term for the first time at the Dartmouth Summer Research Project on Artificial Intelligence.[5]

In today's age, the massive increase in computing power enables Artificial intelligence and all its branches like Machine Learning, Knowledge Mining, Computer Vision and many more to find uses in a wide variety of real-world application. Benefitting greatly from the rise of the internet the enormous amount of data being gathered, assisting in the training process of complex AI models that will more closely resemble human thinking.

For many decades after its conception, artificial intelligence was considered to be a "predictive" tool, trained on collected data and attempting to find patterns in order to forecast, classify values or events and provide insight based on those historical data.

The continuous evolution of AI is remarkable and with the recent introduction of "generative" artificial intelligence, AI has started to offer solutions previously thought to be unattainable or cost prohibitive by human labor, while simultaneously expanding the ways in which it can be used.

As game developers reach for higher and higher with the games they are creating, it is inevitable that the requirements will be so high that the use of AI will be unavoidable. The modern game development process can be greatly benefited by utilizing the artificial intelligence technology, some of the pros of artificial intelligence in video games are, increasing employee productivity, reducing labor costs and identifying future trends based on collected player data.

(Sub-chapter 1.3) Thesis Motivation and Objective

The scientific community is working hard on constantly progressing the state of the art AI technology through research in various specialized fields. Image, video, audio, language and 3d geometry processing are well researched on an individual level and are all involved in the video game development industry.

The motivation for this thesis came from noticing a lack of substantial work on reviewing and discussing the potential uses and benefits of utilizing the state-of-the-art AI in the videogame development workflow, as well as realizing the simulation capabilities of game engines for cost effective and reliable data collection.

This thesis can be broken down in two parts, the theoretical part and the practical part.

1. The theoretical part

Chapter 2 – A review of scientific literature proposals on artificial intelligence techniques that have application in the domain of game development.

Chapter 3 – An attempt to define what a "Video Game" is and an in-depth analysis of the development process as well as breakdown of the technical components of a video game.

Chapter 4 – a highlight of the challenges faced during game development in both large and small scale teams and a discussion of the benefits of employing artificial intelligence for assisting in or replacing parts of the development process.

Chapter 5 – A showcase of the necessary software needed to develop a video game and web-based artificial intelligence tools that can substitute them or greatly reduce the developers' workload.

Chapter 6 – How to leverage the Unreal Engine as a simulation environment to train agents using deep reinforcement learning.

2. The practical part

A video game will be developed using the latest version of Unreal Engine and the proposed AI tools, that were showcased in the theoretical part of the thesis, will be used in order to prove the practical application of the claims made.

After completion of the game development process, an agent will be trained using deep reinforcement learning inside the virtual game world.

Many techniques from the academic field are utilized in creating believable characters and varied behaviors in video games but sometimes complex and costly in terms of computational power are most of the time avoided, because the developers desire a degree of control in certain interactions.

(Section 2.1.a) Finite State Machines

The first and simplest form of creating a behavioral AI inside a game is using a finite state machine (FSM) in order to simulate the behavior of an NPC. [9, 10, 11, 12, 13, 14, 15]

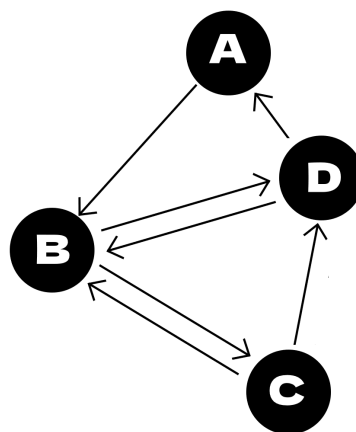


Figure 2: Example of Finite State Machine Behavior Control

This example FSM represents a simple enemy NPC behavior. Each circle represents a state of the FSM, A: patrol, B: Hunt, C: Attack, D: Seek. The enemy keeps track of many in game variables and transitions between states on the FSM depending on those.

Consider the scenario of an enemy being in state A: patrol, after player character is in sight transition to state B: Hunt, if player is within attack range transition to state C: attack. If the player runs away, then transition back to state B: Hunt. If the player leaves line of sight transition to state D: Seek. If certain window of time passes and the player is not found, then transition to state A: patrol. By following the flow of a predefined FSM, a game can simulate the behavior of an enemy character with ease.

Various propositions are made from the scientific community to combine the concept of FSMs with multiple different technologies like fuzzy logic [16], A* algorithm [17], evolutionary programming [18, 19] and more in order to create better models for decision making, path finding and NPC emotion status.

The complexity of FSMs increases rapidly based on the number of states, they start to become more and more difficult to design. A Hierarchical Finite State Machine (HFSM) [20] was the next step to combat the shortcomings of traditional FSMs. HFSM is nothing more than an extension of the basic FSM by creating composite states containing inside them states and transitions, thus making a hierarchy of FSMs.

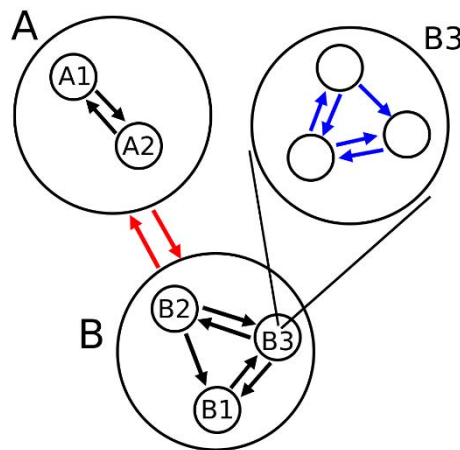


Figure 3: Hierarchical FSM example. Same colored arrows represent interactions between states of the same level of hierarchy. B3 is zoomed out view of inner state of B.

(Section 2.1.b) Behavior Trees

Using Behavior Trees (BT) for game AI is the most common technique used in current game development; they are a hierarchical structure of nodes that control the flow of the decision-making process. Compared to FSM approaches, BTs provide easier state transitions and are a simpler method to create more complex behaviors. [21]

There are 3 types of nodes used in BTs, Composite, Decorator and Leaf nodes.

1. Composite nodes: can have one or more children of different types.
 - a. Sequence: Behavior analogous to an AND logic gate. Executes all its children from first to last. If a child returns a failure the sequence stops and returns fail to the parent node. If the last child succeeds, then the sequence returns success to the parent node.
 - b. Selector: Behavior analogous to an OR logic gate. Executes all its children from first to last. If the first child fails then continues to the second and so on, but immediately breaks and returns success to the parent when a child succeeds. Returns fail only if all children fail.
 - c. Random Selector/Sequence: Identical functionality as Selector and Sequence but the order of visiting the children is randomized.
 - d. Parallel: executing all its children at the same time
2. Decorator nodes: has only one child and affect its behavior or result.
 - a. Inverter: Inverts the result of the child node. Similar to NOT logic gate
 - b. Succeder: always returns success regardless of the child's outcome.
 - c. Repeater: repeat the process of the child node every time child node returns a result or for a set number of iterations.

- d. Repeat until fail: repeats the process of the child node until it returns a fail.
3. Leaf nodes: Task nodes that implement the actual actions of the AI.

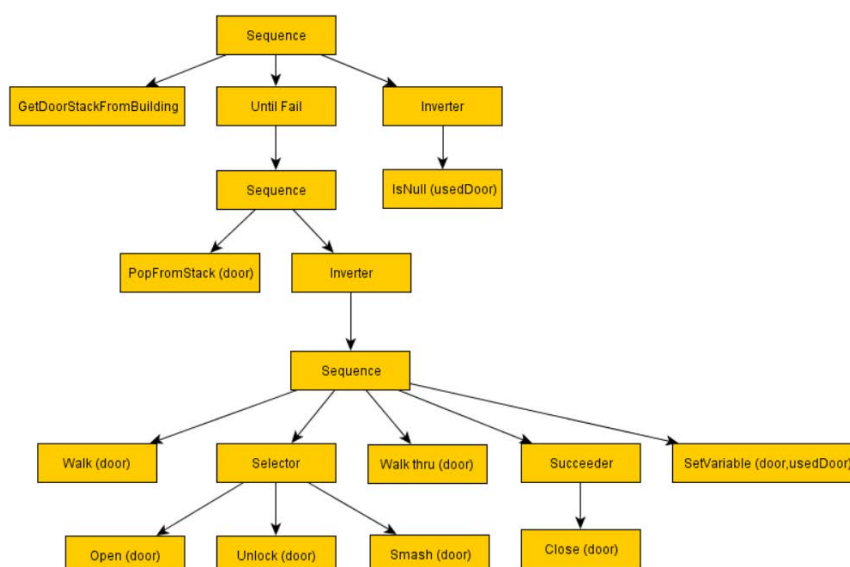


Figure 4: A behavior tree that will retrieve and then try to enter every single door into a building and return success if the character succeeded in getting in any of the doors, and it will return failure if they did not. Source:[22]

On top of this basic concept of a BT, many implementations have been proposed by the scientific community as well as the development industry. The main problem in creating BTs is in how to generate and improve them in a manner that doesn't require much human intervention.

Using Genetic Algorithms to generate BTs has been explored in research[23], Darwinian evolutionary methods can yield good results in decision making, where genetic factors are used to generate improved trees. The challenges of genetic algorithm implementations of BTs are that they require a predefined initial population and the design of a specific fitness function in order to evaluate the quality of generated BTs comprising the population. Depending on the case this kind of approach may not result in improved BTs.

A better method for generating and improving BTs is by observing expert players and using their decision patterns as a means of constructing a BT. Two such methodologies have been proposed, the first by Robertson and Watson and the second by Tomai and Flores.

The Robertson-Watson[24] method creates a maximally specific BT combining every action made by the players and then attempting to reduce the tree by implementing a pattern recognition mechanism using the Gapped Local Alignment of Motifs (GLAM2) software. Passing all sequences to the GLAM2, which uses a simulated annealing approach to evaluate matching patterns, and returning the new sequences and their scores when they cannot be further improved after a set number of iterations. The resulted sequences are evaluated by posing a threshold which discards every sequence that scored below it and the remaining sequences are used to construct the new BT.

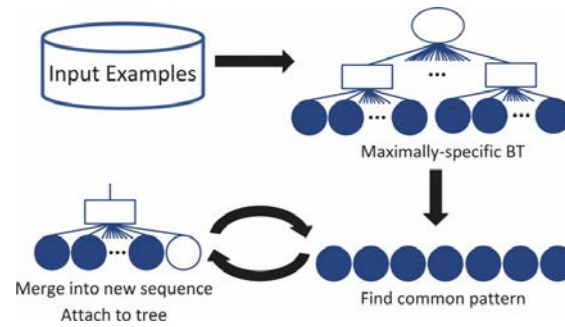


Figure 5: Overview of the Robertson-Watson approach. Source:[24]

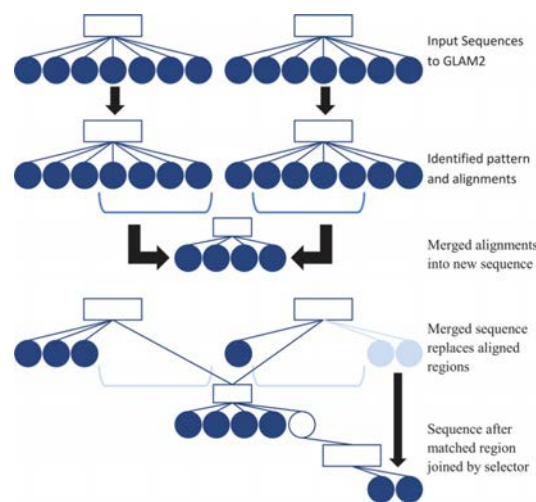


Figure 6: Reducing the BT using GLAM2. Source:[24]

The Tomai-Flores[25] method proposes a Learning BT approach by using a predefined BT, running an agent in sync with a playback recording of a player and trying to match the actions of the player with the behavior defined by the BT. If the player action diverges from the expected, then the algorithm automatically tries to alter the structure of the tree by adding Decorator nodes in order to create matching human-like behavior. The most common example is the addition of delay as players take time to consider their next action while the AI immediately processes the next action in the BT. Another example of diverging player behavior is when an enemy is killed the BT will automatically have the agent loot the corpse, but in some circumstances, players may decide not to loot and proceed to the next action.

The Tomai-Flores approach tries to closely imitate player actions in order to create agents with human-like behavior. That has the presupposes that the players always take objective oriented logical actions, but in fact that is also not true, as many players run around and jump aimlessly or make frantic movements while playing the game, making the evaluation of the resulted agent behavior very challenging as it is difficult to define what constitutes human-like behavior inside a game environment.

Instead of observing players' actions, another proposal to improve predesigned BTs is by implementation of utility theory approaches, to attempt recreating believable game AI. This

method does not alter the structure of the BT but instead adds principles of calculating utility for the purpose of selecting actions that will maximize the profit for the agent.

A group of Chinese researchers[26], proposed a reinforcement learning method for adding utility in BTs. Traditional Selector nodes have a left to right priority in child processing, they argue that a more natural consideration would be to attach weight to each child and by utilizing a reinforcement learning method of calculating those weights it would result in a more believable agent behavior. To achieve that they implemented the Q-Learning algorithm for selector node optimization. Q-Learning introduces a state-action evaluation function which calculates the profit of each action given the agent's state and chooses the one that will result in the highest reward in an attempt to mimic human player behavior.

Another utility-based approach is presented by Simonov- Zagarskikh- Fedorov [27], they introduce an intention-action system that chooses an action based on the agents' intentions. In the design phase developers create behavior groups and intentions, they attach a list of appropriate intentions to each behavior group depending on the personality they desire the agents of that group to have. During gameplay the agents get a list of available intentions and choose the best one given by a proposed utility evaluation function which considers environmental observations and personal characteristics, after that they calculate the utility trade-off for each intention-action pair and select the best action.

This approach produces believable agent behavior for agents of different behavior groups, but agents of the same group will have predictable behavior. To address this, they introduce Behavior Characteristics (BC) variables that influence the utility calculation process and diversify each agent based on individual circumstances. An agent may receive temporary BCs depending on factors such as game area, health state, mental state or game state.

One of the most popular game engines on the market, Unreal Engine (UE) implements an Event-Driven approach of BTs [28], instead of checking for potential changes in every game tick, UE system passively listens to relevant events that can be used to trigger changes in the BT. To achieve this, conditionals are removed from the leaf nodes functionality and instead are implemented as Conditional Decorators used as gates to block off entire sub-trees or as observers by listening to events triggers and if the conditions are met then they allow execution of the subtree. By doing this the computational cost of visiting the BT is decreased and thus achieving better game performance.

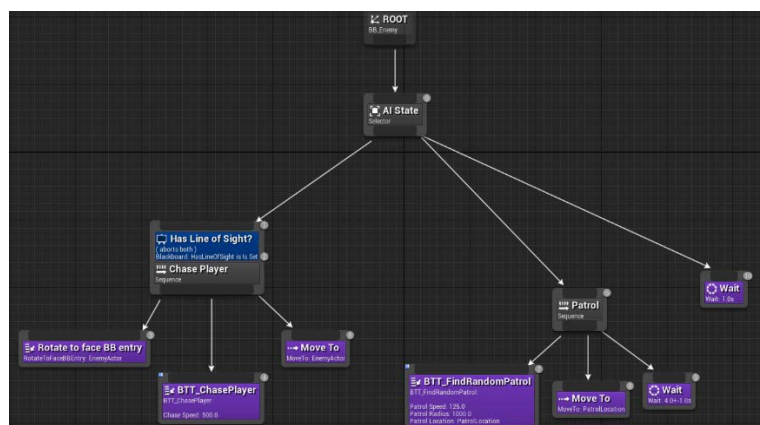


Figure 7: Simple Unreal Engine 4 "Zombie" Behavior Tree. Source:[28]

(Section 2.1.c) Neural Networks & Deep learning

In order to simulate human brain function inside a digital environment the scientific community developed the concept of Artificial Neural Networks. These networks are represented as weighted directed graphs consisting of "a layer of input neurons(nodes), one or two or even three layers of hidden neurons and a final layer of output neurons"[29]. Video game environments are ideal training grounds for neural networks as they provide a plethora of use cases and training variables. Games can leverage this technology to interpret input data and make probabilistic decisions to approximate realistic responses in whatever scenario they are put through.

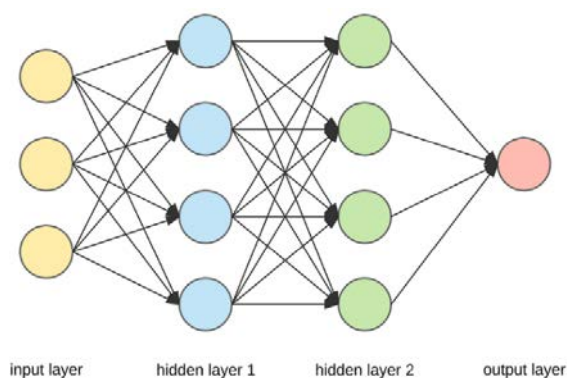


Figure 8: Simple neural network graph representation. Source:[30]

Some of the most common uses of neural networks inside video games include:

- **Environmental Scanning and classification:** The game AI can be taught to recognize audiovisual input from the environment and choose the appropriate response based on that.[12]
- **Memory:** The AI can learn a set of responses based on previous experiences and can choose how to respond to new situations. [12]

- **Prediction:** The network can be trained to recognize patterns in players' behavior, try to predict their next actions and respond according to this prediction.[31]
- **Behavior control:** The neural network can directly control the game AI by connecting the network's output as an input to various game engine variables. [12]

DEEP LEARNING

"Deep learning is a subset of machine learning that uses multi-layered neural networks, called deep neural networks, to simulate the complex decision-making power of the human brain. By strict definition, a deep neural network, or DNN, is a neural network with three or more layers. In practice, most DNNs have many more layers. DNNs are trained on large amounts of data to identify and classify phenomena, recognize patterns and relationships, evaluate possibilities, and make predictions and decisions." [32]

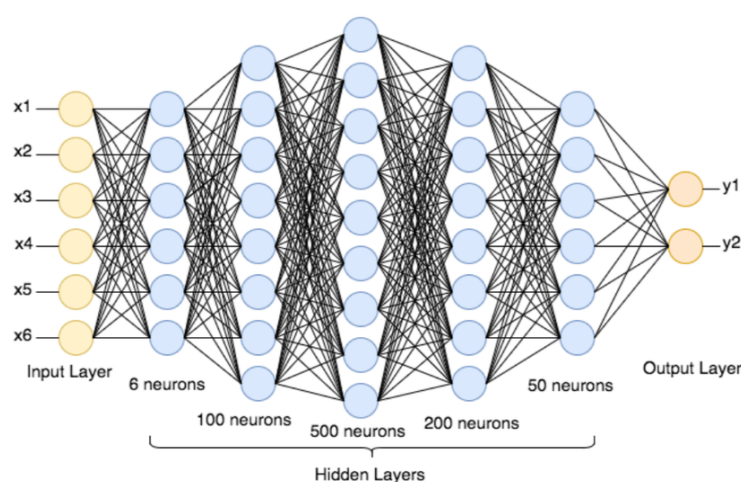


Figure 9: Deep Neural Network Architecture Source:[33]

By utilizing deep learning techniques to create the behavior of autonomous agents, video games can make the next step in terms of realistic behavior simulation.

The most popular deep learning approach in game development is deep reinforcement learning, that combines the advantages of neural networks and reinforcement learning technics to create agents who learn through trial and error.

Model-Free Policy Gradient methods are commonly used as a means to solve reinforcement learning problems such as the training of an agent inside a video game environment and depending on the task at hand an on-policy or an off-policy approach may be preferred.

On-Policy	Off-Policy
• Agent can pick actions. • Agent calculates state-action pairs to determine behavior policy. • Agent always follows his own policy.	• Agent cannot pick actions. • Agent learns through explorations and plays without exploration. • Agent can learn from expert (imperfect human) • Agent can learn from sessions (recorded data)

Table 2: Comparing On-policy and Off-policy

According to Sutton and Barto (2018 2nd edition)[34], On-policy methods result in an agent that follows a safer policy and as such don't take risky decisions, that makes the algorithm converge faster but it is more likely to be trapped in local minima.

On the other hand, off-policy methods are greedy and agnostic, that makes them less likely to get stuck in local minima thus increasing the possibility to achieve optimal policy. Due to their greedy nature off-policy techniques result in an agent that takes more risky decisions as it assumes that the agent will always take optimal future action.

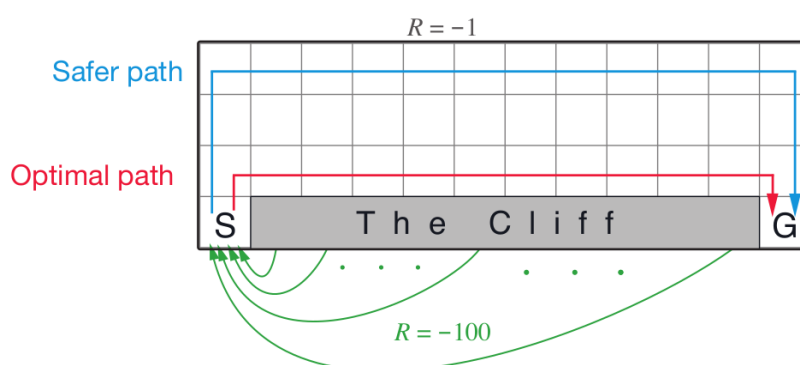


Figure 10: Path finding comparison Blue is on-policy and Red is off-policy Source:[34]

Value-based and Policy-based

Louis Owen in his article "Bird's-Eye View of Reinforcement Learning Algorithms Taxonomy" [35] explains the distinction between value/action based and Policy based approaches.

"The value-based methods try to pick the action which *maximizes* the action-state value function which will improve the policy in the direction to the best policy (faster and lower variance), while policy-based methods just take a little step and smoothly update in that direction which is more stable but at the same time is less efficient and sometimes leads to higher variance.

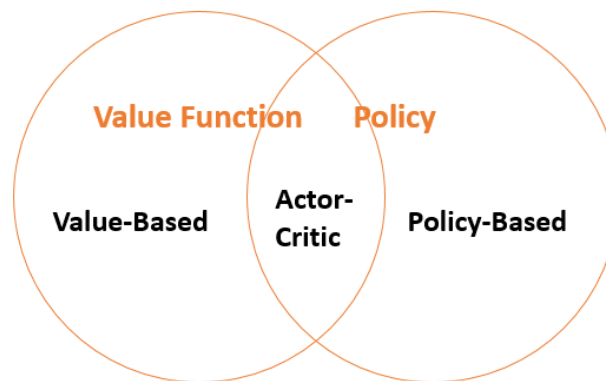


Figure 11: Value-Based and Policy-Based Algorithms. [Image by Owen Louis, Reproduced from David Silver's RL Course]

- **Value-based:** Agent aims to learn the value/action-value function in order to generate the optimal policy (i.e. the optimal policy is generated implicitly).
- **Policy-based:** Agent aims to learn the policy directly using a parameterized function.
- **Actor-Critic:** Agent aims to learn both the value function and the policy."

In implementations of video game AI value based and actor-critic algorithms are commonly used, here we provide a list of popular algorithms categorized appropriately.

	On-Policy	Off-Policy
Value based	Deep SARSA	Deep Q Network (DQN) Double DQN Dueling DQN
Actor-Critic	A3C A2C TRPO PPO	DDPG TD3 SAC

Table 3: List of common Deep Reinforcement learning Algorithms

Online and offline

Charles and McGlinchey [31] in their 2004 work on neural networks denote that video game AI can fall in two categories online and offline depending on whether the learning process stops or not after the release of the final product.

The most common approach is to fully train the agent while the game is in development and after the game's release no further improvements will be possible on the agent's behavior. If that's the case, then the AI falls in the offline category as its behavior has been developed only prerelease by the game makers.

The second approach is to begin the agents learning process during the game's development phase but continue to let the AI train itself after the final release of the game by collecting data

from players and develop its behavior as an accumulation of different playstyles. This method constitutes online learning as the AI continues to improve and adapt its behavior by interacting with the end users.

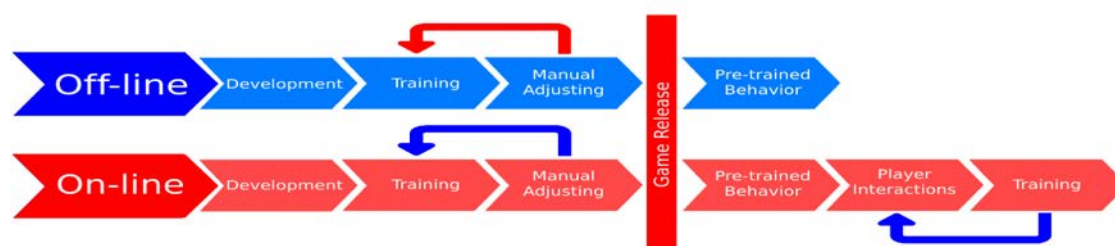


Figure 12: Online and offline learning processes comparison.

Both categories come with their respective advantages and disadvantages. The offline method may produce a realistic agent behavior and the developers can make changes to account for undesirable or buggy behavior before the game is released publicly, but that comes with the drawback that the agent has a set behavior making it predictable after some time and thus the players will learn to exploit its play patterns eventually.

On the other hand the online method will not produce predictable gameplay for the agent as it continuously learns and changes its behavior depending on the players it encounters in the game, but as Barnes and Hutchens point out in "Testing Undefined Behavior as a Result of Learning" (2002) [36] "The most significant issue with on-line learning is that it may produce unpredictable results; sometimes these effects serve to enhance but more often it leads to erratic game behavior that reduces the quality of gameplay, and in worse scenarios will introduce dynamic game bugs. Testing, debugging and balancing games that incorporate learning is a challenging task."

With online learning, the agent selects actions in real time based on interactions with the video game environment, it is free to train the policy using on-policy or off-policy updates. Offline techniques employ a pre-collected dataset without real-time interactions; therefore, they are closely connected with off-policy methods.

OPENAI FIVE

OpenAI is considered one of the leading figures in the research of artificial intelligence. In 2018 developed a multi-agent deep reinforcement learning model named "OpenAI Five" [37] for playing Valve's DOTA2 game. Dota2 is a Multiplayer online battle arena (MOBA) where two teams comprised of five players each compete to subjugate each other's base, inside a predefined area (arena).

The model is a scaled-up adaptation of the Proximal Policy Optimization algorithm (PPO) that also includes a Long Short-Term Memory (LSTM) network for each individual agent. Each LSTM is constituted by a single layer of 1024-units that extract data from Valve's BOT API.

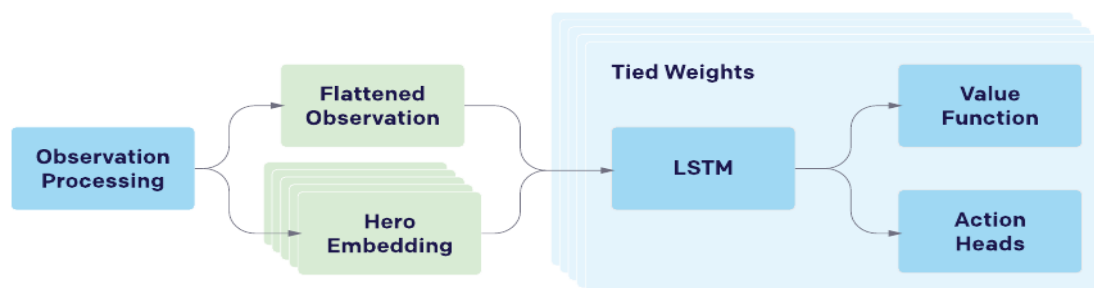


Figure 13: Overview of the OpenAI Five model. Source:[38]

Open AI Five utilizes self-learning only and does not rely on data gathered from human replays, the agent trains 80% of its games against itself and the other 20% against its past selves. It starts from randomized weights and tries to maximize the exponentially decayed sum of future rewards, weighted by an exponential decay factor valuing future rewards with a half-life of 46 seconds.

The metrics used to calculate the reward are the same that humans use to calculate performance: net worth, kills, deaths, assists, last hits etc.

The AI heroes do not communicate with one another, the "team spirit" hyperparameter is used to dictate how much each agent should care about his individual reward versus the overall team's average reward.

Slowly but steadily the AI learned to play the game. In the beginning the workers just wandered around aimlessly, after some training they started to learn the concepts of laning, farming, fighting over mid and steadily developed base level as well as high level strategies like stealing bounty, champion rotations, five people push and more. OpenAI Five averages around 150-170 actions per minute with a theoretical maximum of 450 and has an average reaction time of 80ms, which is faster than humans. In live testing it managed to beat teams of amateur (93rd percentile) and semi-professional (99th percentile) level.

(Sub-chapter 2.2) Generative AI

In the classical approach, video game content generation, also referred commonly as procedurally generated content (PGC), is different than modern generative AI. PGC focuses on generating adjustable game content, "refers to all aspects of a game that affect the player experience but are not non-player character (NPC) behavior or the game engine itself"[39], such as textures, props, enemies, terrains, maps, levels, quests, created by using a predefined set of assets as puzzle pieces to mix and match together based on predefined algorithms.

A very popular approach on PGC proposed by Giannakakis and Togelius is Experience Driven PGC [39], which leverages videogames as a prime ground for data mining from the player base in order to create models that generate content based on player preferences.

The main difference between PCG and Generative AI is that the first is done at run time, resulting in a different experience for the player each time he plays the game. Generative AI

focuses on the development phase of the game, as an assistive tool during the artistic process.

The objective of Generative AI is to create new synthetic data and content from existing data sets by utilizing latent spaces which are defined as abstract multi-dimensional spaces that encode a meaningful internal representation of externally observed events. It works by transforming raw data (image pixel values, soundwaves etc.) to suitable internal representations inside a latent space. The distance between two data points inside the latent space is dependent on the similarity of the data before transformation. The more similar the raw data the closer the distance of the latent representations. [40]

The focus of this chapter will be on Latent Variable models, explaining their inner workings and their practical uses as well as discussing some of the scientific community's proposals and implementations of those models that can be leveraged in game development applications.

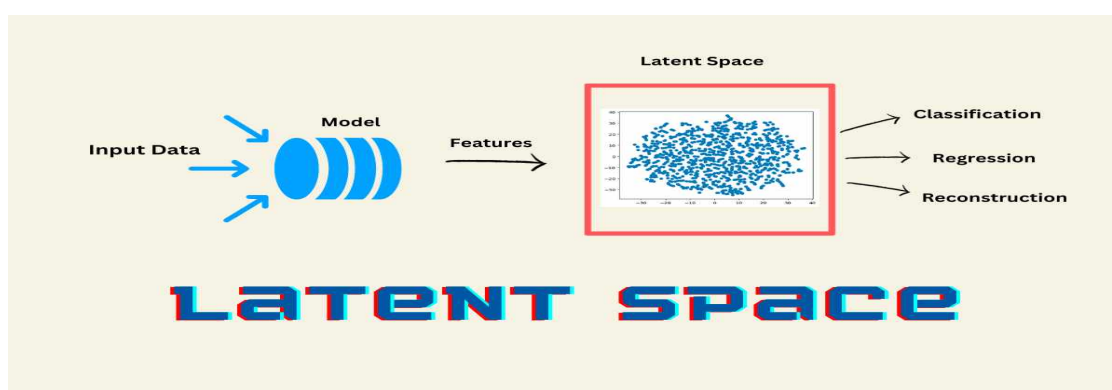


Figure 14: Simplified latent space representation. Source:[41]

(Section 2.2.a) Variational Auto-Encoders – VAE

Neural networks are a very useful tool in the machine learning world for learning representations of complex data in a lower dimension domain. The autoencoder architecture is based on two neural networks, the first is the encoder and the second is the decoder. The "auto" part of the name comes from the ability to automatically learn the best way to encode the input data so that the decoder results will be as close as possible to the initial input.

The encoder turns the input x to a latent representation z and the decoders tries to decode z to x' while being penalized for how much difference exists from x to x' .

Traditional autoencoder architecture results in sparsely populated latent space which is desired in compression and denoising applications but for generative purposes it is not optimal for predicting the distribution of the values inside z .

Variational Auto-Encoders (VAE) have been developed to tackle that sparsity problem, instead of feeding the latent variables to the decoder they try to force the data to become normally distributed and therefore more predictable. By utilizing the Kullback–Leibler divergence during training they attempt to make a distribution of the latent space, for the decoder to sample from, by calculating the mean standard deviation to match as close as possible to the standardized normal distribution.

This process makes the VAEs capable of exploring variations in a more specific direction

retaining the characteristics of the input data while providing the much desirable variation. Furthermore, with the inclusion of meta-data in terms of labels can produce an even more guided VAE model.

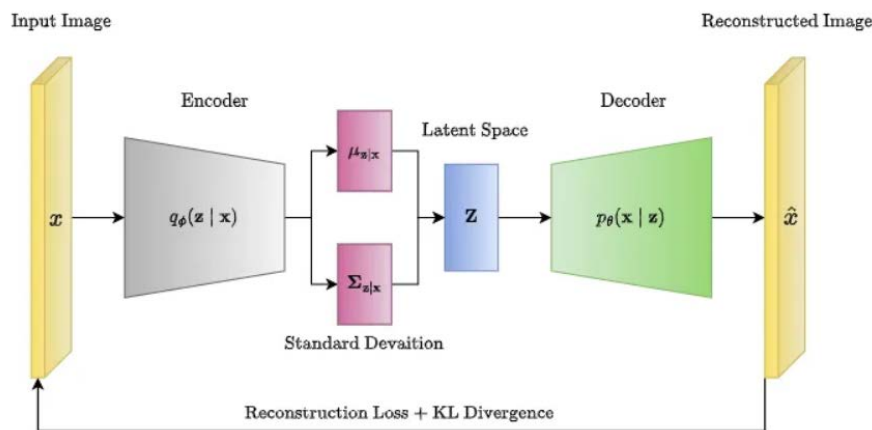


Figure 15: VAE Architecture Source:[42]

The main uses of VAEs include synthetic data generation, image classification, image segmentation, image reconstruction and anomaly detection, but most importantly they can be leveraged for their excellent nonlinear latent feature representation learning capability.[43]

Hugo Wai Leung Mak and his colleagues Runze Han and Hoover H. F. Yin [44] did an extensive case study on applications of VAE model for image processing in game design (2023). They experimented with level generation from levels of the Arknights game, stylized anime girl avatar generation and data clustering with the well-known MINST dataset.

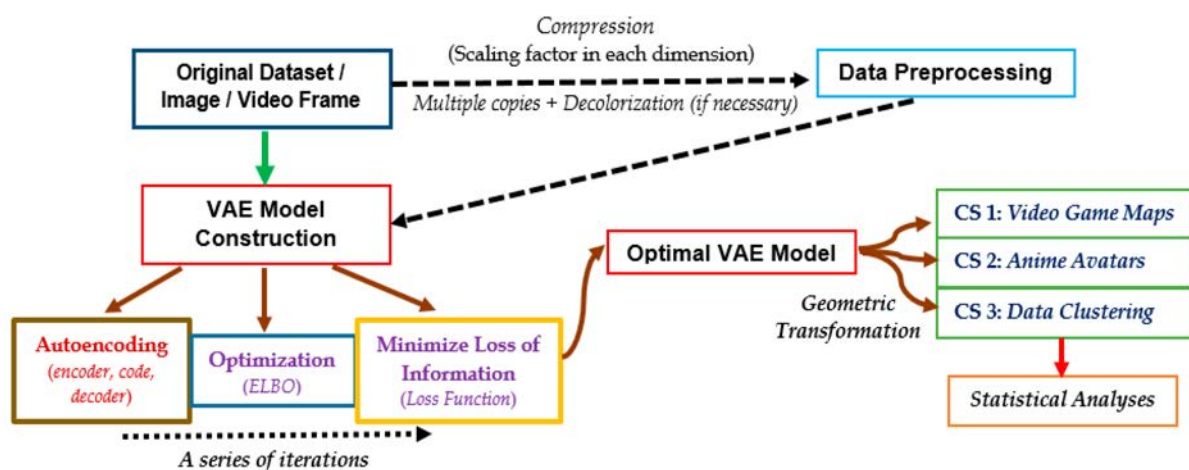


Figure 16: Mak, Han, Yin VAE model. Source:[44]

It was clearly illustrated in their research outcomes that VAEs are capable of generating images in a specific pattern and show a lot of promise in data clustering and in-game avatar generation, with the major drawback being the lack of resolution in the generated results.

They concluded that VAE models don't produce satisfactory results when used as a standalone model, thus in order to overcome their shortcomings and make use of their strength, they should be combined with other machine learning technologies such as Generative Adversarial Networks-GANs and Long Sort Term Memory- LSTM with the goal to increase the resolution of outputs, reduce training time and minimize the change of "mode collapse".[45]

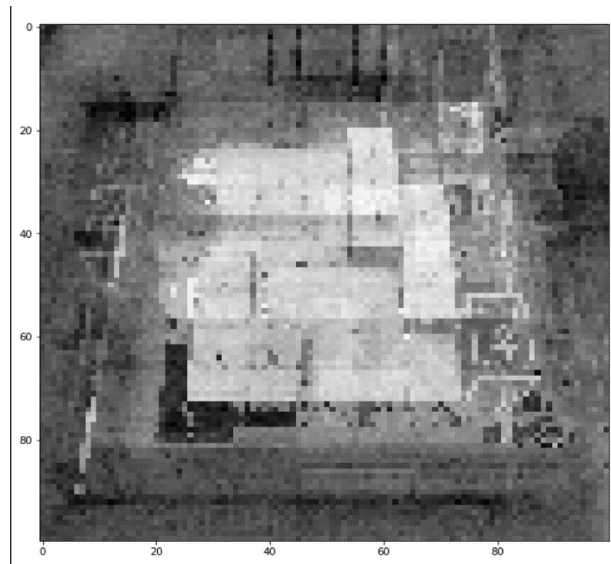


Figure 17: The zoom-in version of an output game map of Arknights after the mixing of several processed images in the VAE model. Source:[44]

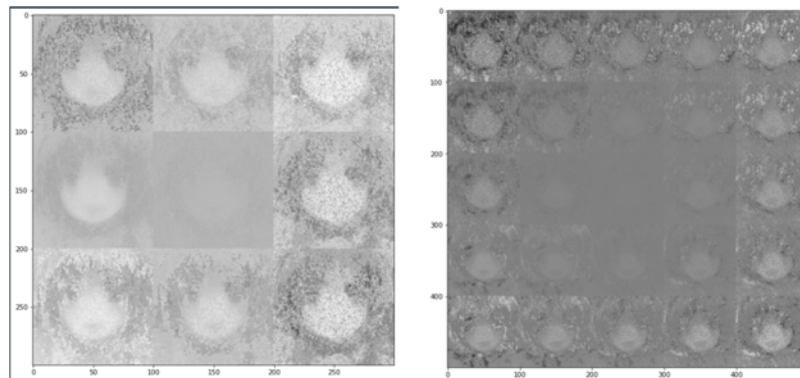


Figure 18: Two sample output images obtained from the VAE model based on the anime-girl dataset. Source:[44]

Furthermore, the ability of VAEs to learn nonlinear latent feature representations has made them an ideal sub-system of state-of-the-art diffusion model frameworks, which will be discussed in a later section, increasing their computational efficiency immensely as well as data compression and reconstruction capabilities which are required due to the large amount of data needed to train diffusion models.

(Section 2.2.b) Generative Adversarial Network - GAN

A GAN model consists of two neural networks which are trained at the same time resulting at an adversarial competition. Jakub M. Tomczak[46] in his book "Deep Generative Modelling" explains the concept as a competition between a fraud and an expert. The fraud tries to create a data in a specific style and the expert examines the data created by the fraud in order to determine if they are real or fake. The more they compete with each other the better they both will get but given enough time the fraud will be able to create fake data that are indistinguishable from the real data from the expert's perspective. We name the fraud as generator and the expert as discriminator, we can break the model as follows:

- A generator creates fake data from noise.
- A discriminator that examines and classifies the generated data as real or fake.
- Both the generator and the discriminator are parameterized using deep neural networks.
- The networks are learned by using adversarial loss.

Most Common Types of GANs

1. **Conditional GAN-cGAN:** Introduced labels in order to generate data with specific attributes. Enables Text-to-Image generation. Most popular implementations Pix2Pix, Pix2PixHD.[47]

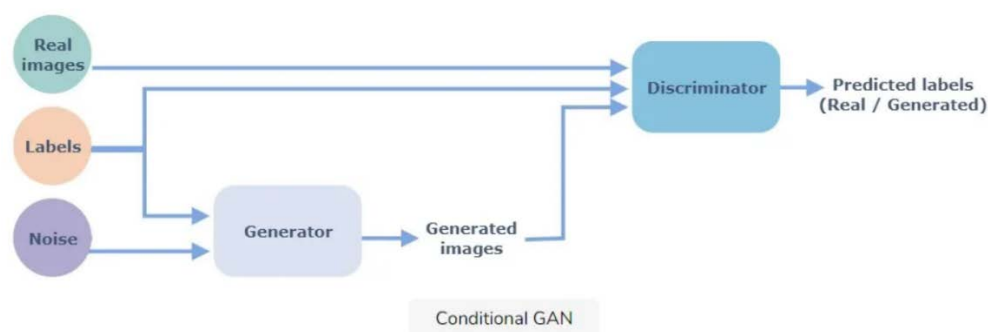


Figure 19: Overview of C-GAN architecture. Source:[47]

2. **GAN with Encoders:** Uses encoder to learn the inverse mapping of the generated data back to the latent space.[48] Most popular implementations BiGAN, ALI.

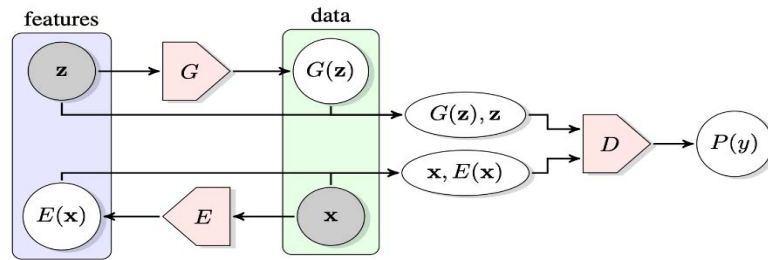


Figure 1: The structure of Bidirectional Generative Adversarial Networks (BiGAN).

Figure 20: BiGAN Architecture. In addition to the Generator G , an encoder maps data to latent representations. The BiGAN discriminator discriminates jointly in data space and latent space, where the latent component is either an encoder output or a generator input. Source:[49]

3. **StyleGAN:** Alters the generator architecture in order to generate photorealistic images of faces as well as images based on a specific style by using an intermediate latent space that contains style vectors.[50]

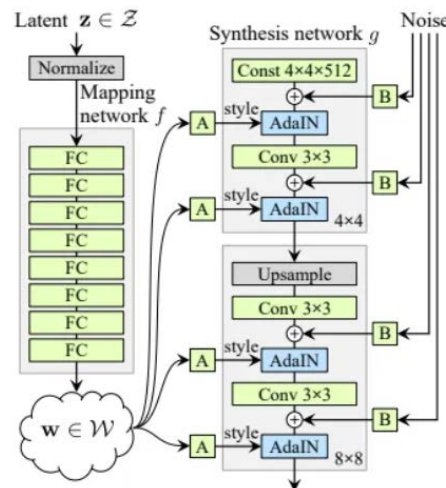


Figure 21: StyleGAN Generator. Source:[51]

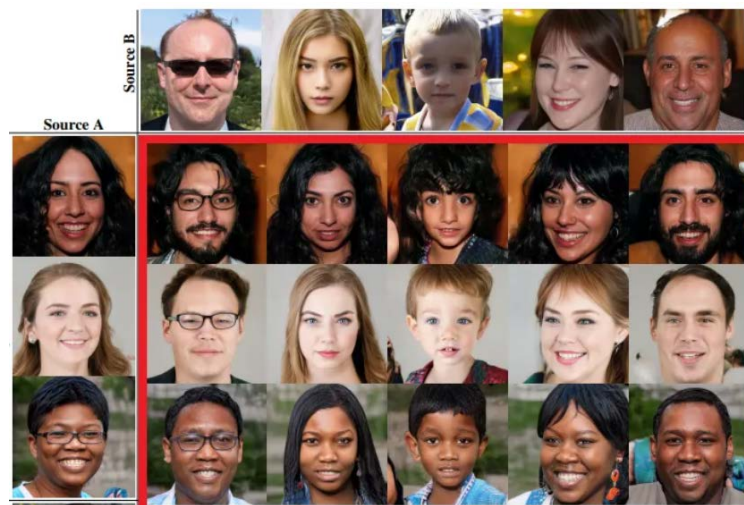


Figure 22: StyleGAN Face mixing example. Source:[51]

4. **CycleGAN:** Image-to-image translation without the need for one-to-one mapping from source to target. [50]

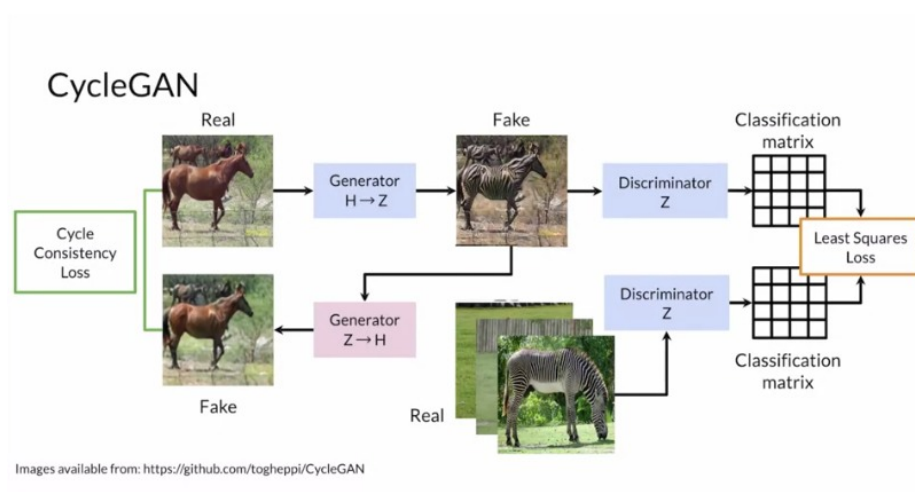


Figure 23: CycleGAN architecture Horse to Zebra to Horse example. Source:[52]

5. **Deep Convolutional GAN-DCGAN:** Generally used for unsupervised learning. Replaced the MLP for a Convolution Neural Network.[53]

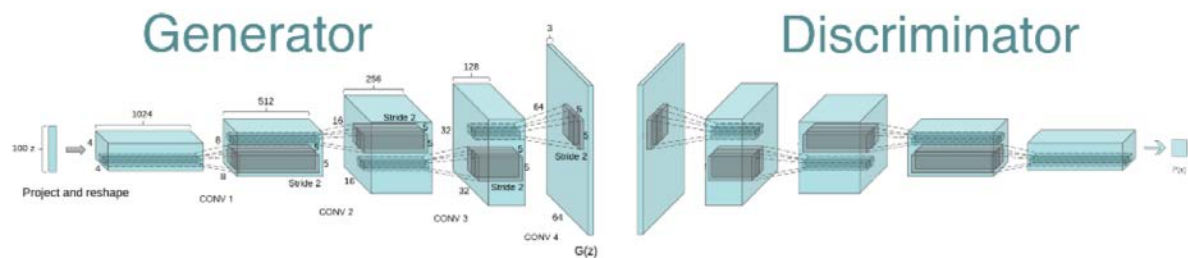


Figure 24: DCGAN architecture. Source:[54]

There are even more types of GANs that alter the metrics with which the adversarial loss is calculated such as Wasserstein GANs and f-GANs that use Wasserstein distance and f-divergence respectively.

The main limitation of the GAN model is called "mode collapse" that happens "when the generator focuses on producing a limited set of data patterns that deceive the discriminator. It becomes fixated on a few dominant modes in the training data and fails to capture the full diversity of the data distribution.

Just like the artist's fear of trying new styles, the generator avoids exploring the entire data distribution in GANs. It produces repetitive data samples, missing out on the rich variations present in the real data." [45]

Another problem that GANs suffer from is the inherent difficulty of non-standardized evaluation metrics which depending on the application may require the selection of a different metric in order to produce the desired results.

In respect of game development applications, GANs prove a very useful tool for many areas, such as generating dynamic game environments, seamless tileable textures, PBR material maps realistic terrains, sprites, post-process effects, 3D models and super-resolution.

Dynamic Game Environments

The problem of synthetic dynamic environment generation presented many difficulties in the past years, and many proposed methods were riddled with problems. "Action-LSTM models achieved success in scaling the generated images to higher resolution but do not handle complex stochasticity present in environments like Pacman. Recently proposed VAE-based frameworks to capture the stochasticity of the task. However, the resulting videos are blurry, and the generated frames tend to omit certain details. GAN loss has been previously used in several works and adversarial loss was used to disentangle pose from content across different videos." [55]

In 2020 a team of researchers proposed a model called GameGAN [55] by framing the problem as recurrent 2D image generation task to simplify it. The general idea of the model is to train itself in an unsupervised way by watching existing game footage, PACMAN and VIZDOOM were used as examples. The model is capable of distinguishing between static and dynamic game components as well as understanding the rules of the game by observing the agent's actions during training.

A challenging problem in traditional image processing is the detection of loop closures (returning to the same location), in this scenario the GameGAN simulator must generate the virtual environment and maintain temporal consistency during gameplay.

GameGAN is comprised of three components:

1. **Dynamics Engine:** implemented as an action conditioned LSTM for the purpose of learning non-permitted agent actions like walking through walls and how other game components should behave in response to agent's actions.
2. **Memory module:** in cases where maintaining long-term temporal consistency is needed an external memory module is used to remember every scene generated.
3. **Rendering engine:** a special implementation of standard transposed convolution layer-based renderer accounting for long-term consistency.[56]

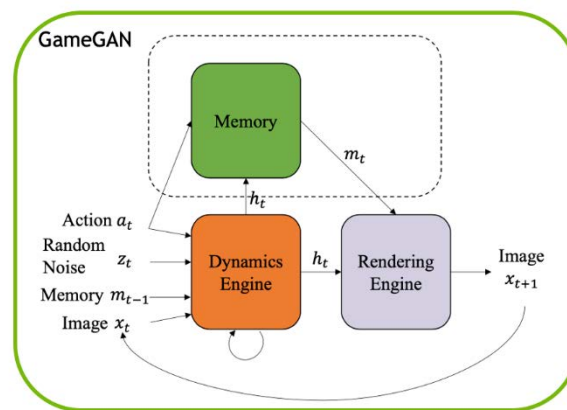


Figure 25: GameGAN architecture. Source:[57]

As the model is capable of understanding the difference of static and dynamic game components the researchers were able to create mix and match games by combining static and dynamic components of two different games at a time.

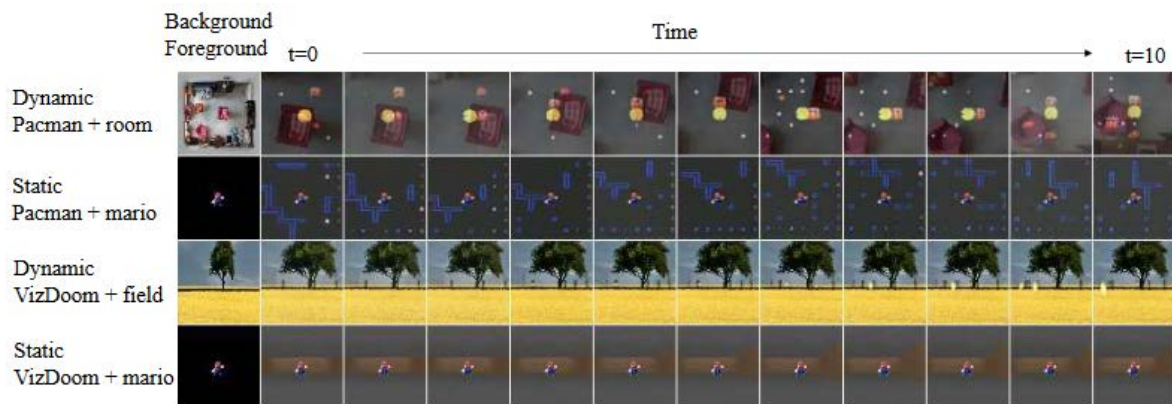


Figure 26: GameGAN Element Swapping. Source:[55]

One year later (2021) Ghasemshirazi et al. [58] proposed a model improving upon the work done in GameGAN, the new model was called Multi-Class GameGAN (MC-GAN). MC-GAN adds upon the player agent egocentric approach of the GameGAN and uses a MultiGAN Analyzer framework in order to train an Inner Model for each individual enemy type agent of the game, the player agent as well as every static element of the game. They achieve that by training a new version of the game with the camera focused on the desired component, for agents of the same type; the camera perspective is changed and for static components the camera is focused on them while the player agent seems to be fixed.

It is very obvious that the data complexity produced by the MultiGAN Analyzer is very big, therefore a Data Unifier is used to combine the data from all the classes and get them ready for rendering.

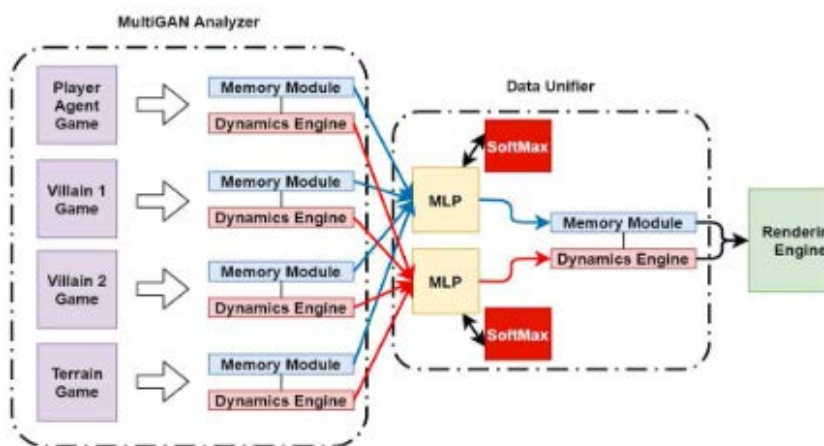


Figure 27: MC-GAN Architecture. Source:[58]

With this MultiGAN approach the model can not only disentangle static from dynamic components but can understand the different static and dynamic components. This enables the model to be able to swap between individual components from games, not all the static or dynamic sets. Another advantage of the MC-GAN is the ability to create additional new levels without the need of retraining for each one like the original GameGAN required.

As advancements continue, these methods start pointing to a new generation of AI-powered game engines, that may emerge in the near future, capable of creating mashup-games from combinations of existing ones only by observing them.

Seamless tileable textures

The game development process move often than not requires textures to be applied in large surfaces, when that's the case then the artists responsible for creation of those textures need to take in account the tileability component and transform the texture in such a way that when multiple copies of itself are placed side to side then no seam will be visible on the touching side.

The manual method inside an image editing software involves cropping the texture image, transforming the parts in specific orientation, and using stamps to remove the seams. It is labor intensive, and the process has to be replicated for every texture individually.

Considering the advances in image inpainting technology utilizing Context Encoders [59] to generate a missing part of an image based on surrounding context, someone might experiment creating seamless textures by using the classic manual method but replace the removal of the seams from stamps to context aware fill using technology.

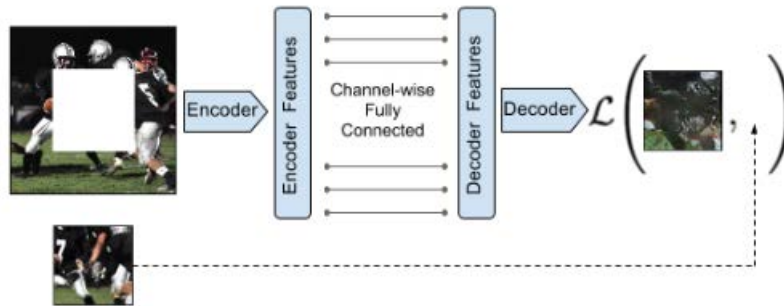


Figure 28: Context Encoder Architecture with example. Source:[59]

Michael Cohen et al.[60] proposed a Wang Tiles method of creating seamless textures by sampling smaller parts of a bigger exemplar and position samples with connecting borders next to each other. It has shown great results, but a big problem remains, an initial texture is required. Nowadays with the huge explosion in image synthesis technology this doesn't pose such a difficulty.

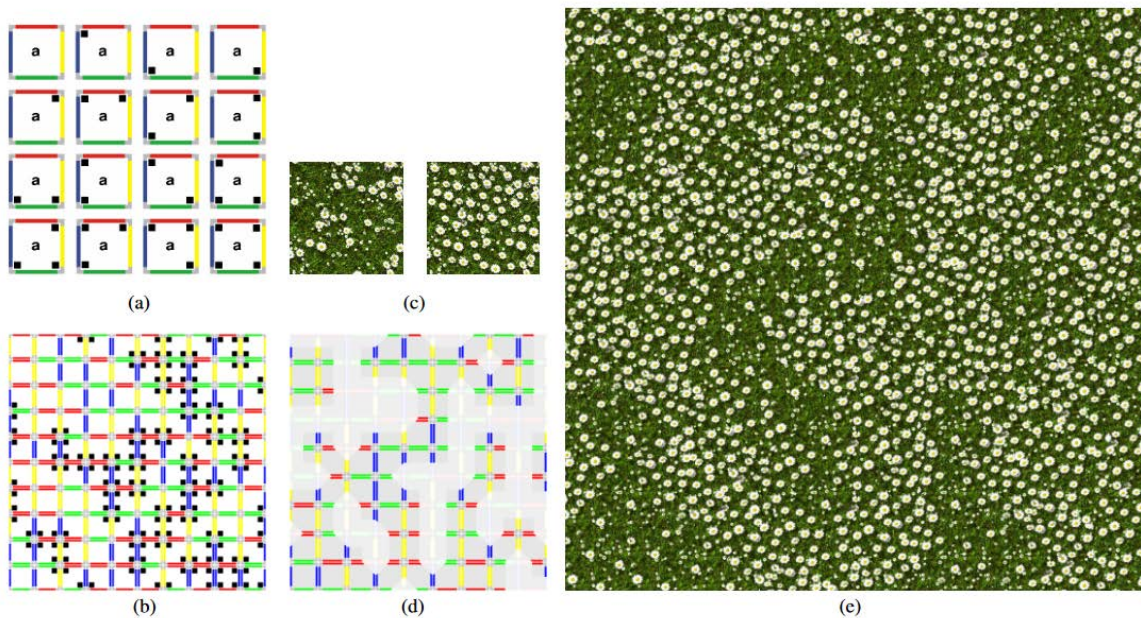


Figure 29: a) A Wang tile with all combinations of marked corners. b) Tiling from the expanded tile set that shows various combinations of marked/unmarked tile corners. c) two source images d) tiling that contains density information. e) a tiled plane (12x12) constructed from a tile set with inhomogeneities. Source:[60]

Another approach is to try generating a new seamless texture instead of trying to make an existing one seamless. "Creating tileable textures is a very challenging problem, as it requires a semantic understanding of the repetitive elements, often at multiple scales." [61] A proposal for a SeamlessGAN model was made by Rodriguez and Garces in 2023 [61]. The model is capable of generating an entire seamless texture stack (different PBR maps) given an untileable input texture stack.

It works in two steps, in the first step uses crops of the input stack to train a GAN that can generate new untileable textures double the spatial resolution of the input, in the second step a latent space concatenation is performed by repeating the latent space, created by the GAN's generator, horizontally and vertically, after the new latent field is passed through a decoder to remove discontinuities as well as artifacts and after the decoding process we obtain four copies of the original input crop with seamless transitions between them. The final seamless texture is achieved by cropping the central area of the decoder output textures put in 4x4 grid.

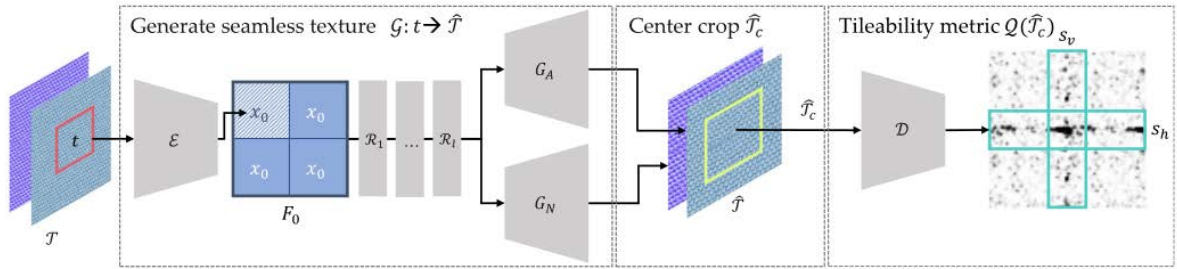


Figure 30: SeamlessGAN Architecture. Source:[61]

The SeamlessGAN can generate different tileable textures depending on the initial crop of the input texture and is currently the state-of-the-art approach to seamless texture stack synthesis, with a GAN, from single texture input and shown to outperform every other competing implementation. A limitation of the model is that the inter-map consistency, there is no guaranty that the different maps (albedo, normal, roughness etc.) will have pixel-wise coherence as the loss function is calculated separately for each one. Performing quality evaluation on the generated textures reveals that the results are more than satisfactory as the potential pixel inter-map inconsistencies are most of the time not noticeable by the human eye.

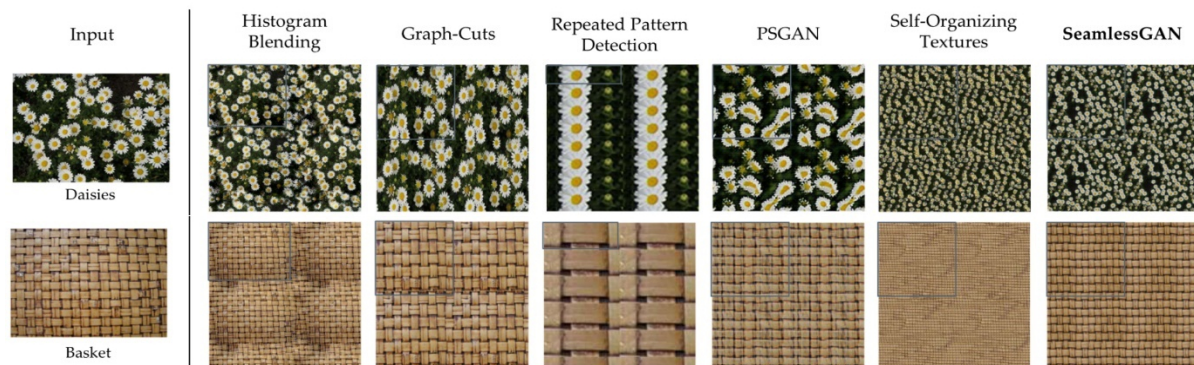


Figure 31: Comparison of SeamlessGAN with other technics. SeamlessGAN method generally captures better the overall structure of the texture, while providing seamless and semantically coherent borders, for enhancing tileability. Source:[61]

PBR material maps

As discussed previously, generating pixel-perfect coherent texture stacks is very difficult even if we account for inter-map generation loss there is no guaranty that it can be eliminated completely. Tom Moore implemented a model for predicting PBR maps by utilizing the pix2pix Conditional GAN [62]. The pix2pix method is dedicated for image translation and can achieve better results in converting an albedo texture to the respective PBR maps.

The method uses a simple albedo texture also referred as diffuse, or color map can be referenced and used to generate the different maps used in the PBR materials. For every texture map a separate model must be trained therefore, at least 4 models are needed to produce the basic maps normal, roughness(smoothness), height, ambient occlusion (AO).

- Albedo2normal: translates the albedo to the corresponding normal map.
- Albedo2smoothness: translates the albedo to the corresponding smoothness map.
- Albedo2height: translates the albedo to the corresponding height map.
- Height2AO: translates the height map to the corresponding ambient occlusion map.

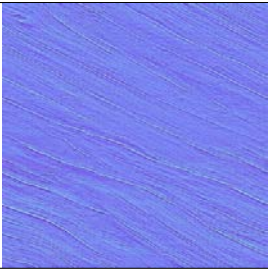
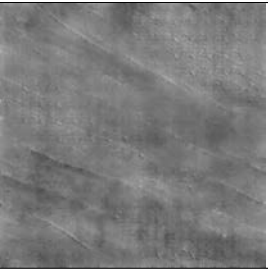
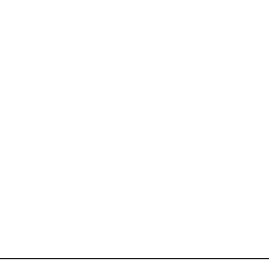
Albedo	Normal	Roughness	Height
			
Height	AO		
			

Table 4: Tom Moore's Pix2Pix model generated PBR maps. Source:[62]

Post-process effects

Post-process effects refer to rendering filters used in videogames in order to enhance the visual experience of the player. Motion blur and color correction are one of the essential uses of post processing effects, but developers also use them to create alternative artistic representations without the need to make changes to the game, it just gets rendered differently.

Style transferring is a very prominent domain of GANs and there is a plethora of methods proposed for image-to-image translation, but we will focus on a videogame-based approach from Mittermueller, Ye and Hlavacs.[63]

These three researchers from the university of Viena proposed a method to leverage the G-buffers used in game engines for rendering. "GBuffer is short for Geometry Buffer, which means a buffer that stores geometry information. More specifically, it refers to a group of render textures used to describe object positions, normals, colors, and other information needed for lighting." [64]

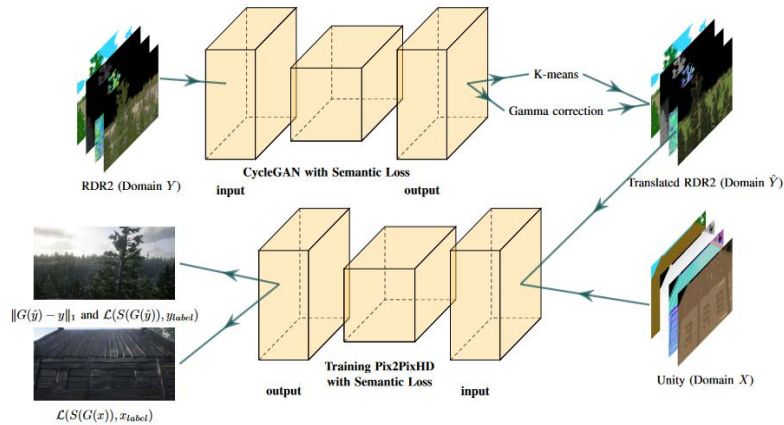


Figure 32: Overview of EST-GAN Architecture. Source:[63]

The model is called Enhanced Style Transfer GAN or EST-GAN for short. EST-GAN uses a combination of a CycleGAN and a Pix2PixHD cGAN to produce a game-to-game style transfer, Y is the game from which we want to extract the style and X the game to which we want to apply the style transfer.

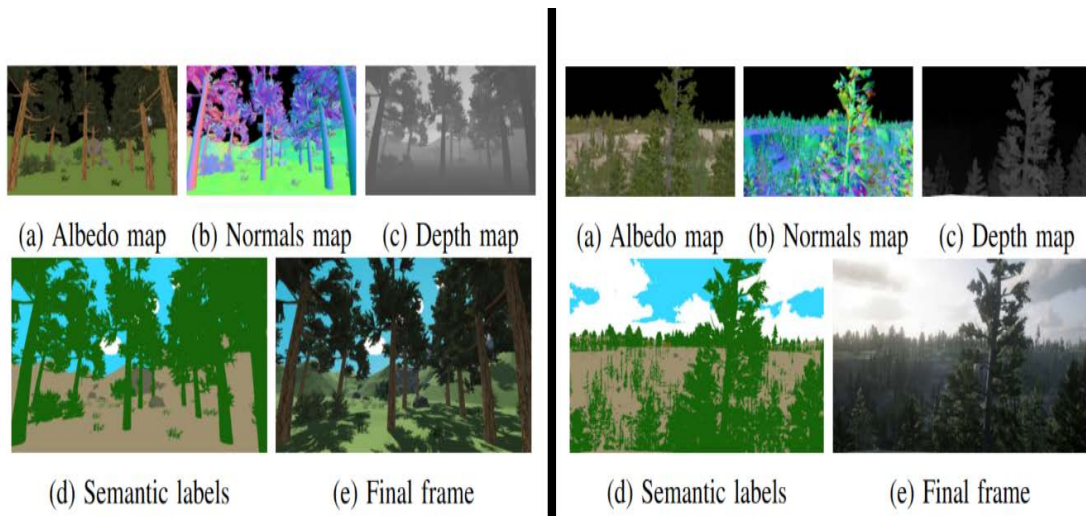


Figure 33: Example of Sampled G-Buffer Frames. Left from Unity Domain X. Right from RDR2 Domain Y. Source:[63]

The first step in the model used a CycleGAN to translate the G-Buffers of the domain Y to the domain X but they observed that the default cycle loss calculation method was producing undesirable results mainly resulting in the model hallucinating vegetation where it should not be, so a semantic loss method was used instead combined with mean absolute error to result in matching probability distribution over all labels. The new domain of translated G-buffers is called $\hat{Y}$.

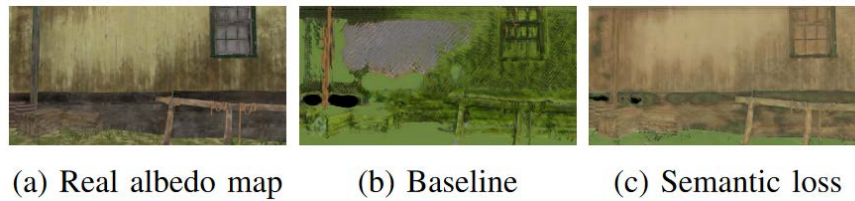


Figure 34: Comparison of a) Sampled Albedo b) CycleGAN generated Albedo c) CycleGAN+Semantic Loss Albedo. Source:[63]

The second step is using the Pix2PixHD cGAN to create a translation between the X and $\hat{Y}$. The inputs to the cGAN are the G-buffers from X as well as the translated Y G-buffers. In this step semantic loss was used as well as it was already implemented for the CycleGAN and produced very impressive results with minor artifacts on normal maps in cases where there were low polygon assets present

The third and last step was focused on removing artifacts produced by the pix2pix cGAN in the normal maps as well as in the albedo maps from the CycleGAN. The artifacts were a result of unsimilarity between the sampled G-Buffers, due to low polygon objects in Domain X , compared to Domain Y . A k-means clustering algorithm was used on the images to remove those artifacts and produce sharp edges desired in the final normal maps.

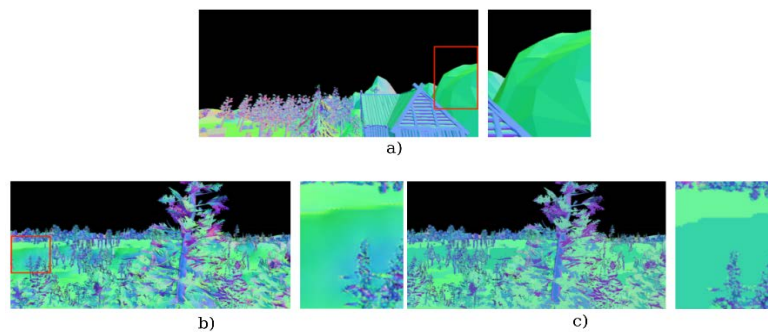


Figure 35: a) Domain X sharp normal edges b) Domain Y smoother edges c) Domain Y after Clustering. Source:[63]

The researchers created a completely synthetic dataset with the use of RenderDoc [65], which is a graphic debugger tool, used to capture single frames as well as additional information like the G-Buffers. The popular Red Dead Redemption game [66] was used for data collection and as a target game the researchers created a simple nature scene inside the Unity game engine to test the style transfer model.



Figure 36: Left Baseline Unity Frame. Right EST-GAN final model produced Frame. Source:[63]

3D Models

The generations of 3D models can be separated into two categories, general 3D objects and 3D terrains which are a special case of 3D geometry.

a. General 3D objects

In literature 3D objects are represented in three different ways: point clouds, meshes and voxel grids. Point clouds are sets of multidimensional data points that contain space coordinated but can also include additional information like color or velocity values. Meshes are the most common representations of 3D geometry that are collections of polygons that form the faces of the mesh in space. Each polygon consists of edges, vertices, and an orientation vector. Voxel grids are 3D grids organized in layers, rows, and columns. Voxel grids can be thought of as fixed-size point clouds, where each voxel has a fixed size and discrete coordinates.

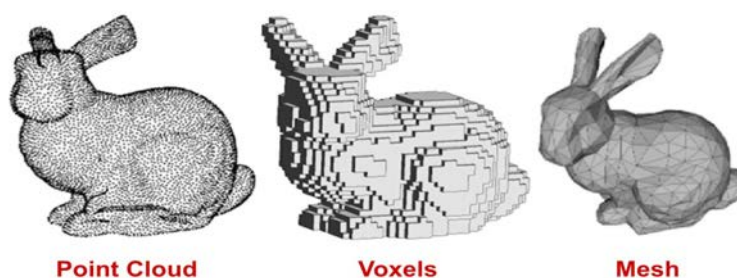


Figure 37: Different 3D representations of a rabbit. Source:[67]

Due to the complexity of using point clouds as input of GAN models only a few researchers try and tackle the problem, Point Cloud GAN or PC-GAN proposed by Chun-Liang Li et al. in 2018 [68] was capable of processing point clouds but despite the impressive results, the model did not take into account the shape of the object or the relationship between neighboring point therefore resulting in a suboptimal solution.

The use of point clouds as direct input to deep learning algorithms is still an under-researched topic. Most implementations of point clouds use the PointNet [69] that utilizes Multi-Layer Perceptron (MLPs) and as a result of not using convolution, no relationships of neighboring points can be found. Ben Hamu et al.[70] introduced a model in 2018 to take meshes as input and generate human body and tooth meshes, based on the progressive growing methodology used in the ProGAN(2017) model by Karas et al [71]. The results are promising but nowhere near realistic enough.

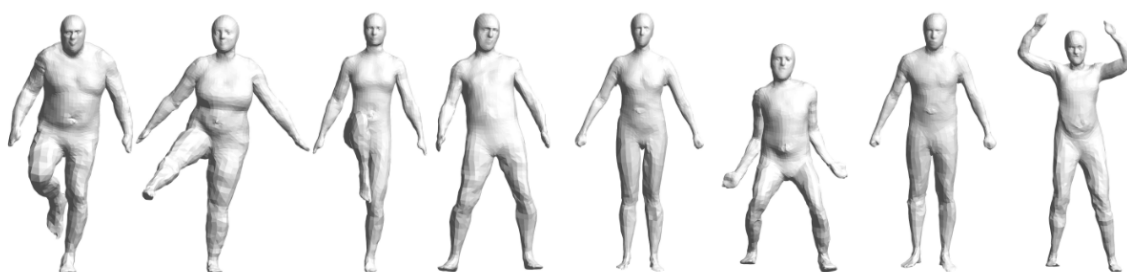


Figure 38: Ben Hamu et al model results. Source:[70]

These problems are leading researchers to convert irregular data into a regular format such as 3D voxel grids which can be used as input to the convolution architectures. Even though point clouds and meshed do not have equivalent voxel representation there are a lot of proposals for conversion to voxels. [72, 73, 74]

Wu Jiajun, et al. [75] demonstrated in their 2017 research that GANs can be extended to use voxel-based input for training a GAN model to generate high quality voxelized representations of 3d meshes. Some years later (2020) Ryan Spick et al. [76] proposed an extension of the DCGAN for a 4-channel voxel-based approach for mesh-to-mesh generation with the inclusion of colors. The important step in their methodology is the voxelization process of the meshes, they convert all the high resolution meshes from the initial data set to lower resolution voxel representations in order to use them as input to the GAN. The proposed model learns the shape as well as variety of characteristics of different texture patterns.

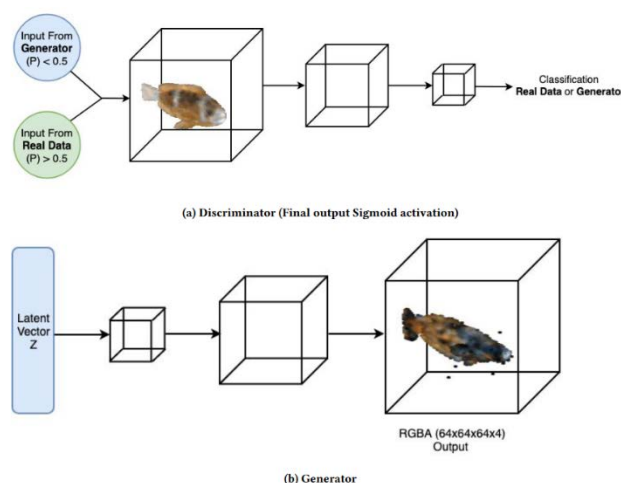


Figure 39: Spick et al. GAN model architecture. Source:[76]

During the data preprocessing to create the voxel representations, every mesh is placed inside a $64 \times 64 \times 64$ voxel grid, and a 3D kernel is run through the mesh to decide if a voxel should appear or not. The data could be visualized as RGBA with RGB channels representing the color of the voxel and the A channel was -1 or 1 depending on if the voxel should be visible or not in the given position.

The network used was based on the DCGAN model but extended the input by 1 channel, for a total of 3, for the purpose of including the depth. The network was able to down-sample the 64^3 to a final fully connected sigmoid output. The learning rate of the generator plays an important role in the variety of output produced, a high learning rate results in lower variation in respect to the training meshes.

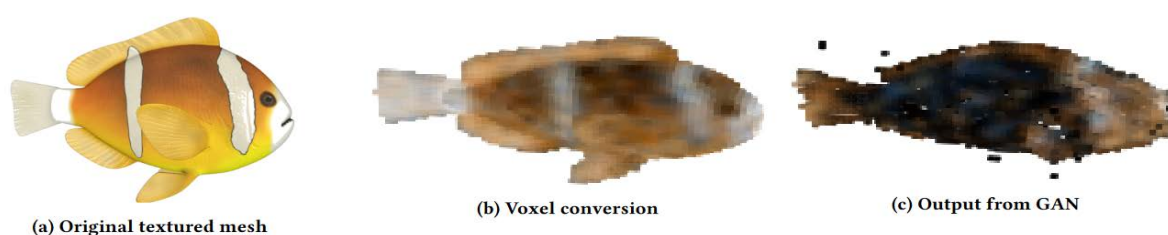


Figure 40: Spick et al. GAN model Results. Source:[76]

After the 3D DCGAN generates the voxel then the next step is converting that to a mesh as the initial proposition was mesh-to-mesh generation. The marching cubes algorithm was used to convert the voxel data to a mesh while preserving the colorization of the model by extrapolating the voxel RGB values to the corresponding faces.



Figure 41: Spick et al. GAN model Results after Marching Cubes. Source:[76]

Due to hardware limitations the results of their model were coarse, by limiting the voxel grid to 64^3 lowers the resolution of the final mesh data produced by the marching cubes algorithm. Acquiring hardware with more memory would solve the problem and result in higher quality meshes.

Another approach to produce voxelized representations of 3D objects is the inverse graphics approach, using 2D images to generate the desired 3D objects. Most proposals on the inverse graphics method require the use of a Differentiable Rendering. A Differentiable Renderer is different from a classic industry renderer in the fact that it can render as well as "De-render", that can be achieved by identifying and distinguishing the role each input played to result in the final render frame and thus the name Differentiable.

The problem of differentiability occurs during the rasterization that is an inherently non-differentiable process in which the renderer must convert the 3D object to an array of pixels to render on the frame. Only a handful of handmade Differentiable Renderers are created by the research community from scratch to overcome the rasterization step. The video game industry as well as most photorealistic industrial applications use non-differentiable renderers such as OpenGL, DirectX and Vulkan, due to their efficiency.

Sebastian Lunz et al. [77] proposed in 2020 an Inverse Graphics GAN (IG-GAN) model for generating 3D shapes from unstructured 2D image data, the difference between their approach was that it was aiming to be the first to utilize a non-differentiable off-the-shelf renderer.

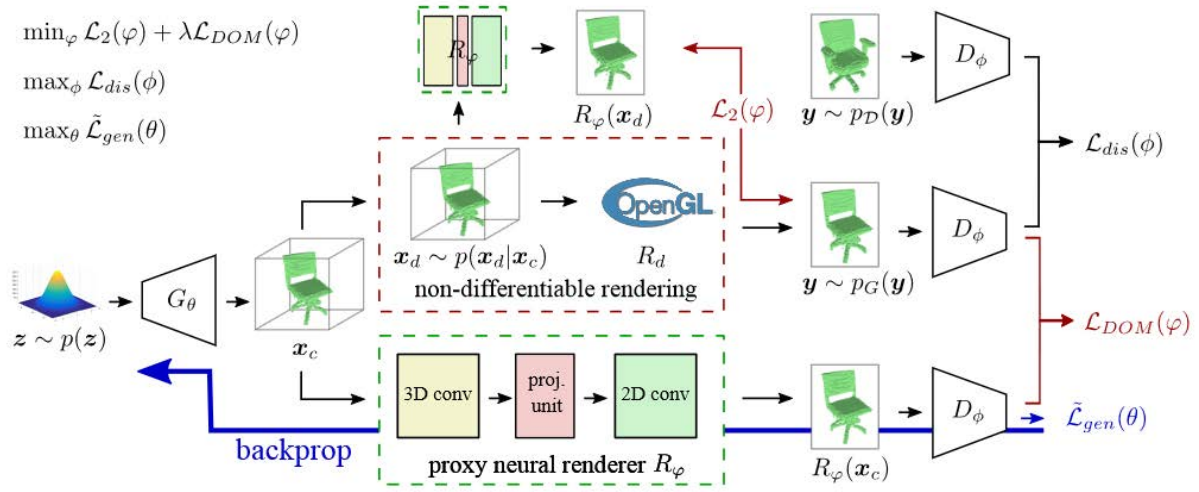


Figure 42: IG-GAN architecture. Source:[77]

The IG-GAN was trained exclusively on 2D data, but they opted to use voxel-based representation for the 3D object generations. Another problem in the non-differentiable requirement is that generated objects are represented as continuous voxel grid, sampling that grid to generate a mesh is not differentiable, but they can be made differentiable by representing the occupancy of each voxel by a real number $[0, 1]$ which identifies the probability of voxel occupancy.

To fool the discriminator, the 3D generative model must generate continuous voxel grids which correspond to meaningful representations of the underlying 3D shapes.

The researchers propose the use of an Intermediate differentiable neural renderer that will be utilized for backpropagation for the generator's training to match the output of the off-the-shelf renderer. The continuous voxel grid representation created by the generator is rendered as a 2D image by the proxy renderer, furthermore the neural renderer is trained to minimize the loss of the voxel discretization process of the continuous voxels.

As discussed previously the Marching Cubes algorithm can be used to convert the final discrete voxel grid representation but it is apparent that the output meshes of all the methods showcased are still very low resolution due to memory limitations, that is should not be a deterrent as the advancements in super resolution technics and especially 3D super resolution based on voxel upscaling is impressive. [78]

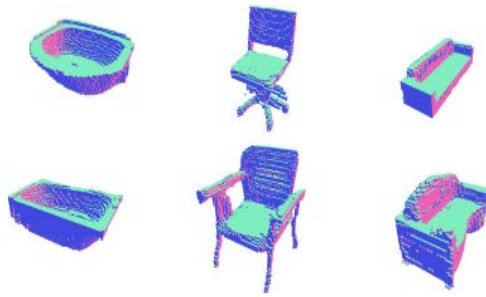


Figure 43: Normal maps of objects generated by IG-GAN using multiple images of the same object from different points of view. Source:[77]

In conclusion, let's show a complete table comparing all the discussed approaches in 3D model generation using Generative Adversarial Networks.

Model	Point cloud Based	Mesh-Based	Voxel-based	Textured	Image-to-3D	Shape Interpolation	Output
PC-GAN	Yes	No	No	No	No	Yes	Point-cloud
Hamu[70]	No	Yes	No	No	No	Yes	Mesh
3D-DCGAN	No	Yes (input)	Yes	Yes	No	No	Mesh
IG-GAN	No	No	Yes	No	Yes	No	Voxel

Table 5: Comparison of discussed 3D GAN models

b. 3D Terrains

A sub-category of 3D models, terrains are virtual representations of geographical terrains created by displacing a flat grid of vertices with a given set of data points.

The most common way for 3d terrain construction with the use of a height map also known as displacement map, that is a 2-dimensional raster image of values points which are used to displace a mesh to create 3-dimensional terrain in computer graphics.

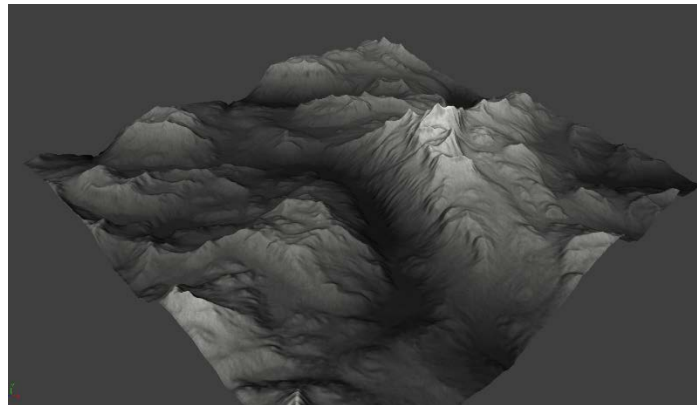


Figure 44: Terrain rendered from height-map using TerreSculptor.

In 2017 a team of researchers proposed an interactive GAN-based model for terrain authoring using user input in the form of a sketch image [79]. The basic concept is heavily relying on the data preprocessing, extracting the specific data from the training data set of terrains in order to be capable of matching the user input sketches to terrains and train a conditional GAN to generate a corresponding terrain.

They implemented 4 different types of models called synthesizers, "**S** creates a terrain from a sketch containing ridges, rivers, altitude cues or any combination of the three; the levelset-to-terrain synthesizer **L** turns a binary levelset image into a terrain; the eraser synthesizer **R** removes a user-specified part of the terrain and completes it, and, finally, the erosion synthesizer **E** transforms an input terrain into the corresponding eroded terrain." [79]

- In **S** the sketches are RGB images that have each channel representing a specific detail, **R** is mountain ridges, **G** is altitude cues and **B** is rivers.
- In **L** the sketches are black and white images representing the part of the terrain with altitude above a threshold (60%).
- In **R** the sketches are two channel images, one channel has the height map of a terrain and the second channel the parts to be erased, the values of first channel are set to 0 where the second channel is equal to 1.
- **E** does not require a sketch; it just runs an erosion simulation algorithm on top of a terrain.

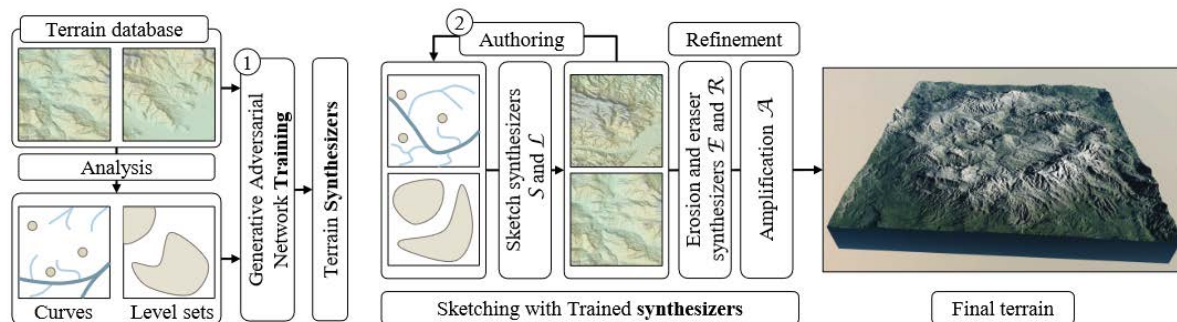


Figure 45: Eric Guérin et al. terrain model architecture. Source:[79]

Artists can use the tool to incrementally generate desired terrains by providing appropriate input sketches. The sketching process proved to be very intuitive in both novice and expert artists enabling them to create realistic terrains parameterized to their desire with minimal effort. This approach produces very desirable results, but it requires manual work by the users which may be problematic in many cases requiring vast amount of input images.

Ryan Spick and James Alfred Walker [80] introduced in 2019 a fully automated approach using the NASA's SRTM data set to train a GAN model on real world terrain data, satellite images and height maps combined, to be able to generate realistic variants only with a longitude and latitude coordinate input that corresponds to a desired region on the earth's surface. This method did not only generate variants of the satellite images but also the corresponding height map in order to generate a ready-to-render textured terrain.

The model works by cropping the terrain data image to 32 random parts in each iteration and training a SpatialGAN, which is similar to a Deep convolutional GAN but is non-spatially dependent. They observed that there was an imbalance between the generator and the discriminator, the discriminator was getting too good, and the generator could not keep up, therefore they opted to train the generator twice for every update of discriminator. Post-process to height map channel of the GAN output was necessary to remove noise from the training process, structural similarity was used to choose the median blur kernel size in order to minimize visual data loss.

A different approach was taken by Panagiotou and Charou [81], two Greek researchers who used a ProGAN to generate the satellite image variations and then a pix2pix conditional GAN to obtain the height data, they also chose to implement the Discriminator with a PatchGAN that works by classifying patches of the image and then it classifies it by averaging over all the patches.

The results of this framework are very impressive as the ProGAN approach is capable of generating variations of high resolution as well as imitating basic elements like riverbanks, islands with greener water near the surface and snow.

Super-resolution

It is not uncommon for very successful videogames to get re-released after many years, these cases are called "Remakes", and they are done manually by developer studios. This requires huge effort and expense from the developer's standpoint. AI automated approach to reproduce old game low resolution texture images in modern day aesthetically appealing ones would be very helpful.

Many implementations of GANs have been suggested over the years to deal with the single image super-resolution problem. A group of Chinese researchers proposed the ESRGAN[82] model, that found popularity as it was released pre-trained and with easy-to-use code. The model was based on the SRGAN[83] and added many improvements on top of the base model in order to minimize artifacts and generate representations closer to the original image. The model makes significant alterations to the neural network of SRGAN, specifically:

- The Batch normalization layers are removed and replaced with Residual-to-Residual Dense Block
- They reimplemented the Discriminator as a Relativistic average GAN which learns "whether one image is more realistic than the other" rather than "whether one image is real or fake."
- Lastly, they propose an improved version of the perceptual loss by using the VGG[84] features before activation instead of after activation.

The ESRGAN model produces very impressive results compared to its predecessor while being easier to train.



Figure 46: Left original image generated by PolycamAI 512x512. Right 4x upscaled image 2048x2048 using Real-ESRGAN (a free to use online implementation of the ESRGAN model).

(Section 2.2.c) Latent Diffusion Models

In previous chapters we discussed various ways to generate synthetic data, mainly images, with various degrees of success in terms of quality. VAEs though stable; they perform badly in terms of image resolution. GANs show promising results but they are unstable during training and suffer from "mode-collapse", also do not scale well in more complex multi-modal data distributions.

Researchers in the pursuit of producing higher resolution outputs to match the current demands have shown that Diffusion-based Models (DM) were the next step of generative AI. The

first attempt that successfully implemented a model with those requirements was the Jonathan Ho et al. "Denoising Diffusion Probabilistic Models" [85] which proposed a hierarchical denoising autoencoder architecture.

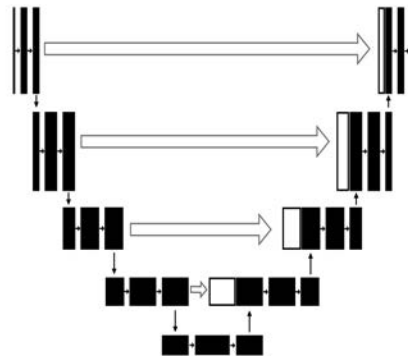


Figure 47: Simplified example of a Hierarchy of Denoising Autoencoders. Forward process starts from top left and Backward process ends at top right.

Not to be confused with variational autoencoders, denoising autoencoders purposely corrupt the input by adding noise of a certain distribution, usually Gaussian, to the data. Mapping the data to a lower dimensional manifold makes the attempt to de-noise more manageable. The corrupting step is called forward process and the de-noising step backward process as it tries to go back to the original non-corrupted data by reversing the mapping learned in the forward process.

In the DM approach, by stacking the denoising autoencoders, the data are repeatedly corrupted until they have no resemblance to the initial input, but they become almost entirely a noise distribution. The trained DM tries to recreate the data by consecutively passing the learned noise distribution through the backwards process. This results in a highly compressed mapping of input data and consequently on high resolution reconstructed variations of that data.

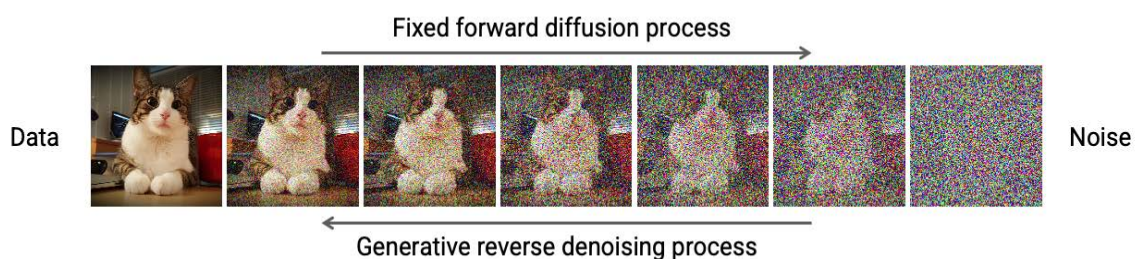


Figure 48: Diffusion process example. Source:[86]

DMs have been shown to produce very impressive results in image synthesis as well as audio generation. They "belong to the class of likelihood-based models, whose mode-covering behav-

ior makes them prone to spend excessive amounts of capacity (and thus compute resources on modeling imperceptible details of the data." [87]

Research comparing DM to GANs on image synthesis [88] concluded that for training the GPU days needed can range from 150 to 1000 V100 for DMs depending on the size of the dataset and the complexity of the model. Aiming to mitigate the immense hardware requirements, Rombach et al. [87] proposed a latent-based approach to diffusion models. Their method broke the process down to two steps, the first step was to train an autoencoder to provide a lower dimensional representation but perceptually equivalent space, the second step was to train the DMs to the learned latent space which provides better scaling in respect to spatial dimensionality. They named this approach as Latent Diffusion Model or LDM.

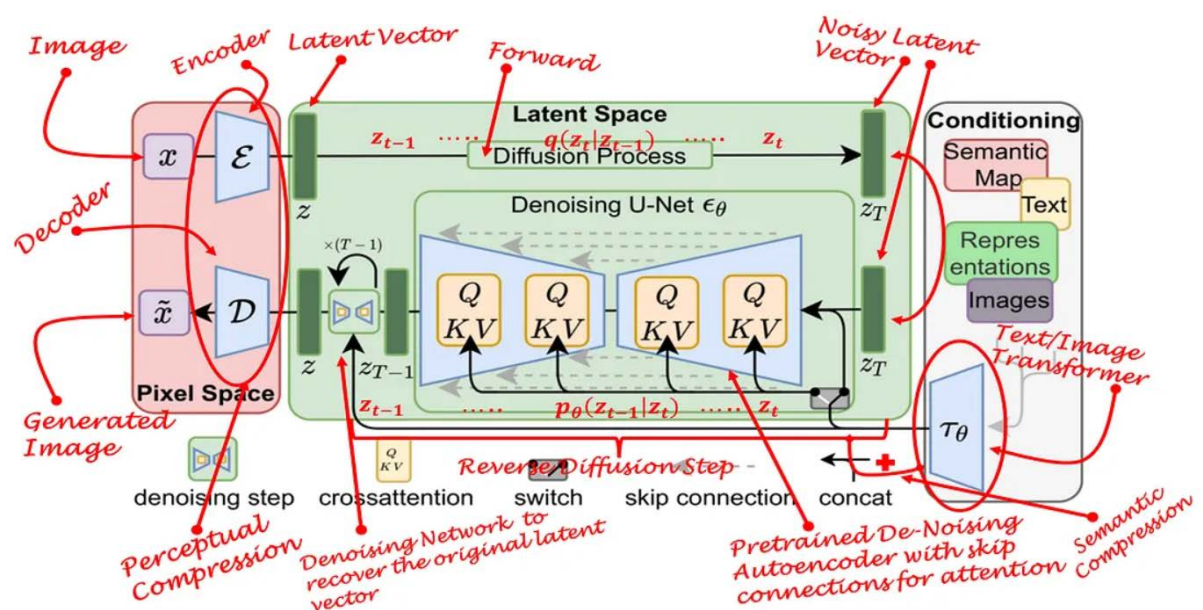


Figure 49: Latent Diffusion Model Architecture with notes. Source:[89]

It is important to mention that the first step of training the universal autoencoder can be done once and then multiple DMs can be explored for different tasks. Furthermore, transformers can be connected to the DM in order to add conditional generation such as text-to-image, layout-to-image, image-to-image and more.

Advantages of LDMs:

- They lower the computational costs significantly while achieving competitive performance on several tasks compared to pixel-based DMs.
- They work at a compression level that retains high levels of detail and can reproduce them very accurately.
- They can be used efficiently in higher resolutions in the scale of megapixels.
- Capable of densely conditioned generation on large images of 1024x1024 pixels

After the publication of the LDM model many different implementations hit the scientific community as well as the commercial space for synthetic image generation, some of the most well-known commercial models are DALLE2[90], Midjourney [91] and Stable diffusion[92].

DALLE2 was released on April 6, 2022, which was their second attempt at an image generative model after the first DALLE released on January 5, 2021. DALLE2 is an LDM based model compared to its autoregressive predecessor and boasted very impressive results in comparison.

Midjourney is a small independent research lab created by David Holtz. They developed Midjourney AI and released it July 12, 2022, which is their variation of an image generation LDM and is implemented on top of the popular Discord community platform [93].

Stable Diffusion was developed by a research group from the University of Munich along with researchers from the Runway AI, an American based company. The project received support from many non-profit organizations in terms of donations as well as all the training data used. An important contributor was Stability AI that offered computational resources to the project and the final model was released in August 2022 with the name Stable Diffusion.

Stable Diffusion is a latent diffusion model similar to the other two mentioned above but there is a very important distinction that makes it a special case, the code as well as the trained weights of the model are open source and free for public use. The model can be run natively on a user's computer with the requirement of a GPU with at least 4gb VRAM, that was an important feature due to all other competitors relying solely on online cloud services.

	DALLE2	Midjourney	Stable Diffusion
Text-to-image	Yes	Yes	Yes
Image-to-text	No	Yes	No
Image-to-Image	Yes	Yes	Yes
In-painting	Yes	Yes	Yes
Out-painting	Yes	Yes	Yes
Animate images	No	No	Yes
Image-to-Video	No	No	Yes
Text-to-Video	No	No	Yes
Image Blend	No	Yes	No
Seamless Image	No	Yes	No
Super Resolution	Yes (limited 1024x1024)	Yes	Yes

Table 6: List of most important features of the three discussed models. Stable diffusion is clearly more advanced in terms of features due to the open-source nature of the project.

3D OBJECTS

Latent diffusion models are not confined only in image synthesis even though they were developed with that in mind, they can be applied in other tasks that require high-resolution precise output such as audio synthesis and 3D shape generation.

As discussed in length on the previous section, GANs have been used successfully for 3D

shape generation but the results are novel and might possibly serve as prototypes in actual commercial cases, due to the lack of detail fidelity. LDMs come to the table bringing efficient big data computational capabilities making them the next step in high resolution and complexity applications such as 3D shape generation.

Compared to GAN models falling short in understanding high resolution point cloud representations LDMs can model highly complex distributions and don't suffer from mode collapse as they are likelihood-based probabilistic models and have been shown to produce great results in 3D object synthesis.

Luo and Hu [94], inspired by thermodynamics, proposed an LDM in 2021 capable of generating 3D shapes in point cloud representation. The model was based on a non-equilibrium thermodynamic system in contact with a heat bath, the point cloud was seen as particles inside the system that under the effect of the heat bath disperse and diffuse through the entire space. By simulating this diffusion process, the researchers converted the initial point cloud distributions to simple noise distributions by adding noise in each time step. Consequently, the point cloud generation would coincide with the reverse diffusion process.



Figure 50: 3D point cloud diffusion. Source:[94]

This reverse diffusion is modeled as a Markov chain with the objective of converting the initial distribution to a target distribution, thus the goal of the LDM is to learn the transition kernel so the chain can reconstruct the shape. The Markov chain alone is not capable of reconstructing multiple shapes, for that purpose a shape latent variable is introduced as a condition to the kernel. This shape latent is learned end-to-end in the autoencoder step via normalizing flows parameterization. The training objective of the model is to maximize the variational lower bound of the likelihood of the point cloud conditional on the shape latent.

The model shows very promising results in shape reconstruction as well as latent space interpolation between two different point clouds, notably when trained on the entire ShapeNet dataset [95] it was shown to outperform all other competition (AtlasNet [96], PointFlow [97], ShapeGF [98], suggesting LDMs have higher capacity to encode different shapes.

Zhou et al. [99] proposed a Point-Voxel hybrid LDM (PVD) capable not only of 3D shape generation but also multimodal shape completion, making a huge step forward in generated variations. The PVD model utilizes as its backbone the Point-Voxel CNN [100] that voxelizes the point cloud data for 3D convolution, aiming to exploit the strong spatial correlation inherent in point cloud data.

The shape generation part of the model is very similar to Luo and Hu that was discussed previously, the important addition and a very desirable feature is the multi-modal shape completion. Most implementations follow an encoder-decoder architecture for shape completion that offers a deterministic result and therefore only a single possible completed final shape. PVD on the

other hand adopts a probabilistic approach on shape completion and can generate many probable variations of a shape from an initial observation of a partial set of points as well as from a single-view depth-map. In the case where it is provided with a depth map, the model converts it to a partial point cloud representation. The partial is named z_0 and it remains unchanged throughout all the time steps, only the missing parts are diffused.

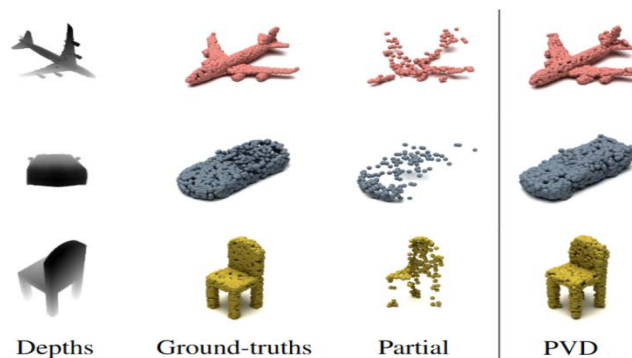


Figure 51: Partial point clouds from depth map and PVD completed shape. Source:[99]

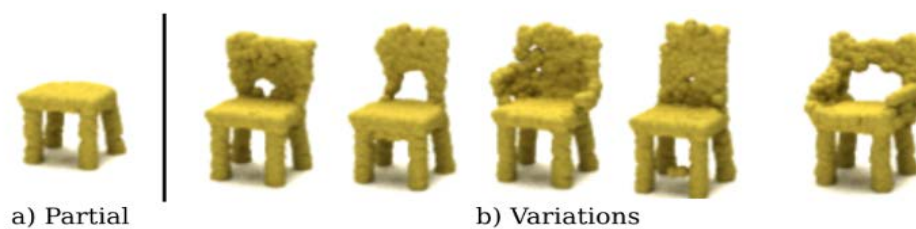


Figure 52: a) partial point cloud b) Possible variations. Source:[99]

Both the Zhou et al as well as Luo and Hu models show very promising results but they both suffer from the same problem. Point cloud and voxel representations are mostly useful for research purposes, the artistic community as well as the commercial industry prefer working with smooth high resolution mesh representations as they coincide with their preferred workflow.

A group of researchers in collaboration with Nvidia, University of Toronto and the Vector Institute proposed in 2022 [101] a framework aiming to fulfill multiple criteria and create a useful tool corresponding to the needs of digital artists. The model is called Latent Point Diffusion Models for 3D shape generation or LION for short and is structured as a variational autoencoder (VAE) with a hierarchical latent space that combines a vector-based global shape latent representation with a point-structured latent space, along with two denoising diffusion models (DDM) trained in those latent spaces.

The training process involves a two-stage process, in the first stage the model is trained as a regular VAE, trying to maximize the ELBO and in the second stage the encoder and decoder

networks of the VAE are frozen and two different DDMs are trained one on the shape latent space and another on the point cloud latent space. The implementation of the model's encoder, decoder and point cloud DDM is based on the Zhou et al. approach using the Point-Voxel CNN. The shape latent DDM uses a ResNet [102] which is a type of convolution network with fully connected layers.

One of the main objectives of LION is flexibility in real time conditional generation, the VAE approach enables the model to be adapted for various tasks without the need for retraining the DDMs. LION can be guided by multiple input methods:

- Low resolution voxel grid
- Single view RGB image
- Text

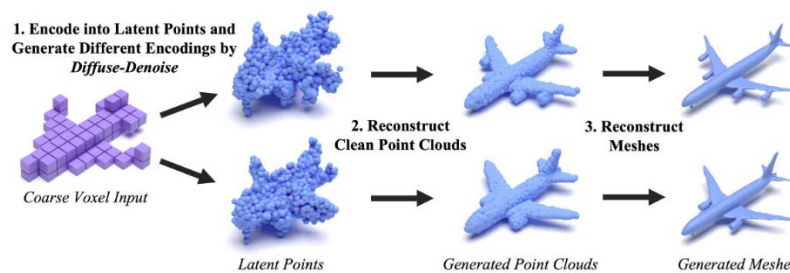


Figure 53: LION same voxel input resulting in different final generations. Source:[101]

Furthermore, the model is capable of multimodal shape generation providing a variety of generated outcomes for the same object as well as shape interpolation between two different objects.

The defining feature of LION is the implementation of a state-of-the-art surface reconstruction technic in order to provide the end user with a desirable mesh representation instead of a voxel or a point cloud. The method used is called "Shape as Points" (SAP) [103] and is based on differentiable Poisson surface reconstruction that can be trained on noisy point cloud data. The SAP is topology agnostic and is further fine-tuned on point clouds generated by LION by encoding "clean" shapes and denoise-diffuse them only a few times so that no details are lost but the noise generated resembles that of a LION generated shape.

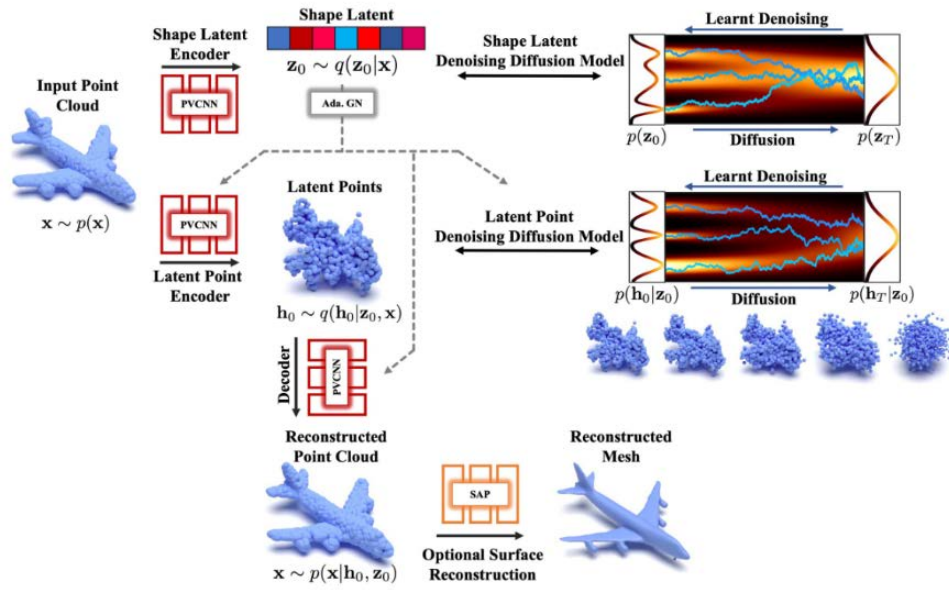


Figure 54: LION Architecture. Source:[101]

Model	RGB-D Input	RGB Image Input	Voxel Input	Text Input	Partial Completion	Shape Interpolation	Mesh Output	Point Cloud Output
PVD	Yes	No	No	No	Yes	Yes	No	Yes
LION	No	Yes	Yes	Yes	No	Yes	Yes	Yes

Table 7: Comparison of PVD and LION models.

AUDIO

There are two main ways of seeing audio in digital format, in waveform and in Musical Instrument Digital Interface (MIDI). The waveform is a graph representation of an analog audio signal as a function over time, "where the magnitude of the signal is sampled regularly at uniform intervals, then quantized to a series of symbols in a digital (usually binary) code"[104]. The MIDI format is a protocol developed for musical tool communication, it does not make sound, it is a symbolic representation of musical notes and different pitches that can be interpreted by a corresponding MIDI instrument or software. [105]

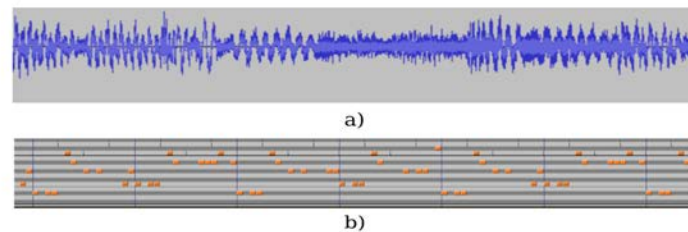


Figure 55: Example of a) Waveform b) MIDI representation.

The research community has proposed methods of generating audio using both formats as inputs for training latent diffusion models with various degrees of success. Mittal et al. [106] made a proposal in 2021 for a symbolic music generation model trying to tackle the problem using MIDI audio samples as they offer many advantages over traditional waveform.

Their model utilized a VAE to encode the discrete MIDI input to a continuous latent space where a diffusion model will be trained on, more specifically they opted to use the pretrained 2-bar MusicVAE [107] which uses bidirectional recurrent neural networks for the encoder and an autoregressive decoder. The model was trained with the Lakh MIDI Dataset (LMD) [108] containing over 170,000 MIDI files, each file was comprised of 1024 discrete tokens that were divided to 32 parts and fed to the MusicVAE encoder in order to produce 32 continuous latent variables that will be fed to the diffusion model.

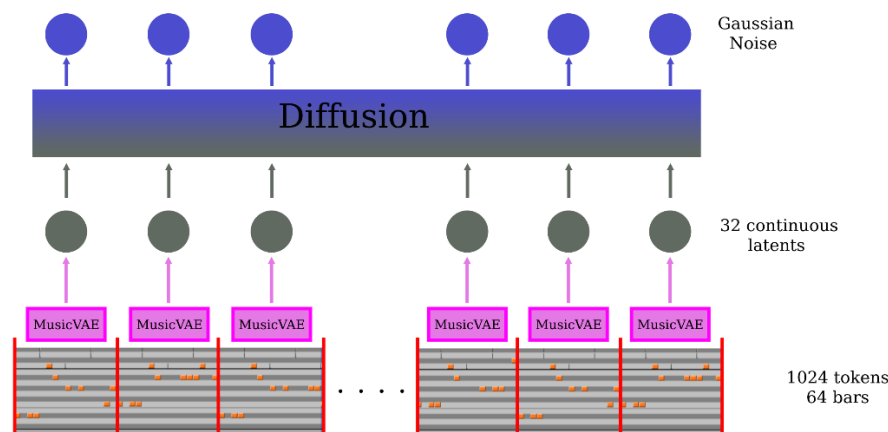


Figure 56: MIDI latent diffusion model Architecture.

This multi-stage model is capable of unconditional MIDI music generation as well as conditional infilling. To demonstrate conditional infilling of latent embedding in an unconditionally trained diffusion model the researchers remove from the total 32 sections the 16 middle ones, leaving only 8 on the start and 8 on the end of the MIDI sample for the purposes of generating the missing sections by conditioning the model on those remaining 16.

Despite the promising results in MIDI audio synthesis demonstrated by Mittal et al., there are no practical applications apart from the infilling which is still limited with no way of guided generation, due to the unconditional nature of the model. Most recent audio generation approaches focus on text guided generation but have moved away from the symbolic MIDI format and focus on the classic waveform representation.

Research in previous years focused on labeled audio data, but the quantity of those data was limited to a small set of labels like the 10 different sound classes in the UrbanSound8K [109] dataset, resulting in limited flexibility and quality of the generated results along with high computational costs. AudioLDM [110] proposed by a team of researchers and showcased in the Hawaii International Conference on Machine Learning in 2023, focusing on natural language text descriptions of audio data in order to guide the diffusion model.

The AudioLDM model uses a mel-spectrogram-based VAE along with contrastive language-audio pretraining (CLAP)[111] embeddings for latent variable generation. By leveraging the audio-text-aligned embedding space in CLAP, paired audio-text data are not required during training of the LDM. Along with audio generation the model can achieve text-guided audio style transfer, super-resolution, and inpainting, without fine-tuning.

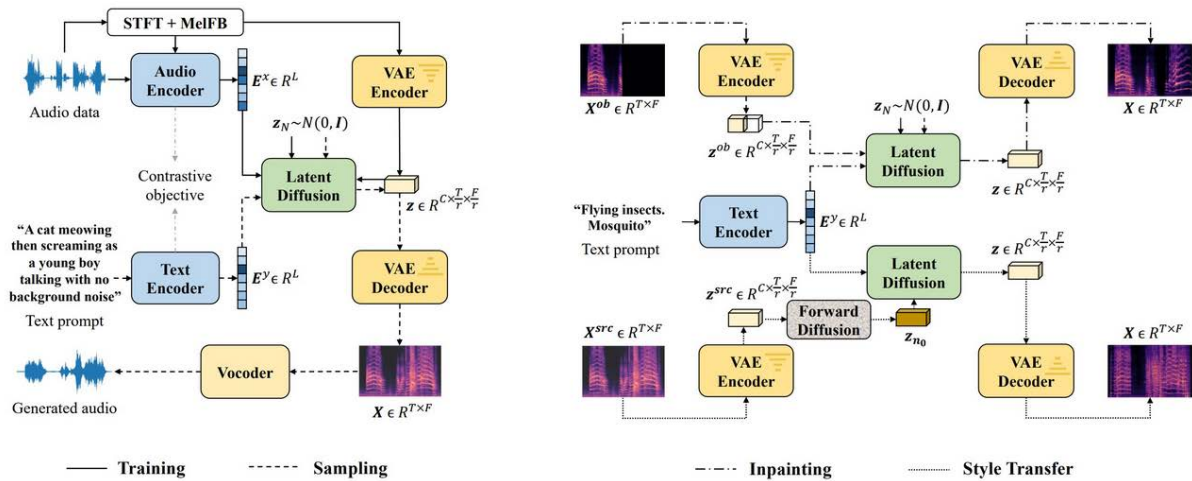


Figure 57: AudioLDM Architecture. Source:[110]

The audio encoder is based on HTS-AT[112] which is a hierarchical token-semantic audio transformer used for sound classification and detection and the text encoder is based on RoBERTa [113], an improved version of the BERT model used for language pretraining. Text-To-Audio (TTA) requires generation of compositional audio that is consistent with the text descriptions provided in natural language, but the datasets are not large enough to achieve desirable results. AudioLDM uses the AudioGEN[114] method which mixes pairs of audio samples and concatenated the textual context for the purpose of creating new paired data, thus increasing the size of the dataset significantly. As mentions previously AudioLDM uses audio only conditioning for the LDM and as a result increasing the audio samples with this mix-up process results increased robustness to CLAP embeddings and at the same time removing the need for language-audio paired data.

The decoder of the model takes the generated compressed latents and transforms them to a mel-spectrogram representation which is then used as input to a vocoder. This vocoder is responsible for transforming the mel-spectrogram generated by the LDM to a waveform representation, it is implemented using the Hifi-GAN [115] trained on the AudioSet [116] dataset. The GAN has two sets of discriminators, a multi-period discriminator, and a multi-scale discriminator and is widely used for speech waveform generation due to the enhanced perceptual and generated waveform quality.

Experiments done with AudioLDM show great flexibility and versatility in a large number of cases as well as comprehensive understanding natural language textual input for guided generation of synthetic audio.

List of AudioLDM capabilities:

- Text-to-Audio Generation
- Audio Style Transfer
- Audio Super-Resolution
- Audio Inpainting
- Environment Control (e.g. huge room, studio etc.)
- Music Generation
- Music Control (e.g. instruments)
- Pitch Control
- Material Control (e.g. “hit wood”, “hit metal”)
- Temporal Order Control

Although impressive, AudioLDM has limitations in the audio sample quality and length, all the audio used in training the model is resampled to 16khz mono format and length of 10 seconds. Another problem of the model is that it is not fine-tuned end-to-end, all the subsystems are trained independently which may result in module misalignment. For example, the VAE may not learn an optimal latent space representation for training the LDM. Moreover, there is no filter for sensitive language context and abusing the model’s capabilities can result in creation of harmful audio content or misleading information.

With the recent revolution of large language models (LLM), zero- or few-shot performance in natural language processing has increased significantly. Inspired by the LLMs and the AudioLDM model Ghosal et al. [117] from the University of Singapore proposed a large language based TTA model named TANGO, improving upon the existing AudioLDM approach in two aspects. The first improvement is the removal of the CLAP-based text-encoder and replacement with the FLAN-T5[118] which is initialized with a T5 checkpoint and fine-tuned on instructions and chain-of-thought reasoning. The second improvement came in the audio data augmentation with the mix-up technique, instead of random audio pair mixing the TANGO model inspired by Tokozume et al.[119], considered a pressure-level consideration in the mixing process in order to avoid high-pressure loud audio to be mixed with low-pressure quitter samples which would result in the latter being overwhelmed and get drowned. Moreover, the energy of a sound wave which is proportional to the square of the amplitude is also considered for the mixing step.

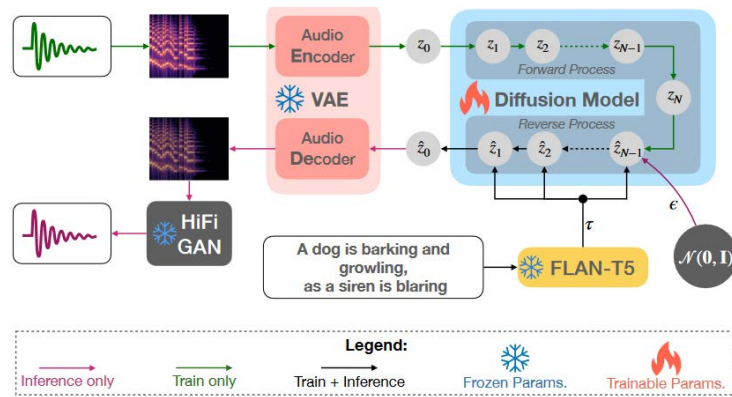


Figure 58: TANGO Architecture. Source:[117]

The LDM used in TANGO is trained in paired text-audio data from the AudioCaps[120] dataset instead of using exclusively audio data like AudioLDM. The dataset is 45,438 audio clips with length of 10 seconds each, paired with 5 different human-written captions. Tango achieves better or comparable performance in metrics with a dataset of samples 63 times smaller than that needed to train AudioLDM.

Nevertheless, regardless of all the improvements in terms of data efficiency and a more comprehensive LLM offering more nuanced natural language processing, TANGO still has some inherent limitations, firstly the size of the dataset limits the model ability to correlate text input to audio in a highly precise degree, slightly different prompts can result in very similar generated audio output. As demonstrated by the researchers "Chopping tomatoes on a wooden table" and "Chopping potatoes on a metal table" are very similar and due to limited size of the training data TANGO does not have the ability to differentiate between materials like the AudioLDM, which could be solved by training the model in larger datasets. Secondly TANGO follows in the steps of AudioLDM in the HiFi-GAN vocoder module, therefore the quality of the output is limited to 16khz which might be enough for novel uses but is low quality for digital audio professional applications.

The next improved implementation of AudioLDM called AudioLDM2[121] comes to introduce a "language of audio" (LOA), which is a sequence of vectors that represent the semantic information of an audio clip. This approach allows a universal representation of audio, which enables large-scale self-supervised pretraining of the core latent diffusion model without audio annotation and helps to combine the advantages of both the autoregressive and the latent diffusion model.

AudioLDM2 utilized the CLAP audio encoder like its predecessor, combined with 3 other modules, FLAN-T5 to capture additional temporal data, a Phoneme Encoder for encoding textual input into phonemes which is a widely used module in text-to-speech applications and ImageBind which is similar to CLAP but with more embedded modalities in order to condition the model in image, video, depth map, thermal images and inertia measurement units. All this conditioning information is translated to LOA using GPT-2 initialized with pretrained weight and fine-tuned on the conditioning information and ground truth data.

Instead of using a hand-build VAE, this time they opted for AudioMAE which is an extension of the image-based masked autoencoder architecture but conditioned to learn audio spectrogram

representations from unlabeled audio data without relying on manually labeled annotation. First the audio is converted to a mel-spectrogram and then it is encoded to patches with a high masking ratio, feeding only the non-masked tokens through encoder layers. The decoder then re-orders and decodes the encoded context padded with mask tokens, in order to reconstruct the input spectrogram.

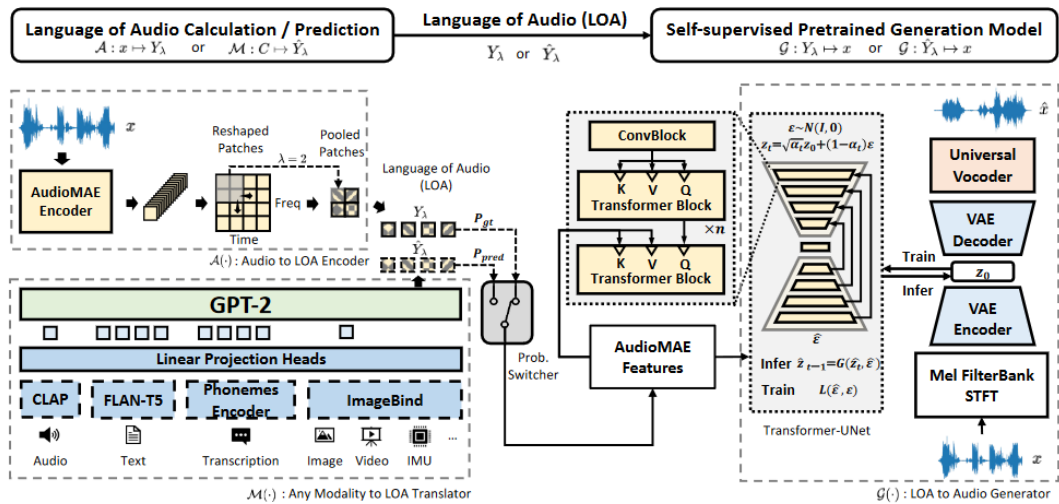


Figure 59: AudioLDM2 Architecture. Source:[121]

The LDM part of the model does not use the AudioMAE[122] latent representation because it is not focused on reconstruction quality and has a low compression rate. On the other hand, the VAE architecture shows better spectrogram reconstruction ability and higher compression rate making it better for training the diffusion model.

The AudioLDM2 model shows state of the art results in text to audio, text to music and text to speech applications, although it is trained on resampled 16khz audio in order to compare with previous models, it is capable of generating 48khz mono audio.

Generating audio of high quality is very computationally demanding, that's why the models we discussed so far focus on 16khz mono samples. Also, another problem is the length of audio samples used to train the model, the input samples will determine the final generated audio length, training on 10 seconds samples will result in generating 10 seconds audio clips. Another approach to TTA generation is proposed by Evans et al.[123] moving away from LLM for text input and focusing on quality and length of the generated audio output. They proposed a model named Stable Audio, capable of generating 44.1 kHz long-form variable length stereo audio with maximum of 95 seconds which revolutionized the TTA capabilities.

Following the specification of commercial standard audio, 44,1khz stereo, Stable Audio introduces TTA using timing embeddings for the purpose of generating audio of variable length depending on the desired time input. It uses two timing variables "seconds_start" and "seconds_total", during training chunks of 95 seconds from longer clips can be used by correctly aligning the 95 seconds window using the seconds_start. In practice, the model always generates 95 seconds of audio, it uses the seconds_start variable set to 0 and the seconds_total set to whatever second the user chooses, then pads the rest with silence and at the end cuts the clip to

the desired `seconds_total`, achieving variable length generation.

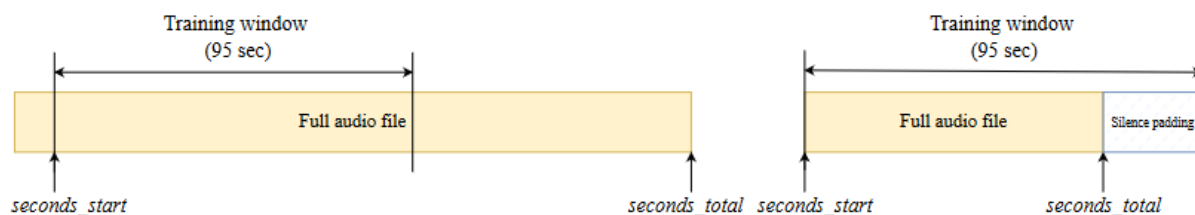


Figure 60: Stable Audio timing variables. Source:[123]

Stable audio uses CLAP embeddings for textual conditioning, a VAE model based on Descript Audio Codec[124] but removing the quantizer module and utilizing Snake activations for improved audio reconstruction and an LDM implementation focused on high memory-efficiency attention layers in order to scale more efficiently with longer sequences.

Regarding training data, the model is using the AudioSparx [125] provider with corresponding metadata for each sample. For prompt preparation the model used random concatenation of those metadata which include natural language descriptions as well as audio specific attributes such as genre, mood, tempo Beats Per Minute and instruments used per sample.

Stable audio is able to generate state of the art audio with the relatively small dataset, and is capable of generating high quality stereo music, sound effects and specific instruments with considerably less inference time compared to it equivalent competitor AudioLDM2 model.

Model	Channels/kHz	Audio Length (sec)	Inference time (sec)
AudioLDM2-large	1/16	95	37
AudioLDM2-48kHz	1/48	95	242
Stable Audio	2/44.1	95	8

Table 8: Quantitative results on MusicCaps.

Model	Channels/kHz	Audio Length (sec)	Inference time (sec)
AudioLDM2-large	1/16	10	14
AudioLDM2-48kHz	1/48	10	107
Stable Audio	2/44.1	95	8

Table 9: Quantitative results on AudioCaps.

In conclusion, let's show a comparison between all the audio generative models utilizing latent diffusion.

Model	Format	Channels	kHz	Length (original model)	CLAP	LLM	Dataset size (original)
Mittal[106]	MIDI	-	44.1	Variable (tempo based)	No	No	1 million
AudioLDM	Waveform	1(mono)	16	10 sec	Yes	No	3.3 million
AudioLDM2	Waveform	1(mono)	16	10sec	Yes	Yes	3.3 million
TANGO	Waveform	1(mono)	16	10sec	No	Yes	0.046 million
Stable Audio	Waveform	2(stereo)	44.1	Variable (max 95 sec)	Yes	No	0.8 million

Table 10: Comparison of the discussed audio generative models.

(Sub-chapter 2.3) Chapter Conclusions

The second chapter of the thesis was focused on showcasing the existing scientific literature on the field of artificial intelligence in a game development perspective by discussing methodologies that assist in some part of the game making process.

Firstly, the focus was on different ways behavior can be implemented inside a video game world in order to create realistic representation of human interactions. FSMs, Behavior Trees and Neural Networks serve as a core technology that is expanded upon in order to optimize it for use inside video games.

Secondly, the new field of generative AI is analyzed in depth as it can be used as a tool to assist the developers in many parts of the process. VAE, GAN, and LDM technology is discussed along many proposed models by the scientific community that were specifically researched for their game development applications or were proposed independently but their results could potentially be proven useful for video game makers.

From NPC behavior to music and prop generation, modern AI is getting very sophisticated and can provide very impressive results that would otherwise require time, money and specialized experts.

Chapter 3 Modern Video Game Development and Analysis of a Video Game's Components

(Sub-chapter 3.1) Definition of a Video Game

There is not a general consensus on what a video game is defined as. The word "videogame" is used as an umbrella term to encompass a vast variety of types of games, from puzzle, to board, to sport simulation, to fighting and many more. James Newman in his book Videogames [126], states that "scholars identify a range of social, cultural, economic, political and technological factors that suggest the need for a (re)consideration of videogames by students of media, culture and technology".

An attempt for a definition by Nicolas Esposito[127] is as follows: "A videogame is a game which we play thanks to an audiovisual apparatus and which can be based on a story".

Appears simple at first glance but with a deeper look at the wording, becomes evident that this definition is dependent on the definitions of each individual element.

What is a "game". What is "play". What is an "audiovisual apparatus"?

"Game" Frasca[128] in his thesis said "Even if it sounds obvious, videogames are, before anything else, games" trying to convey the connection between videogames and traditional games despite the change in playing medium.

In the words of Eric Zimmermann [129], one of the leading figures in the videogame world, "A game is a voluntary interactive activity, in which one or more players follow rules that constrain their behavior, enacting an artificial conflict that ends in a quantifiable outcome."

Certainly, a very comprehensive definition but still some types of games cannot be classified. It can be argued that some games do not have a quantifiable outcome and some do not enact an artificial conflict.

Roger Caillois in his approach[130], identifies two orientations "paida" and "ludus", translated as "play" and "game" respectively. Games without a quantifiable outcome who are not bound by strict rules are categorized as play or toy-games.

Puzzle games are still an exception as they have a quantifiable outcome, the solution of the puzzle, but they don't enact artificial conflict.

"PLAY" Play in the realm of video games refers to the term "gameplay". "The gameplay is the component of the computer games that is found in no other art form: interactivity. A game's gameplay is the degree and nature of the interactivity that the game includes" [131]. A person "plays" a game while interacting in a virtual world that may or may not have artificial rules, to govern the player-game world relationship and receives pleasure for doing so.

A video game can offer players competition, accomplishment, system mastering, narrative enjoyment and audiovisual experience. [132]

"Audiovisual Apparatus" The means in which the game is played constitutes the "audiovisual apparatus", the electronic computing system that is responsible for "running" the game as well as the input/output devices. From arcade cabinets, personal computers, dedicated gaming consoles, handheld systems and smartphones are examples of different platforms videogames can be experienced upon.

The way a player interacts with the game affects the experience greatly. Video games are regarded as an intricate interactive art form, each individual element has importance and every modification, no matter how minute, can alter the feeling of the game. Consider playing a shooter game with keyboard and mouse, the feeling is vastly different from playing the same game with a console controller.

(Sub-chapter 3.2) Breakdown of Video Game Development

(Section 3.2.a) Evolution of the Game Development Industry

In their early days videogames were "hardware bound" to one machine

- OXO (1952) [133]
- Tennis for two (1958) [133]
- Space Wars! (1961) [133]

Later became "system bound" as multiple companies released their own proprietary systems(consoles) and made games exclusively for those.

- Nintendo's Entertainment System(1983)
- Commodore's Amiga(1985)
- Sega Genesis/Mega Drive (1988)
- Nintendo's Game Boy(1989)

Periods of heavy competition among console manufacturers known as "Console Wars"[134] were prevalent in 80s and 90s.

- Sega vs Nintendo
- Sony vs Microsoft

As the years passed, three companies emerged as winners in the console industry Microsoft with XBOX, Sony with the PlayStation and Nintendo, who gave up on the home systems and focused on Handheld gaming, with various innovative devices.

Personal Computers-PCs were and will always be one step ahead of the competition when it comes to video games, easy upgradability and customization offers the user a personalized experience but comes with higher financial cost. In recent years a niche field of gaming using Virtual Reality headsets and motion controls, has become more popular with Meta's Oculus Quest being the first affordable solution available in the market.

(Section 3.2.b) Roles of a Development Team

The development process can be broken down into several parts and roles can be assigned depending on the task and corresponding responsibilities.

1. The Designer

The game designer is responsible for coming up with new ideas and creating the world and rules of the game. As the scope of the game gets bigger the role of designer can be divided into sub-roles.

- The Creative Director is responsible for maintaining the broad vision of the game by looking at the big picture.
- Level designer creates the world and populates it with props, enemies, rewards etc.
- System Designer is responsible for the systems (economy, combat, skills) the game must implement and how they will interact with each other. Depending on the number and complexity of those systems, multiple system designers may be needed.

2. The Artist

Creating virtual objects that will come together to shape the game world is the job of the artist. Creating either realistic or stylized art, in 3D or 2D depending on the designers' vision.

3. The Sound Engineer

A game without sound is half the experience. The sound engineer creates the sound that the game will have when the player interacts with it, like footsteps, doors closing, guns firing etc.

4. The Composer

Tasked with creating music that will accompany the player on the entire playthrough. The music must capture and enhance the feeling of the game every step of the way.

5. The Producer

A managerial role taking care of the necessary paperwork for the project. He is responsible for hiring developers, assigning teams, scheduling and budget management.

6. The Writer

The writer is not a full-time job, acting more as a supporting role, he is needed for the later stages of development to rewrite the script with clear and better character dialogs.

7. The Tester

Testing the game is one of the most important jobs of the development process. Testers are constantly playing the game during its development for the purpose of providing valuable feedback about the game-feel to the development team. Tasked to find bugs, glitches and how to reproduce them in order to assist in finding their cause, game testers are essential in the creation of a quality game that will be shipped to the players without problems.

The following table shows the core development team behind the popular title of Bethesda Game Studios called The elder Scrolls V:Skyrim(released 2011)[135], which is an open world, role playing game focused on a medieval fantasy setting with swords,magic and dragons.

Game Design	Bethesda Game Studios
Game Director	Todd Howard
Lead Programmer	Guy Carver
Lead Artist	Matthew Carofano
Lead Designer	Bruce Nesmith
Co-Lead Designer	Kurt Kuhlmann
Production Director	Ashley Cheng
Lead Producer	Craig Lafferty
Audio Director	Mark Lampert
Senior Producers	Jeff Gardiner Tim Lamb
Producers	Angela Browder Kevin Kauffman Nathan X. Mcdyer Phil Nelson
Lead Systems Programmer	Brett Douville

Systems Programming	Ryan Ashford Shannon Bailey Joel Dinolt Mike Dulany Jason Hammett Mat Krohn Ryan Lea Mike Lipari Jeff Lundin Christopher Rodriguez Jean Simonet
Lead Graphics Programmer	Scott Franke
Graphics Programming	Jonathan Bilodeau Paul Graber Jeff Sheiman Orin Tresnjak
Gameplay Programming	David Diangelo Joseph Diangelo Steve Meister Dan Teitel Craig Walton
Lead AI Programmer	Ahn Hopgood
AI Programming	Brian Robb Jay Woodward
Lead Interface Programmer	Erik Deitrick
Interface Programming	Ricardo Gonzalez
Additional Programming	Markus Alind Magnus Auvinen Ken Cockerham Chris Doyle Chris Esko Jim Kjellin Andrew Meggs
Concept Art	Adam Adamowicz Ray Lederer

World Art	Cory Edwards Tony Greco Daniel T. Lee Nate Purkeypile Rashad Redic Ryan Sears Clara Struthers Rafael Vargas Robert Wisnewski
Lead Environment Art	Noah Berry
Environment Art	Andy Barron Ryan Salvatore Megan Sawyer
Additional Environment Art	Gabrielle Adams Alex Schwartz-Rudd Christopher Zdana
Special Effects	Grant Struthers Mark Teare
Additional Effects	Liz Rapp
Lead Animator	Josh Jones
Animation	Jeremy Bryant Jangjoon Cha Lianne Cruz Gary Noonan Juan Sanchez Ricardo Vicens Alex Utting
Additional Animation	Ray Arnett
Lead Character Artist	Christiane H. K. Meister
Character Art	Ben Carnow Charles Kim Jonah Lobe Dennis Mejillones Dane Olds Yan Qin
Additional Character Art	Liz Beetem Hui Lai Chong Lucas Hardi Massive Black
Interface Art	Natalia Smirnova
Additional Graphic Design	Istvan Pely
Senior Designer & Writer	Emil Pagliarulo

Quest Design & Writing	Brian Chapin Jon Paul Duvall Shane Liesegang Alan Nanes William Shen
Additional Quest Design & Writing	Matt Daniels Nate Ellis
Lead Level Designer	Jeff Browne
Level Design	Daryl Brigner Joel Burgess Steve Cornett Ryan Jenkins Andrew Langlois Justin Schram
Additional Audio	Dave Schreiber
Production Assistant	Emily Sears

Table 11: Bethesda Studio's TES V:Skyrim Development team(Excluding Voice actors, musicians and production staff).Source:[136]

(Sub-chapter 3.3) Components of a Video Game

(Section 3.3.a) Theoretical Components

Developing a game is a very labor-intensive endeavor and it is comprised of various components that come together to create the final product. Here a thorough break down of the building blocks of every game will be presented.

The idea

In the process of brainstorming ideas come and go in an instant and most of them are discarded. It is very difficult but important, to successfully choose the correct idea that will shape the game.

Initial ideas most often than not do not sound very promising, sometimes even sound bizarre, but can lead to a very interesting and fun game.

- Donkey Kong

"A gorilla throws barrels down the stairs and a man tries to climb up to stop him".

- Super Mario

"A plumber fights mushrooms and turtles sent by a dinosaur who kidnaped his girlfriend".

- PacMan

"A yellow circle eating dots is hunted by ghosts inside a maze".

- Toki

"An evil wizard steals a man's girlfriend and transforms him to a monkey that can spit fireballs".

The idea will shape the story as well as the core gameplay mechanics of the game and is the job of a good designer is to come up with an interesting idea that will be shaped into an interactive experience for the players.

A comprehensive list of skills needed to design a game is given by Jesse Schell, in his 2008 book "The Art of Game Design A Book of Lenses"[137]

- | | | |
|------------------|--------------------|---------------------|
| • Animation | • Creative Writing | • Psychology |
| • Anthropology | • Economics | • Public Speaking |
| • Architecture | • Engineering | • Sound Design |
| • Brainstorming | • History | • Technical Writing |
| • Business | • Management | • Visual Arts |
| • Cinematography | • Mathematics | |
| • Communication | • Music | |

Might seem daunting but a team of skilled designers who will collectively share their knowledge and actively cooperate, solutions can be found on every problem that will be encountered during the designing or development process.

The World

Every videogame must have a virtual world for the player to interact in. From pixels and moving images to 3D geometry and from realistic graphics to stylized pixel art, every game world is unique and can have many forms. It can be comprised of small independent levels, it can have smaller levels that are interconnected and create a big cohesive world, it can have a big world for the player to navigate through or a combination of the above depending on the designers' vision.

The setting where the games take place, will greatly affect the world. From Middle Ages fantasy with castles and peasants to spaceships and alien planets, no matter where and when the game's setting is, one of the most important aspects is how alive the world feels.

Creating buildings, cities, forests, or any type of environment is not as simple as it may initially be perceived.

Populating the world with props, people, creatures, sounds, visual effects that give a sense of life to the world when at the same time maintaining a clutter free experience for the player is

difficult, especially when optimization is taken into account so the game can run smoothly on the players machine.

A common term used in gaming is "Immersion" and the game's world plays a big role in that. One of the designers' goals is to make the player feel that he is transported into another world that runs without him but at the same time make him feel like his actions have an impact inside that world.

The Gameplay

A unique aspect of videogames as an interactive art form, gameplay is not bound by graphics or realistic physics simulations, it is entirely based on the experience the player has while playing the game. Depending on what the players' style is the gameplay experience can vary greatly, some players seek challenging competition and complete mastery of the game, some seek relaxing exploration of the world, some a thrilling storyline and interesting characters. The same game can be made to a completely different experience but still enjoyed the same by individual players.

Sometime game makers point to the direction they want the game to be played, other times they fill the game with different systems and offer the players endless choices, but more often than not players tend to disregard directions and try finding themselves how they like to play the game.

A crucial part of the gameplay experience is the different systems designers add to the game or commonly referred as "mechanics". Some examples are:

- Responsive control
- Skill trees
- Unique combat
- In-Game economy
- Crafting

Multiple types of mechanics can be part of a game. The issue for the designers is to make all the systems work with one another flawlessly.

A huge challenge in designing a game is to choose which mechanics the game must have. Developers often overdesign their games and fill them with all types of systems and consequently lead them in the creation of games that overwhelm and deter casual players.

Another issue that developers will most definitely face, at some point of the process, is the difficulty level of the game. It is evident that all game makers want to offer a enjoyable experience to the players but deciding how challenging the game should be is not an easy task and depends on a lot of variables like game genre and player demographic.

An integral part of every game is the difficulty curve, which affects greatly the gameplay experience.

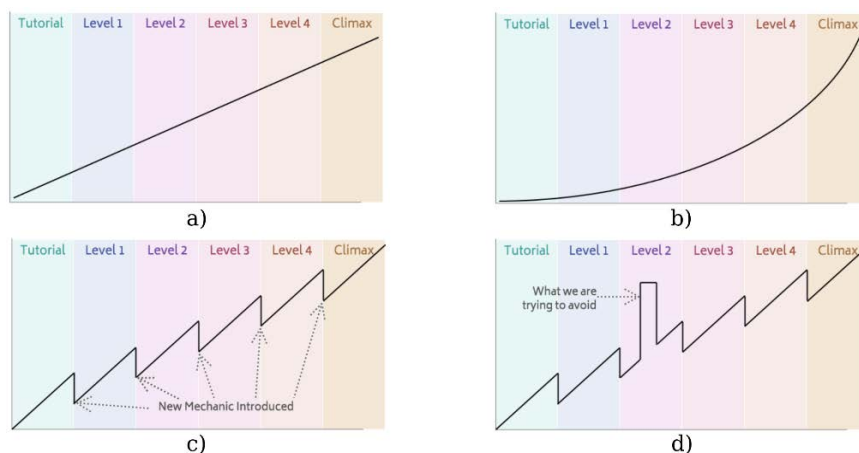


Figure 61: Difficulty curve examples a) Steady increase b) Easy to learn hard to master c) Teaching new mechanics d) Sudden difficulty spike, bad design. Source:[138]

Some more sophisticated games, include complex systems that dynamically adjust the difficulty levels in real time, by taking into account numerous variables. Those systems can personalize the gameplay experience based on each individual's skill level to make the game enjoyable to first time players to skilled veterans.

Valve's Left 4 Dead games, employ a system called The Director[139] "that features a dynamic system for game dramatics, pacing, and difficulty." Instead of predetermined scenarios "the Director places enemies in varying positions and numbers based upon each player's current situation, status, skill, and location, creating a new experience for each play-through. The Director also creates mood and tension with emotional cues, such as visual effects, dynamic music, and character communication."

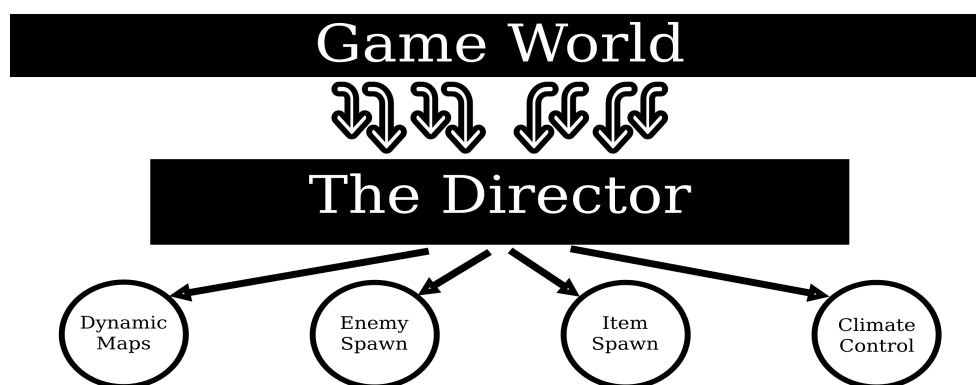


Figure 62: Overview of the Left 4 Dead 2 director's influence in the game.

The games' task is to control the pacing and make the player feel satisfied when overcoming challenges while at the same time not overwhelming him.

Huge outliers are games with absolutely non-existent gameplay by design. Often called experiences or walking simulations that are games designed just to offer a visual journey in awe

inspiring environments.

Gameplay along with the World make a great part of the game's experience. Managing to combine a beautiful alive world with an intuitive gameplay can elevate the players' immersion and make them feel like a part of the virtual world.

(Section 3.3.b) Technical Components

Every game can be dissected down to two individual parts, the Assets and the Logic. The word "Asset" is a broad term referring to a multitude of things that constitute the videogame world, and "Logic" is the underline code that creates a complex network of connections between the Assets.

Assets As previously discussed, a lot of different things can be a game asset, we can classify them as:

- Objects
- Sounds
- Visual effects

OBJECTS

Objects are the characters, buildings, weapons, props, and every item the player may or may not be able to interact with.

Every object is comprised of geometry, materials and collision, some objects can also have animations that make them move in some kind of way. Depending on the type of game, an object can be 2D or 3D, another approach is to use 3D but from a locked 2D camera perspective, that method is called 2.5D. The focus of this section will be on 3D game development with brief mentions to the 2D counterpart techniques.

1. Geometry

An Object's geometry or often called "Mesh" is defined as collection of 3 dimensional vectors that form triangles which combined create a visualization of a shape in the game world.

In early 3D games, objects were limited to a small number of triangles, but as computational power increased so did the triangle count.

- 1996: Crash bandicoot's model was 512 polygons, roughly 1024 triangles
- 2004: Half life's 2 Alyx was 8000 triangles
- 2011: Elder Scrolls V: Skyrim player character with full armor and weapon, estimated around 30.000 triangles.
- 2024: Characters are about 100-200 thousand triangles.

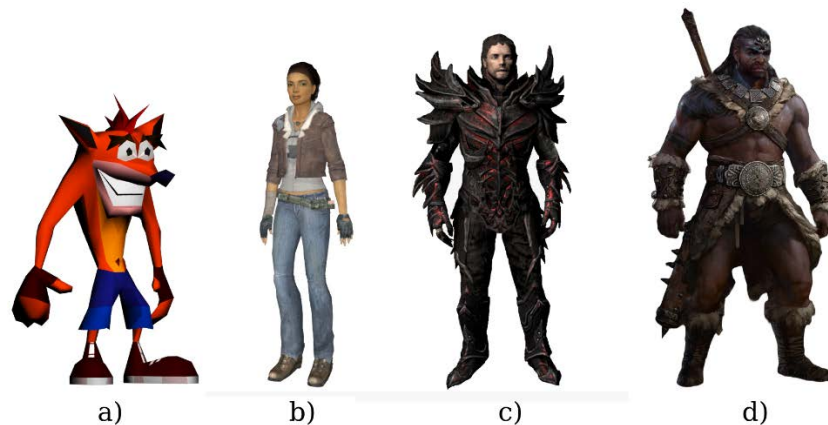


Figure 63: a) Crash Bandicoot b) Half-life 2 - Alyx c) Skyrim Daedric Armor d) Diablo IV Barbarian.

In modern video games employ a variety of technics to reduce the hardware strain from geometry. A common practice is Level of Detail or more commonly referred as LODs.

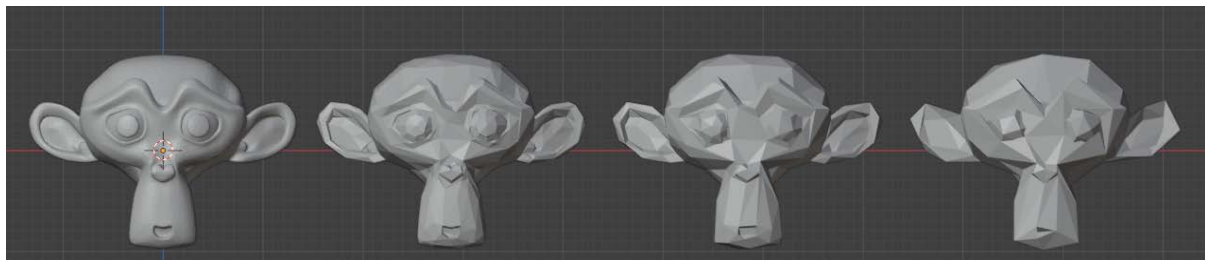


Figure 64: Example of LODs. Left-most is closest to camera Right-most the furthest away.

LODs are a variation of the same object's geometry but with a lower triangle count which changes depending on the object's distance from the player. It allows for rendering of detailed models close to the camera while far away objects become low resolution for performance's sake.

2. Materials

A material is a combination of different texture maps that make up the surface rendered on the geometry.

Modern videogames use the Physical Based Rendering, PBR for short, material pipeline that includes multiple variables that control the rendering of the surface.

- Color
- Normal
- roughness
- metallic
- specular
- ambient occlusion

- emissive
- opacity

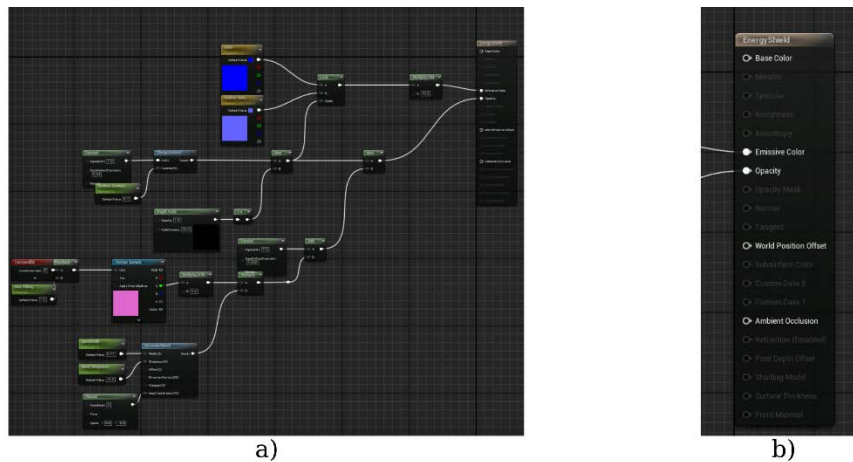


Figure 65: a) Animated Material example from unreal engine 5. b) List of all supported maps used in unreal engine's PBR workflow. In this case (Opaque) only the supported ones are enabled for use.

Those are the most commonly used in the game development process and each one is represented by a specific image map that has a significant effect on the final material.

Although not necessary to use the PBR pipeline, only a single texture map can be the material, the surface rendering quality is far greater than the flat texture on the geometry.

Despite the evolution of 3D graphics, development of 2D games have not stopped. The technical approach is similar but restricted due to the 2D nature of the game. The object geometry is confined to square plains with one or more materials rendered on them.

3. Collision

In game development in order to make different objects interact with each other or the player, a system must be in place to detect when one geometry is intersecting another or it is within a very small distance tolerance, that system is called collision detection and is a very useful technic applied in various parts of the game.

Making objects rigid and not having them just be a 3d projection of an image is important. A player character should be able to stand on top of a surface and not fall through it, so every object along with the geometry and the materials, it has a collision field.

There are two types of collision:

- Simple collision can be an assortment of basic shapes like boxes or spheres that wrap around an object.
- Complex collisions can be either the objects geometry itself or more commonly an LOD of the same mesh.

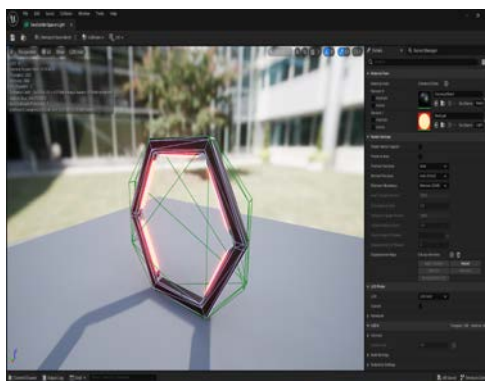


Figure 66: Collision example from unreal engine 5. The GREEN lines represent the simple and the WHITE lines the complex collision.

Collisions give objects the ability to interact with other objects' collision and creating a perception of rigidity inside the game world, but more complex collisions have a performance impact on the game, it is advised to use simple collisions in the majority of circumstances and complex collisions must be reserved for special cases.

SOUND

An essential part of every game, sounds give life to the game and boost the immersion of the player inside the virtual world.

Michael Cullen of the University of California, offers a break down in Sound Design featured in games[140] and can be categorized as:

" DX = Dialogue - any verbal speech in the game.

MX = Music - any non-diegetic music.

SFX = Sound Effects (Hard Effects) - any sound from a real-life object.

FOL = Foley - any sound effect that the player makes.

BG = Backgrounds (ambience) - noise from the environment." A lot of players do not find it appealing to read long story scripts or dialogues while having to stop their gameplay. DX offers the players the ability to continue playing while a narrator tells the story and hear the characters talking with each other.

Game music and ambient sounds set the mood and tone of the scene, adding tension and suspense to the gameplay.

SFX are sounds recorded from real life and imbedded inside game objects to make the player's interactions have audio feedback along with the visual, greatly increasing the feeling of immersion.

Foley sounds are similar to SFX, they include character footsteps, breathing, cloths rubbing while walking or running and give an added feel of realism to the game.

Modern games implement sound localization technologies, also known as spatial sound or 3D sound, used with SFX and DX to simulate the sense of space inside a 3D game world.

Sound is not only for aesthetic purposes but has a role in gameplay too as they can be used to enhance and assist various game mechanics.

Spatial sounds give the world a lot more realism and should not be left out of any game. The most basic example is pinpointing enemy location based on where the gun sound is coming from instead of using visual indicators.

VISUAL EFFECTS

Visual effects referred commonly as VFX were part of mainstream industries like film and TV, but as computing power increased, games started adopting it immediately as is the next level of realism and creating life in a digital environment. The difficulty of implementing VFX systems inside games is greatly impacted by the need of real time rendering.

Films have the advantage of being prerecorded and can be rendered as long as needed in huge distributed systems to get the desired quality, games on the other hand are played and interacted with in real time, that poses a huge trade-off between hardware limitation and desired VFX quality.

Developers try make the most efficient use of the screen space and find workarounds to minimize the trade-off in creative ways by:

- limiting the use of VFX only where the player can see.
- using a technic similar to object LODs to set the appropriate quality depending on the VFX.

Game VFX Categories:

- Lighting
- reflections
- animation
- particle systems

1. Lighting

There are 3 types of lighting used in the game development process.

1. Static lighting rendered during the development process on top of the affected surfaces and can not change after the fact, also known as Baked lighting.
2. Dynamic lighting rendered in real time by calculating the light source position in space and lighting the surfaces dynamically inside a specific area of effect.
3. Ray Tracing lighting "is a rendering technique that can realistically simulate the lighting of a scene and its objects by rendering physically accurate result".[141]

Most would immediately choose the use of Ray Tracing as it delivers the most realistic result possible, but taking in consideration the performance costs the player base will dramatically be reduced due to the minimum hardware requirements, it is a new technology used on the best and most realistic games.

Baked lighting has no performance cost at run-time but has the huge limitation of not being able to effect moving objects or static objects that are added during gameplay.

Dynamic lighting can affect moving objects and can have a light source that moves around in space, it is the most realistic approach but also has a high-performance cost during run-time.

Both have their advantages and disadvantages, the optimal solution is a per case mixture of both technics, the choice is in the hands of developers to make based on the desired result.

2. Reflections

Reflections are very similar to lighting as they are split between static/baked and dynamic/real-time reflections but both categories employ more than one technic for rendering.

Reflection Technics

- Planar Reflections
- Screen Space Reflections
- Cube Maps
- Ray Tracing Reflections

All technics can be used baked or real-time depending on the developer's decision, but a combination of technics is advised for achieving optimal results.

a) Planar reflections

Using a virtual camera to capture the reflections and render them in the scene. Offers good quality reflections but it is a very resource demanding approach because a separate render pass is required.

b) Screen Space Reflections

The most efficient and widely used type of reflection generation technic uses the already rendered frame to calculate the reflections but comes with a significant limitation, it can not render what the camera does not see, resulting in many artifacts and medium quality reflections when combined with harsh camera angles and rough surfaces.

It is most often used on water or floor surfaces and not as an independent system for all reflection needs of the game.

c) Cube Maps

Introduced by Nvidia in 1999 but still being a relevant technic for reflective but not mirror surfaces. Cube mapping uses an invisible square and it's 6 facets to capture the reflections from the surrounding scene and create 6 texture maps to stitch and wrap around the object.

d) Ray tracing Reflections

Ray tracing is not used only for lights, by bouncing the rays on top of each object it generates physically accurate reflections [141]. The resulting render is impressive but similar to Ray traced lighting it is the most resource expensive technic.

3. Animation

Having animations is the first and most important step for giving it life to every game. Depending on the type of game animations can be separated as 2D and 3D.

a) 2D Animation

Making games in 2D is very similar to making traditional 2D cartoons. Hand drawn or digitally created images are played in quick succession in order to give players the perception of moving objects.

Despite being based on the same core technic game animations have a very significant difference, the scene is not drawn as a single picture frame, each object is animated independently and composited on the scene.

b) 3D Animation

Animating 3D objects is not as easy as drawing the next 2D animation frame, it requires the use of a special technic known as a "Rig". Every 3D object must be "rigged" before being animated. The role of a rig is to influence the coordinates of the object's vertices in space. On simple objects rigging is the placing of one or more hinging point but on more complex objects like humanoid characters a more advanced rig is required that plays the role of the bones and it's often called "Armature" or "Skeleton". Depending on the desired animation fidelity the skeleton's bone count changes.

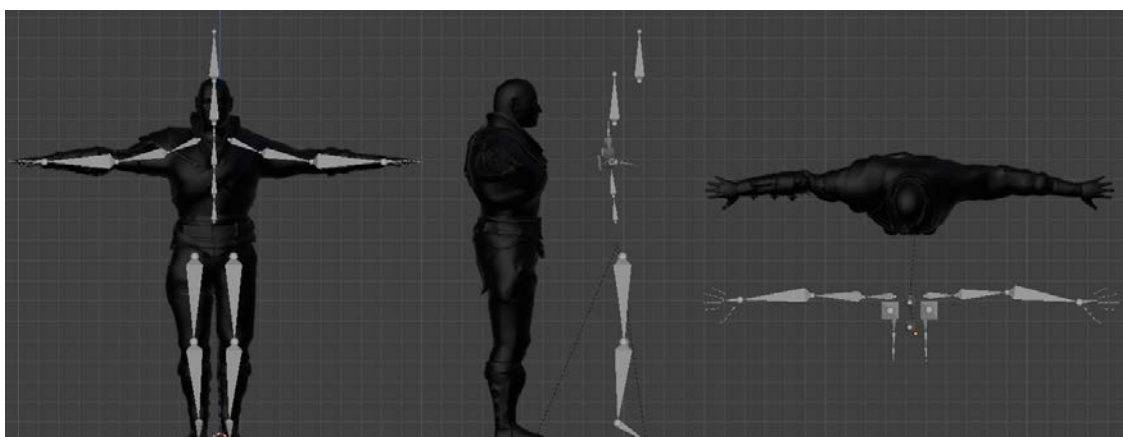


Figure 67: Armature and mesh(displaced) from front, side and top view.

The animators create key frames or poses by manipulating the bones. Interpolating between poses creates the final animation. Animations can have 2 or more key frames depending on their

complexity. Mathematical 3D vector coordinate interpolation is not enough to provide smooth animations, increasing or decreasing the number of key frames will dictate the fluidity of each animation.

Creating simple animations is not a hard task on its own, making physics accurate complex animation that blend seamlessly with each other is another matter. Thankfully there are tools on the market that assist with the physics simulation part of the animating procedure, enabling developers to create realistic results with less work, thus increasing individual productivity, immensely important in small development teams.

4. Particle Systems

Particle systems are a core aspect of visual effects, they are a vector field consisting of small images or 3D objects that act independently based on forces acting upon them.

They are used for "fuzzy" effects like fire, smoke, explosions and other similar vfx. The number of particles used will often dictate the quality and realism of the effect, more particles mean more quality but their independent nature links the computational cost to the particle count making them a very resource expensive system.

The main component of a particle system is called particle emitters and is responsible for the way the particles are emitted from the source and keeping track of each individual particles variables. A particle system can have multiple emitters active at the same time or one after another, in order to achieve the desired effect.

Most important particle variables

- color
- size
- velocity
- rotation

Every particle has a huge number of variables, some of them remain static and some of them follow curves in order to change their value during every particle's life span.

List of emitter types

"Sprite Emitters - These are the most basic types of emitters and the most frequently used. Particles are emitted as polygonal sprites (2-polygon cards) which always face the camera. Useful for smoke, fire, and a wide variety of other effects.

AnimTrail Data - Used to create trails from the sockets of animated skeletons.

Beam Data - Used to create beams such as lasers, streaks of lightning, and similar effects.

GPU Sprites - These are special types of particles in which the bulk of runtime calculation is passed off to the GPU. This boosts the number of possible particles from the several thousand possible with CPU particles to several hundred thousand, depending on the type of GPU in the target system.

Mesh Data - Instead of emitting a series of sprites, this type of emitter will instead emit polygonal meshes. Great for rockslides, debris, and similar effects.

Ribbon Data - This produces a string of particles that are attached end to end to form a ribbon which trails behind a moving emitter." [142]

Particle system Optimization

As discussed previously, particle systems are a very resource expensive technic and as a result there was a need for optimizing their performance.

Particle System Level of Detail is the most common way to enable developers to minimize that cost by reducing texture quality or even disabling entire emitters when the camera is at a far enough distance from the system that the player would not notice a difference.

Logic

In the previews segment we list and analyzed in depth all the components that constitute a video game, but those components themselves can not make a video game. As chess pieces do not make chess what it is, objects, vfx and animations do not make a video game, they are just the means with which the games rules and gameplay are enacted by the players.

In the realm of video game development, the rules of the game are the logic imparted within the other components, interconnecting them with each other and creating a complex web of in-game system interactions. Every game is an event driven interactive experience that each player delves into and with his actions effects those events inside the virtual world.

Types of Events

1. Button press Events.

The simplest type of in-game interaction. When a player presses a keyboard or a controller button an in-game event is triggered resulting on a predefined scenario to be executed.

Common button press events:

- Character animations like walking, running, jumping.
- Weapons firing attacks
- Opening menus
- User interface mouse clicks

2. Collision Based Events

Collision Detection in videogames is not only used for rigid bodies, another crucial implementation of collision is also event triggering.

Primitive shapes like boxes and spheres are employed to detect when another object's geometry is overlapping them in order to trigger a specific scenario.

Those shapes are called collision boxes and remain invisible to the player but they play an important role in the developer's part.

Common Collision Based Events

- Proximity Opening Doors
- Playing a Sound
- Assuring proper player character position
- Player in range to allow specific action.

3. System Events

Many games implement complex systems that do not require direct player action to initiate, they exist in the game so the player may interact with them.

Including mechanics such as simulating a damage system, an enemy AI system, an in-game economy, or a reputation system. Their role is keeping track of a vast array of game variables, listening to changes that may happen during gameplay and triggering events depending on certain conditions the developers set.

Summary

The logic in every game, gives life to otherwise lifeless objects and enables player interaction inside the world created by the game makers. Game Logic can be complex or simple depending on the game's intentions, but a more complex game does not necessarily mean a better game and a simple game does not mean a boring game. Creating interesting and unique gameplay is not a matter of complexity, but rather a matter of creativity and understanding of the player-base desires.

The majority of players are not playing a game to become the best and making them feel that is necessary to "study" the game mechanics in order to enjoy it, will deter them from playing. Implementing intuitive systems and gradually introducing the complex ones to the players is key in developing a successful game.

(Sub-chapter 3.4) Chapter Conclusions

In this chapter it was attempted to define the concept of video games in a philosophical and sociological way by their impact to the world and offer a comprehensive analysis on the theoretical and technical parts of the process of making a video game as well as the required specialized roles that contribute to the final product from the conception of the idea to the release of the game.

It is evident that before attempting to tackle the technical aspect of the process and assume a role in a development team, someone must have a good understanding of the nature of video games as well as a broad view of the entire process as each part interlocks with each other to produce a single coherent game without inconsistencies between the different parts.

CHAPTER 4 Challenges of Video Game Development and Benefits of Employing AI

(Sub-chapter 4.1) Challenges Depending on the Scale

Modern Gaming has become a huge entertainment industry. As games set higher and higher standards, major titles required the involvement of more and more people, that lead to the formation of huge publishing studios comprising of many development teams. Today's AAA titles need hundreds of skilled people, each one being specialized in one aspect of the process and take multiple years to develop.

Despite the difficulty, individuals did not stop pursuing game development. With open-source software solutions available and companies offering free public access to their software in exchange for royalties, many people joined the development scene individually or in small teams. Independent studios were born and unrestricted from corporate interests, they were free to explore innovative and niche ideas.

(Section 4.1.a) Large Scale game development

Large scale game development produces games of great quality, pushes the technological limits and moves the gaming industry forward, but all that comes with a lot of challenges that need to be addressed and overcome.

Size

In the first years of video game development, one person or teams of a few people were developing videogames. As videogames moved from a research field to a small niche part of the entertainment industry, players as well as developers started demanding for higher quality games, that inevitably led to the addition of more and more people in the development teams to help with the growing demand and quality expectations.

When games hit the mainstream and major corporations started seeing them as an investment opportunity, they started pouring money expecting good turnovers. Increasing pressure from investors and from the player base caused the development studios to exponentially increase their size.

Game development Studio approximate size list:

- Small: 15-50 people
- Medium: 50-100 people
- Large: 100-500 people
- Extra Large: 500+ people

Human Resources Management

Large scale development regularly faces the problem of employee management due to the large number of people involved in the game development process but also because more often than not the studios are split across multiple locations and countries.

It is very difficult to create and maintain an inter-team communication system spanning hundreds of people, when at the same time trying to maximize productivity and have strict production time frames.

Late-Stage Technical difficulties

At the later stages of the development process, unexpected technical difficulties may be presented and delay the release date. Especially in Massive Multiplayer Online (MMO) games that want to release in multiple countries, the server solutions that seemed adequate during the testing process, may prove to be problematic in a larger player influx and require complete redesign of the game's infrastructure.

Another late-stage challenge is finding pacing problems during the final larger scale testing process. Players completing the game faster than expected or having a hard time understanding the mechanics of the game, thus losing interest. It can prove to be a major step back and create huge delays.

Art out-sourcing

It is not uncommon for development studios to outsource art in order to focus on the technical aspects of the game and the mechanics, but poor communication between the two parties can create problems. Realizing after the fact that the outsourced art is not on par with the quality requirements after spending huge sums of time and money while creating the need to outsource again with a different studio or create it in-house will dramatically delay production and will increase costs.

Huge production times and market pressure

It is not uncommon in large scale game development production time to be in the multiple years ranging approximately between 3 to 7 years [32] and in some rare cases even more. During such large development periods the industry's advancements may be so big that the game ends up seeming outdated in terms of graphics or mechanics, forcing studios to try catching up with current trends and technology. Changing game engines one or multiple times during the development will inevitably lead to major delays and technical difficulties.

Code Rot

Code rot, also known as software rot or bit rot, describes the tendency of software to fail when it encounters an exceptional condition. If the design was insufficiently robust and assumptions made during development become out of date, this may cause it to fail in unexpected ways.

It is common that code written for the game in the prototyping stages may not be optimized for long time maintenance or account for every possible scenario and will need to be redesigned in the later stages of development.

A prime example of code rot was when some early video games tied some of the games' functions to the CPU clock speed. Later when users upgraded to a faster system the games ended up being unplayable due to bad foresight and design decisions.[143]

Uncertainty

The game designing process is a very difficult task and always includes a level of uncertainty as it tries to appeal to many peoples' subjective opinion on what a good video game is.

During the development of a game, huge chunks of the game may be discarded or redesigned from scratch. Revising the games' vision midway through the production and radically changing the art style of the game or redesigning the mechanics of the game to fit better with the overall feeling and storyline, will cause a major setback in both time and money for the studio.

Maintenance

Most prevalent in online games, maintenance is a core aspect of a game's lifecycle. Balancing, bug fixing, updating the outdated and releasing new content is essential in prolonging a video game's life and maintain relevance for a long period time.

It is very costly to even maintain a game in it's released state, keep adding content is even harder. Many companies try to utilize micro-transaction systems for in-game purchases in order to keep financing the post-release support of the game. These micro-transactions can be for purely cosmetic, or they can give an in-game advantage by affecting the player's performance, the first do not bother the player base but the latter are considered as underhanded pay-to-win methods. Companies who used those pay-to-win approaches are heavily criticized for being greedy and for using underhanded paywalls.

(Section 4.1.b) Small Scale game

When a small team tries to tackle an enormous challenge, such as developing a game, each individual will be assigned with multiple roles and the workload as well as the responsibilities are increased exponentially. Often "projects have ambitious goals that may or may not be underestimated during the design phase, causing delays in trying to reach those goals to a high degree. They get stuck in development for an extended period of time, never make it to production and are gradually abandoned by the people involved"[144]. This is called "Development hell" and it is very common among inexperienced independent teams.

According to Juego Studio [144], a software and game development company based in India and operating internationally, this is a list of challenges that make projects fall in Development Hell:

- Poor project planning
- Financial challenges
- Unforeseen technical issues
- Licensing and legal issues
- Change in management.
- Technology changes
- Unrealistic deadlines and overly ambitious project
- Internal conflict

Careful planning in the early stages of development, depending on the team's capabilities but also resource restrictions such as budget and setting an appropriate shipping deadline, are essential to avoid falling in Development Hell.

(Sub-chapter 4.2) Benefits of Employing AI in Game Development

The videogame industry depends on the cooperation of many specialized experts for every aspect and component that constitutes a video game, making it very resource expensive in both time and money. The employment of AI technology is a natural step forward in the quickly expanding field of video game development.

Artificial intelligence can be used to assist in the development of video games by analyzing players' data mined from playing games as well as player reviews, making it possible to identify trends and patterns in the player base's behavior. Such analysis would be impossible without the use of data mining and machine learning algorithms that provide the developers with valuable data and enable them to create the best games possible.

A new leap in AI technology is Generative Artificial Intelligence that is capable of language and media generation by recognizing patterns of the data provided in the training dataset and can generate:

- Text
- Code
- Image
- 3D geometry
- Video
- Sound

There is a plethora of trained Generative AI tools on the web, most of them are utilizing a text or tag-based input system in order to generate media using keywords and descriptions the users provide.

(Section 4.2.a) Prototyping

Generative AI can prove a very useful tool in the prototyping phase of the development, enabling designers to visualize their ideas while brainstorming, shift between different art styles, character, and level designs to see what will suit their vision of the game.

On the technical side of things generative AI can provide crud and simple models and textures so the developers have an initial start point that can be later refined as necessary.

Fast prototyping can impact the pace of the production process and give a huge head start to the developers.

(Section 4.2.b) Productivity

As generative AI continues to evolve, it can be used to increase productivity during the development process.

Artists don't have to start painting textures from scratch or design the 3D models vertex by vertex, AI can give them a head start and let them refine and build upon a generated base.

Make story telling easy by writing the main points of a script and let AI fill the not so important details or let it create stories branching away from the main plot in order to fill the game with more content without interfering with the basic premise of the game.

Composing songs and sound effects can become easier for the developers as AI can give them original ideas and stop them from getting stuck during the creation of new and innovative music.

Having AI do the repetitive and mundane jobs during the early stages of development can dramatically increase productivity. AI voice overs and narration of the game can help developers get a feel of the final product before investing the time and money hiring voice actors.

Level and environment composition with topology and topography-based AI algorithms can reduce the workload of designers immensely, as they only need to go and refine the levels after the AI has done the repetitive job of placing models one by one to populate the scene.

(Section 4.2.c) Individual or Small teams

Individual and small teams must shoulder the burden of all the required roles necessary for game development. The quality of the generated content provided by the latest AI can be enough so it can be assigned and take over the responsibilities of a specific role.

By splitting the workload and assigning multiple AIs for their respective tasks, developers can make it easier for them to pursue game development and not get overwhelmed from the knowledge requirements, by reducing the repetitive tasks or filling in for a missing spot of a person with the appropriate specialized knowledge of a field.

(Section 4.2.d) Realism, Immersion and Replay-ability

A huge field of study in AI used for game development are implementations of behavioral AI models that enhance the process of interaction within game characters humanoid or not and bring the virtual game world closer to a more realistic representation.

Immersion can be elevated as conversations and interactions will not just be a set of questions and answers read from a script and repeated the same way every time the game is replayed, but they will be dynamic, guided by the script but changing every time thus producing an infinite amount of interactions and encourage more replay-ability.

(Section 4.2.e) Post-Release Maintenance

Online games, besides the development process, require constant post-release maintenance and balancing. It is one of the most important and difficult aspects of online games especially if they are in a competitive e-sport scene. That is the reason why small developer studios do not even try to get into that part of the video game market due to the financial cost but most importantly the cost to sacrifice a team member from development to maintenance.

There are discussions by many video game developers to implement an artificial intelligence system to assume the role of the game manager and oversee a single game or a platform including multiple games in real time as the players interact with them. It may have the ability to control

some parameters to steer the player behavior in a certain predefined preferred way or just log and make reports for the game developers to address.

It would be a major asset that would enable more games and ecosystems of games to be managed by a single AI entity while freeing the hands of the developers and letting them focus on producing new content and games.

(Sub-chapter 4.3) Negative Effects of AI

Even with all its benefits AI technology does come with a fair share of negative effects as well. In this section we will discuss three of the most important ones in order to showcase that nothing comes with only benefits, there are always some drawbacks in some way or another.

(Section 4.3.a) Reduction of jobs

With the constant evolution and improvement of AI based solutions, many mundane and repetitive tasks were outsourced to AI as it was very cost effective and didn't require training to do the job like a person would. That led to a reduction of job positions especially in the entry-junior developer level. Making it harder for new people to get in the industry not only for game development but in general software development.

As a result aspiring beginners have to upskill fast on their own before being able to have all the requirements for a junior-mid developer position.

(Section 4.3.b) Sentiment of uncertainty

After seeing the capabilities demonstrated by the state of the art AIs, a sentiment of uncertainty was cultivated even to the most seasoned of developers. Many fear that AI will take over most of the tasks, which will lead to many fields of development being obsolete in the future. Rushing to specialize in a field which they believe to be safe at least for the foreseeable future as well as deterring everyone who might show interest in any of the fields they believe to be vulnerable to AI overtaking.

(Section 4.3.c) Over-reliance on AI

An over-reliance on AI happens when users believe the AI without considering if its suggestions are correct or not. Most people have an idealistic view of AI and are not able to determine the appropriate levels of trust. An AI tool, even if it is capable of delivering impressive results, it should not be trusted completely. Regardless of its complexity, it remains an algorithmic procedure which still lacks qualitative capabilities compared to the human mind. The outputs generated by AI technology should always be scrutinized by at least one human expert and cross-referenced in order to verify the correctness of the output.

(Sub-chapter 4.4) Chapter Conclusions

This chapter is divided in two parts the first was focused on an in-depth analysis of the challenges faced in modern video game development during the development process and after the release of the game in both small and large-scale scenarios. It showed that making a video game is a very hard endeavor and without a doubt many challenges will need to be faced until its completion. In the second part there was a discussion on how utilizing artificial intelligence can improve the game development process in multiple aspects. By automating parts and even eliminating entire processes will make game development more approachable and financially viable for small or individual teams. Lastly, some of the negatives effects that come with the rise of AI technology were discussed as well for the purpose of eliminating any misconception of AI having only positive impact on the development scene.

CHAPTER 5 Software used in Game Development and Replacement AI tools

(Sub-chapter 5.1) Software

(Section 5.1.a) Modeler

Started as Computer Aided Design, CAD, software for digital recreation of the real world in architecture, engineering, and construction industries but developed into a complete 3D Model designing suit that expanded its use in film and game industries. A modeler uses a collection of interconnected points representing various shapes such as triangles and polygons in 3D space, in order to create a digital 3d model reproduction of a physical object inside a virtual environment.

Every 3D model is comprised of

- Vertices - points in space represented by a (x, y, z) vector coordinates.
- Edges - the lines connecting different Vertices.
- Faces - polygons created when a vertex group is closed.

Faces can be further analyzed depending on how many vertices create by

- Triangle or tris = 3 vertices
- Quad = 4 vertices

Even though quads are used in the modelling process, in practice every 3d model is saved as a collection of triangles and every quad is converted internally to triangles, because a polygon that has more than 3 vertices is not deterministic.

A modeler is capable of creating and editing an immense number of Vertices, Edges and Faces in order to create models with great fidelity.

Brief timeline of important 3D CAD software

- | | |
|------------------|-------------------|
| • 1960 Sketchpad | • 1990 Cinema 4D |
| • 1964 DAC-1 | • 1994 Blender |
| • 1971 ADAM | • 1995 SolidWorks |
| • 1982 AutoCAD | • 1996 3Ds Max |
| • 1988 Cobalt | • 1996 Houdini FX |

All 3d modelling software developed in the 1990s are prevalent and still industry standard to this day, as they keep evolving to match the technology advancements.

(Section 5.1.b) Painter

After creating the 3d models the textures that will become the PBR material need to be painted on top of the geometry. Almost all modelers come bundled with a simple painter or an independent painting software with more specialized functionality can be used, depending on the needs of the artist.

Some of the industry standard painters are:

- Substance painter
- ZBrush
- Blender
- Quixel Mixer
- ArmorPaint

(Section 5.1.c) Audio Software

To record or edit the sound that will be used in the game, an assortment of software is needed for recording and editing sound effects, narration and mixing music.

Audio software for used in game development falls in two categories:

1. Game Audio Tools
2. Digital Audio Workstations

Game Audio tools, also referred as middleware, are different tools designed with game development applications in mind. They focus on 2D as well as 3D audio and also on offering solutions for interactive scenarios by understand different states of the game and dynamically parameterizing the sounds being played in real time during gameplay.

They are easily integrated inside the game engine of choice to provide an easy workflow for non audio expert users.

A list of the most used Game Audio Tools

- Wwise
- FMOD
- Miles Sound System
- Fabric

Digital Audio Workstations are the more traditional audio editing software offering a vast array of functionalities to the users.

A list of the most used Digital Audio Workstations

- Adobe Audition
- Cubase
- Komplete
- Audacity

(Section 5.1.d) Game Engine

The most important software for developing videogames is a game engine. It can be described as a framework designed for the development of video games, used by game developers for creating games in different types of platforms such as mobile phones, consoles, pc, virtual reality.

A game engine includes a variety of subprograms such as:

1. **Graphics Renderer** - Responsible for combining geometry and materials in order to show the final rendered graphics, 2D or 3D, to the player.
2. **Object Handler** - Keeping track of all game level instances and every object instance inside each level.
3. **Material editor** - Combining all imported texture maps and handling "material specific code" to create the final rendered PBR Material.
4. **Animation Handler** - Handling all independent animations and interpolating between user input and animation being played on screen.
5. **Lighting and reflection Handler** - Calculating lighting and reflections either dynamically in real time or static baked during the development process.
6. **Collision Detection Handler** - Detecting geometry overlaps between two or more in game objects.
7. **Particle System Handler** - System for creating and rendering in game particle system simulations.
8. **Audio Handler** - Subsystem for handling all audio being played in game.
9. **Physics Handler** - Subsystem responsible for calculating and simulating physics interactions between game objects.
10. **AI Handler** - Subsystem for creating predetermined AI behavior scenarios based on real time in game variable changes.
11. **Input Handler** - Automatic input handling subsystem for keyboard, game controller or other input device.
12. **Memory Handler** - Automatic computer memory handling based on real time game requirements.

Those are the core subsystems that constitute the game engine ecosystem, but on top of them a lot more can be added to increase functionality in the form of plug-ins or integrated subsystems included with new versions of the game engine.

A list of the most popular game engines:

- Unreal Engine 4/5

- Unity
- Godot
- Lumberyard (former CryEngine)

(Sub-chapter 5.2) AI Tools

(Section 5.2.a) 3D Models – Sloyd

Training an AI model for 3D object generation is no trivial task, especially when there is a requirement for optimized game ready topology. For the purpose of generating 3d models for use inside a game development environment Sloyd[145] offers a solution by combining parametric modeling and AI, trained on a plethora of different types of objects.

The available objects are categorized as:

- | | |
|------------|-------------|
| • Sci-fi | • Medieval |
| • Military | • Furniture |
| • Urban | • Modular |

Sloyd offers two ways of object generation 1) by interactive settings menu.
2) by text-prompt.

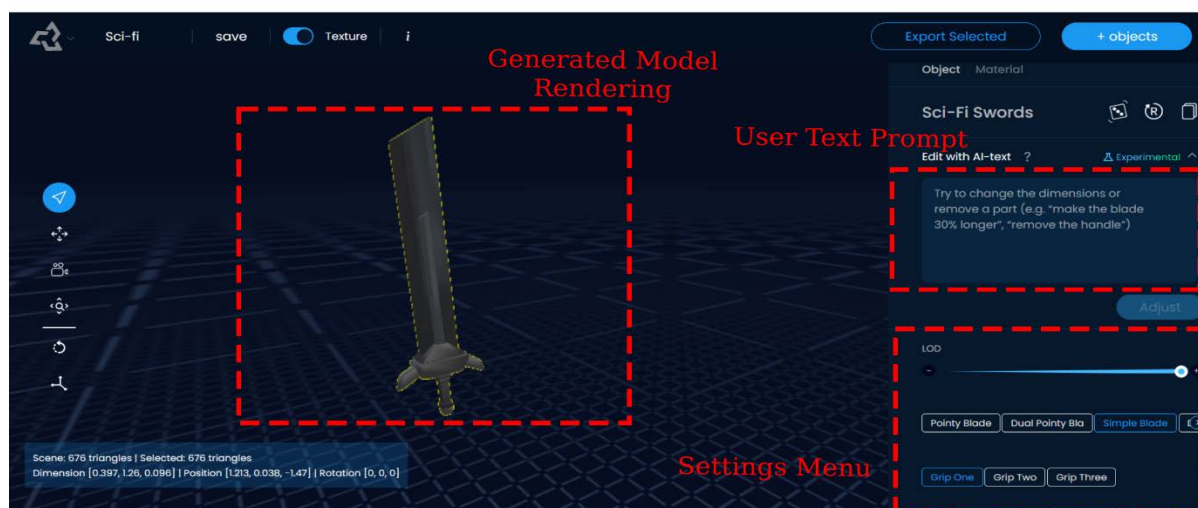


Figure 68: SLOYD AI playground interface.

Settings Menu approach

Sloyd offers an interactive menu comprised of sliders, buttons and check-boxes. By changing any of those elements, users can create great variation by enabling/disabling certain features,

setting different slider variables and choosing between available component variations of the object they are attempting to generate.

Text-Prompt approach

The user can provide a text-prompt describing the desired characteristics of the object he is trying to generate, and the AI tool will automatically change the sliders or enable/disable components.

It is important to mention that this method requires the user to study the different components of the object he is trying to generate a variation of and explain in detail the variable he wants to change.

After much experimentation with the text-prompt system it was concluded that the best approach is to prompt Sloyd to generate the model without mentioning any detail characteristics and after the model is generated prompt in the "Edit with AI-text" menu option and prompt the AI to change the components of the model or dimensions of them.

Generate a model Text-prompt examples:

- "Laser Blaster"
- "Sword"
- "Drone"

Edit a model Text-prompt examples:

- Sword – "Dual Angled, blade 0.8 length, tapered grip, enable grip end detail, pyramid grip end detail."
- Laser Blaster- "antennae barrel, round back, elaborate handle, enable barrel antenna, antenna 0.5 length."

Export Process

After generating the desired object, the user can export it for use outside of the Sloyd environment.

All objects generated by Sloyd come pre-textured with simple materials and a large assortment of different palettes of material presets is available. Users can export the entire model including the mesh and textures or can opt for the mesh only export option and texture the model themselves in a different software of choice.

Export file format:

- .OBJ - combines all component meshes into one mesh and exports it without any material.
- .GLB – Exports the model as individual separate components including all the materials

It is recommended to export the model as GLB format and join the meshes in a separate modeler software as well as perform changes in the material slots or set appropriate mesh origin as needed.

(Section 5.2.b) Textures – Polycam and Real-ESRGAN

1. Texture Generation

Polycam [146] is a company that offers a variety of different tools for free public use on their website. Their primary focus is Lidar photography and photogrammetry, but they also offer a text-prompt based AI texture generator that also includes image-to-image generation.

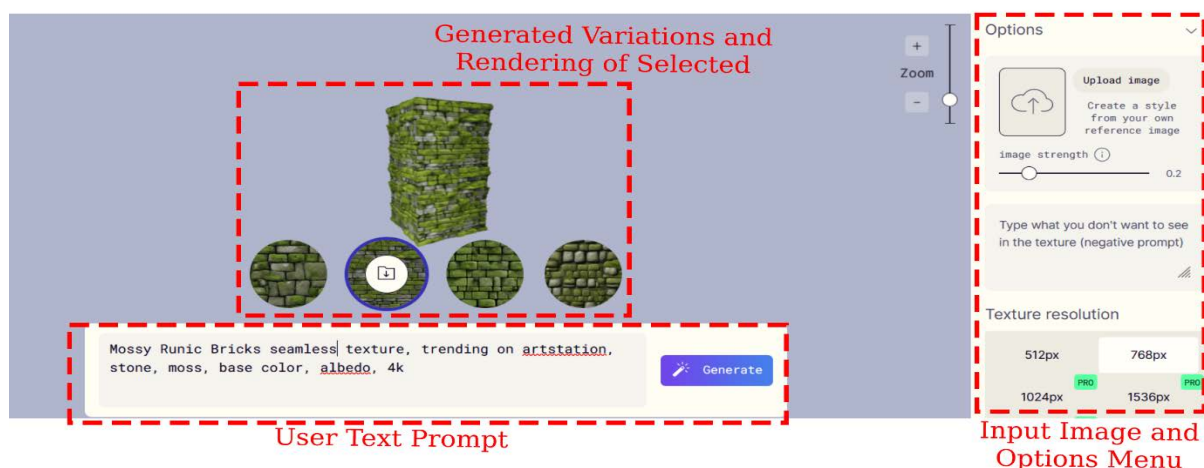


Figure 69: Polycam Texture Generator interface. Prompt is used to generate 4 different variations of the described texture.

The texture generation process is very simple. The user provides the AI with a set of keywords that describe the desired texture, and the tool generates four different variations based on them as well as showing a rendered preview of the PBR material rendered on top of a cuboid.

Another way of generating a texture is by using an existing image as an input sample for the AI to reference, fine-tuned with a variable that controls how close to the reference image should the generated texture be.

The AI texture generator is not only generating the color map, also known as albedo, but also generates the Normal, Displacement and Roughness maps.

This is an important distinction between image and texture generators, as the albedo alone cannot create a good result on the PBR material workflow. The results are flat and lack realism when rendered on top of 3D surfaces.

Another aspect of the polycam texture generator is that all textures generated from this tool are seamless and can be infinitely tiled on 3d surfaces without a noticeable seam.

Example of prompts and generated results:

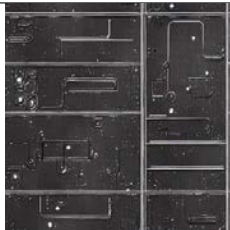
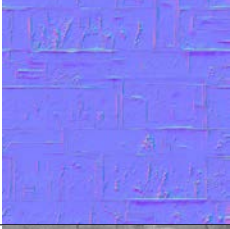
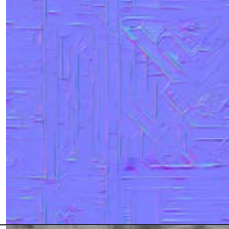
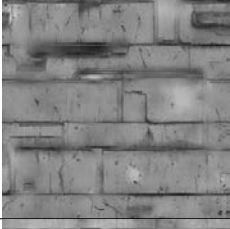
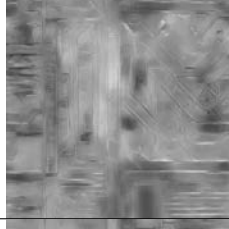
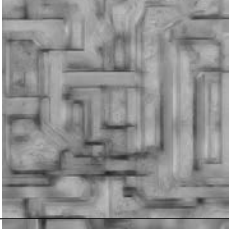
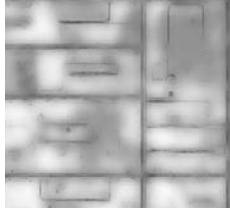
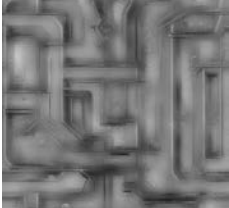
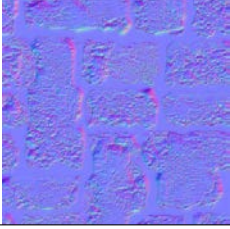
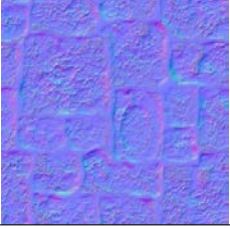
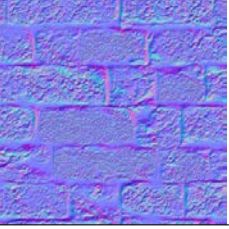
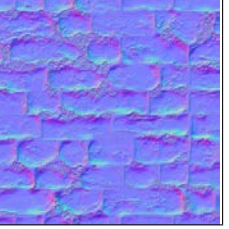
Albedo				
Normals				
Roughness				
Displacement				

Table 12: Prompt: "Cyberpunk, metal, wall panel, sci fi, technology"

Albedo				
Normals				

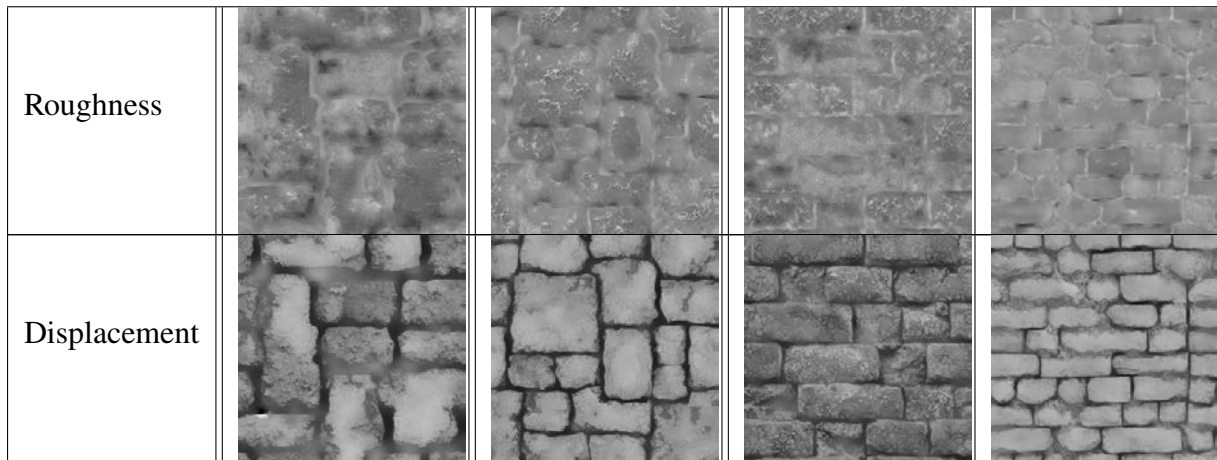


Table 13: Prompt: "Mossy Runic Bricks, stone, moss"

2. Texture Upscaling

After generating the desired textures and corresponding maps, we could immediately import them inside the game engine of choice and use them to render the PBR material. The problem is that the Polycam AI texture generator creates and exports images in 512x512 resolution (on the free version) which for modern game development is considered to be a very low resolution.

In order to solve this problem, we need to utilize a super resolution technique on your generated textures. An online implementation of the ESR-GAN [82] that was discussed in the second chapter of the thesis will be used to make the individual texture images in higher resolution before importing them into the game engine. The Real-ESRGAN AI tool [147] offers the option to upscale 2x, 4x and 8x super resolution, so in this case we can go from 512x512 to 1024x124 to 2048x2048 or 4096x4096 resolution.

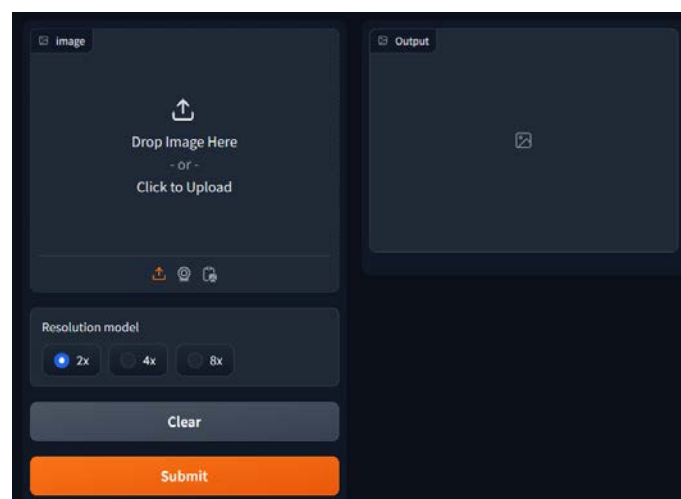


Figure 70: Real-ESRGAN playground interface. Source:[147]

For the purpose of developing a game inside unreal engine only the albedo, normal and roughness maps will be upscaled to 2048x2048 and utilized in the PBR workflow.

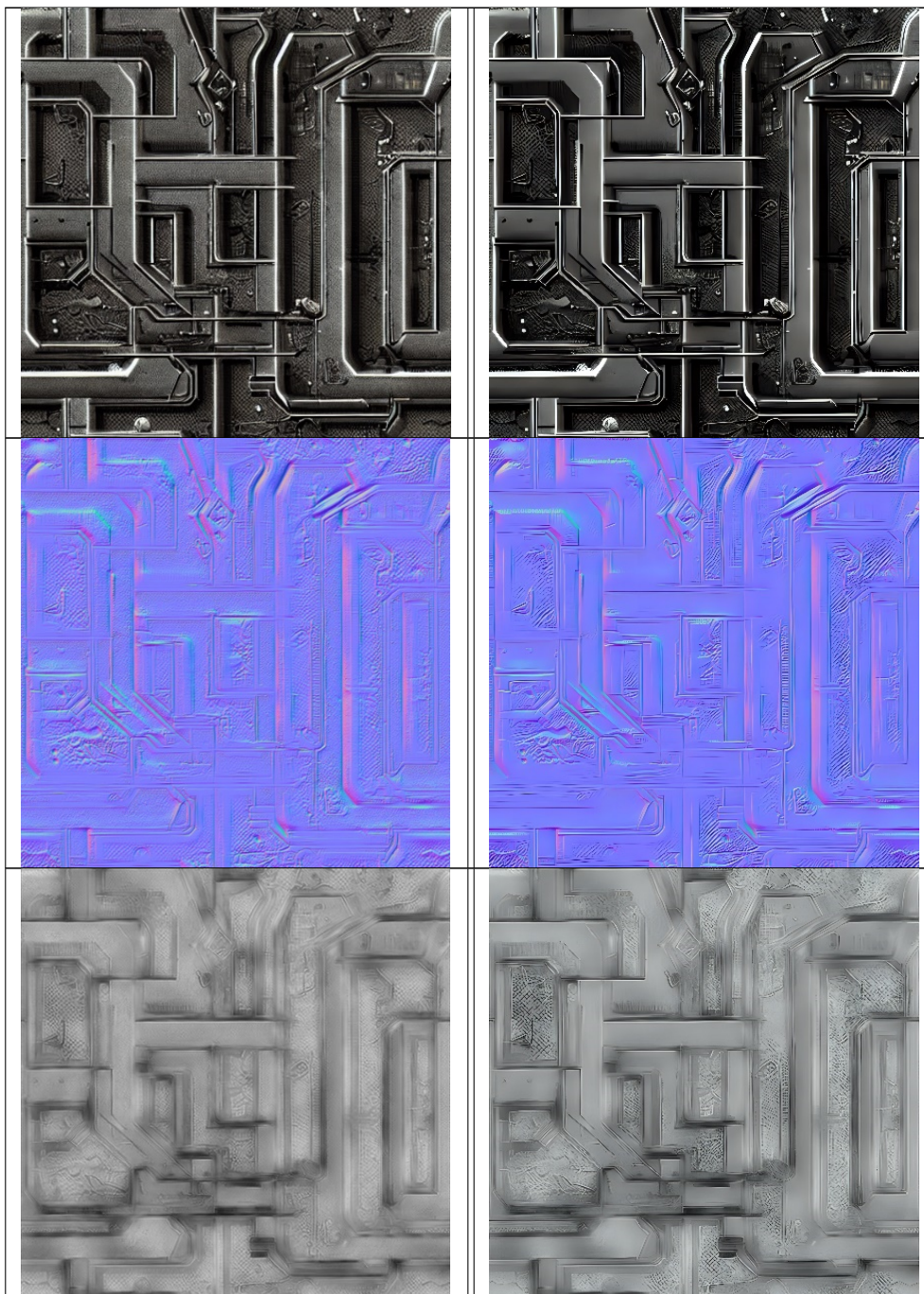


Table 14: Left the generated 512x512 textures. Right their upscaled versions of Real-ESRGAN

The quality of the upscaled textures is outstanding, with no observable artifacts from the Real-ESRGAN method. It produces a crisper image while at the same time removes artifacts produced by the texture generation model.

With careful examination, it can be seen that some details have been lost in the normal map upscaled texture, most likely because the model was not trained specifically with texture generation and the original image is of low resolution. Nevertheless, the model performs better than expected in texture upscaling compared to other methods tested (e.g. Pixelcut Upscaler AI[148]) which left many artifacts in the final image.

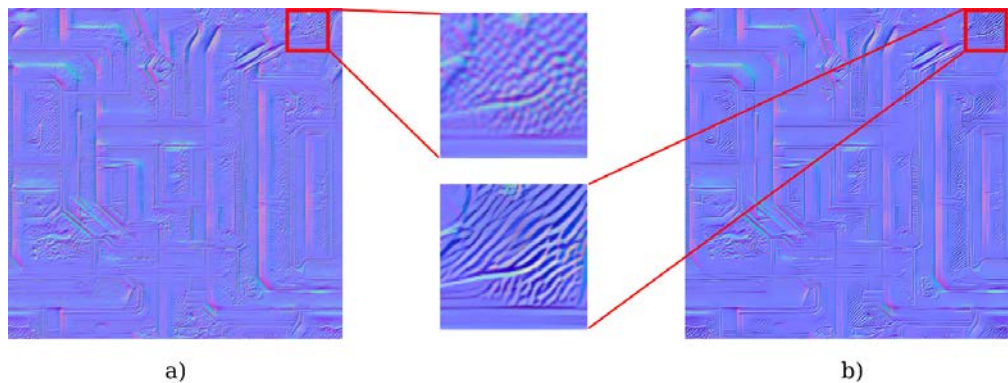


Figure 71: a) Original generated texture b) Real-ESRGAN upscaled version.

(Section 5.2.c) Audio – SOUNDRAW, Stable Audio and EvokeMusic

Audio can be separated in two categories:

- **Music**
- **Sound Effects**

Music - SOUNDRAW

For generating music, the SOUNDRAW AI[149] will be used as it offers a better environment for real time editing of the generated music than other similar music generating AI tools.

In order to generate music, the SOUNDRAW AI has a tag-based approach. The user must select one or more tags that describe the characteristics of the music track he wants, and the AI generates a list of song tracks obeying the selected tags.



Figure 72: Soundraw Interface.

Tag Categories

1. Genre

Hip-Hop, Drill, Trap, R&B, Jersey Club, Latin, Acoustic, Rock, Ambient, Beats, Drum n Bass, Electro & Dance, Funk, House, Techno & Trance, Tokyo night pop, Pop, Lofi Hip Hop, World, Electronica, Orchestra, Tropical House, Afrobeats, Phonk, Christmas

2. Mood

Epic, Happy, Hopeful, Laid Back, Angry, Sentimental, Busy & Frantic, Romantic, Funny & Weird, Dark, Glamorous, Mysterious, Elegant, Dreamy Euphoric, Fear, Heavy & Ponderous, Peaceful, Restless, Running, Sad, Scary, Smooth, Suspense

3. Theme

Corporate, Photography, Motivational& Inspiring, Cinematic, Comedy, Cooking, Fashion& Beauty, Nature, Sports & Action, Technology, Tutorials, Travel, Workout & Wellness, Gaming, Wedding & Romance, Horror & Thriller, Ads & Trailers, Broadcasting, Holiday Season, Drama, Documentary, Vlogs

4. Length

10sec, 15sec, 1min, 1:30min, 2min, 3min, 4min, 5min

5. Tempo

Slow, Normal, Fast

6. Instruments

Piano, Keyboard, Synth, Acoustic Guitar, Electric Guitar, Bell, Maller, Ethnic, Strings, Voice, Brass, Woodwind

By choosing one or more tags from the Genre, Mood, and Theme categories the user can make more specific generations.

The Length tag sets the approximate length of time the tracks will have.

The Tempo tag specifies to the AI the approximate maximum Beats per Minute or BPM the generated music should have.

The Instruments tag, enables or disables instruments included in the generated music. The AI will automatically choose all available instruments depending on the other tags chosen previously but the user can keep only those he wants.

After choosing a tag from a category, some tags from the other categories may be excluded automatically and the user will have to choose from the other available choices. Making combinations of tags that relate closer to one another will produce more desirable results.

Example of Tag list

- Genre - Electro & Dance
- Mood - Dark, Gaming
- Length - 5mins
- Tempo - slow/normal/fast tempo
- Instruments - piano, keyboard, synth, electric guitar, strings, voice, brass

After generating the music and finding a desirable track, the user has the option to generate similar music based on the track selected.

Track Editing

After generating and selecting the desired track, Soundraw offers the ability to edit it. The generated track is sliced in equal time intervals and the user can change the characteristics of each segment in the edit menu.

Moreover, the user has access to each segment's component level editing such as Melody, Backing, Bass, Drums, Fills and can edit them individually.

The available options are:

- duplicate segment.
- delete segment.
- change component energy level.
- Increase/Decrease track length

- 3 choices to change BPM.
- 3 choices to change instrument for each component.
- 3 choices to change Key.
- Component Volume control.



Figure 73: Soundraw Track Editing Interface.

After editing the track and reaching the desired result the song can be downloaded and used royalty free in any project. Soundraw offers a variety of export file formats like MIDI, MP3, WAV etc. The recommended type is WAV as it's the standard lossless, uncompressed format supported by almost every software.

Sound Effects – STABLE AUDIO and EVOKE MUSIC

1. Stable Audio

Stable Audio which was discussed in technical detail in the literature review part of the thesis, is an audio generating AI, capable of creating music as well as sound effects in high quality, by using a text-based prompt system as well as audio-to-audio generation. [123, 150]

It supports variable length stereo audio generation with maximum of 90 seconds with the AudioSparx 1.0 dataset and 180 seconds with the new AudioSparx 2.0.

The audio-to-audio feature supports maximum of 3 minutes input audio.

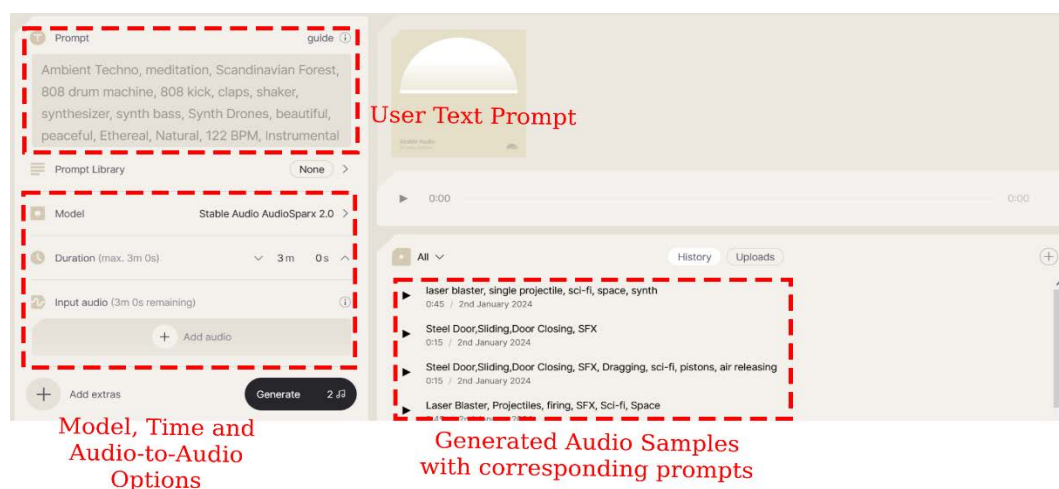


Figure 74: Stable Audio Interface.

Text-prompt construction

The more detailed the text prompt is the more specific the generated track will be, a good prompt must include descriptive words and phrases in order to guide the AI properly.

A combination of musical and emotional terms, setting, instruments and tempo will produce a very specific generation.

Prompt example:

- *"Dark, Sci-fi, Gaming, Atmospheric, Electronic music, Beats, Synth, Electric guitar, Bass, Drums, Action, Suspense"*
- *"Laser Blaster, Single Projectile, SFX, Sci-fi, Space, synth"*
- *"Steel Door Sliding, Dragging, Door Closing, Sci-fi, SFX"*

Before generating you can choose the duration of the track in seconds, minding of course the limits of the subscription package.

After generating the and selecting the desired track, it can be exported in MP3 or WAV formats.

As mentioned before, the free subscription package doesn't allow commercial use of the content generated, only the professional and enterprise packages allow for commercial use of the generated soundtracks.

2. Evoke Music

A different approach is to utilize Evoke Music[151], an online library comprised by collection of over 60.000 different royalty songs and sound effects created and composed by the company's proprietary Artificial Intelligence songwriting model named Amadeus Code.

The user has access to the AI generated content and is able to browse the website or search manually using key words that describe the song or sound effect he wants. Those key words

can be genre, mood or instruments included in the song and as for the sound effects they can be setting, item or action.

Examples of search prompts:

- "Door, Sliding, wooden"
- "Electric, Buzzer"
- "Laser, gun, space, sci-fi"
- "Rock, Electric Guitar, Epic"
- "Orchestral, heroic, strings"
- "Electronic, beats, space, sci-fi"

The first three examples are focused on SFX-sound effects and the latter three on music, but every one of those will try to find results in both categories.

A unique characteristic of Evoke Music is that every song generated is comprised by smaller segments called loops and the user is able to download not only entire songs but individual loops that he desires as well.

Amadeus Code AI is powered by the Music Trinity Generative Algorithm or MusicTGA. It is an artificial intelligence generative algorithm capable of generating "rhythm, melody, and harmony based on "Licks," which are minimal viable sound phrases". The data set used to generate music and SFX is comprised of "9 million licks, of which 99.5% are generated by MusicTGA itself " [152] and the rest 0.5% are licks are human-created by the company's employees.

(Section 5.2.d) AI assistant - ChatGPT

In order to have an AI personal assistant inside a videogame's environment, we need to leverage the power of large language models and more specifically OpenAI's ChatGPT. [153]

Communication with ChatGPT from inside the Unreal Engine, by using a community Plugin created by Kellan Mythen, that allows to call the OpenAI API. [154]

ChatGPT Unreal Engine Integration Process

Step 1: Acquire a key for the OpenAI API.

Step 2: Download and Install the plugin inside the Unreal Engine project.

Step 3: Access the library of functions created by the plugin.

Step 4: Implement the API call and response handling procedures as desired.

The API requires an input string to prompt ChatGPT and responds back with a string. How the game developers decide to handle the input and the output is entirely dependent on them. Designing proper User Interfaces for the user to provide input and then handle how the output response is presented back, is very important for the game's immersion.

(Section 5.2.d) Concept Art – Stable Diffusion/Easy Diffusion

While Stable Diffusion implementations are widely available on the internet, running such a model locally requires high computational capabilities on the users end, which can be problematic for indie developers who have tight budgets and cannot afford the best hardware or to rent cloud based virtual machines.

Gigantic Work [155] has created a free online implementation of the very popular Stable Diffusion model called Easy Diffusion. More specifically they offer three different variations being:

- SD XL 1.0 Base
- SD XL 1.0 Refiner
- SD 1.5 Pruned EMA

Developers can leverage the capabilities of those models without the need of running them natively, to create concept art for their games, for the purpose of visualizing their ideas and experiment with different ones, before spending resources and more importantly time to create them by hand.

With Easy Diffusion its possible to generate images using text prompts or image2image generation and the environment provided by Gigantic Works allows access to all the needed variables for model parameterization inside an intuitive layout for easy use even for someone totally inexperienced.

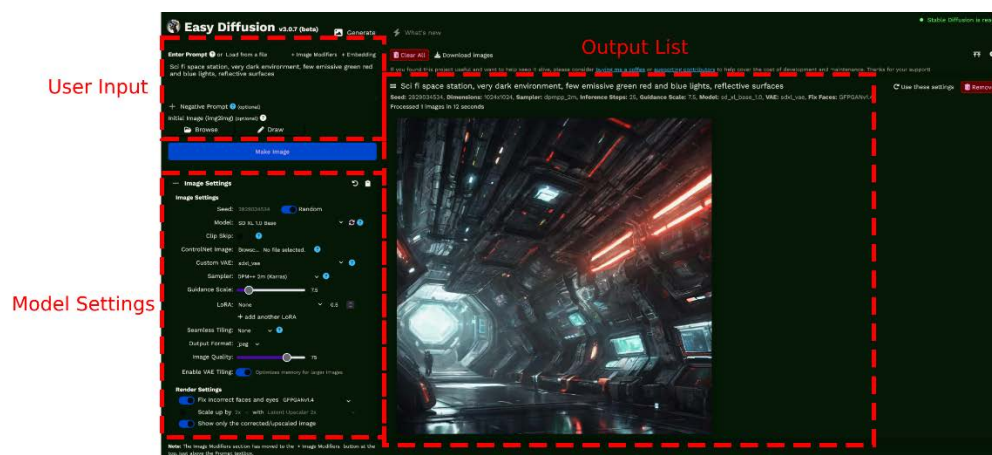


Figure 75: Easy Diffusion Interface.

The interface provided by Easy Diffusion includes 3 main fields as shown in figure 75.

- **User Input** Includes text prompt field, image input, user sketch input and image modifiers.
- **Model Settings** Include all the necessary settings for controlling the model used for generation.

- **Output List** Includes a history of all the previous prompts, as well as the corresponding settings and generated images for the current session.

Example prompts and generated images that was used as concept art for a sci-fi space station environment:

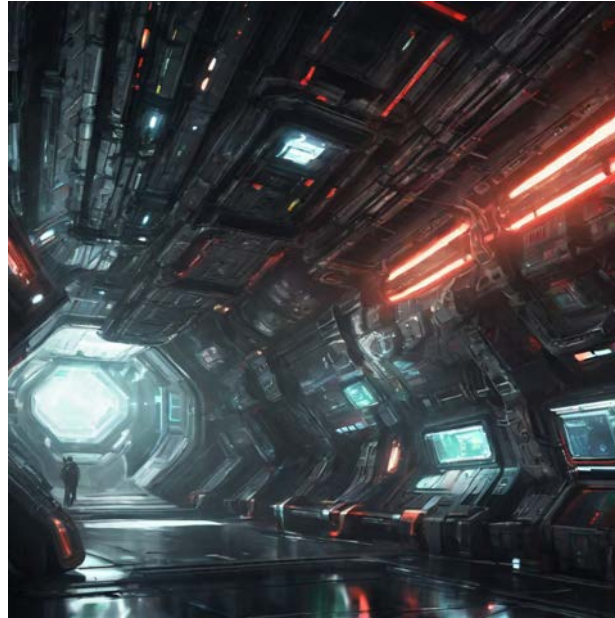


Figure 76: Prompt: "Sci fi space station, very dark environment, few emissive green red and blue lights, reflective surfaces"

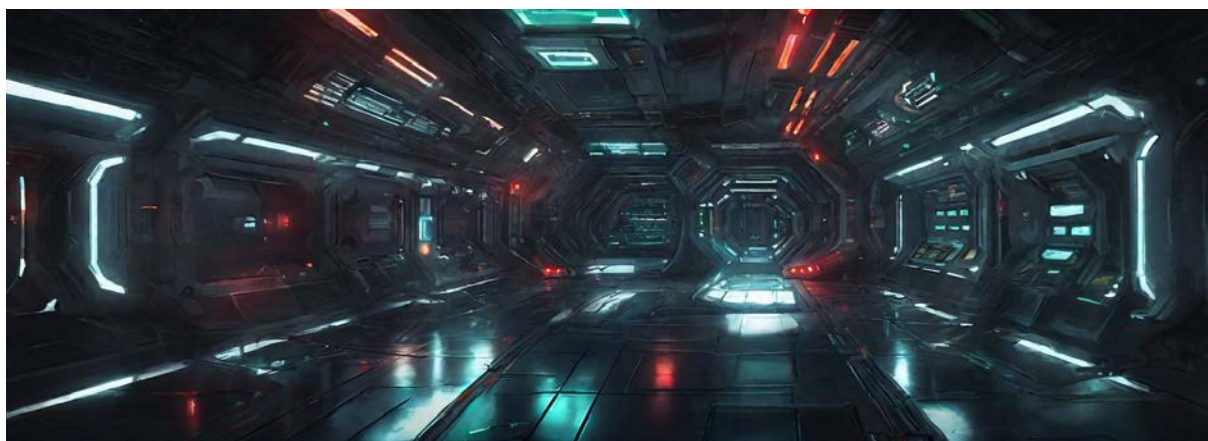


Figure 77: Prompt: "Sci fi space station, very dark environment, few emissive green red and blue lights, reflective surfaces"



Figure 78: Prompt: "Sci fi space station, dark environment, emissive lights"

(Sub-chapter 5.3) Chapter Conclusions

In this chapter, an overview of the necessary software needed in video game development was shown and available online AI tools that could replace or speed-up considerably the developers' workflow were proposed for use in game making.

Modern AI can offer a high degree of control in order to generate the best result possible in correspondence with the developer vision of the game.

With a focus on free or low-cost AI solutions developers can generate an array of assets from objects to textures to audio for use, as prototypes or as is, inside their video games.

It is a cutting-edge field of study that is constantly evolving with each passing day. As people start using the different AI technologies, scientists can collect even more data and slowly improve their models or create better ones by analyzing and identifying weaknesses after large scale public testing.

Tool	Open Source	Subscription Based	Free Generations	Free Generations per month	Commercial Use
Sloyd	No	Yes	Yes	3	Yes
Soundraw	No	Yes	Yes	unlimited	Yes
Stable Audio	Yes	Yes	Yes	10	Yes
Polycam	No	Yes	Yes	unlimited	Yes
Real-ESRGAN	Yes	No	Yes	unlimited	Yes
Easy Diffusion	Yes	No	Yes	unlimited	Yes

Table 15: The action space and low and high bounds of each variable.

CHAPTER 6 Deep Reinforcement Learning inside Unreal Engine

Gathering adequate amounts of data for training agents can be proven to be difficult depending on the task at hand. Discovering information required to train agents in real world can be difficult and very expensive or cost prohibitive in many cases, requiring specialized equipment that has a chance to be damaged especially in the first stages of training while the agent learns the environment.

Modern video games can be a very detailed representation of the real world inside a virtual space which enables easy access to every possible variable that might be needed. In this chapter the focus will be on a practical application of deep reinforcement learning (DRL) using a video game and leveraging the game-world as simulation environment for gathering input data to train an agent.

(Sub-chapter 6.1) Agent Training inside Unreal Engine

Unreal engine is one of the top free-access game engines, boasting many state-of-the-art features and a very active developer community, from big studios to indie teams and many individual developers.

Although it has some game AI features, state machines and a custom implementation of behavior trees it does not include built-in deep reinforcement learning capabilities. In order to train our agent, we will use middleman approach. We extract the useful data from the game world, encode and send them outside of the engine to be processed by a deep reinforcement learning algorithm which will output a series of actions for the agent to take, which will be send back to the engine.

(Section 6.1.a) Introduction to the MIND MAKER plug-in

Inspired by OpenAI Gym [156], an API library enabling simulation and agent training inside python for general reinforcement learning problems, Aaron Krumins created an engine plug-in for enabling deployment of the same cutting-edge algorithms used in the OpenAI Gym inside the engine environment called Mind Maker [157].

An important difference between the Mind Maker and OpenAI Gym is that the latter is confined in training with old Atari Games, classic card games and mostly 2D simulation environments, although there is an option of creating custom environments, it is limited to what pure coding can achieve. Mind maker on the other hand is focusing only on the connection between the unreal engine and the desired algorithm, therefor the possible virtual environments that can be created inside Unreal are endless. Making a transition from simple games and simulations to one of the most advanced and detailed world building platforms.

Mind Maker is based on a client-server approach thus it can used natively in the same machine unreal is running on or it can be configured to connect with a remote server, getting access to a cloud-based system with more computational power when running larger models.

Furthermore, Mind Maker can be used for Multi-Agent Reinforcement Learning or MARL and work with multiple clients. It is as simple as changing the port number, inside unreal, for each agent and the training can be done simultaneously and in parallel for up to 100 agents.

(Section 6.1.b) Custom Environment for Agent Training

For the purposes of training an agent inside the unreal engine a custom environment will be created from scratch using the generative AI tools discussed in the previous chapter.

List all AI generated assets used:

- 3D models
 - 2 types of sci-fi Consoles
 - Console screen(taken from the console model)
 - Laser Gun
 - Twin Barrel Gun
 - 2 types of Sci-fi crates
 -
- Materials
 - 2 Sci-fi Panels
 - Sci-fi Panels with LED circuit board connections effect
 - Console Screen with random texture and color based on mesh world location
 - Caustic texture effect
- Sounds
 - 10 background music tracks
 - Hydraulic door opening
 - Laser sound effect
 - Shield whoosh
 - Projectile explosion
 - Shoot sound
 - Multiple explosions
 - Rolling sound

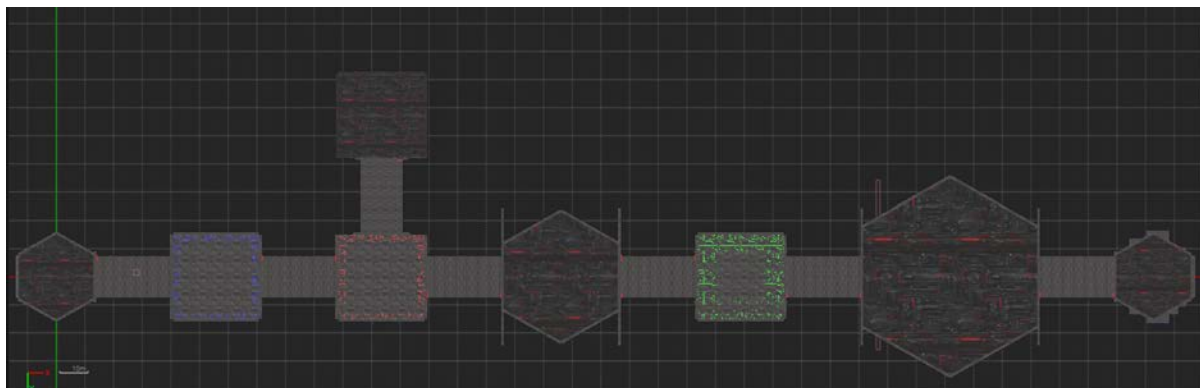


Figure 79: Unlit top-view of the level to be used as a simulation environment.



Figure 80: Screenshot 1 taken from the UE5 Simulation Environment.

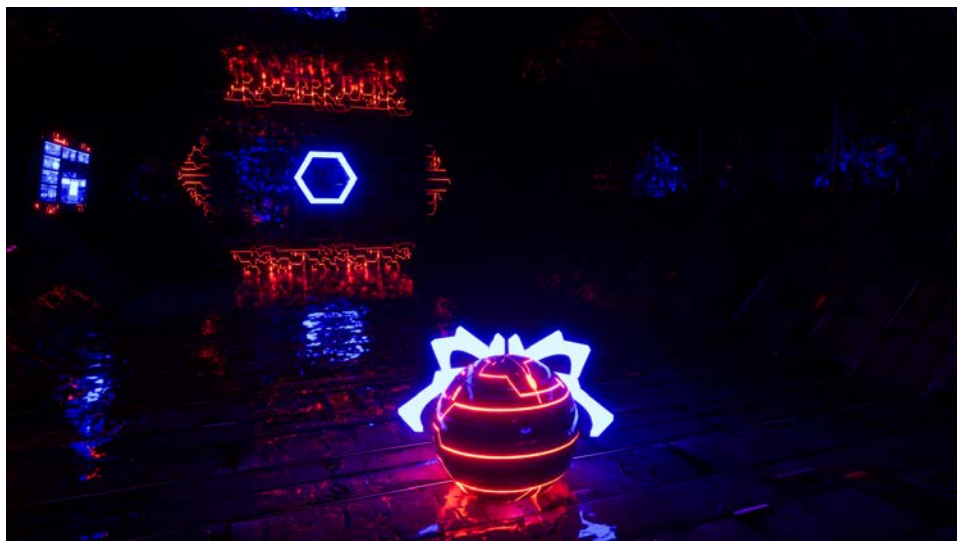


Figure 81: Screenshot 2 taken from the UE5 Simulation Environment.



Figure 82: Screenshot 3 taken from the UE5 Simulation Environment.



Figure 83: Screenshot 4 taken from the UE5 Simulation Environment.

(Section 6.1.c) Designing an Agent

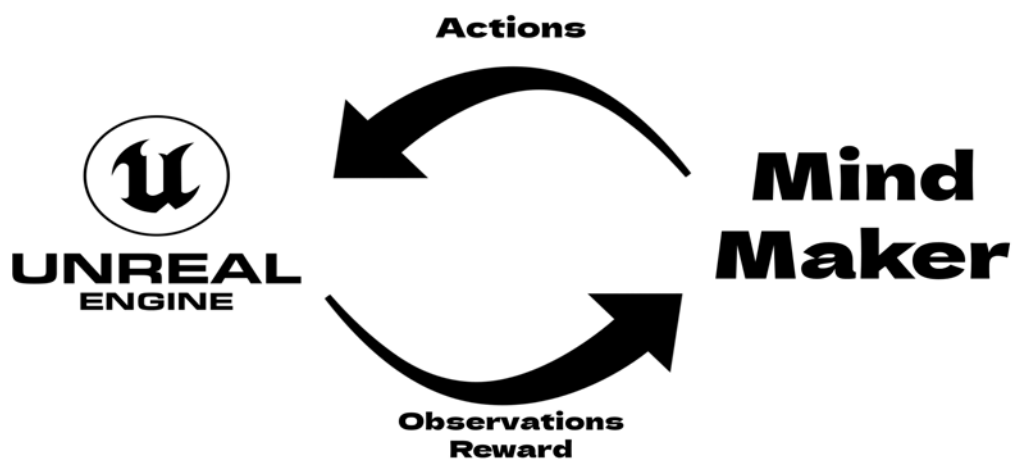


Figure 84: Agent training loop within the unreal engine environment.

As Mind Maker is based on Open AI's Gym python library it uses the same standardized input format as well as algorithms. The models supported by Mind Maker are Stable Baselines 3 implementations of popular models which include:

- Advantage Actor-Critic(A2C)

- Deep Deterministic Policy Gradient (DDPG)
- Twin Delayed DDPG (TD3)
- Deep Q Network (DQN)
- Proximal Policy Optimization (PPO)
- Soft Actor Critic (SAC)

Users can choose to use any of those models for training agents inside Unreal Engine and also have full blueprint control of each model's neural network hyperparameters.

Deep RL has three core aspects, actions, observations and a reward. The agent does one or more actions, then observes potential changes made to the environment from the actions taken and lastly a reward is calculated based on if certain conditions, set by the programmer, are met.

On the demo game created for this thesis the actions, observations and the reward function are way more complex than the simple Atari simulations.

ACTIONS

The agent's Action Space includes the number of actions the agent can make as well as their respective value ranges.

- Movement Control: Broken down to two axis horizontal and vertical.
- Camera/Gun Control: Broken down to two axis horizontal and vertical.
- Shoot Action: Binary Value for agent to shoot or not.
- Shield Action: Binary Value for agent to use shield action or not.

Movement

The game demo is designed in such a way that the movement input needs to be two variables that each can take 3 possible values. Horizontal -1 to go left, 0 to do nothing and 1 to go right, similarly in vertical -1 to go backwards, 0 to do nothing and 1 to go forward.

Camera/Gun

The two input variables for the camera control also control the gun orientation, therefore whether the camera is pointing affects where the gun is aiming. They need to be continuous variables from -1 to 1 but were limited to -0.6 to 0.6 in order to reduce the camera panning speed.

Shoot

Is a binary variable that controls if the agent should shoot or not. If it is 1 then it shoots if it is 0 then it does not.

Shield

Similarly to shoot action, shield is a binary variable but the shield mechanic is implemented differently, more specifically unlike shoot that is simulated as a single key press, shield has a toggle mechanic. Meaning that if the agent chooses 1 then the shield is turned on and to turn it off the agent must pick 1 again, 0 only means that the shield is kept to its current state regardless of whether it's enabled or not.

Action Space limitation and shape

Due to Mind Maker default implementation having restrictions on the action and observation space types, only Discrete and Box shapes can be used, lacking more appropriate Multi-Discrete and Dictionary shapes, the only option for the agent to take multiple actions is to use Box shape. Inevitably the action space has become continuous for all 6 variables.

Action	Movement Horizontal	Movement Vertical	Camera/Gun Horizontal	Camera/Gun Vertical	Shoot	Shield
Low	-1	-1	-0.6	-0.6	0	0
High	1	1	0.6	0.6	1	1

Table 16: The action space and low and high bounds of each variable.

OBSERVATIONS

The agent's Observation space, also known as state, includes all the environment variables that need to be taken into consideration after a certain set of actions is performed. Generally as a rule of thumb, every variable that will be used in the reward calculation must be included in the agent's observations, but when the agent has multiple objectives and a very complex reward function, some variables considered for the reward might hinder the training by overwhelmingly increasing the state-space.

Deciding which variables should be part of the observation/state of the agent is of at most importance during the problems' modelization. A very common mistake occurs when trying to reward the agent based on past successful actions, similar to a game's score. On model-free algorithms meaning the agents take actions in a probabilistic manner based on the current state/observation of the environment. If variables referring to the agents' past are considered when taking an action, it will cause confusion in the decision process.

Simplified example of a decision process:

Possible action: shoot/not shoot

State: enemy targeted/not targeted, can shoot/can not shoot

"if enemy targeted and can shoot if you choose action shoot then expected reward +100"

Simplified example of a decision process with wrong design:

Possible action: shoot/not shoot

State: enemy targeted/not targeted, can shoot/can not shoot, enemies killed

"if enemy targeted and can shoot and enemies killed equal X, if you choose action shoot then expected reward +100"

With the correct design, when an agent encounters the state [enemy targeted, can shoot] on a random timestep then he will have a high probability to choose to shoot due to past iterations that observed the same state and chose the action shoot resulted in a positive reward. On the

other hand, in the second example even if the agent has successfully killed some enemies he will not gain knowledge based on the previous experiences because each time the state will be different as the kill count will increase.

When it is needed to reward the agent for performing a desired action we can keep a local variable in the reward function to give rewards but not include it in the state, thus resulting in an increased probability of selecting that action when encountering this state.

The observation space can include:

Combat State

Binary variable that shows if Agent is in combat or not. The agent constantly senses if it has enemies in line of sight and sets it to 1 if enemies found and to 0 otherwise.

Can shoot

Binary variable to control shooting interval after each shot. The agent makes actions a lot of times per second, this variable is used to deter him from spamming the shoot action constantly.

Hit to Enemy Distance

Real variable to control how good or bad the agent's aim is. Constantly a calculation is made for predicting the projectile trajectory if a bullet is fired in that moment, also when the agent's state becomes 1, meaning its in combat, the closest in line-of-sight enemy becomes the currently focused enemy. Hit to Enemy Distance variable is the Euclidean distance between the predicted hit location and the focused enemy location.

Enemy Targeted

Binary variable that shows if an enemy is on target. When calculating the projectile trajectory, we can get access to the item the projectile will collide with, thus we check if the said item is an enemy and set the Enemy Targeted Variable to 1.

Health Points

Variable that shows how much damage the agent can take from enemies.

Bullet Points

Variable that shows how many bullets the agent has remaining.

Shield Energy Points

Variable to show the damage the agent can block with shield.

Can use Shield

Binary variable to deter agent from spamming the shield action. Works similar to "Can Shoot" variable.

Coordinate

Every object has a vector representing its coordinates in the 3D space, xyz can be included separately in the Observation or a subset of them if one does not change during playing.

Goal Distance

Real variable to keep track of the agent's distance from the current

VARIABLES TO AVOID:

Hits on Target

Variable that shows the agent's successful hits on enemies. When an agent shot projectile inflicts damage to an enemy it rises the count of the hits on target variable.

Kill Count

Variable that shows how many enemies the agent killed. When an enemy dies it rises the count of the kill count variable.

Variable	Low	High
State	0	1
Can Shoot	0	1
Hit-to-Enemy Distance	0	999999
Enemy Targeted	0	1
Bullet Points	0	100
Shield Points	0	50
Can Shield	0	1
Goal Distance	0	999999

Table 17: The observation space as well as low and high bounds of each variable. Some variables have no actual upper bound, arbitrary 999999 value is set.

REWARD

After the agent performs the chosen action and makes observations of the changes to the game environment variables, the last thing to do is to see if the actions resulted in positive or negative outcomes by running the reward function to calculate the reward received for those actions.

The Reward function is of at most importance as it steers the agent's behavior towards the designer's expectations. By setting action-observation conditions for the agent to meet, the designer can reward or punish the agent based on the desired behavior he wants him to achieve.

When designing a reward function careful consideration must be given on detecting and avoiding trapping states that can steer the agent in behavior in which he can maximize rewards without progressing towards a desired objective.

Reward Types

There are many types of methods to give rewards to an agent but in this implementation the focus will be on dense and sparse rewards.

Using dense rewards means that the agent will be rewarded positively or negatively according to a set of conditions in each timestep. On the other hand, sparse rewards are given when an agent completes an objective.

In this custom designed reward function a combination of sparse and dense rewards will be implemented to fulfill the different types of behavior desired. Dense rewards will be used

for the main movement and aiming and sparse rewards for finetuning combat behavior such as successful hits on an enemy.

Reward Conditions

Following a rigorous design process as well as a lot of trial and error, the reward function to be used for training the demo agent can include one or more of these conditions:

- Punishment on health damage/ reward collecting health orbs, equal to the health lost or restored respectively.
- Punishment on shield damage/ reward on regenerating shield, equal to half of the shield points lost or regenerated to encourage the agent to take damage to shield instead to health.
- Punishment on continuous shield spamming.
- Punishment on out of combat shooting.
- Punishment on out of combat use of shield
- Punishment on wrong shoot timing, deter shoot spamming during intervals.
- Punishment on shield spamming during shield turns on or off.
- Reward correct shoot timing when enemy is targeted.
- Reward on aiming improvement.
- Reward heading towards the current goal location.
- Reward reaching the goal line.

Due to the limitation of the Mind Maker plug-in for multiple actions to Box shape which only supports continuous variables we need a way to choose only between the desired actions. To achieve that we round the input action variable before the agent makes the action inside the simulation environment.

- Round Movement horizontal to -1 if input action $[-1, -0.3)$, round to 0.0 when $[-0.3, 0.3)$ and round to 1.0 when $[0.3, 1]$.
- Round Movement vertical to -1 if input action $[-1, -0.3)$, round to 0.0 when $[-0.3, 0.3)$ and round to 1.0 when $[0.3, 1]$.
- Round action shoot to 0 when input action is $[0, 0.5)$ and to 1 when $[0.5, 1]$.
- Round action shoot to 0 when input action is $[0, 0.5)$ and to 1 when $[0.5, 1]$.

Pseudocode representing the reward function based on the conditions listed above:

#punish health damage

```
Reward = Reward + health_points - previous_health_points
previous_health_points = health_points
```

#punish shield damage

```
Reward = Reward + (shield_energy_points - previous_shield_energy_points)
previous_shield_energy_points = shield_energy_points
```

#punish shield spamming

```
if can_use_shield == 0 and_action_shield == 1:
    Reward = Reward - 1
```

#punish shoot spamming

```
if can_shoot == 0 and action_shoot == 1:
    Reward = Reward - 1
```

if combat state==0 # exploring

#reward heading to goal (punishment on moving away)

```
if previous_goal_distance - goal_distance < 0
    Reward = Reward + (previous_goal_distance - goal_distance)
else:
    Reward = Reward + (previous_goal_distance - goal_distance)
    previous_goal_distance = goal_distance
```

#punish out of combat shooting

```
if action_shoot == 1:
    Reward = Reward - 100
```

#punish out of combat use of shield

```
if action_shield == 1:
    Reward = Reward - 100
```

elif combat state == 1: # combat

#reward aiming improvement

```

if enemy_targeted == 0:
    Reward = Reward +
        (previous_hit_to_enemy_distance - hit_to_enemy_distance)
    previous_hit_to_enemy_distance = hit_to_enemy_distance

#punish shooting when enemy not targeted

if action_shoot == 1:
    Reward = Reward - 100

#reward shoot when enemy targeted

elif enemy_targeted==1:

    If can_shoot==1 and action_shoot==1
        Reward = Reward + 50

```

Training Environment

Most DRL agent training simulations from Open AI Gym are done on a single level with only one objective, "get on top of the hill", "land the spaceship upright", "balance the pendulum" etc. To train the agent in this demo game is magnitudes more difficult in terms of complexity as the agent has multiple mechanics to be trained on:

- Finetune controls
- Path finding
- Aiming
- Shooting
- Avoiding damage

That's why the training level was designed in distinct rooms which step by step guide the agent to the next goal. The agent will be trained in a level specifically designed towards an incrementally increasing difficulty in both exploration and combat scenarios. It will include six different rooms each one with a specific training objective in order to steer the agent to a desired behavior.

Goals

1. Enter room 1.
1. Find path and enter room 2.
2. Find path between lasers to collect battery from room 3.

3. Go back to room 2 and unlock the door with the collected battery.
4. Go to room 5 after fight in room 4.
5. Go to room 6 after pathfinding in room 5, with intentional local maxima path.
6. Go to last goal by climbing a ramp with incline.

Starting Room

The room in which the agent is spawned in the beginning of the training. Goal 1 is to go from the starting room to Room 1, there are no obstacles, and the objective is for the agent to learn the basic controls as well as to head towards a specific goal location.

Room 1

In this room the target is for the agent to learn how to navigate and find a path to reach the second room. Two possible paths are given as bridges that allow the agent to cross the gap.

Room 2

A transition type of room with a locked door that the agent needs to go to a different room to find the key. It doesn't introduce a new objective as the agent only sees a goal location and doesn't have understanding of the locked door mechanic, it is a harder variation of the previous room that the agent needs to navigate in order to go to Room 3.

Room 3

This room has the key for the locked door (Battery), that is protected by an array of lasers. The agent needs to find the correct path between the lasers that inflict contact damage to his health points each second. The agent in this room could learn punishment mitigation with the use of the shield and ultimately find the path which will result in him not taking damage.

Room 4

First contact with a stationary enemy turret. Agent must defeat the enemy to progress to the next goal. Objective is learning how to aim, shoot, avoid enemy hits and use shield. The agent gets a handicap at the beginning to not overwhelm him, the turret is deactivated for the first 5 times the room is repeated, so the agent can learn to correctly aim the gun, afterwards the enemy starts shooting projectiles at the agent.

Room 5

Same bridge navigation type of level but with an introduced path that doesn't reach the other side. A purposely introduced local maxima for the agent to learn to navigate out of.

Room 6

The final room is an advanced combat room, the room includes a new type of enemy a laser turret, which has more health points and does contact damage per second.

Training Level Completed Room

After defeating the enemy in room 6, the agent must navigate to the final room by passing over a low friction ramp which has a big incline. If the agent does not have the proper velocity at the beginning or if he loses velocity during the climb, the low friction will not allow him to proceed and will fall back. To complete the room the agent must reach the top of the ramp.

Training Loop

Each goal must be completed 10 times before the agent can progress to the next one. After reaching the final goal the 10th time, the agent starts the level from the beginning.

Training Method

Due to the complexity of the demo game created, the number of training steps the agent will need in order to learn the game is unknown. We couldn't try to make an educated guess just by looking at each individual problem (path finding, aiming etc.) and sum our prediction to derive the final number because combining all of them makes the problem exponentially more difficult.

Due to hardware and time limitations we will implement multiple agents to test each training objective separately as the goal of the thesis is not to produce a complex agent but to prove the simulation capabilities of the unreal engine environment.

Someone can still design big multi-objective agent to be trained in this environment by combining all the objectives together in a large and complex reward function.

Another useful feature of the Mind Maker plugin is Saving and Loading Models, which allows us to locally save the trained model and load it for prediction or more importantly for continuous training which is ideal for our training application.

(Sub-chapter 6.2) Deep RL Algorithm Chosen for Agent Training

The MindMaker tool gives us access to many different algorithms to use for training Agents, between all the available ones Proximal Policy Optimization-PPO was chosen due to its compatibility with the training objectives and capability of handling continues variable types.

Characteristics of PPO:

- Model-free
- Gradient based
- On-Policy
- Actor-Critic
- Policy Clipping

The basic idea behind PPO:

- The **actor** calculates a probability distribution for the next action the agent should take based on the current state/observation.
- The **critic** estimates an expected cumulative reward based on the current state/observation.

The goal of PPO is for the actor to choose actions which will be more likely to result in a higher reward than expected by the critic. In order to achieve that, a metric value called "Advantage" is used in the algorithm, which measures how wrong the critic estimation was for any given state. If the advantage is negative then the action resulted in a lower cumulative reward than the critic expected thus we make that action have a lower probability of being chosen. On

the other hand if the estimated advantage is positive then it means that the action resulted in a higher reward than expected, thus we make it have a higher probability of being chosen.

To have good advantage estimation, a critic that can predict as accurately as possible the value of any given state. The Mean Square Error of the estimated and the observed cumulative rewards is implemented as the critic's loss function.

PPO training loop:

```
int iterations= Training_episodes / N_steps
for iteration in range(iterations):
    for actor in range(x):
        #Run policy for N_steps
        #calculate advantage for each step

    #Update policy
```

(Sub-chapter 6.3) Experiments, Training Results and Observations

In this section the results of the agent training inside the unreal engine will be presented. The MindMaker plugin allows for saving tensorboard logs after each training session.

All the experimental agents will be designed and trained using the Proximal Policy Optimization (PPO) algorithm using continuous training method discussed previously.

The hyperparameters of the algorithm will be kept the same for each of the experiments for simplicity and to assure a degree of consistency between all the agents, even if they might be not optimal for all scenarios.

Policy	MlpPolicy
Learning rate	0.00025
N steps	2048
Batch size	32
N epochs	4
Gamma	0.99
Gae lambda	0.0
Clip range	0.2
Clip range vf	None
Normalize advantage	False
Ent coef	0.01
Vf coef	0.5
Max grad norm	0.5
Use sde	False

Sde sample freq	0
Target kl	None
Tensorboard log	Filepath
Create eval env	False
Verbose	1
Seed	0
Device	Auto
Init setup model	True
Activation function	Th.nn.Tanh
Network arch	[512,512]

Table 18: Custom hyperparameters used for the PPO algorithm.

Due to technical challenges and hardware unavailability, we ended up breaking up the multi-objective reward function and focus on training multiple smaller agents experimentally. The setup is a personal computer with an i5 10400f CPU, 16gb ram and 3060ti GPU, which is not nearly capable of achieving significant results in a realistic time frame. Nevertheless, these experimental attempts showcase greatly the DRL simulation capabilities of a game development environment such as unreal engine.

(Section 6.3.a) Experiment 1- Movement and camera control

The first experiment will call for the agent to handle 4 different inputs for movement and camera control as they were designed as a linked system in the demo game. The axis for the ball movement are not static with the world but are rotating based on the camera's forward vector, that means that forward is where the camera is facing at.

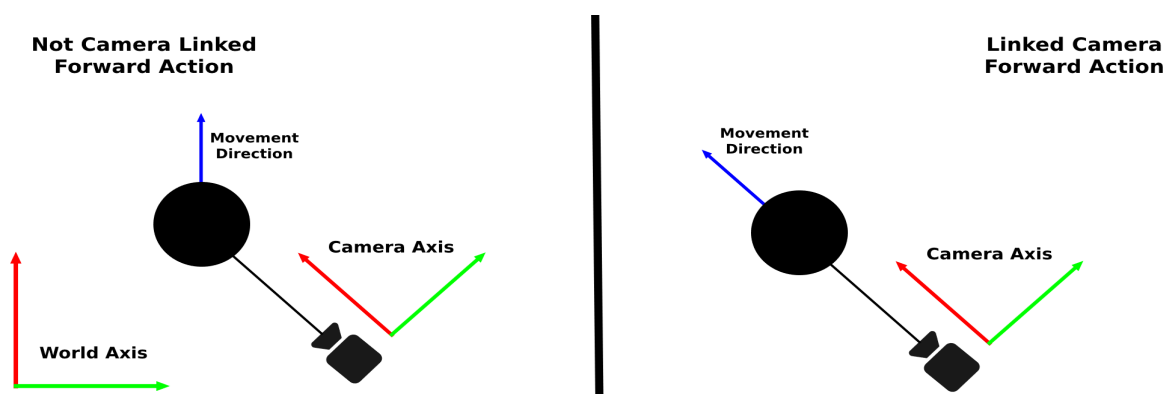


Figure 85: Demonstration of the movement-camera linked system.

Action	Movement Horizontal	Movement Vertical	Camera/Gun Horizontal	Camera/Gun Vertical
Low	- 1(backwards)	-1(left)	-0.6(look left)	-0.6(Look down)
High	1(forwards)	1(right)	0.6(look right)	0.6(look up)

Table 19: Experiment 1 action space and low and high bounds of each variable.

The observation space will only be the Euclidean Distance between the agent's location and goal location.

Observation	Distance to goal location
Low	0
High	10000

Table 20: Experiment 1 observation space/ State with low and high bounds.

Reward function that the agent will try to maximize:

```
float previews_goal_distance=goal_distance #initialize in the first step

#in each step calculate the reward

if previous_goal_distance - goal_distance< 0
    Reward = Reward + (previous_goal_distance - goal_distance)*x
else:
    Reward = Reward + (previous_goal_distance - goal_distance)

previous_goal_distance = goal_distance
```

In this reward function the objective is to get a positive outcome from the previous goal distance minus the current goal distance, that will only occur when the agent selects an action that will make the current goal distance smaller in each step which will make him head towards the goal location. The x variable is a multiplier for the negative reward, in this case it was kept at 1 and does not interfere, but it can be made larger to increase the negative reward when moving away from the objective.

An important observation in this first experiment is that the agent was very confused in certain spots in the room, it was determined that the agent is sensitive in distance symmetric positions. Due to the fact that the agent has only a distance observation, he makes associations between distance measurements and actions that will result in a higher expected reward, but if the same state can be observed in two or more different locations in the level it results in a confused agent.

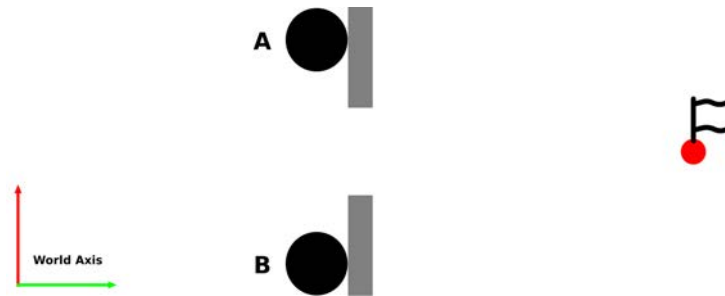


Figure 86: The symmetric distance problem. A and B are the same distance away from the goal but in different locations

(Section 6.3.b) Experiment 2- Improved Movement and camera control

The second experiment will try to clear the confusion in the agent's decision selection by incorporating the x and y coordinates (z does not change so it was not included) in order to create distinct action-Observation associations based on the agent's current location as well as distance from the goal.

Another important problem the agent faces is the linked camera-movement system, meaning that the camera rotation around the z axis affects the result of any given agent movement action. as demonstrated by the figure below the agent can be on state with the same distance, identical xy coordinates but depending on the camera rotation the input action will result in a different outcome.

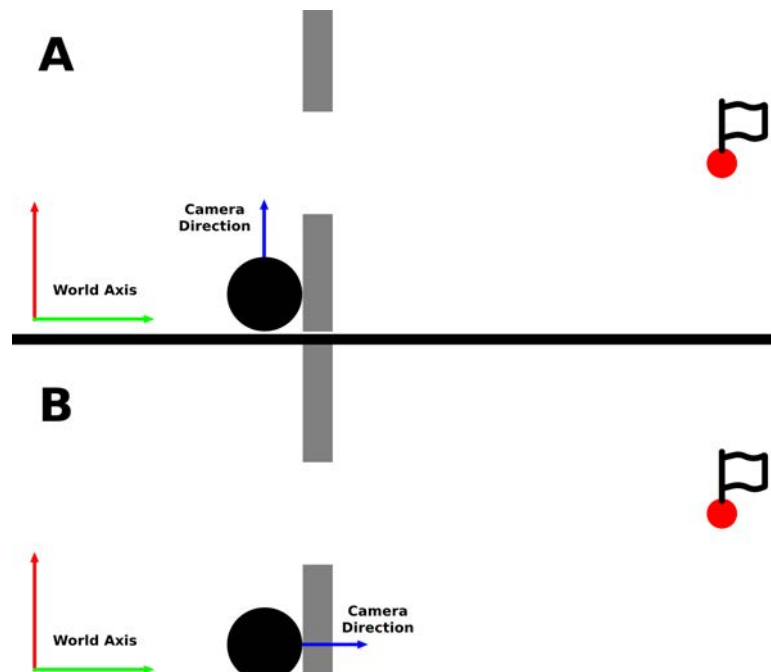


Figure 87: The camera Z rotation problem. A and B are in the same location but have different camera Z rotation

In example A from the figure above a sideways left action will result in a negative reward as the agent will increase the distance from the goal, on the other hand the same action will produce a favorable out come in the example state B. It is necessary to include the camera z rotation in order to help the agent select the correct action for each circumstance.

The action space will remain the same as the first experiment, but the observation space will change as follows.

Observation	Distance to goal location	x coordinate	y coordinate	Camera Z rotation
Low	0	-500	-2500	-180
High	10000	9000	2500	180

Table 21: Experiment 2 observation space/ State with example of low and high bounds.

Reward function that the agent will try to maximize will remain the same as the previous experiment, as the objective does not change, only the observation space is expanded in order to provide more accurate information to the agent and avoid confusing the algorithm with duplicate states that require different actions to be selected.

Additional improvements:

After multiple experimental runs of this setup the agent was observed spending too much time stuck on the walls and especially in the corners of the room, in some cases resulting in the algorithm getting trapped in the corner. In order to get the agent unstuck from the walls and promote more exploration of the room, two measures were taken:

- Adding another face to the hexagon room and making it an octagon, removing the problematic corners
- Implementing a "Don't touch the wall" mechanic in the game. If the agent touches the wall he is returned to the starting position and receives a 1000 points punishment to the reward
- All Observation variables were rounded to the nearest integer, meaning reducing the location accuracy of the agent to centimeters and the rotation accuracy to non-fractional degrees. This accuracy reduction is negligible compared to the state space reduction it gives us.

A huge improvement in the agent's exploration process of the room was observed as well as better "runs" to the desired goal location.

(Section 6.3.c) Experiment 3- Stationary Aiming

The third experiment will focus on training an agent to aim at an enemy. For simplicity, both the agent and the enemy will be stationary and the enemy will not try to damage the agent. The

agent's objective is similar to the movement as he tries to minimize the distance between where he is aiming and the target enemy. The aiming system is linked with the camera rotation, has 360 degrees of freedom in the horizontal axis but is limited in the vertical axis from 20 to -20 degrees.

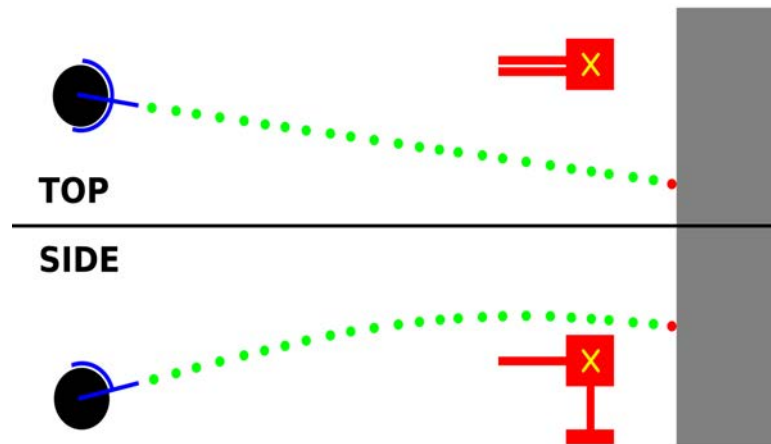


Figure 88: Agent aiming diagram. The red dot represents the predicted hit location and yellow X the target location.

At every timestep a calculation of the location that the projectile will make if shot is made and the Euclidean distance between the aimed location and the current enemy's location is computed. The agent is rewarded for each centimeter he aims closer to the enemy and is punished when that distance becomes larger.

Action	Camera/Gun Horizontal	Camera/Gun Vertical
Low	-0.6(look left)	-0.6(Look down)
High	0.6(look right)	0.6(look up)

Table 22: Experiment 3 action space and low and high bounds of each variable.

The observation space will include the Euclidean Distance between the current aimed location and enemy's location, as well as the xyz coordinates for the aimed location.

Observation	Hit to enemy Distance	x coordinate	y coordinate	z coordinate
Low	0	15000	-2500	-100
High	10000	19000	2500	1500

Table 23: Experiment 3 observation space/ State with low and high bounds.

Reward function that the agent will try to maximize:

```
float previews_distance=current_distance #initialize in the first step

#in each step calculate the reward
Reward = Reward + (previous_distance - current_distance)
previous_distance = current_distance
```

(Section 6.3.d) Experiment 4- Stationary Aiming and Shooting

The fourth experiment will focus on extending the third one by including a shooting mechanic on top of the aiming. The agent's objective is more complex this time as he tries to aim and simultaneously decide when to shoot. The agent's action space has a "shoot" action which when chosen a projectile will be fired, this action has a 0.5 second cooldown period, if the agent tries to spam the action during the cooldown time he is punished.

Another variable the agent takes into consideration is if the enemy is targeted, that is a binary variable that is 0 when the calculated projectile hit location is not on top of the enemy and becomes 1 when this location is on the enemy even if it does not match the focused location the agent tries to aim at. If the enemy decides to shoot when the enemy is targeted then he is rewarded. The purpose of this variable is to increase the number of states in which the agent can be rewarded for correct shooting.

Action	Camera/Gun Horizontal	Camera/Gun Vertical	Shoot
Low	-0.6(look left)	-0.6(Look down)	0(No shoot)
High	0.6(look right)	0.6(look up)	1(shoot)

Table 24: Experiment 4 action space and low and high bounds of each variable.

The observation space will include be the Euclidean Distance between the current aimed location and enemy's location, as well as the xyz coordinates for the aimed location. Additionally a "Can Shoot" variable for the cooldown between shots and a "enemy targeted" variable to show is enemy is targeted or not at any given observation/state.

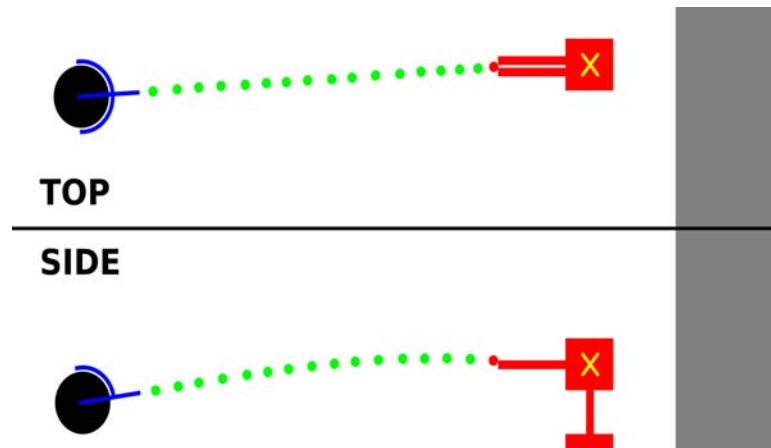


Figure 89: Agent aiming on target, despite the red dot not being on the yellow X.

Adding a second mechanic to the agent no matter how simple it might be, both the action and Observation/state spaces are increased. In this case the it is hard to judge how much the action space will be increased because we use a continuous variable which gets rounded in the simulation, but if we assume the algorithm will choose to use only the high and low bounds similar to the other 3 experiments then the action space will be increased by a factor of 2. As for the Observation/state space it will be multiplied by a factor 2 for each one of the new variables, becoming 4 times larger due to the shooting mechanic.

Observation	Hit to enemy Distance	x coordinate	y coordinate	z coordinate	Can Shoot	Enemy Targeted
Low	0	15000	-2500	-100	0	0
High	10000	19000	2500	1500	1	1

Table 25: Experiment 4 observation space/ State with low and high bounds.

Reward function that the agent will try to maximize:

```

if can shoot == 0 and action shoot ==1
    Reward = Reward - 100

float previews_distance=distance #initialize in the first step

#in each step calculate the reward

if previous_goal_distance - goal_distance< 0
    Reward = Reward + (previous_goal_distance - goal_distance)*x
else:
    Reward = Reward + (previous_goal_distance - goal_distance)

```

```

previous_goal_distance = goal_distance

if enemy targeted==1 and action shoot ==1
    Reward = Reward + 1000
elif enemy targeted==0 and action shoot ==1
    Reward = Reward - 500

```

(Section 6.3.e) Training Observations and Scalar Curves

1. Dense rewards were proven to be the best in steering the agent towards the desired behavior.
2. Sparse rewards either positive or negative needed to be very large compared to the dense ones, in order to have any significant impact in the agent's behavior.
3. The PPO algorithm has a tendency to choose the upper or lower bound of continuous input variables.
4. Due to PPO being an on-policy algorithm the agent can be observed to take the "safer" long way round to reach the goal location in the path finding objective.
5. When getting "stuck" for multiple timesteps the agent did "track-back" to a state that had confidence for the next action even if it meant getting negative reward to reach it.

Experiment	Timesteps
1 dist	276.5K
1 dist+loc	251.9K
1 dist+loc+rot	251.9K
2	7.225million
3	3.411million
4	2.207million

Table 26: Total timesteps of continuous training per experiment. sessions.

Proposed future improvements:

The model would need to be trained using Multi-Agent training on a distributed system in order to produce an agent with the desired finetuned behavior within a realistic training period. By separating the behavior in multiple agents each one will have an easier objective to be trained on. The best separation point in this agent implementation would be around the state variable, one agent could handle the path finding and another the combat portion of the behavior. The Mind-Maker plug-in is out of the box capable for remote training, it is easy to set up by configuring the SocketIO component in UE accordingly.

The MindMaker source code must be modified to include Multi-Discrete action space and avoid rounding the continuous variables. Choosing between discrete values reduces the agent's action space significantly.

Expand the MindMaker plug-in to support imitation learning in combination with the SB3 reinforcement learning, to leverage human player input data as well as the reward function to improve the learning rates of the model. A similar approach to the DeepMimic paper [158] which focused on complex humanoid physics-based movements.

PPO evaluation variables:

value loss: mean loss of the value function updates. Increases while the reward increases and should decrease when the reward becomes stable.

std: learned standard deviation adds noisiness in draws during training to help the agent explore new promising actions.

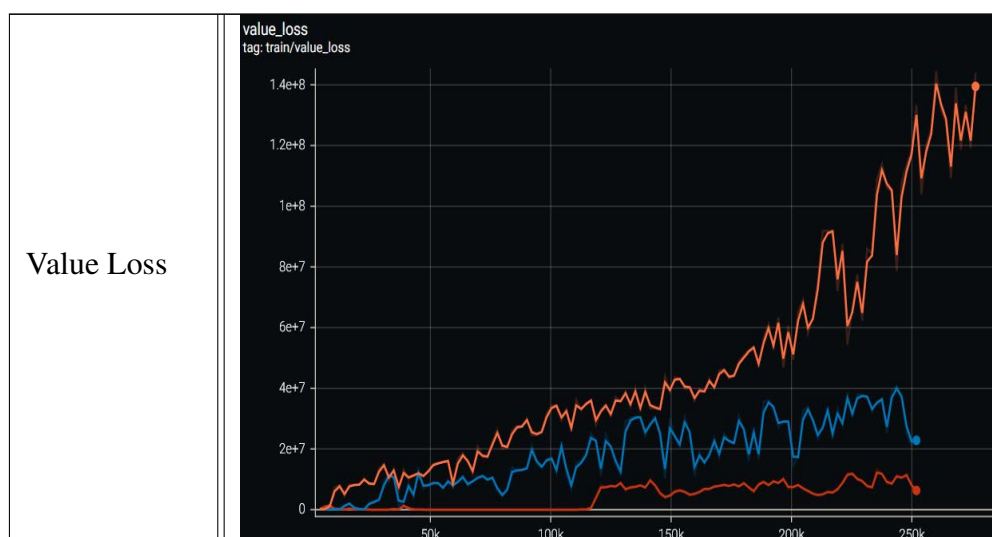
entropy loss: measurement of the noisiness of the action draws. Legacy metric from TRPO.

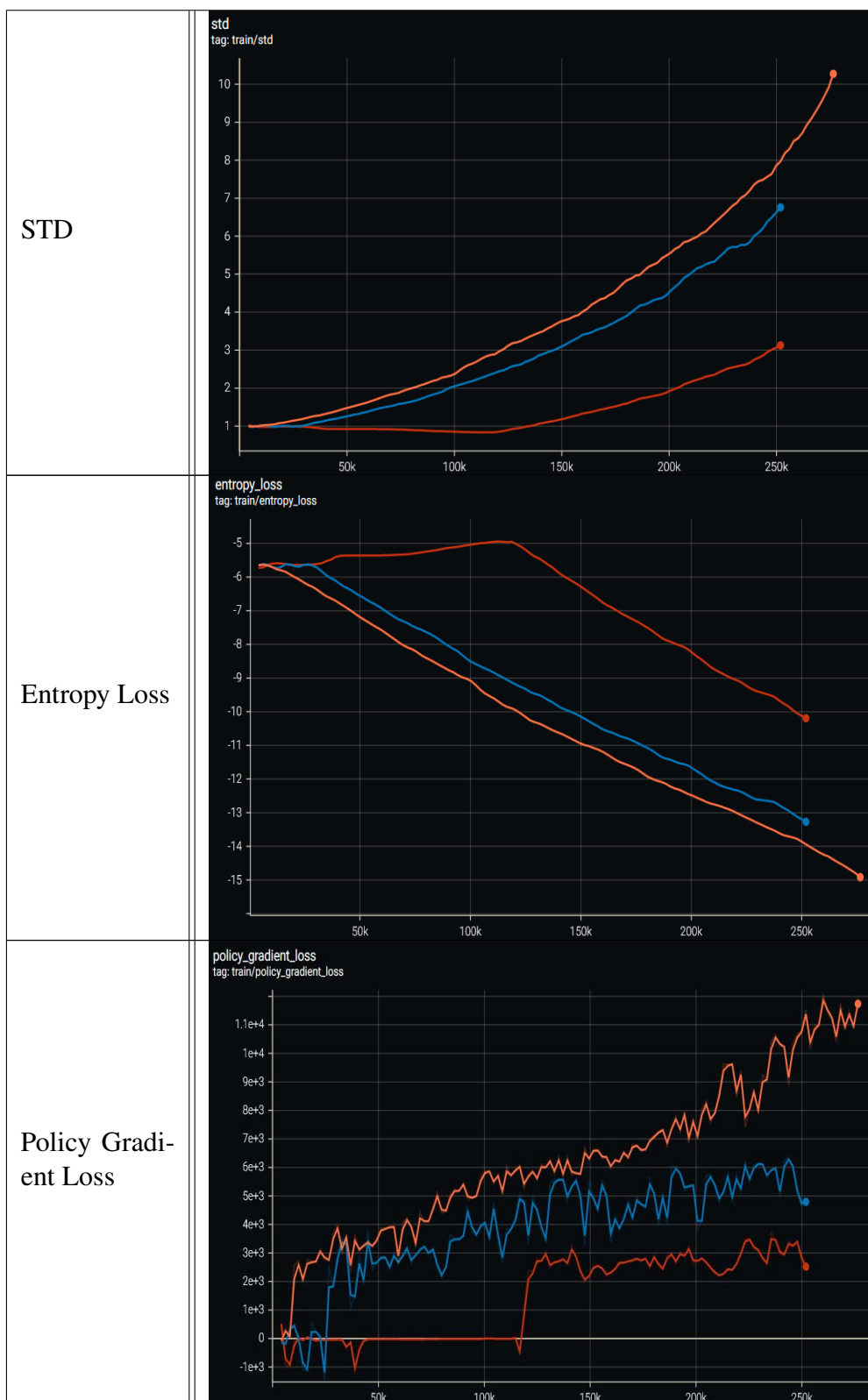
explained variance: calculates how good the value function is as predictor for reward. When $ev < 0$ the value function is worse than predicting nothing.

approximate kl: indicates how different the new vs the old policy.

clip fraction: indicates how different the new vs the old policy.

policy gradient loss: loss for the policy gradient part of the algorithm.





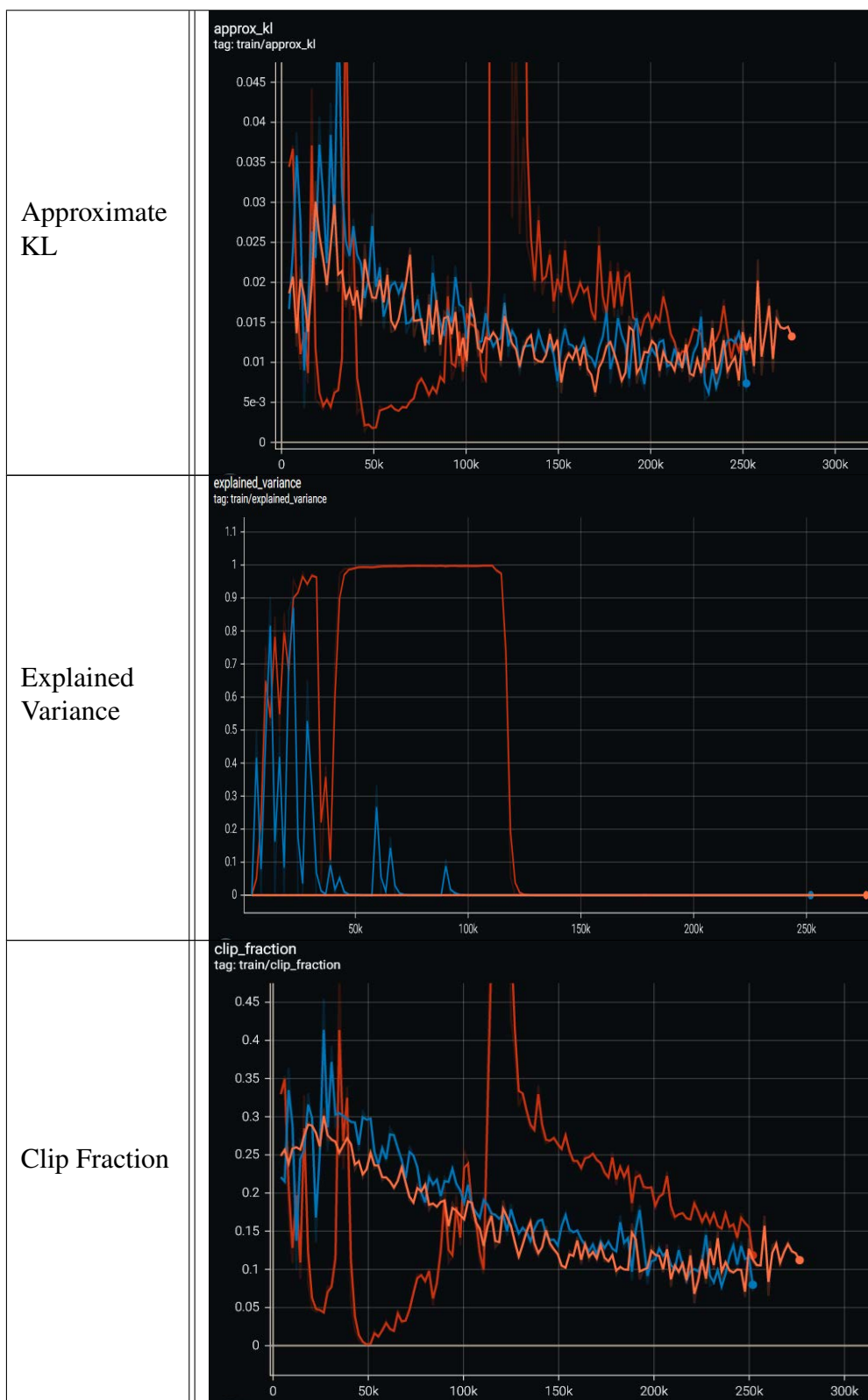


Table 27: Movement experiment 1 scalar curves. Three runs of the first experiment with different Observation spaces. Orange is a run with distance only, Blue is a run with distance + xy location, Dark Orange is a run with distance + xy location + camera rotation.

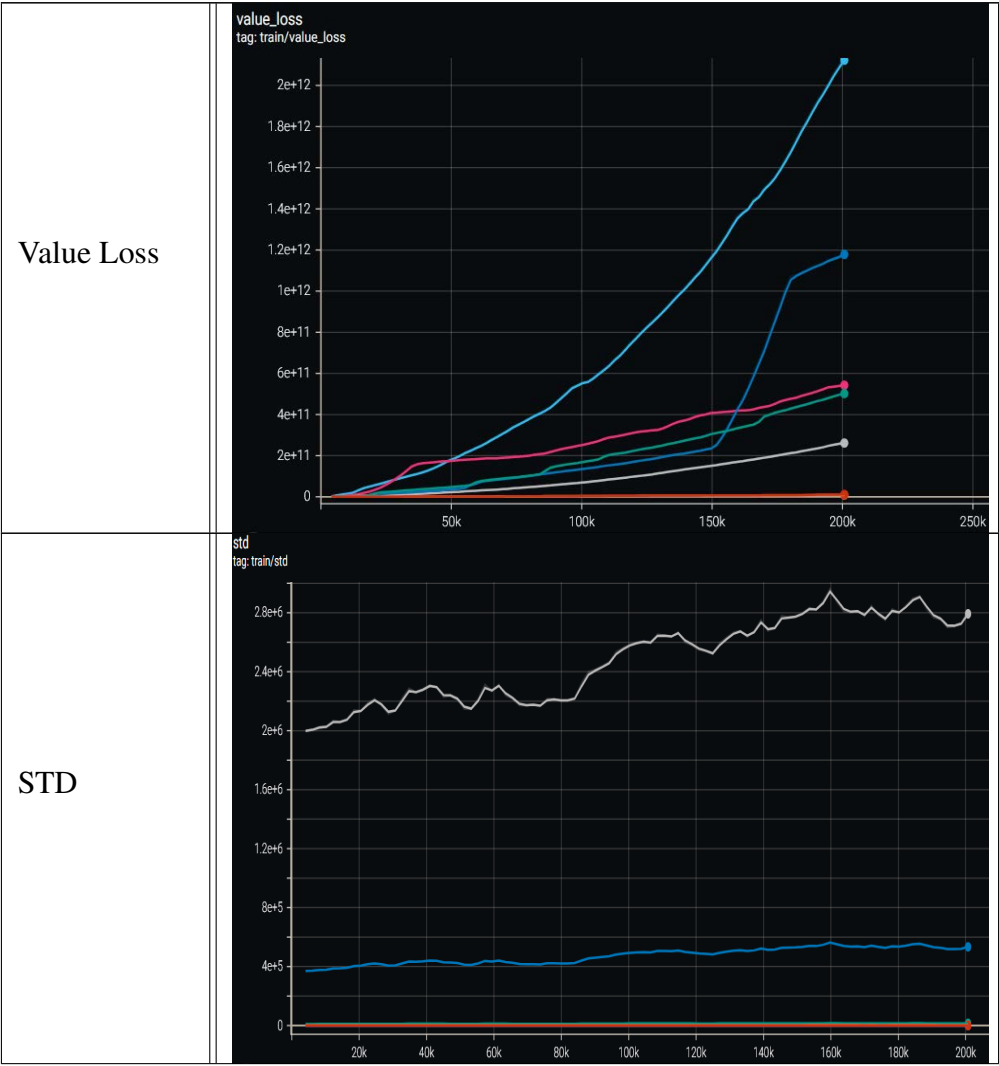
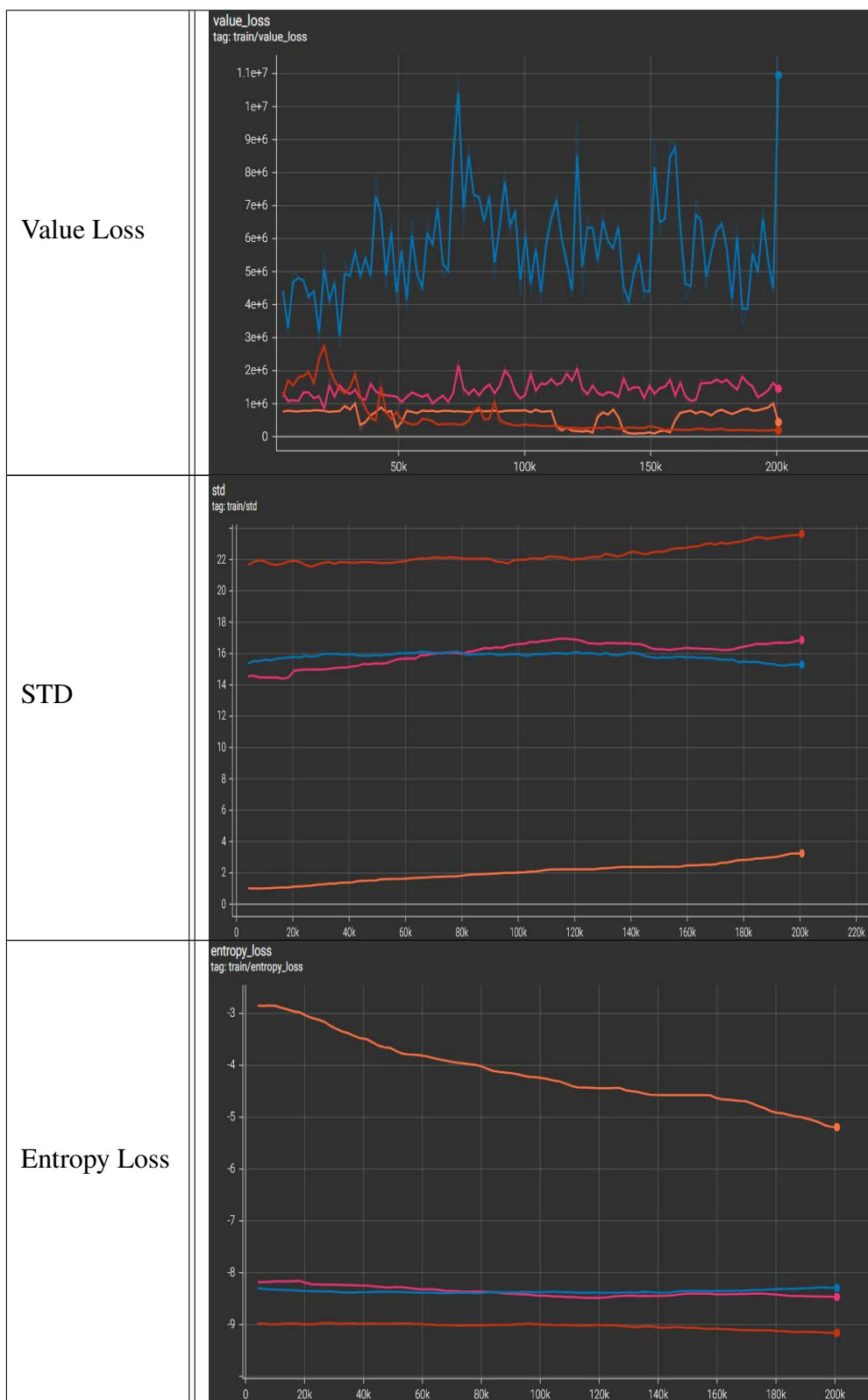






Table 28: Movement experiment 2 scalar curves. Light Blue is session 1, Pink is session 6, Dark Orange is session 11, Orange is session 16, Dark Green is session 21, Blue is session 31, White is session 36





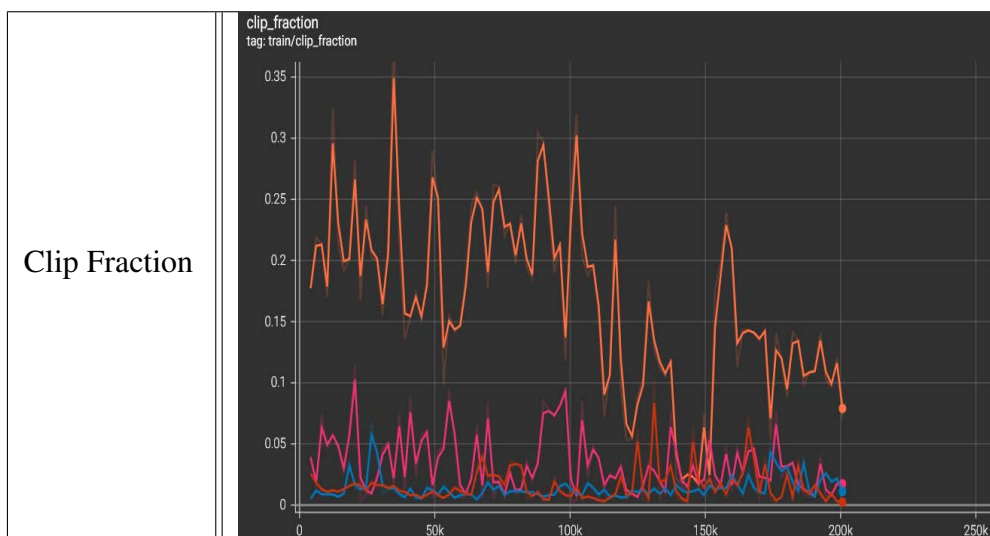
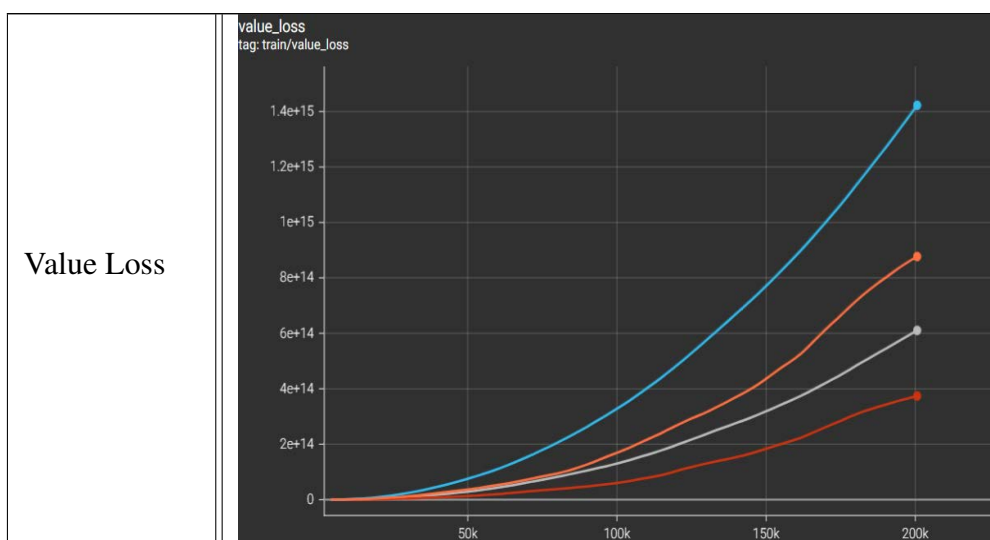
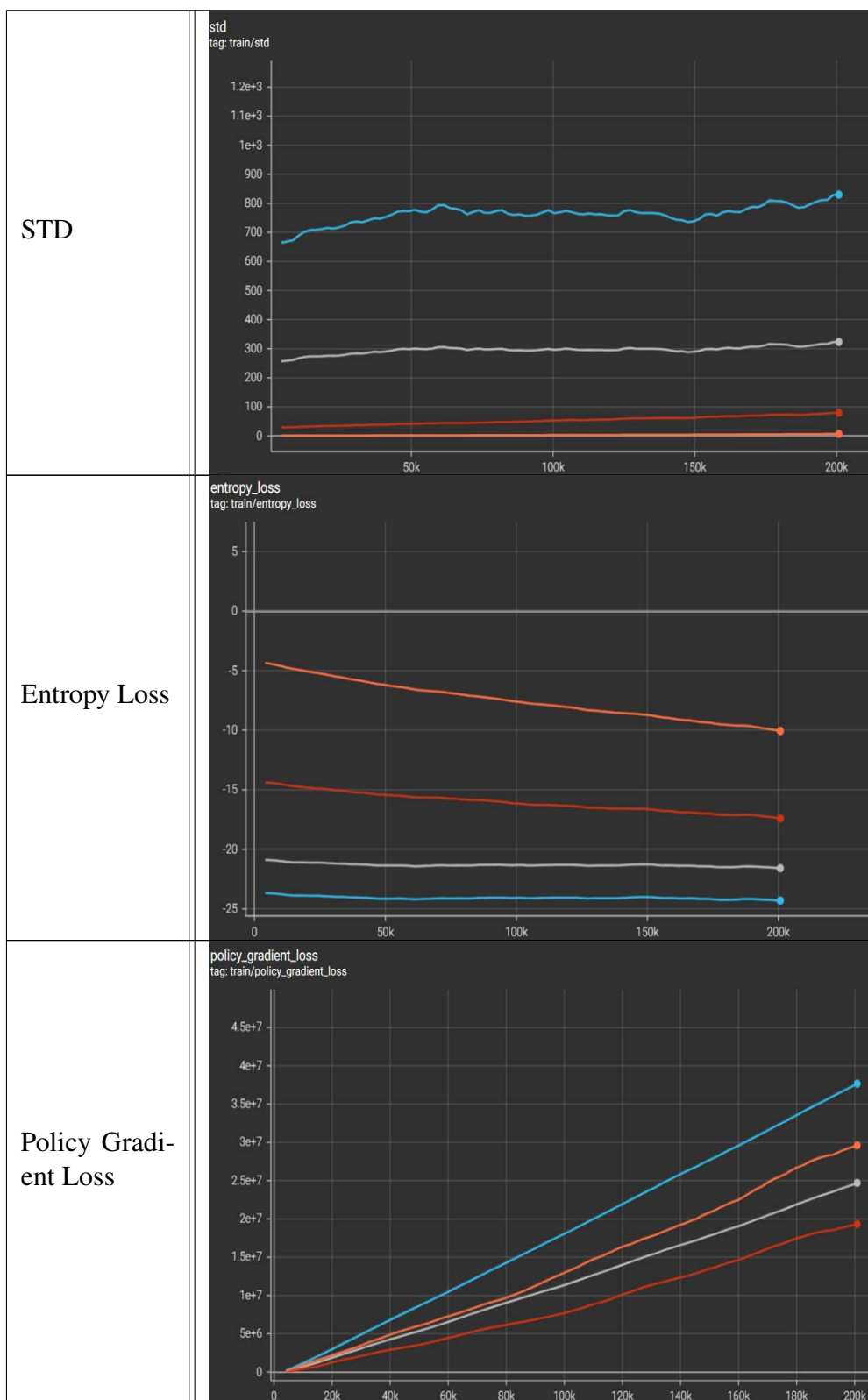


Table 29: Aiming experiment 3 scalar curves. Orange is session 1, Pink is session 5, Dark Orange is session 10, Blue is session 16





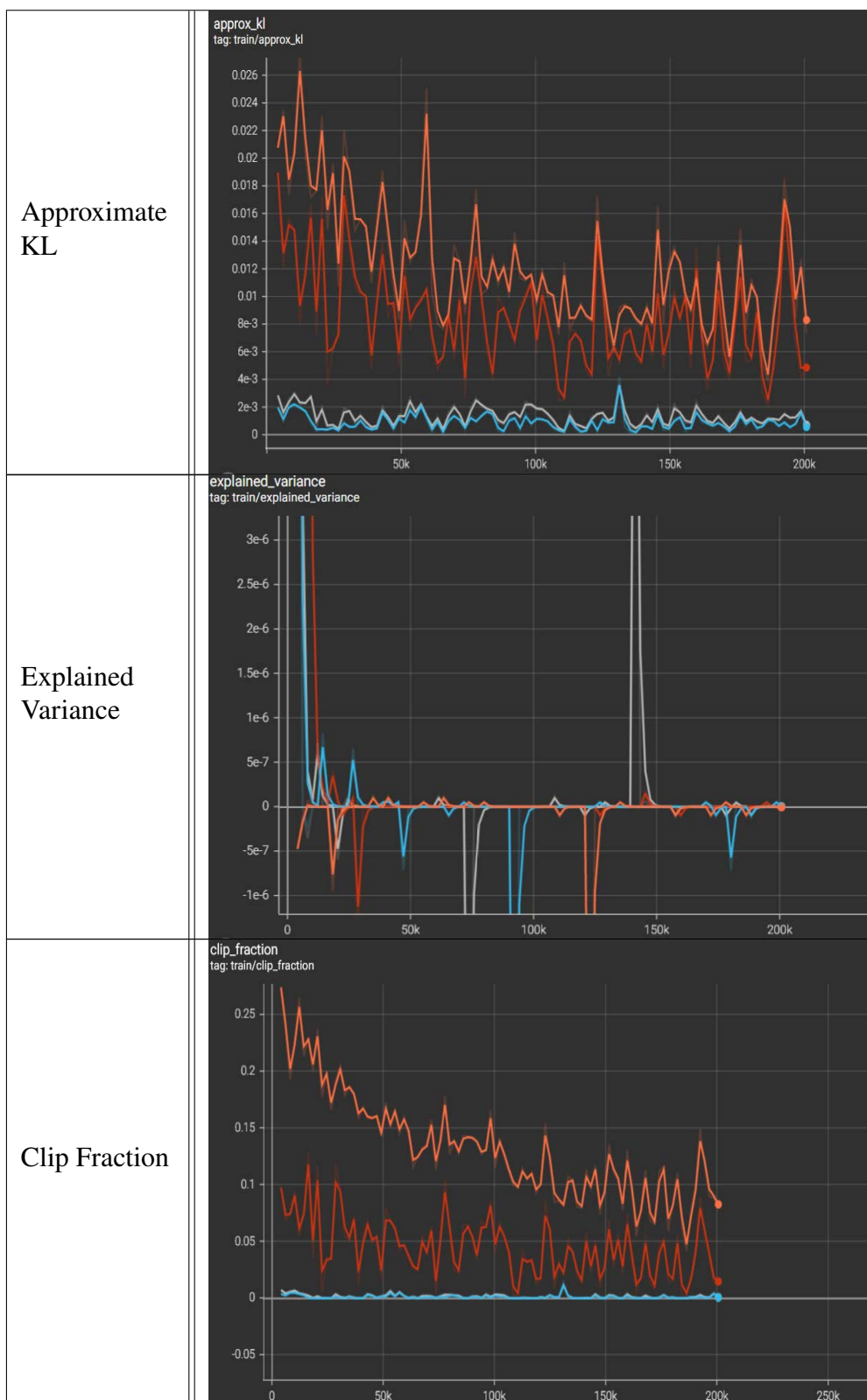


Table 30: Aiming+shooting experiment 4 scalar curves. Orange is session 1, Dark Orange is session 3, White is session 7, Blue is session 11

(Sub-chapter 6.4) Adapting Unreal Engine Environment for Real-World Applications Simulation

Training agents inside the unreal engine just like the demo game created for the thesis is only one of many possibilities focused on training game AI behavior. Most autonomous robots or other autonomous decision-making systems were mainly driven by complex algorithms which analyzed the sensor input data to perform predefined actions. With the rise of machine learning along with the significantly increased hardware performance, research has made a shift to training autonomous agents for commercial and industrial applications. Bonsai[159] was one of the first companies to focus on implementing RL to industrial applications and was acquired by Microsoft in 2018 that wanted to dive into creating "brains" for autonomous systems[160].

A way of utilizing this combination of Game development and Deep Reinforcement learning technology, is to use the capabilities of the realistic simulation environment to recreate real-world scenarios and train agents to deal with them, with a final goal of exporting the trained AIs for use in real-world applications.

The designing process is not much different as we can get access to every variable in real-world just like inside the simulation environment using specialized hardware.

Most Common conversions to hardware:

- Get world location $\longleftrightarrow$ GPS
- Line Trace Hit $\longleftrightarrow$ LiDAR Sensor
- In game Camera $\longleftrightarrow$ Conventional Camera
- Pixel Depth $\longleftrightarrow$ LiDAR Camera
- Post Process Effect $\longleftrightarrow$ Infrared Camera

After carefully analyzing every possible real-world interaction and recreating a detailed approximation of the environment, that the agent will be eventually placed into, training inside unreal engine can be conducted safely and with very low cost, avoiding hardware damage in the initial phases when the agent will learn the environment.

There are three different options for utilizing this technology:

1. Part of the training can be done by simulation and when it is decided the agent has reached a certain degree of the desired final behavior, it can be exported and continue to train using the real hardware.

2. Most of the training (85-95%) can be done inside the simulation environment and then for the last 5-15% the model can be finetuned in the real world.
3. All the training can be done inside the simulation environment and then export the final model for use in the real world.

Applications vary from commercial, industrial, medical and military use.

- Robot control (commercial and industrial)
- Vehicle driving (assistive or complete control)
- Fire prevention and suppression
- Patient Vitals Monitoring
- Drug dosing
- Remote terrain reconnaissance
- Unmanned combat (Air and Land)

As this technology matures, applications utilizing autonomous AIs will become more common, people will start getting comfortable being around autonomous systems and having them assist or take over control in certain scenarios, which will drive the market to invest in researching this specialized field of computer science.

CHAPTER 7 CONCLUSIONS

Video games earned a place among the largest entertainment industries, even though there was a big gap between the scientific community and game development industry in the past, it is gradually getting smaller in recent years. Researchers have started to take video games seriously and focus on implementing current technology and even developing new one, for use in video games.

Video game development is a complex process combining artistic and technical components in a sophisticated way to create a final product. With the rise of deep learning and generative AI, capabilities have increased significantly, revealing endless possibilities.

In this thesis some of the latest scientific literature was discussed with a game development perspective, from behavioral artificial intelligence building and improving upon traditional techniques, to the latest generative AI methods using GANs, VAEs and Diffusion based models for content generation. Only a few scientists focus on videogame specific research, but the promising results will surely drive the video game industry to invest more resources for enticing further specialized research.

A wide variety of AI based tools can be found online and developers can have access to them in order to greatly enhance their arsenal with minimal or no cost, compared to the traditional method of creating everything by hand. Although the technology has not progressed enough to meet the AAA game standards, a very good result can be achieved.

The video game industry is continuously changing and evolving in order to meet and even exceed consumer expectations, it is inevitable that artificial intelligence will definitely make its way inside the workflow of game development process. Not as much as replacing the human aspect but optimizing productivity, driving down costs and creating the environment for faster turnover times. Big studios will be the last to integrate such systems at least at the production level, as the technology has not matured enough to meet the highest standards demanded.

The last chapter focused on how a game engine can be leveraged as a simulation environment to train Deep RL agents and how to design and implement them inside Unreal Engine. It is a very intuitive process especially with the use of the Blueprint visual scripting environment, but it is a very computationally demanding task that will require incredible amounts of training time. The more complex the desired behavior the more difficult it will be to train a single agent, the use of methods such as multi-agent training may be necessary for reducing training times significantly and creating a realistic timeframe in which a model can be trained and transferred for use in a real-world application but with a huge increase of computational cost.

In the near future, AI technology will penetrate many industries including the video game development scene. The use of AI will become commonplace during development as well as in postproduction applications due to its many advantages it can bring to the table but will also leverage the simulation capabilities of the video game world to continue evolving itself.

BIBLIOGRAPHY

- [1] B. Dictionary. *Game Definition & Meaning*. www.britannica.com. URL: <https://www.britannica.com/dictionary/game> (visited on 04/15/2024).
- [2] J. Huizinga. *Homo ludens : a study of the play-element in culture*. Humanitas, Beacon Reprints in Humanities, 1955.
- [3] E. R. Pianka. *Convergent Evolution*. www.zo.utexas.edu. URL: <http://www.zo.utexas.edu/courses/THOC/Convergence.html> (visited on 04/15/2024).
- [4] A. Turing. "Computing Machinery and Intelligence". In: *Mind* LIX (Oct. 1950), pp. 433–460. DOI: 10.1093/mind/lix.236.433. URL: <https://redirect.cs.umbc.edu/courses/471/papers/turing.pdf>.
- [5] J. Moor. "The Dartmouth College Artificial Intelligence Conference: The Next Fifty years". In: *AI Magazine* 27 (2006), pp. 87–89. URL: <https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1911/1809> (visited on 04/15/2024).
- [6] "Alexander (Sandy) Shafto Douglas 1921-2010". In: *The Computer Journal* 54 (June 2010), pp. 187–188. DOI: 10.1093/comjnl/bxq054. URL: <https://academic.oup.com/comjnl/article-abstract/54/2/187/500815> (visited on 04/15/2024).
- [7] D. Cohen. *OXO aka Noughts and Crosses - The First Video Game*. Lifewire, Mar. 2019. URL: <https://www.lifewire.com/oxo-aka-noughts-and-crosses-729624> (visited on 04/15/2024).
- [8] L. Piispanen et al. "The History of Quantum Games". In: *arXiv (Cornell University)* (Sept. 2023). DOI: <https://doi.org/10.48550/arxiv.2309.01525>.
- [9] M. F. Syahputra et al. "Historical Theme Game Using Finite State Machine for Actor Behaviour". In: *Journal of Physics: Conference Series* 1235 (June 2019), p. 012122. DOI: 10.1088/1742-6596/1235/1/012122. (Visited on 04/14/2024).
- [10] D. Jagdale. "Finite State Machine in Game Development". In: *International Journal of Advanced Research in Science, Communication and Technology* 10 (Oct. 2021), pp. 384–390. DOI: 10.48175/ijarsct-2062. (Visited on 04/14/2024).
- [11] T. S. Riyan, A. M. H. Pardede, and F. Y. Manik. "Implementation of Finite State Machine Models on the Artificial Intelligence System of Characters in The Game "MMORPG" using RPG Maker". In: *Journal of Artificial Intelligence and Engineering Applications* 3 (Oct. 2023), pp. 287–291. DOI: 10.59934/jaiea.v3i1.311. (Visited on 04/14/2024).
- [12] P. Sweetser and J. Wiles. "Current AI in games : a review". In: *Australian Journal of Intelligent Information Processing Systems* 8 (2002), pp. 24–42. URL: <https://eprints.qut.edu.au/45741/> (visited on 04/14/2024).
- [13] M. Utesch et al. "The Finite State Trading Game: Developing a Serious Game to Teach the Application of Finite State Machines in a Stock Trading Scenario". In: *Online Engineering & Internet of Things* (Sept. 2017), pp. 570–582. DOI: 10.1007/978-3-319-64352-6_54. (Visited on 04/14/2024).

- [14] K. Fathoni, R. Y. Hakkun, and H. A. T. Nurhadi. “Finite State Machines for Building Believable Non-Playable Character in the Game of Khalid ibn Al-Walid”. In: *Journal of Physics: Conference Series* 1577 (July 2020), p. 012018. DOI: 10.1088/1742-6596/1577/1/012018. (Visited on 04/14/2024).
- [15] D. Aversa, A. S. Kyaw, and C. Peters. *Unity Artificial Intelligence Programming : Add Powerful, Believable, and Fun AI Entities in Your Game with the Power of Unity 2018!, 4th Edition*. Packt Publishing Ltd, 2018.
- [16] R. Andrea and Nursobah. “Combination of Finite State Machine (FSM) and Sugeno Fuzzy for Game Agent in “Battle of Etam Earth””. In: *Advanced Science Letters* 24 (Nov. 2018), pp. 8663–8667. DOI: 10.1166/asl.2018.12320. (Visited on 04/14/2024).
- [17] A. A. Adegun et al. “Design and Implementation of an Intelligent Gaming Agent Using A* Algorithm and Finite State Machines”. In: *International Journal of Engineering Research and Technology* 13 (Feb. 2020), p. 191. DOI: 10.37624/ijert/13.2.2020.191-206. (Visited on 04/14/2024).
- [18] M. Miranda and F. Peinado. *Improving the Performance of a Computer-Controlled Player in a Maze Chase Game using Evolutionary Programming on a Finite-State Machine*. nil.fdi.ucm.es, 2015. URL: <http://nil.fdi.ucm.es/?q=node/657> (visited on 04/14/2024).
- [19] A. M. Mora et al. “Creating autonomous agents for playing Super Mario Bros game by means of evolutionary finite state machines”. In: *Evolutionary Intelligence* 6 (Jan. 2014), pp. 205–218. DOI: 10.1007/s12065-014-0105-7. (Visited on 04/14/2024).
- [20] W. Hu, Q. Zhang, and Y. Mao. *Component-based hierarchical state machine — A reusable and flexible game AI technology*. IEEE Xplore, Aug. 2011. DOI: 10.1109/ITAIC.2011.6030340. URL: <https://ieeexplore.ieee.org/abstract/document/6030340> (visited on 04/14/2024).
- [21] Y. A. Sekhavat. “Behavior Trees for Computer Games”. In: *International Journal on Artificial Intelligence Tools* 26 (Apr. 2017), p. 1730001. DOI: 10.1142/s0218213017300010.
- [22] C. Simpson. *Behavior Trees for AI: How They Work*. July 2014. URL: <https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work>.
- [23] Q. Zhang et al. “Behavior Modeling for Autonomous Agents Based on Modified Evolving Behavior Trees”. In: *2018 IEEE 7th Data Driven Control and Learning Systems Conference (DDCLS)* (May 2018). DOI: 10.1109/ddcls.2018.8515939. (Visited on 04/14/2024).
- [24] G. Robertson and I. Watson. “Building behavior trees from observations in real-time strategy games”. In: (Sept. 2015). DOI: 10.1109/inista.2015.7276774. (Visited on 04/14/2024).
- [25] E. Tomai and R. Flores. “Adapting In-Game Agent Behavior by Observation of Players Using Learning Behavior Trees”. In: *Foundations of Digital Games 2014* (Jan. 2014). URL: https://scholarworks.utrgv.edu/cs_fac/11/ (visited on 04/14/2024).

- [26] *A Reinforcement Learning Behavior Tree Framework for Game AI*. Proceedings of the 2016 International Conference on Economics, Social Science, Arts, Education and Management Engineering. Atlantis Press, Aug. 2016. DOI: 10.2991/essaeme-16.2016.120. URL: <https://www.atlantis-press.com/proceedings/essaeme-16/25860289> (visited on 04/16/2024).
- [27] A. Simonov, A. Zagarskikh, and V. Fedorov. “Applying Behavior characteristics to decision-making process to create believable game AI”. In: *Procedia Computer Science* 156 (2019), pp. 404–413. DOI: 10.1016/j.procs.2019.08.222. (Visited on 04/16/2024).
- [28] U. Engine. *Behavior Tree Overview*. docs.unrealengine.com. URL: <https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/ArtificialIntelligence/BehaviorTrees/BehaviorTreesOverview/> (visited on 04/14/2024).
- [29] S. C. Wang. “Artificial Neural Network”. In: *Interdisciplinary Computing in Java Programming* 743 (2003), pp. 81–100. DOI: 10.1007/978-1-4615-0377-4_5. (Visited on 04/14/2024).
- [30] A. Dertat. *Applied Deep Learning - Part I: Artificial Neural Networks*. Aug. 2017. URL: <https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6>.
- [31] D. Charles and S. McGlinchey. “The Past Present and Future of Artificial Neural Networks in Digital Games”. In: *www.academia.edu* (Jan. 2004). URL: https://www.academia.edu/63592036/The_Past_Present_and_Future_of_Artificial_Neural_Networks_in_Digital_Games (visited on 04/14/2024).
- [32] IBM. *What is Deep Learning?* www.ibm.com, 2023. URL: <https://www.ibm.com/topics/deep-learning> (visited on 03/18/2024).
- [33] M. Bahi and M. Batouche. “Deep Learning for Ligand-Based Virtual Screening in Drug Discovery”. In: Oct. 2018, pp. 1–5. DOI: 10.1109/PAIS.2018.8598488.
- [34] R. S. Sutton and A. Barto. *Reinforcement learning : an introduction*. 2nd. The Mit Press, 2020, p. 132. URL: <http://incompleteideas.net/book/RLbook2020.pdf> (visited on 04/14/2024).
- [35] L. Owen. *Bird’s-eye View of Reinforcement Learning Algorithms Taxonomy*. Medium, Oct. 2020. URL: <https://towardsdatascience.com/birds-eye-view-of-reinforcement-learning-algorithms-landscape-2aba7840211c> (visited on 04/14/2024).
- [36] J. Barnes and J. Hutchens. “Testing Undefined Behavior as a Result of Learning”. In: ed. by S. Rabin. *AI Game Programming Wisdom*. Charles River Media, 2002, pp. 615–623. URL: <https://theswissbay.ch/pdf/Gentoomen%20Library/Game%20Development/Programming/AI%20Game%20Wisdom.pdf> (visited on 04/17/2024).
- [37] A. OpenAI. *OpenAI Five*. openAI. URL: <https://openai.com/research/openai-five> (visited on 04/14/2024).
- [38] C. Berner et al. *Dota 2 with Large Scale Deep Reinforcement Learning*. Dec. 2019.

- [39] G. N. Yannakakis and J. Togelius. “Experience-Driven Procedural Content Generation”. In: *IEEE Transactions on Affective Computing* 2 (July 2011), pp. 147–161. DOI: 10.1109/t-affc.2011.6. (Visited on 03/13/2024).
- [40] P. Antoniadis. *Latent Space in Deep Learning | Baeldung on Computer Science*. Mar. 2022. URL: <https://www.baeldung.com/cs/dl-latent-space> (visited on 03/13/2024).
- [41] R. Verma. *A Detailed overview on Latent Space and Representation*. Oct. 2023. URL: <https://mlwithaarvy.com/a-detailed-overview-on-latent-space-and-representation/>.
- [42] R. Shende. *Autoencoders, Variational Autoencoders (VAE) and β -VAE*. Apr. 2023. URL: <https://medium.com/@rushikesh.shende/autoencoders-variational-autoencoders-vae-and-%CE%B2-vae-ceba9998773d>.
- [43] R. Wei and A. Mahmood. “Recent Advances in Variational Autoencoders With Representation Learning for Biomedical Informatics: A Survey”. In: *IEEE Access* 9 (2021), pp. 4939–4956. DOI: 10.1109/access.2020.3048309. (Visited on 01/15/2021).
- [44] L. Mak et al. “Application of Variational AutoEncoder (VAE) Model and Image Processing Approaches in Game Design”. In: *Sensors* 23 (Mar. 2023), p. 3457. DOI: 10.3390/s23073457. (Visited on 03/28/2024).
- [45] M. TOPAL. *WHAT IS MODE COLLAPSE IN GANS?* Medium, Jan. 2024. URL: <https://medium.com/@miraytopal/what-is-mode-collapse-in-gans-d3428a7bd9b8> (visited on 03/24/2024).
- [46] J. M. Tomczak. *Deep Generative Modeling*. Springer Nature, Jan. 2022. DOI: 10.1007/978-3-030-93158-2. URL: <https://link.springer.com/book/10.1007/978-3-030-93158-2> (visited on 03/14/2024).
- [47] Melanie. *What is a Conditional Generative Adversarial Network (cGAN)?* Data Science Courses | DataScientest, Sept. 2023. URL: <https://datascientest.com/en/what-is-a-conditional-generative-adversarial-network-cgan> (visited on 03/14/2024).
- [48] S.-H. Tsang. *Review — BiGAN: Adversarial Feature Learning (GAN)*. Nerd For Tech, May 2021. URL: <https://medium.com/nerd-for-tech/review-bigan-adversarial-feature-learning-gan-535eb76be2ca> (visited on 03/14/2024).
- [49] J. Donahue, P. Krähenbühl, and T. Darrell. *Adversarial Feature Learning*. Apr. 2017. DOI: 10.48550/arXiv.1605.09782. URL: <https://arxiv.org/abs/1605.09782v7>.
- [50] B. Dhakal. *Diving into Different GAN Architectures*. Medium, Feb. 2021. URL: <https://towardsdatascience.com/diving-into-different-gan-architectures-a96d05c03c5c> (visited on 03/14/2024).
- [51] J. Khan. *StyleGAN: Use machine learning to generate and customize realistic images*. June 2023. URL: <https://www.comet.com/site/blog/stylegan-use-machine-learning-to-generate-and-customize-realistic-images/>.
- [52] H. T. Center. *CycleGAN: a GAN architecture for learning unpaired image to image transformations*. Nov. 2020. URL: <https://www.haikutechcenter.com/2020/11/cyclegan-gan-architecture-for-learning.html>.

- [53] S. Karthika and M. Durgadevi. “Generative Adversarial Network (GAN): a general review on different variants of GAN and applications”. In: *2021 6th International Conference on Communication and Electronics Systems (ICCES)* (July 2021). DOI: 10.1109/icc51350.2021.9489160.
- [54] A. Hachcham. *Training and Debugging Deep Convolutional Generative Adversarial Networks*. July 2022. URL: <https://neptune.ai/blog/deep-convolutional-generative-adversarial-networks>.
- [55] S. W. Kim et al. “Learning to Simulate Dynamic Environments with GameGAN”. In: *ArXiv (Cornell University)* (May 2020). DOI: 10.48550/arxiv.2005.12126. (Visited on 03/18/2024).
- [56] A. Anwar. *What Is Transposed Convolutional Layer?* Medium, Apr. 2021. URL: <https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11> (visited on 03/18/2024).
- [57] NVIDIA. *GameGAN*. URL: <https://research.nvidia.com/labs/toronto-ai/GameGAN/>.
- [58] *MC-GAN: A Reimplementation of GameGAN With a Gaming Perspective*. May 2021. DOI: 10.1109/icwr51868.2021.9443138. (Visited on 04/17/2024).
- [59] D. Pathak et al. “Context Encoders: Feature Learning by Inpainting”. In: *ArXiv (Cornell University)* (Apr. 2016). DOI: 10.48550/arxiv.1604.07379. (Visited on 03/18/2024).
- [60] M. F. Cohen et al. “Wang Tiles for Image and Texture Generation”. In: *ACM Transactions on Graphics* 22 (July 2003), pp. 287–294. DOI: 10.1145/882262.882265. URL: <https://dl.acm.org/doi/10.1145/882262.882265> (visited on 03/17/2024).
- [61] C. Rodriguez-Pardo and E. Garces. “SeamlessGAN: Self-Supervised Synthesis of Tileable Texture Maps”. In: *IEEE Transactions on Visualization and Computer Graphics* 29 (June 2023), pp. 2914–2925. DOI: 10.1109/tvcg.2022.3143615. (Visited on 03/17/2024).
- [62] T. Moore. *pix2pix_tf_albedo2pbrmaps - Run with an API on Replicate*. replicate.com. URL: https://replicate.com/tommoore515/pix2pix_tf_albedo2pbrmaps (visited on 03/16/2024).
- [63] *EST-GAN: Enhancing Style Transfer GANs with Intermediate Game Render Passes*. 2022 IEEE Conference on Games (CoG). Aug. 2022. DOI: 10.1109/cog51982.2022.9893673. (Visited on 03/22/2024).
- [64] K. JiangWei. *A Comprehensive Explanation of Deferred Rendering*. www.linkedin.com, May 2023. URL: <https://www.linkedin.com/pulse/comprehensive-explanation-deferred-rendering-kylin-jiangwei> (visited on 03/23/2024).
- [65] R. Org. *RenderDoc*. renderdoc.org. URL: <https://renderdoc.org/> (visited on 03/22/2024).
- [66] RockstarGames. *Red Dead Redemption 2*. Rockstar Games. URL: <https://www.rockstargames.com/reddeadredemption2> (visited on 03/22/2024).
- [67] C. Wolfe. *Shape Reconstruction with ONets*. Feb. 2023. URL: <https://towardsdatascience.com/shape-reconstruction-with-onets-1c1afe89c50>.

- [68] C.-L. Li et al. “Point Cloud GAN”. In: *arXiv (Cornell University)* (Oct. 2018). DOI: 10.48550/arxiv.1810.05795. (Visited on 03/24/2024).
- [69] C. R. Qi et al. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. openaccess.thecvf.com, 2017. URL: https://openaccess.thecvf.com/content_cvpr_2017/html/Qi_PointNet_Deep_Learning_CVPR_2017_paper.html (visited on 03/24/2024).
- [70] H. Ben-Hamu et al. “Multi-chart generative surface modeling”. In: *ACM Transactions on Graphics* 37 (Jan. 2019), pp. 1–15. DOI: 10.1145/3272127.3275052. (Visited on 03/24/2024).
- [71] T. Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. In: (Oct. 2017). DOI: 10.48550/arxiv.1710.10196.
- [72] *3D Object Reconstruction from a Single Depth View with Adversarial Learning*. Oct. 2017. DOI: 10.1109/iccwv.2017.86. (Visited on 04/17/2024).
- [73] V. V. Kniaz, P. V. Moshkantsev, and V. A. Mizginov. “Deep Learning a Single Photo Voxel Model Prediction from Real and Synthetic Images”. In: *Studies in computational intelligence* 856 (Sept. 2019), pp. 3–16. DOI: 10.1007/978-3-030-30425-6_1. (Visited on 04/17/2024).
- [74] N. Nozawa et al. “3D car shape reconstruction from a contour sketch using GAN and lazy learning”. In: *The Visual Computer* 38 (Apr. 2021), pp. 1317–1330. DOI: 10.1007/s00371-020-02024-y. (Visited on 04/30/2023).
- [75] J. Wu et al. *Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling*. arXiv.org, Jan. 2017. DOI: 10.48550/arXiv.1610.07584. URL: <https://arxiv.org/abs/1610.07584> (visited on 08/07/2023).
- [76] R. Spick, S. Demediuk, and J. J. Walker. “Naive Mesh-to-Mesh Coloured Model Generation using 3D GANs”. In: *Proceedings of the Australasian Computer Science Week Multiconference* (Feb. 2020). DOI: 10.1145/3373017.3373067.
- [77] S. Lunz et al. “Inverse Graphics GAN: Learning to Generate 3D Shapes from Unstructured 2D Data”. In: *arXiv:2002.12674 [cs]* (Feb. 2020). URL: <https://arxiv.org/abs/2002.12674> (visited on 03/26/2024).
- [78] E. Smith, S. Fujimoto, and D. Meger. “Multi-View Silhouette and Depth Decomposition for High Resolution 3D Object Representation”. In: *arXiv (Cornell University)* (Feb. 2018). DOI: 10.48550/arxiv.1802.09987. (Visited on 03/26/2024).
- [79] E. Guérin et al. *Interactive Example-Based Terrain Authoring with Conditional Generative Adversarial Networks*. HAL Archives Ouvertes, Sept. 2017. URL: <https://hal.science/hal-01583706v2> (visited on 04/16/2024).
- [80] R. Spick and J. Walker. “Realistic and Textured Terrain Generation Using GANs”. In: *European Conference on Visual Media Production* (Dec. 2019). DOI: 10.1145/3359998.3369407. (Visited on 03/21/2024).

- [81] E. Panagiotou and E. Charou. “Procedural 3D Terrain Generation using Generative Adversarial Networks”. In: *arXiv (Cornell University)* (Oct. 2020). DOI: 10.48550/arxiv.2010.06411. (Visited on 03/21/2024).
- [82] X. Wang et al. “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks”. In: *Lecture Notes in Computer Science* 11133 (Jan. 2019), pp. 63–79. DOI: 10.1007/978-3-030-11021-5_5.
- [83] C. Ledig et al. “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”. In: (Sept. 2016). DOI: 10.48550/arxiv.1609.04802. (Visited on 04/16/2024).
- [84] A. A. Grigoryan. *Understanding VGG Neural Networks: Architecture and Implementation*. Aug. 2023. URL: <https://thegrigorian.medium.com/understanding-vgg-neural-networks-architecture-and-implementation-400d99a9e9ba>.
- [85] J. Ho, A. Jain, and P. Abbeel. “Denoising Diffusion Probabilistic Models”. In: *Advances in Neural Information Processing Systems* 33 (2020). URL: <https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html> (visited on 03/30/2024).
- [86] K. Kreis, R. Gao, and A. Vahdat. *Denoising Diffusion-based Generative Modeling: Foundations and Applications*. Sept. 2022. URL: <https://cvpr2022-tutorial-diffusion-models.github.io/>.
- [87] R. Rombach et al. “High-Resolution Image Synthesis with Latent Diffusion Models”. In: *ArXiv (Cornell University)* (Dec. 2021). DOI: 10.48550/arxiv.2112.10752. URL: <https://doi.org/10.48550/arXiv.2112.10752> (visited on 03/30/2024).
- [88] P. Dhariwal and A. Nichol. “Diffusion Models Beat GANs on Image Synthesis”. In: *arXiv:2105.05233 [cs, stat]* (June 2021). URL: <https://arxiv.org/abs/2105.05233> (visited on 04/03/2024).
- [89] R. Siddiqui. *What are Stable Diffusion Models and Why are they a Step Forward for Image Generation?* Sept. 2022. URL: <https://towardsdatascience.com/what-are-stable-diffusion-models-and-why-are-they-a-step-forward-for-image-generation-aa1182801d46>.
- [90] OpenAI. *DALL-E 2*. OpenAI, 2022. URL: <https://openai.com/dall-e-2> (visited on 04/04/2024).
- [91] Midjourney. Midjourney. URL: <https://www.midjourney.com/home> (visited on 04/04/2024).
- [92] StableDiffusion. *Stability AI Image Models*. Stability AI. URL: <https://stability.ai/stable-image> (visited on 04/17/2024).
- [93] Discord. *Discord — A New Way to Chat with Friends & Communities*. Discord, 2024. URL: <https://discord.com/> (visited on 04/04/2024).
- [94] S. Luo and W. Hu. “Diffusion Probabilistic Models for 3D Point Cloud Generation”. In: (June 2021). DOI: 10.1109/cvpr46437.2021.00286. URL: https://openaccess.thecvf.com/content/CVPR2021/papers/Luo_Diffusion_Probabilistic_Models_for_3D_Point_Cloud_Generation_CVPR_2021_paper.pdf (visited on 04/08/2024).

- [95] ShapeNet. *ShapeNet*. shapenet.org. URL: <https://shapenet.org/> (visited on 04/17/2024).
- [96] T. Groueix et al. “AtlasNet: A Papier-Mache Approach to Learning 3D Surface Generation”. In: *arXiv (Cornell University)* (Feb. 2018). DOI: 10.48550/arxiv.1802.05384. URL: <https://doi.org/10.48550/arXiv.1802.05384> (visited on 04/08/2024).
- [97] G. Yang et al. “PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows”. In: (June 2019). DOI: 10.48550/arxiv.1906.12320. URL: <https://doi.org/10.48550/arXiv.1906.12320> (visited on 04/08/2024).
- [98] R. Cai et al. “Learning Gradient Fields for Shape Generation”. In: *arXiv (Cornell University)* (Aug. 2020). DOI: 10.48550/arxiv.2008.06520. URL: <https://doi.org/10.48550/arXiv.2008.06520> (visited on 04/08/2024).
- [99] L. Zhou, Y. Du, and J. Wu. “3D Shape Generation and Completion through Point-Voxel Diffusion”. In: *arXiv (Cornell University)* (Apr. 2021). DOI: 10.48550/arxiv.2104.03670. URL: <https://doi.org/10.48550/arXiv.2104.03670> (visited on 04/08/2024).
- [100] Z. Liu et al. *Point-Voxel CNN for Efficient 3D Deep Learning*. arXiv.org, Dec. 2019. DOI: 10.48550/arXiv.1907.03739. URL: <https://arxiv.org/abs/1907.03739> (visited on 04/08/2024).
- [101] X. Zeng et al. “LION: Latent Point Diffusion Models for 3D Shape Generation”. In: *arXiv (Cornell University)* (Oct. 2022). DOI: 10.48550/arxiv.2210.06978. (Visited on 04/09/2024).
- [102] K. He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016), pp. 770–778. DOI: 10.1109/cvpr.2016.90. URL: <https://doi.org/10.1109/CVPR.2016.90> (visited on 04/09/2024).
- [103] S. Peng et al. “Shape As Points: A Differentiable Poisson Solver”. In: *arXiv (Cornell University)* (June 2021). DOI: 10.48550/arxiv.2106.03452. (Visited on 04/09/2024).
- [104] *Linear Pulse Code Modulated Audio (LPCM)*. www.loc.gov, Feb. 2008. URL: <https://www.loc.gov/preservation/digital/formats/fdd/fdd000011.shtml> (visited on 04/10/2024).
- [105] Instructables. *What Is MIDI?* Instructables, Sept. 2012. URL: <https://www.instructables.com/What-is-MIDI/> (visited on 04/10/2024).
- [106] G. Mittal et al. *Symbolic Music Generation with Diffusion Models*. arXiv.org, Nov. 2021. DOI: 10.48550/arXiv.2103.16091. URL: <https://arxiv.org/abs/2103.16091> (visited on 06/22/2023).
- [107] A. Roberts et al. *A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music*. proceedings.mlr.press, July 2018. URL: <http://proceedings.mlr.press/v80/roberts18a.html> (visited on 04/11/2024).
- [108] C. Raffel. “Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching”. PhD thesis. July 2016. DOI: 10.7916/d8n58mhv. URL: <https://doi.org/10.7916/D8N58MHV> (visited on 04/11/2024).

- [109] UrbanSound. *UrbanSound8K*. Urban Sound Datasets, 2019. URL: <https://urbansounddataset.weebly.com/urbansound8k.html> (visited on 04/17/2024).
- [110] H. Liu et al. *AudioLDM: Text-to-Audio Generation with Latent Diffusion Models*. arXiv.org, Sept. 2023. DOI: 10.48550/arXiv.2301.12503. URL: <https://arxiv.org/abs/2301.12503> (visited on 04/12/2024).
- [111] Y. Wu et al. “Large-scale Contrastive Language-Audio Pretraining with Feature Fusion and Keyword-to-Caption Augmentation”. In: *arXiv (Cornell University)* (Nov. 2022). DOI: 10.48550/arxiv.2211.06687. (Visited on 04/12/2024).
- [112] K. Chen et al. “HTS-AT: A Hierarchical Token-Semantic Audio Transformer for Sound Classification and Detection”. In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (May 2022). DOI: 10.1109/icassp43922.2022.9746312. (Visited on 04/12/2024).
- [113] Y. Liu et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *arXiv (Cornell University)* (July 2019). DOI: 10.48550/arxiv.1907.11692. (Visited on 04/12/2024).
- [114] F. Kreuk et al. “AudioGen: Textually Guided Audio Generation”. In: *arXiv (Cornell University)* (Sept. 2022). DOI: 10.48550/arxiv.2209.15352. (Visited on 04/12/2024).
- [115] J. Kong, J. Kim, and J. Bae. *HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis*. arXiv.org, Oct. 2020. DOI: 10.48550/arXiv.2010.05646. URL: <http://arxiv.org/abs/2010.05646> (visited on 04/12/2024).
- [116] AudioSet. *AudioSet*. research.google.com. URL: <https://research.google.com/audioset/index.html> (visited on 04/17/2024).
- [117] D. Ghosal et al. “Text-to-Audio Generation using Instruction-Tuned LLM and Latent Diffusion Model”. In: *arXiv (Cornell University)* (Apr. 2023). DOI: 10.48550/arxiv.2304.13731. (Visited on 04/12/2024).
- [118] H. W. Chung et al. “Scaling Instruction-Finetuned Language Models”. In: (Oct. 2022). DOI: 10.48550/arxiv.2210.11416. (Visited on 04/12/2024).
- [119] Y. Tokozume, Y. Ushiku, and T. Harada. *Learning from Between-class Examples for Deep Sound Recognition*. arXiv.org, Feb. 2018. DOI: 10.48550/arXiv.1711.10282. URL: <https://arxiv.org/abs/1711.10282> (visited on 04/13/2024).
- [120] J. Burstein, C. Doran, and T. Solorio, eds. *AudioCaps: Generating Captions for Audios in The Wild*. Vol. 1. June 2019, pp. 119–132. DOI: 10.18653/v1/N19-1011. URL: <https://aclanthology.org/N19-1011/> (visited on 04/17/2024).
- [121] H. Liu et al. *AudioLDM 2: Learning Holistic Audio Generation with Self-supervised Pretraining*. arXiv.org, Sept. 2023. DOI: 10.48550/arXiv.2308.05734. URL: <https://arxiv.org/abs/2308.05734> (visited on 04/13/2024).
- [122] P.-Y. Huang et al. “Masked Autoencoders that Listen”. In: (July 2022). DOI: 10.48550/arxiv.2207.06405. (Visited on 04/13/2024).
- [123] Z. Evans et al. “Fast Timing-Conditioned Latent Audio Diffusion”. In: *arXiv (Cornell University)* (Feb. 2024). DOI: 10.48550/arxiv.2402.04825. (Visited on 04/13/2024).

- [124] R. Kumar et al. "High-Fidelity Audio Compression with Improved RVQGAN". In: *arXiv (Cornell University)* (Jan. 2023). DOI: 10.48550/arxiv.2306.06546. (Visited on 04/13/2024).
- [125] AudioSparx. *Commercial Music for Video, TV, Film and Media from AudioSparx.com*. AudioSparx.com. URL: <https://www.audiosparx.com/> (visited on 04/14/2024).
- [126] J. Newman. *Videogames*. Routledge, 2004.
- [127] *A Short and Simple Definition of What a Videogame Is*. Proceedings of DiGRA 2005 (Changing Views: Worlds in Play). Vancouver (Canada), 16–20 june. 2005. URL: <http://www.digra.org/wp-content/uploads/digital-library/06278.37547.pdf> (visited on 04/15/2024).
- [128] G. Frasca. "VIDEOGAMES OF THE OPPRESSED: VIDEOGAMES AS A MEANS FOR CRITICAL THINKING AND DEBATE". PhD thesis. 2001. URL: <https://ludology.typepad.com/weblog/articles/thesis/FrascaThesisVideogames.pdf> (visited on 04/15/2024).
- [129] E. Zimmerman. "Narrative, Interactivity, Play, and Games". In: *Electronic Book Review* (July 2004). URL: <https://electronicbookreview.com/essay/narrative-interactivity-play-and-games/>. (visited on 04/15/2024).
- [130] R. Caillois. *Les Jeux Et Les Hommes*. University of Illinois Press, 2001. URL: <https://voidnetwork.gr/wp-content/uploads/2016/09/Man-Play-and-Games-by-Roger-Caillois.pdf> (visited on 04/15/2024).
- [131] R. Rouse. *Game Design*. 2nd. Wordware Publishing, Inc, 2005. URL: <https://gamifique.wordpress.com/wp-content/uploads/2011/11/5-game-design-theory-and-practice.pdf> (visited on 04/15/2024).
- [132] A. L. Diberder and F. L. Diberder. *L'univers des jeux vidéo*. Editions La Découverte, 1998.
- [133] R. Kowert and T. Quandt, eds. *The Video Game Debate: Unravelling the Physical, Social, and Psychological Effects of Video Games*. Routledge, Sept. 2015, pp. 2–4.
- [134] C. War. *Console war*. Wikipedia, Aug. 2021. URL: https://en.wikipedia.org/wiki/Console_war (visited on 04/17/2024).
- [135] Wikipedia. *The Elder Scrolls V: Skyrim*. Nov. 2019. URL: https://en.wikipedia.org/wiki/The_Elder_Scrolls_V:_Skyrim.
- [136] UESP. *Skyrim: Credits - The Unofficial Elder Scrolls Pages (UESP)*. URL: <https://en.uesp.net/wiki/Skyrim:Credits>.
- [137] J. Schell. *Art Of Game Design: A Book of Lenses*. CRC Press, Aug. 2008.
- [138] D. Strachan. *Making difficulty curves in games - Just Things Made By Dave*. 2018. URL: <http://www.davetech.co.uk/difficultycurves>.
- [139] *The Director*. Left 4 Dead Wiki, 2019. URL: https://left4dead.fandom.com/wiki/The_Director (visited on 04/15/2024).
- [140] M. Cullen. *Basics of Sound Design for Video Games*. URL: <https://frost.ics.uci.edu/ics62/BasicsofSoundDesignforVideoGames-MichaelCullen.pdf> (visited on 04/17/2024).

- [141] Nvidia. *Ray Tracing*. NVIDIA Developer, Aug. 2018. URL: <https://developer.nvidia.com/discover/ray-tracing> (visited on 04/17/2024).
- [142] U. Engine. *Key Particle Concepts*. dev.epicgames.com. URL: <https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/ParticleSystems/Overview/> (visited on 04/17/2024).
- [143] J. Salemi. "Slowing Down a Fast System". In: *PC Mag* (Jan. 1992), p. 286. URL: <https://books.google.gr/books?id=uEkrL23rU98C&printsec=frontcover&hl=en#v=snippet&q=%22slowing%20down%20a%20fast%20system%22&f=false> (visited on 04/16/2024).
- [144] JuegoStudio. *How Long Does it Take to Develop a Video Game? A Detailed Guide*. Juego Studio, July 2023. URL: <https://www.juegostudio.com/blog/how-long-does-it-take-to-develop-video-game#> (visited on 04/17/2024).
- [145] Sloyd - *Generate 3D. Fast*. sloyd.ai. URL: <https://www.sloyd.ai/> (visited on 04/16/2024).
- [146] Polycam. *AI Texture Generator for Blender, Unreal, Unity* | Polycam. poly.cam. URL: <https://poly.cam/tools/ai-texture-generator> (visited on 04/16/2024).
- [147] *Real-ESRGAN: Advanced Image Enhancement & Super-Resolution Online Tool*. real-esrgan.com. URL: <https://real-esrgan.com/#Playground> (visited on 04/25/2024).
- [148] *Pixelcut*. create.pixelcut.ai. URL: <https://create.pixelcut.ai/upscaler> (visited on 04/16/2024).
- [149] *Soundraw*. soundraw.io. URL: <https://soundraw.io/> (visited on 04/16/2024).
- [150] StabilityAI. *Stable Audio - Generative AI for music & sound fx*. stableaudio.com. URL: <https://stableaudio.com/> (visited on 04/17/2024).
- [151] EvokeMusic. *Check out Evoke Music!* evokemusic.ai. URL: <https://evokemusic.ai/music> (visited on 04/16/2024).
- [152] AmadeusCode. *About* | Amadeus Code, Inc. Amadeus Code. URL: <https://amadeuscode.com/en/about/> (visited on 04/16/2024).
- [153] OpenAI. *ChatGPT*. URL: <https://chat.openai.com/> (visited on 04/16/2024).
- [154] K. Mythen. *OpenAI API Plugin*. GitHub, Mar. 2023. URL: <https://github.com/KellannM/OpenAI-Api-Unreal> (visited on 04/16/2024).
- [155] GiganticWork. *Easy Diffusion*. stablediffusion.gigantic.work. URL: <https://stablediffusion.gigantic.work/> (visited on 06/15/2024).
- [156] OpenAI. *Gym Documentation*. www.gymlibrary.dev. URL: <https://www.gymlibrary.dev/> (visited on 06/15/2024).
- [157] EpicGames. *MindMaker Machine Learning AI Plugin in Code Plugins - UE Marketplace*. Unreal Engine. URL: <https://www.unrealengine.com/marketplace/en-US/product/mindmaker-ai-plugin> (visited on 06/15/2024).
- [158] X. Peng, P. Abbeel, and M. Van De Panne. "DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills". In: *ACM Trans. Graph* 37 (2018). DOI: 10.1145/3197517.3201311. URL: <https://arxiv.org/pdf/1804.02717.pdf>.

- [159] Bonsai. *Bonsai | LinkedIn*. July 2020. URL: <https://www.linkedin.com/company/bonsai-ai>.
- [160] P. Gurdeep. *Microsoft to acquire Bonsai in move to build “brains” for autonomous systems*. June 2018. URL: <https://blogs.microsoft.com/blog/2018/06/20/microsoft-to-acquire-bonsai-in-move-to-build-brains-for-autonomous-systems/>.