ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ
Πρόγραμμα Μεταπτυχιακών Σπουδών
Τεχνητής Νοημοσύνης

Πτυχιακή Εργασία

Δημιουργία περιεχομένου για Video Games με μεθόδους Procedural Content Generation και Generative Adversarial Neural Networks.

Ευφροσύνη Καλτιριμίδου

Επιβλέπων:
Νίκος Νικολαίδης
Καθηγητής

Θεσσαλονίκη 2020

Περιεχόμενα

Κεφάλαιο 1

Procedural Content Generation

Το αντικείμενο του Procedural Content Generation (PCG) όπως ορίζεται στο [1] είναι η *Δημιουργία περιεχομένου για ηλεκτρονικά παιχνίδια με την χρήση αλγορίθμων και με την παροχή ελάχιστων ή καθόλου εισόδων από τον χρήστη.* Αποτελεί μια μεγάλη κατηγορία έρευνας και ανάπτυξης για την επιστήμη της πληροφορικής [2] και την βιομηχανία των ηλεκτρονικών παιχνιδιών (Video Games). Για συντομία, στο υπόλοιπο κείμενο της εργασίας θα αναφέρετε ως PCG.

Το PCG, όπως και πολλά άλλα αντικείμενα της πληροφορικής, αντιπροσωπεύεται από ιδιαίτερα προβλήματα και περιορισμούς, τόσο στην πολυπλοκότητα των αλγορίθμων όσο και στην αυθεντικότητα και πρωτοτυπία των αποτελεσμάτων. Ως γνωστικό αντικείμενο της επιστήμης της πληροφορικής, μπορεί να ταξινομηθεί κάτω από την κατηγορία της Τεχνητής Νοημοσύνης. Βασικός στόχος του PCG είναι η δημιουργία αλγορίθμων που μπορούν να προσομοιάσουν την ανθρώπινη δημιουργικότητα και ευφυΐα. Επιπλέον στα προβλήματα και στις προσεγγίσεις επίλυσης παρατηρούμε πολλά κοινά στοιχεία με άλλα πεδία της Τεχνητής Νοημοσύνης.

Όπως έχει παρατηρηθεί και σε πολλά άλλα αντικείμενα της Τεχνητής Νοημοσύνης, έτσι και το PCG είχε μικρή εξάπλωση και χρήση στο ξεκίνημα της επιστήμης. Στη συνέχεια όμως επεκτάθηκε γρήγορα και εδραιώθηκε ως ένας ξεχωριστός τομέας με δικά του προβλήματα, αλγορίθμους και εφαρμογές. Η σταδιακή άνοδος του δεν οφείλεται στις δυνατότητες των αλγορίθμων και της θεωρία πίσω από το αντικείμενο του PCG, αλλά κυρίως στην αδυναμία του υλικού (hardware) των υπολογιστών εκείνων των περιόδων να εφαρμόσουν αλγόριθμους τέτοιας πολυπλοκότητας. Τα τελευταία χρόνια με την ανάπτυξη των δυνατοτήτων των προσωπικών υπολογιστών και των κινητών συσκευών έχει διευρυνθεί η χρήση του PCG για την παραγωγή διαφόρων ειδών περιεχομένου για παιχνίδια (game content). Μαζί με την αύξηση

στην χρήση μεθόδων PCG ήρθε και η αύξηση των προβλημάτων που καλείται να επιλύσει.

Σχήμα 1.1: Screenshot από το παιχνίδι No Man's Sky (2016). Οι κόσμοι που δημιουργεί είναι εξολοκλήρου κατασκευασμένοι με τη χρήση ντετερμινιστικών μεθόδων PCG.

1.1 Περιεχόμενο

Για να κατανοήσουμε καλύτερη το πεδίο του PCG πρέπει πρώτα να καταλάβουμε τι θεωρείται περιεχόμενο (Game Content) σε ένα παιχνίδι. Ο τομέας του PCG αναφέρεται και έχει χρησιμοποιηθεί σε ηλεκτρονικά παιχνίδια (Video Games), επιτραπέζια παιχνίδια (Board Games), παιχνίδια με κάρτες (Card Games) κ.ά. Το αντικείμενο της συγκεκριμένης εργασίας, επικεντρώνετε στην δημιουργία περιεχομένου για Video Games.

Game Content Όπως αναφέρεται και στο όνομα, περιεχόμενο, είναι κάτι που περιέχεται σε ένα παιχνίδι. Αυτός ο ορισμός όμως είναι πολύ γενικός και ευρύς και δεν περιορίζετε μόνο στο περιεχόμενο που αντιστοιχεί στο PCG. Στην βιβλιογραφία μπορούμε να ξεχωρίσουμε συγκεκριμένα είδη περιεχομένου [9] που φαίνεται να μπορούν να δημιουργηθούν με μεθόδους PCG. Αυτά είναι:

- Γραφικά (Textures)
- Επίπεδα (Levels) και χάρτες (Maps)

- Αντικείμενα (Items)
- Αποστολές (Quests)
- Ιστορίες (Stories)
- Μουσική (Music)

Ο παραπάνω διαχωρισμός γίνεται με βάση το είδος του κάθε περιεχομένου, για παράδειγμα η μουσική ως περιεχόμενο ενός παιχνιδιού, διαφέρει από τα αντικείμενα. Αντίστοιχα οι μεθοδολογίες που έχουν αναπτυχθεί για την παραγωγή μουσικής διαφέρουν από τις μεθόδους που χρησιμοποιούνται για την παραγωγή αντικειμένων. Αυτό βέβαια δεν σημαίνει ότι δεν υπάρχουν κοινοί αλγόριθμοι που μπορούν να εφαρμοστούν σε παραπάνω από ένα είδος με επιτυχία, αντίθετα υπάρχουν πολλοί αλγόριθμοι που έχουν ως βάση έναν πιο γενικό αλγόριθμο και αποτελούν ειδικές εκδόσεις του για το κάθε είδος περιεχομένου.

Ο διαχωρισμός του περιεχομένου βοηθάει στην καλύτερη κατανόηση των ιδιαιτεροτήτων και των περιορισμών που εμφανίζει το κάθε είδος το οποίο οδηγεί στην δημιουργία καλύτερων μεθόδων και αλγορίθμων.

1.2 Η χρησιμότητα του PCG

Η πρώτη ανάγκη που παρουσιάστηκε και οδήγησε στην υιοθέτηση μεθόδων PCG αφορούσε την μείωση του αποθηκευτικού χώρου που καταλάμβανε ένα παιχνίδι. Το PCG δίνει την δυνατότητα της δημιουργίας του game content μόλις παρουσιαστεί η ανάγκη να χρησιμοποιηθεί ή να εμφανιστεί στον παίκτη. Αυτό σημαίνει ότι δεν χρειάζεται να καταλαμβάνει χώρο στην μνήμη εάν υπάρχει η δυνατότητα της δημιουργίας του ακριβώς την στιγμή που χρειάζεται. Με την μνήμη να αποτελεί έναν διαμοιραζόμενο πώρο μεταξύ των εφαρμογών ακόμα και σήμερα το PCG αποτελεί μια μέθοδο μείωσης του χώρου που καταλαμβάνει ένα παιχνίδι.

Το PCG έχει επίσης χρησιμότητα από τους καλλιτέχνες και σχεδιαστές διαφόρων παιχνιδιών. Η χρήση PCG για την δημιουργία περιεχομένου το οποίο αν και δεν είναι τέλειο ή αρκετά καλό για το παιχνίδι, δίνετε στους σχεδιαστές οι οποίοι το βελτιώνουν, το αλλάζουν και προσθέτουν στοιχεία ώστε να μπορέσει να προστεθεί με επιτυχία στο παιχνίδι. Με αυτόν τον τρόπο το PCG βοηθάει ανθρώπους με τέτοιους ρόλους στην εργασία τους, δίνοντας του έμπνευση και αναλαμβάνοντας ένα κομμάτι της δουλειάς, με σκοπό αυτοί να μπορέσουν να επικεντρωθούν στα κομμάτια που θα κάνουν το περιεχόμενο πραγματικά εντυπωσιακό. Σε αυτές τις

υλοποιήσεις του PCG συνηθίζετε, ο σχεδιαστής να δίνει μέσω διαφόρων εισόδων (inputs) κάποιες παραμέτρους που ορίζουν μια γενική περιγραφή του περιεχομένου που θέλει να δημιουργήσει, όπως το μέγεθος του χάρτη. Στη συνέχεια το PCG δημιουργεί το περιεχόμενα με βάση τις παραμέτρους που πήρε και εμφανίζει το αποτέλεσμα στο σχεδιαστή. Αυτή η διαδικασία μπορεί να επαναληφθεί πολλές φορές μέχρι ο σχεδιαστής να είναι ικανοποιημένος με το παραγόμενο αποτέλεσμα. [13]

Μια επίσης πολύ σημαντική ανάγκη που καλύπτει, εν μέρη, το PCG είναι η δημιουργία ενός παιχνιδιού που δεν τελειώνει ποτέ (endless). Αντίθετα με την συνεχή παραγωγή πρωτότυπου περιεχομένου, ένα παιχνίδι μπορεί να συνεχίζετε ευ άπειρον, προσφέροντας αμέτρητες ώρες διασκέδασης στους παίκτες. Πολλά παιχνίδια έχουν επιχειρήσει να υλοποιήσουν αυτό το χαρακτηριστικό, κάποια με μεγάλη επιτυχία και κάποια με το αντίθετο αποτέλεσμα. Ένα από τα μεγαλύτερα προβλήματα που αντιμετωπίζουν οι υλοποιήσεις του PCG είναι η επαναληψιμότητα και η μονοτονία, όπως θα αναλυθεί και παρακάτω. Είναι ένα από τα πιο σημαντικά κριτήρια για την επιτυχία ενός endless video game.

Τέλος, μια ανάγκη που έχει προκύψει τα τελευταία χρόνια στον χώρο των παιχνιδιών και συνδέετε με μια ακόμα περιοχή της Τεχνητής Νοημοσύνης στα παιχνίδια είναι η εξατομίκευση περιεχομένου, ή personalized content. Αναφέρεται στην δημιουργία περιεχομένου που είναι σχεδιασμένο για τις προτιμήσεις και τις ανάγκες του κάθε παίκτη.

1.3 Παιχνίδια που χρησιμοποιούν PCG

Από την πρώτη στιγμή που ξεκίνησε η διάδοση των Video Games φάνηκε η ανάγκη για την αυτόματη και αυτόνομη δημιουργία *σωστού* περιεχομένου. Κάποια από τα παιχνίδια που εφάρμοσαν με μεγάλη επιτυχία μεθόδους PCG είναι:

Rogue (1980) Ένα από τα πρώτα παιχνίδια που εφάρμοσε PCG για την αυτόματη δημιουργία επιπέδων (**levels**). Το Rogue ενέπνευσε την δημιουργία πολλών παιχνιδιών με αντίστοιχες δυνατότητες και PCG μεθόδους. [4]

Spore (2008) Το Spore είναι ένα life-simulation και strategy παιχνίδι που χρησιμοποιεί PCG για την δημιουργία πλασμάτων και αντικειμένων. Οι αλγόριθμοι του Spore, συνδυάζουν απλά σχήματα και αντικείμενα για να δημιουργήσουν μεγαλύτερα και πιο πολύπλοκα πλάσματα με βάση διάφορους κανόνες και περιορισμούς. [5]

No Man's Sky (2016) Ένα από τα πιο σημαντικά παραδείγματα για τις δυνατότητες του PCG είναι το παιχνίδι no Man's Sky. Το θέμα του παιχνιδιού είναι η εξερεύνηση του διαστήματος και η επιβίωση σε ξένους πλανήτες. Το παιχνίδι δημιουργεί σχεδόν ολόκληρο τον κόσμο με PCG, δηλαδή τους πλανήτες, τα αστέρια, τα φυτά, τα ζώα και τα encounters του παίκτη με τη χρήση ντετερμινιστικών αλγορίθμων PCG. Η αποδοχή του παιχνιδιού από το κοινό ήταν μέτρια. Οι δημιουργοί είχαν υποσχεθεί έναν απέραντο κόσμο γεμάτο περιεχόμενο και οι παίκτες παρατήρησαν ότι το τελικό αποτέλεσμα ήταν μονότονο και επαναλαμβανόμενο, κάτι που τους δημιούργησε άσχημες εντυπώσεις. [6]

1.4 Προβλήματα του PCG

Όπως αναφέρθηκε και παραπάνω, το PCG δεν αποτελεί μια τέλεια και ολοκληρωμένη λύση για να τα προβλήματα που καλείται να αντιμετωπίσει. Αντίθετα οι υλοποιήσεις του PCG πρέπει να λαμβάνουν υπόψη τα καινούργια προβλήματα και περιορισμούς που δημιουργούνται με την χρήση των PCG αλγορίθμων. [8] [7]

Επαναληψιμότητα - Μονοτονία Ένα από τα πιο σημαντικά θέματα είναι η ποικιλία και μοναδικότητα του παραγόμενου περιεχομένου. Όπως αναλύθηκε και παραπάνω, σε πολλές περιπτώσεις το παραγόμενο περιεχόμενο είναι πολύ μονότονο, μοτίβα φαίνονται να επαναλαμβάνονται και κατά συνέπεια το αποτέλεσμα δίνει στον χρήστη την αντίθετη εντύπωση από την επιθυμητή. Οι λόγοι που συμβαίνει αυτό είναι πολλοί και σχετίζονται με το είδος του αλγόριθμου που χρησιμοποιείται κάθε φορά, τις παραμέτρους που δίνονται και τα βασικά assets που χρησιμοποιεί για να δημιουργήσει το περιεχόμενο.

Ταχύτητα - πολυπλοκότητα Πολλοί αλγόριθμοι PCG πρέπει να είναι σε θέση να λειτουργούν σε πολύ λίγο χρόνο και με περιορισμένους πόρους. Σε ένα ηλεκτρονικό παιχνίδι υπάρχουν πολλά κομμάτια που πρέπει να δουλεύουν ταυτόχρονα, όπως τα γραφικά, το UI, το σύστημα κανόνων για την προσομοίωση της φυσικής στον κόσμο του παιχνιδιού κ.ά. Συνεπώς το σύστημα του AI που περιέχει και το PCG δεν μπορεί να καταλαμβάνει πολλούς πόρους ή να αργεί να ανταποκριθεί γιατί το παιχνίδι θα φαίνεται να κολλάει ή να μην δουλεύει όπως πρέπει. Αυτό σημαίνει ότι οι αλγόριθμοι που υλοποιούν το PCG πρέπει να έχουν συγκεκριμένη χρονική και χωρική πολυπλοκότητα και να μην ξεπερνάνε.

Playability Μπορεί να έχουμε σχεδιάσει τον τέλειο αλγόριθμο PCG που χρησιμοποιεί ελάχιστους πόρους του συστήματος και δημιουργεί μοναδικά επίπεδα για

το 2D platformer παιχνίδι μας αλλά ένα στα τρία επίπεδα να μην έχουν είσοδο.Αυτό σημαίνει ότι ο παίκτης δεν θα μπορέσει να το επισκεφτεί ποτέ, ή θα βρεθεί παγιδευμένος μέσα του χωρίς κάποιο τρόπο να προχωρήσει το παιχνίδι. Αυτό φυσικά είναι κάτι που δεν θέλουμε σε καμία περίπτωση να συμβεί. Για αυτό το λόγο πρέπει να ορίσουμε τι θεωρείται ”playable” περιεχόμενο και τι ”unplayable”.

1.5 Επιθυμητά Χαρακτηριστικά του PCG

Με βάση τα παραπάνω στοιχεία μπορούμε να αναλύσουμε ένα σύνολο από επιθυμητά χαρακτηριστικά που θέλουμε να έχει κάθε σύστημα PCG ώστε να διασφαλίσουμε την σωστή και αρμονική λειτουργία του μέσα στο παιχνίδι. [8] [7]

1.5.1 Ταχύτητα - Πολυπλοκότητα

Όπως είδαμε και στο 1.4 ένα μεγάλο θέμα για κάθε παιχνίδι είναι η διαχείριση και κατανομή των πόρων στα επιμέρους συστήματα του. Το σύστημα του AI, που περιέχει και το υποσύστημα του PCG πρέπει να περιορίσει την πολυπλοκότητα των αλγορίθμων του, χρονικά και χωρικά, ώστε να ανταποκρίνονται στους διαθέσιμους πόρους. Αυτός ο περιορισμός είναι ιδιαίτερα σημαντικός για την επιτυχία του παιχνιδιού καθώς η εμπειρία του παίκτη σχετίζετε άμεσα με την πόσο γρήγορα και σωστά ανταποκρίνεται το παιχνίδι.

Εάν το αλγόριθμος που δημιουργεί όπλα ειδικά για τον παίκτη (personalized) αργήσει να ολοκληρώσει την λειτουργία του, θα δημιουργηθεί στον παίκτη η εντύπωση ότι το παιχνίδι έχει ”κολλήσει” και δεν ανταποκρίνεται. Αυτό θα έχει ως συνέπεια να πάρει μια αρνητική εμπειρία από το παιχνίδι. Αντίθετα, τα παιχνίδια που καταφέρνουν να ανταποκρίνονται άμεσα σε κάθε εντολή του παίκτη λαμβάνουν πολύ θετικά σχόλια τόσο από το κοινό όσο και από κριτικούς του χώρου.

1.5.2 Δημιουργικότητα - πρωτοτυπία

Το ιδανικό σύστημα PCG θα δημιουργεί περιεχόμενο αντίστοιχο με το περιεχόμενο που δημιουργεί ένας designer σε θέμα πρωτοτυπίας και δημιουργικότητας. Όπως

είδαμε από τα παραδείγματα παραπάνω αυτό δεν ισχύει καθολικά ή στον βαθμό που θέλουμε πάντα. Είναι πολύ δύσκολη η καταγραφή και έκφραση της δημιουργικότητας με όρους που μπορεί να καταλάβει ένας αλγόριθμος. Αποτελεί ακόμα ένα ανοιχτό πρόβλημα στον χώρο της Τεχνητής Νοημοσύνης και επηρεάζει άμεσα τα αποτελέσματα του PCG.

Παρόλαυτα έχουν γίνει πολλές προσπάθειες και βελτιώσεις σε αυτό το χαρακτηριστικό του PCG ιδιαίτερα με την χρήση βαθιών νευρωνικών δικτύων. Επίσης το παραγόμενο περιεχόμενο θέλουμε να δίνει την εντύπωση ότι δεν δημιουργήθηκε από κάποιον αλγόριθμο, αλλά από κάποιον άνθρωπο. Αυτό το χαρακτηριστικό είναι πολύ δύσκολο να επιτευχθεί. Με την αύξηση της χρήσης του PCG στα παιχνίδια, οι παίκτες "έμαθαν" να ξεχωρίζουν το περιεχόμενο που παράγετε από αλγορίθμους και να προσπαθούν σε πολλές περιπτώσεις να το χρησιμοποιήσουν προς όφελος τους για να "παρακάμψουν" κανόνες του παιχνιδιού.

Για παράδειγμα στο παιχνίδι Mount and Blade II Bannerlord [10] υπάρχει η πιθανότητα το παιχνίδι να δημιουργήσει μάχες κατά την διάρκεια αποστολών για να τις κάνει πιο δύσκολες και ενδιαφέρον. Οι παίκτες γνωρίζοντας ότι το σύστημα του PCG υπολογίζει την πιθανότητα μάχης εκείνη την στιγμή, μπορούν να φορτώσουν το παιχνίδι σε κάποια προηγούμενη στιγμή και όταν φτάσουν στο σημείο της μάχης, το σύστημα να ξανά υπολογίσει την πιθανότητα, αυτή την φορά βγάζοντας αρνητική πιθανότητα μάχης.

1.5.3 Αξιοπιστία

Η αξιοπιστία του παραγόμενου περιεχομένου συνδέετε άμεσα με μια πολύ απλή ερώτηση: Είναι playable? Μπορεί δηλαδή το περιεχόμενο που παράχθηκε να προστεθεί στο παιχνίδι και να το χρησιμοποιήσει ο παίκτης όπως πρέπει ή δημιουργεί προβλήματα, για παράδειγμα ένα όπλο χωρίς σκανδάλη ή χωρίς την λειτουργικότητα της σκανδάλης είναι άχρηστο. Εκτός από το αν είναι χρήσιμο, το περιεχόμενο θα πρέπει να μην "σπάει" το παιχνίδι. Δηλαδή να μην παγιδεύει το παίκτη σε καταστάσεις από τις οποίες δεν μπορεί να συνεχίσει την πρόοδο του ή να του δίνει πλεονεκτήματα που σπάνε τους κανόνες του παιχνιδιού.

Για τον έλεγχο της αξιοπιστίας του παραγόμενου περιεχομένου υπάρχουν οι τα συστήματα αξιολόγησης (evaluators). Είναι αλγόριθμοι του συστήματος PCG και "αποφασίζουν" εάν το περιεχόμενο που παράχθηκε είναι playable ή όχι. Εάν το αξιολογήσουν ως unplayable, ο PCG αλγόριθμος πρέπει να δημιουργεί καινούργιο περιεχόμενο μέχρι κάποιο να περάσει την αξιολόγηση αξιοπιστίας.

1.5.4 Παραμετροποίηση

Σε όλους τους αλγόριθμους PCG δίνονται κάποιοι παράμετροι ως είσοδοι. Από αυτές τις παραμέτρους εξαρτώνται συγκεκριμένα χαρακτηριστικά που θα έχει το παραγόμενο αποτέλεσμα. Αυτή η δυνατότητα είναι πολύ σημαντική για τα συστήματα PCG που χτίζονται για να χρησιμοποιηθούν από τους designers του παιχνιδιού. Επιπλέον, αυτές οι είσοδοι καθορίζουν πόσο ντετερμινιστικό είναι ένα σύστημα PCG.

Για παράδειγμα στο παιχνίδι Oxygen Not Included (2019), ένα παιχνίδι προσομοίωσης και επιβίωσης, δημιουργείτε το επίπεδο στο οποίο ο παίκτης θα πρέπει να χτίσει την βάση του, χρησιμοποιώντας μια αλφαριθμητική μοναδική τιμή, ή όπως λέγετε στην επιστήμη της κρυπτογραφίας (seed). Αυτό το seed μπορεί να παραχθεί τυχαία από το παιχνίδι ή να το παρέχει στο σύστημα ως είσοδο ο παίκτης. Αυτή η δυνατότητα έχει ως αποτέλεσμα οι παίκτες να μπορούν να βρουν και να ανταλλάξουν seeds για επίπεδα που τους φάνηκαν πολύ ευνοϊκά ή πολύ δύσκολα. Το συγκεκριμένο χαρακτηριστικό παρατηρήθηκε να βελτιώνει την εμπειρία των παικτών με το παιχνίδι καθώς δημιουργήθηκε μια κοινότητα παικτών(community) που μοιραζόντουσαν seeds από επίπεδα με άλλους παίκτες και σύγκριναν τις εμπειρίες και τις επιδόσεις τους. [11]

1.6 Procedural Content Generation για 2D επίπεδα

Ένας τομέας με πολλές εφαρμογές του PCG είναι τα παιχνίδια δύο διαστάσεων (2D) και η δημιουργία επιπέδων (levels/dungeon) για αυτά. Ένα τέτοιο επίπεδο μπορεί να οριστεί ως ένας 2D χώρος, ο οποίος περιέχει δωμάτια ή τμήματα, χωρισμένα με διαχωριστικά ή άλλα εμπόδια. Ο παίκτης μπορεί να περιηγηθεί

στο επίπεδο με βάση τους κανόνες του κάθε παιχνιδιού, είτε μέσα από πόρτες και ανοίγματα ή ανοίγοντας τρύπες στα διαχωριστικά. Καθώς εξερευνάει μπορεί να συναντήσει αντικείμενα, εχθρούς, κρυφά περάσματα κ.ά.

Σε αυτή την εργασία αναπτύχθηκαν αλγόριθμοι και μοντέλα για την δημιουργία τέτοιων επιπέδων, συνεπώς είναι σημαντικό να αναλυθούν οι διάφοροι περιορισμοί και ιδιαιτερότητες αυτού του τομέα, καθώς και αντιπροσωπευτικές μέθοδοι και τα χαρακτηριστικά τους.

Η μελέτη και επιλογή αυτού του τομέα είναι ιδιαίτερα διαδεδομένη στην επιστημονική κοινότητα του PCG καθώς προσφέρει ένα χώρο αναζήτησης(search space) που μπορεί εύκολα να αναπαρασταθεί και να αποθηκευτεί σε διάφορες μορφές. Επίσης είναι εύκολη η αξιολόγηση και η δοκιμή τέτοιου περιεχομένου όπως θα αναλυθεί και παρακάτω.

1.6.1 Εισαγωγή

Όπως περιγράφετε και στα [13] [14], το PCG για την δημιουργία τέτοιων επιπέδων περιέχει την δημιουργία της τοπολογίας, της γεωμετρίας και των αντικειμένων του επιπέδου. Στο [14] γίνετε μια ανάλυση ενός συστήματος PCG σε τρία βασικά στοιχεία:

- **Μοντέλο αναπαράστασης** Αποτελεί μια απλοποιημένη και γενική αναπαράσταση ενός επιπέδου.
- **Μέθοδο δημιουργίας του μοντέλου αναπαράστασης** Ο αλγόριθμος που μετατρέπει τα επίπεδα στο *μοντέλο αναπαράστασης* όπως έχει οριστεί.
- **Μέθοδο δημιουργίας του επιπέδου από το μοντέλο αναπαράστασης** Ο αλγόριθμος που αναλαμβάνει να δημιουργήσει το επίπεδο στον χώρο του παιχνιδιού με βάση το *μοντέλο αναπαράστασης* που δημιούργησε η *μέθοδος δημιουργίας*. Αυτό είναι το αποτέλεσμα που θα παρουσιαστεί στον τελικό χρήστη του συστήματος.

Παρακάτω συνοψίζονται μερικές από τις πιο γνωστές οικογένειες αλγορίθμων PCG για 2D επίπεδα:

- **PCG με διαχωρισμό χώρου (Space Partitioning)**

- **PCG με την χρήση πρακτόρων (Agent-based)**
- **PCG με Cellular Automata**
- **PCG με τη χρήση γραμματικών (Generative Grammars)**

1.6.2 Space Partitioning

Αυτή η ομάδα αλγορίθμων χρησιμοποιείται και σε άλλες περιοχές του game development όπως τα γραφικά, οπότε οι μεθοδολογίες και οι δομές δεδομένων είναι γνωστά στους σχεδιαστές και προγραμματιστές παιχνιδιών. Όπως περιγράφει και το όνομα της, εκτελεί έναν διαχωρισμό του χώρου σε υποχώρους. Αυτό μπορεί να χρησιμοποιηθεί στο PCG για την δημιουργία δωματίων και διαδρόμων μέσα σε ένα επίπεδο. [12]

Ένας αλγόριθμος που χρησιμοποιείται πολύ σε υλοποιήσεις space partitioning είναι ο Binary Space Partitioning (BSP) [15] και παραλλαγές του όπως ο Quadtree space partitioning και ο Octree space partitioning. Θα αναλύσουμε τον βασικό αλγόριθμο του BSP, αναλυτικές περιγραφές για τους quadree και octree μπορούν να βρεθούν εδώ [16].

- **Binary Space Partitioning (BSP)** Ο αλγόριθμος είναι αναδρομικός και χτίζει το επίπεδο ιεραρχικά χρησιμοποιώντας ως δομή δεδομένων ένα Δυαδικό Δέντρο (Binary Tree). Στο αρχικό στοιχείο (root node) του Binary Tree περιέχεται "ολόκληρο" το επίπεδο. Στην πρώτη αναδρομή, το επιλεγμένο στοιχείο (node), το root σε αυτή την περίπτωση χωρίζεται σε δύο τμήματα με κάποιον ορισμένο διαχωρισμό, για παράδειγμα κάθετα. Αυτά τα δύο κομμάτια προστίθενται ως παιδιά (child nodes) στο root node και στη συνέχεια ο αλγόριθμος καλείται για κάθε ένα από τα παιδιά.

  Ο αλγόριθμος σταματάει όταν φτάσει σε κάποια προορισμένη τερματική συνθήκη, για παράδειγμα το μέγιστο βάθος δέντρου ή μόλις τελειώσουν τα nodes που υπάρχουν για διαχωρισμό. Για να προστεθεί τυχαιότητα στον αλγόριθμο, άρα και διαφοροποίηση στα παραγόμενα επίπεδα σε κάθε εκτέλεση, μπορεί να αποδοθεί μια πιθανότητα διαχωρισμού σε κάθε node. Αν η πιθανότητα είναι κάτω από ένα ορισμένο κατώφλι να μην γίνετε διαχωρισμός αυτού του node. Επιπλέον το κατώφλι μπορεί να διαμορφώνετε ανάλογα με το μέγεθος του παραγόμενου επιπέδου ή το βάθος που βρίσκεται αυτή τη στιγμή ο αλγόριθμος.

  Η εισαγωγή και δοκιμή τέτοιων παραμέτρων είναι ένα μεγάλο κομμάτι της υλοποίησης, και παίζει καθοριστικό ρόλο στο τελικό αποτέλεσμα. Τέτοιου

είδους παράμετροι μπορούν να χρησιμοποιηθούν από τους σχεδιαστές για να παράγουν διαφόρων ειδών επίπεδα.

Ένα σημαντικό χαρακτηριστικό αυτής της οικογένειας αλγορίθμων, είναι ότι δημιουργεί επίπεδα με δωμάτια τα οποία δεν υπερκαλύπτουν άλλα δωμάτια. Αυτό συμβαίνει προφανώς επειδή ένα δωμάτιο προέρχεται από τον διαχωρισμό ενός μεγαλύτερου δωματίου σε συγκεκριμένα τμήματα, το κάθε ένα εντελώς ξεχωριστό από το άλλο.

1.6.3 Agent-based

Όπως αναφέρει και το όνομα αυτής της οικογένειας, η δημιουργία του επιπέδου γίνετε μέσω ενός πράκτορα [18]. Οι πράκτορες αντίστοιχα με το Space Partitioning είναι συχνά χρησιμοποιούμενοι αλγόριθμοι και από άλλα κομμάτια του παιχνιδιού, όπως για παράδειγμα για την συμπεριφορά χαρακτήρων που δεν ελέγχει ο παίκτης (NPCs).

Σε αντίθεση με το Space Partitioning, οι Agent-Based προσεγγίσεις δεν βλέπουν ολόκληρο τον χώρο καθώς εκτελούνται, αλλά μόνο ένα συγκεκριμένο πεδίο γύρω από τον πράκτορα ανάλογα με τις δυνατότητες που του έχουμε δώσει. Αυτή η διαφορά, κάνει τα αποτελέσματα του Agent-Based PCG να είναι πιο χαοτικά και τυχαία από τα οργανωμένα επίπεδα που αναλύσαμε παραπάνω.

Για την δημιουργία ενός τέτοιου αλγόριθμου πρέπει να οριστεί μια συμπεριφορά για τον πράκτορα και στη συνέχεια να τον "ελευθερώσουμε" στο επίπεδο που θέλουμε να δημιουργήσει. Ο πράκτορας θα "περιπλανιέται" (wander) στον χώρο και με βάση παραμέτρους της συμπεριφορά του θα το αλλάζει με σκοπό να δημιουργήσει έναν χώρο που να είναι αποδεκτός ως επίπεδο στο παιχνίδι μας.

Για παράδειγμα μπορούμε να ορίσουμε ένα State Machine ως την συμπεριφορά του πράκτορα μας το οποίο αποτελείται από τα ακόλουθα States:

- **Περιπλάνηση (Wandering)** Όταν ο πράκτορας βρίσκεται σε αυτό το State, περιπλανιέται στο χώρο σε μια ευθεία. Με μια αυξανόμενη πιθανότητα μετά από κάθε κίνηση σε αυτό το State ο πράκτορας μπορεί να μεταβεί στο State Turn ή στο State Room Placement.
- **Στροφή (Turn)** Μόλις βρεθεί σε αυτό το State, ο πράκτορας επιλέγει μια καινούργια κατεύθυνση και ξεκινάει να κινείται προς αυτήν επιστρέφοντας στο State του Wandering.

• **Τοποθέτηση δωματίου (Room placement)** Σε αυτό το State, ο πράκτορας επιλέγει ένα δωμάτιο τυχαίων διαστάσεων και το τοποθετεί μπροστά του. Στη συνέχεια επιστρέφει στο State Wandering.

Με αυτά τα τρία πολύ απλά States, και τις παραμέτρους που επηρεάζουν τις μεταβάσεις και το μέγεθος των δωματίων έχουμε ορίσει έναν πράκτορα που μπορεί να δημιουργήσει επίπεδα με τυχαία διασκορπισμένα δωμάτια. Αντίστοιχα με το Space Partitioning, αυτές οι παράμετροι που καθορίζουν τις μεταβάσεις των States μπορούν να χρησιμοποιηθούν από designers για να δημιουργούν επίπεδα με τις παραμέτρους που τους βολεύουν.

Ένα μεγάλο πλεονέκτημα των Agent-based αλγορίθμων είναι ότι είναι παραλληλοποιήσιμοι. Δηλαδή μπορούμε να αρχικοποιήσουμε ένα επίπεδο με τέσσερις πράκτορες ώστε να έχουμε πιο γρήγορη δημιουργία και διαφορετικό αποτέλεσμα καθώς η εκτέλεση τεσσάρων πρακτόρων θα δημιουργήσει πολύ διαφορετικό περιεχόμενο απ ότι ο ένας πράκτορας. Αυτή η προσέγγιση συνοδεύεται από επιπλέον λογική στη συμπεριφορά των πρακτόρων για την συνεργασία ή αποφυγή της επικάλυψης του έργου του ενός πράκτορα από τους άλλους.

1.6.4 Cellular Automata

Τα Cellular Automata (ενικός: Cellular Automaton) [17] έχουν χρησιμοποιηθεί πέρα από την επιστήμη της Πληροφορικής, από την Φυσική και την Βιολογία για να προσομειώσουν μοντέλα ανάπτυξης και φυσικά φαινόμενα. Στον τομέα του PCG έχουν χρησιμοποιηθεί με μεγάλη επιτυχία για την δημιουργία δωματίων με πολύ φυσική σπηλαιώδη τοπολογία.

Το περιβάλλον δημιουργίας του επιπέδου αποτελεί ένα NxM επίπεδο (grid) αποτελούμενο από κελιά (cells), ένα σύνολο κανόνων μετάβασης και ένα σύνολο καταστάσεων. Αυτά τα τρία στοιχεία είναι αρκετά για να οριστεί μια υλοποίηση με Cellular Automata.

Ο αλγόριθμος των Cellular Automata εκτελείται σε επαναλήψεις, κάνοντας κάθε φορά αλλαγές μέχρι να φτάσει στο τελικό αποτέλεσμα του επιπέδου. Αυτό γίνετε μόλις φτάσει σε κάποια τερματική συνθήκη ή μόλις σταματήσει να καταγράφει αρκετές μεταβολές στις καταστάσεις των cells.

• **Grid** Το NxM grid αντιπροσωπεύει το επίπεδο που θέλουμε να υλοποιήσουμε, όπου το κάθε στοιχείο του ονομάζετε cell. Αρχικά για να μπορέσει να

λειτουργήσει ο αλγόριθμος πρέπει τα cells του grid να λάβουν αρχικές τιμές από τις διαθέσιμες καταστάσεις. Αυτό συνήθως γίνετε με τυχαία ανάθεση τιμών η οποία μπορεί να ακολουθεί κάποια κατανομή που θέλουμε να έχουν τα cells στο επίπεδο.

- **Κανόνες μετάβασης (Transition Rules)** Αυτοί οι κανόνες εφαρμόζονται σε κάθε επανάληψη σειριακά σε όλα τα cells του grid. Με βάση αυτούς τους κανόνες, σε συνδυασμό με την "γειτονιά" του cell αποφασίζετε αν θα αλλάξει η κατάσταση του και σε ποια κατάσταση θα μεταβεί.

- **Σύνολο καταστάσεων (Set of States)** Περιλαμβάνει όλες τις δυνατές καταστάσεις που μπορεί να έχει ένα cell ανά πάσα στιγμή. Μόνο μία κατάσταση μπορεί να ανατεθεί στο cell κάθε φορά.

Πολύ σημαντική παράμετρος για τα αποτελέσματα του αλγορίθμου είναι η "γειτονιά" του cell. Ως γειτονιά ορίζεται το σύνολο των cells που είναι κοντά στο cell που εξετάζουμε και επηρεάζουν την κατάσταση στην οποία θα μεταβεί. Για ένα μονοδιάστατο cell, γειτονιά ορίζετε ως τα cells που βρίσκονται δεξιά και αριστερά του σε X απόσταση. Αν το X = 1 τότε παίρνουμε μόνο τα cells που είναι ακριβώς δίπλα του. Σε δισδιάστατα cells υπάρχουν δύο γειτονιές ορισμένες στην βιβλιογραφία.

- **Moore Neighborhood** Η Moore Neighborhood μοιάζει με σταυρό, περιλαμβάνει τα cells που είναι πάνω, κάτω, δεξιά και αριστερά του cell που εξετάζουμε.

- **Von Neumann Neighborhood** Αυτή η γειτονιά περιλαμβάνει όλα τα cells που έχει και η Moore Neighborhood και τα cells που βρίσκονται διαγωνίως του cell που εξετάζουμε.

Και οι δύο γειτονιές, αντίστοιχα με τη μονοδιάστατη γειτονιά μπορούν να εκτείνονται κατά X cells μακρυά από το cell που εξετάζουμε. Δεν είναι κανόνας αλλά συνηθίζετε στις υλοποιήσεις το X να είναι 1.

Σε αυτή την εργασία χρησιμοποιήθηκαν μέθοδοι Cellular Automata σε συνδυασμό με το Space Partitioning όπως θα αναλυθεί στη συνέχεια.

1.6.5 Generative Grammars

Οι γραμματικές όπως ορίζονται στον τομέα του Game Development και γενικότερα στην επιστήμη της Πληροφορικής [19], αποτελούνται από μεταβλητές και κανόνες.

Οι μεταβλητές αντιπροσωπεύονται από μικρά λατινικά γράμματα και οι κανόνες από κεφαλαία. Οι κανόνες αποτελούν κανόνες μετατροπής ενός ή περισσότερων κεφαλαίων συμβόλων σε μια ακολουθία από μικρά και κεφαλαία σύμβολα αντίστοιχα.

Για παράδειγμα μπορούμε να ορίσουμε μια γραμματική με μεταβλητές το σύνολο a, b και κανόνες το σύνολο A -> a, B -> Ab. Συνεπώς αν η αρχική μας κατάσταση είναι η AB μόλις εφαρμόσουμε τους κανόνες μετατροπής θα έχουμε την κατάσταση AB -> aAb. Αντίστοιχα μπορούμε να συνεχίσουμε να εφαρμόζουμε τους κανόνες μέχρι να ικανοποιήσουμε κάποια τερματική συνθήκη ή να έχουμε μετατρέψει όλα τα κεφαλαία σύμβολα σε μικρά.

Αυτού του είδους οι γραμματικές είναι πολύ αποτελεσματικές στην δημιουργία φυτών, όπως δέντρα, θάμνους κ.ά. Συνεπώς η χρήση τους είναι ιδιαίτερα γνωστή στον κόσμο του game development αντίστοιχα με τους προηγούμενους αλγόριθμους. Γενικά παρατηρούμε ότι πολλοί αλγόριθμοι έχουν εφαρμογή σε περισσότερους από έναν τομέα.

Για την δημιουργία ενός επιπέδου με την χρήση γραμματικών η διαδικασία είναι λίγο διαφορετική από αυτές που έχουμε δει ως τώρα. Μια ακόμα χρήση των Generative Grammars είναι η δημιουργία αποστολών (Quests). Επειδή υπάρχει μια λογική ακολουθία στην εξέλιξη μιας αποστολής, για παράδειγμα:
Νίκησε τον εχθρό στο δωμάτιο 1 -> Πάρε το κλειδί που κουβαλούσε -> Ξεκλείδωσε το δωμάτιο 2
Η χρήση γραμματικών για την δημιουργία γράφων που αντιπροσωπεύουν αποστολές σε ένα παιχνίδι είναι μια μέθοδος που χρησιμοποιείται. Κατά συνέπεια, εφόσον έχουμε τον συνδεδεμένο γράφο της αποστολής μπορούμε να υλοποιήσουμε έναν αντίστοιχο γράφο που να αντιπροσωπεύει το επίπεδο που θα εκτελεστεί αυτή η αποστολή. Μπορούμε να βάλουμε επιπλέον δωμάτια και διαδρόμους ώστε να φαίνεται πιο ”γεμάτο” το επίπεδο αλλά πρέπει να είμαστε προσεκτικοί και να ακολουθήσουμε ακριβώς την ιεραρχία του γράφου της αποστολής ώστε να μην καταλήξουμε με κάποιο επίπεδο που δεν μπορεί να εκτελεστεί η αποστολή. Ένα τέτοιο παράδειγμα επιπέδου θα ήταν εάν στο παραπάνω σενάριο, το δωμάτιο 1 ήταν μετά το δωμάτιο 2 και δεν υπήρχε τρόπος για τον χρήστη να τον φτάσει παρά μόνο μέσω του δωματίου 2.

Βλέπουμε εδώ μια πιο γενική και αφηρημένη αναπαράσταση ενός επιπέδου, το οποίο έχει διάφορα πλεονεκτήματα. Συγκεκριμένα δίνει μια μεγαλύτερη ελευθερία για τυχαιότητα αλλά επειδή υπάρχει μια συγκεκριμένη ιεραρχία κάποιων δωματίων δεν φαίνεται εντελώς τυχαίο στον παίκτη. Αντίθετα μπορεί να υποθέσει ότι κάποιος άνθρωπος το σχεδίασε ώστε να έχει μια λογική συνοχή.

Επιπλέον επειδή οι αναπαραστάσεις δεν είναι στενά συνδεδεμένες με το παιχνίδι

και τη δομή του μπορούν να μεταφερθούν αυτούσιες σε κάποιο άλλο παιχνίδι αντίστοιχου τύπου και να αναπαρασταθούν σε αυτό. Προσφέρει δηλαδή μια πιο γενική και επαναχρησιμοποιούμενη αναπαράσταση από τις προηγούμενες μεθόδους.

1.7 Αξιολόγηση (Evaluation)

Είδαμε μερικούς από τους αλγόριθμους που μπορούν να χρησιμοποιηθούν για την δημιουργία περιεχομένου. Υπάρχουν πολλοί περισσότεροι, ο καθένας με τις παραλλαγές του. Συμπεραίνουμε ότι είναι πολύ εύκολο να δημιουργήσουμε περιεχόμενο, το δύσκολο κομμάτι του PCG είναι η δημιουργία **καλού** περιεχομένου. Σε αυτή την ενότητα θα αναλύσουμε πως γίνετε την αξιολόγηση PCG συστημάτων με τη χρήση άλλων συστημάτων, των Evaluators. [20]

1.7.1 Πόσο σημαντική είναι η αξιολόγηση

Τα νούμερα και οι μετρήσεις αποτελούν τις βάσεις κάθε επιστήμης. Αυτά δείχνουν την πορεία ενός αλγορίθμου από την αρχική του σχεδίαση μέχρι την τελευταία του αλλαγή. Αποδίδουν την ιστορικότητα και την βελτίωση ή χειροτέρευση μιας κατάστασης με πολύ συμπυκνωμένο και κατανοητό τρόπο για τους ανθρώπους που την παρακολουθούν. Αντίστοιχα και στην επιστήμη του PCG χρειαζόμαστε μεθόδους για να μετράμε την πορεία και την απόδοση του συστήματος. Αυτή την εργασία την αναλαμβάνουν οι Evaluators όπως έχει ήδη αναφερθεί. Συνεπώς μέσα στην σχεδίαση ενός καλού PCG συστήματος περιλαμβάνετε και η υλοποίηση ενός καλού Evaluator για αυτό το σύστημα [21]. Ένας Evaluator έχει τους παρακάτω στόχους:

- Να κατανοήσουμε καλύτερα τις δυνατότητες του PCG. Μπορούμε να εφαρμόσουμε διάφορες μετρικές πάνω στο παραγόμενο περιεχόμενο και να αξιολογήσουμε τα αποτελέσματα συνολικά σε αντίθεση με το να κοιτάμε, ή να ακούμε, στην περίπτωση του ήχου, κομμάτια περιεχομένου που παρήγαγε το PCG.
- Να δημιουργήσουμε playable περιεχόμενα. Ορίζοντας κανόνες και περιορισμούς που πρέπει να εφαρμόζει το παραγόμενο περιεχόμενο, μπορούμε να υλοποιήσουμε έναν Evaluator ο οποίος να εγκρίνει ή να απορρίπτει περιεχόμενο ανάλογα με το τι έχουμε ορίσει.

- Να βελτιώσουμε την διαδικασία υλοποίησης του συστήματος PCG. Η υλοποίηση ενός τέτοιου συστήματος αποτελεί το αποτέλεσμα συνεχών βελτιώσεων μετά από πολλές επαναλήψεις δοκιμής και λάθους (trial and error). Με ένα αυτόματο σύστημα αξιολόγησης των αποτελεσμάτων αυτή η διαδικασία μπορεί να επιταχυνθεί.

- Να συγκρίνουμε διαφορετικά συστήματα PCG μεταξύ τους ή παραλλαγές του ίδιου συστήματος. Έχοντας ένα κοινό σύστημα σύγκρισης, υπάρχει η δυνατότητα να μετρήσουμε και να συγκρίνουμε τις αποδόσεις πολλών διαφορετικών συστημάτων καθώς και του ίδιου συστήματος με διάφορες παραλλαγές. Θα πρέπει να προσέξουμε το σύστημα αξιολόγησης να μην έχει κάποιο προτίμηση(bias) που ευνοεί κάποιο είδος αλγορίθμου περισσότερο έναντι άλλων, θα πρέπει να αξιολογεί τα συστήματα μόνο με βάση το παραγόμενο αποτέλεσμα.

Σχετικά με το τελευταίο στόχο που αναλύθηκε, η πολυπλοκότητα ενός αλγορίθμου δεν περιλαμβάνετε στο σύστημα evaluator. Αυτό αφορά μόνο την αξιολόγηση του τελικού αποτελέσματος. Η χρονική, χωρική ή άλλη πολυπλοκότητα του κάθε αλγορίθμου είναι πολύ σημαντική και παίζει τεράστιο ρόλο για την υλοποίηση ενός τέτοιου συστήματος αλλά είναι κομμάτι διαφορετικής αξιολόγησης και δεν θα πρέπει να μπερδεύεται με την αξιολόγηση του περιεχομένου που παράγει το κάθε σύστημα.

1.7.2 Χαρακτηριστικά Αξιολόγησης

Όπως ορίστηκε και παραπάνω μπορούμε να κατηγοριοποιήσουμε το περιεχόμενο ανάλογα με το αν είναι playable ή όχι. Εάν μπορεί δηλαδή να προστεθεί στο παιχνίδι χωρίς να το "σπάει" (without breaking it). Σε αυτό το κομμάτι θα δούμε τι άλλα χαρακτηριστικά πρέπει να έχει το παραγόμενο περιεχόμενο για να θεωρηθεί καλό και κατά επέκταση τι χαρακτηριστικά πρέπει να έχει το σύστημα PCG μας για να θεωρείται αντίστοιχα αξιόπιστο και καλό.

Πέρα από την ικανότητα του συστήματος να παράγει playable περιεχόμενο, εξετάζουμε επίσης την ικανότητα να παράγει πρωτότυπο περιεχόμενο. Δηλαδή τη "δημιουργικότητα", όπως αν ήταν κάποιος άνθρωπος και έπρεπε να σχεδιάσει επίπεδα για το παιχνίδι μας, θα κρίναμε την πρωτοτυπία του κάθε επιπέδου και αν όλα μοιάζουν μεταξύ τους. Περιεχόμενο που είναι πρωτότυπο και φαίνεται ξεχωριστό μπορούμε να το χαρακτηρίζουμε ως καινοτόμο (novel). Ένας ακόμα πιο δύσκολος στόχος είναι η δημιουργία ενός συστήματος που παράγει playable **και** novel περιεχόμενο. Επίσης δύσκολη είναι η δημιουργία ενός συστήματος

αξιολόγησης που να μπορεί να αναγνωρίζει αυτά τα χαρακτηριστικά στα δεκάδες, καμιά φορά χιλιάδες δείγματα παραγόμενου περιεχομένου, όπως θα δούμε στη συνέχεια.

Όπως είπαμε το novelty είναι το πόσο πρωτότυπο και ιδιαίτερο είναι ένα επίπεδο, σε σύγκριση με άλλα επίπεδα. Είναι από τα πιο δύσκολα χαρακτηριστικά του συστήματος αξιολόγησης για να ορίσουμε και να υλοποιήσουμε. Δεν υπάρχει αλγοριθμική έκφραση για την δημιουργικότητα και πως να την μετρήσουμε. Υπάρχουν διάφορες προσεγγιστικές λύσεις οι οποίες επίσης είναι πολύ στενά συνδεδεμένες με το PCG σύστημα που αξιολογούν και το παιχνίδι για το οποίο σχεδιάστηκαν οπότε η μεταφορά τους σε άλλα συστήματα είναι δύσκολη και καμιά φορά αδύνατη. Αυτό το κομμάτι παραμένει ένα από τα πιο σημαντικά άλυτα προβλήματα του τομέα του PCG. Παρόλαυτα, βλέποντας τις πιο πρόσφατες εξελίξεις στον χώρο των Deep Neural Networks, όπως η μεταφορά στιλιστικού περιεχομένου (style transfer) [22], μπορεί κανείς να οραματιστεί τις προοπτικές που ανοίγονται και για τον τομέα του PCG.

Πέρα από την δημιουργία novel περιεχομένου, υπάρχουν κάποιοι αυστηροί περιορισμοί τους οποίους πρέπει να ακολουθεί το σύστημα PCG . Αυτοί οι περιορισμοί είναι στενά συνδεδεμένοι με το είδος του παιχνιδιού που δημιουργούμε και με το είδος του περιεχομένου που θέλουμε να φτιάξουμε. Για παράδειγμα, ένας αυστηρός περιορισμός για ένα 2D επίπεδο μπορεί να είναι ότι όλα τα δωμάτια πρέπει να έχουν τουλάχιστον μία πόρτα, ώστε να είναι προσβάσιμα από τον παίκτη. Αυτόν τον περιορισμό μπορούμε να τον κωδικοποιήσουμε μέσα στο σύστημα PCG για να δημιουργεί δωμάτια που τον ικανοποιούν και επίσης να τον ορίσουμε και μέσα στο σύστημα αξιολόγησης για να απορρίπτει δωμάτια που δεν τον ικανοποιούν. Αυτός ο περιορισμός είναι πολύ χρήσιμος για να ορίσουμε εάν ένα επίπεδο είναι playable για 2D ή και 3D διαστάσεις, παρόλαυτα δεν μπορεί να χρησιμοποιηθεί σε PCG συστήματα που δημιουργούν αντικείμενα, όπως όπλα. Βάζοντας πολλούς περιορισμούς σε ένα PCG σύστημα μπορούμε να πιστοποιήσουμε ότι τα αποτελέσματα είναι playable αλλά επίσης ταυτόχρονα μπορεί να περιορίζουμε πολύ το είδος και την πρωτοτυπία των αποτελεσμάτων, και κατά συνέπεια το novelty.

Ένα ακόμα χαρακτηριστικό που είναι επιθυμητό αλλά πολύ δύσκολο να οριστεί και να αξιολογηθεί είναι η αντίδραση του παίκτη στο περιεχόμενο που παράχθηκε από το σύστημα μας. Ο παίκτης είναι ο τελικός αξιολογητής του περιεχομένου και κατά επέκταση ολόκληρου του παιχνιδιού, η δυνατότητα να μπορέσουμε να προβλέψουμε την αντίδραση του πριν την κυκλοφορία του παιχνιδιού, ενώ ακόμα το υλοποιούμε είναι ίσως από τα πιο μεγάλα ”όνειρα” της βιομηχανίας του game development. Όπως είδαμε και σε προηγούμενα παραδείγματα, ένα ”κακό” PCG σύστημα, μπορεί να φέρει αρνητικές κριτικές για ολόκληρο το παιχνίδι

και να αποθαρρύνει πολύ το κοινό από το να το δοκιμάσει.

1.7.3 Είδη Αξιολόγησης

Υπάρχουν δύο ειδών αξιολογήσεις που μπορούμε να εφαρμόσουμε σε ένα σύστημα PCG.

- **Top Down Evaluation** Η **top down** αξιολόγηση είναι το σύστημα που περιγράψαμε σε μερικά από τα παραδείγματα παραπάνω. Χρησιμοποιεί μετρήσεις για να αξιολογήσει και να συγκρίνει το κάθε σύστημα PCG.

- **Bottom Up Evaluation** Η **Bottom up** αξιολόγηση στηρίζετε στην αξιολόγηση των παικτών για να βγάλει συμπεράσματα. Περιλαμβάνει την δοκιμή του παιχνιδιού από ομάδες ανθρώπων με συγκεκριμένα χαρακτηριστικά και την καταγραφή των αντιδράσεων και της εμπειρίας τους για το παιχνίδι.

Η **Bottom up** αξιολόγηση μπορεί να γίνει μόνο σε κάποιο τελικό στάδιο της υλοποίησης του παιχνιδιού και του συστήματος PCG καθώς χρειάζεται κάτι που να μπορεί να δοκιμάσει ο παίκτης. Επίσης απαιτεί πολύ καλή γνώση των ανθρώπων που συμμετέχουν στην αξιολόγηση, κάποια βασικά χαρακτηριστικά όπως φύλο, ηλικία και κάποια χαρακτηριστικά σχετικά με τις προτιμήσεις τους σε παιχνίδια, καθώς όλα αυτά επηρεάζουν τα συμπεράσματα που θα βγάλουμε στην **Bottom up** αξιολόγηση.

Και οι δύο αξιολογήσεις είναι χρήσιμες για να υλοποιήσουμε ένα καλό σύστημα PCG αλλά πολλές εταιρείες και ερευνητικές ομάδες αυτού του τομέα δεν μπορούν να διαθέσουν τους πόρους που απαιτούνται και για τις δύο αξιολογήσεις.

Κεφάλαιο 2

Procedural Content Generation με Unity

Την υλοποίηση αυτής της εργασίας μπορούμε να την χωρίσουμε σε δύο ξεχωριστά εννοιολογικά κομμάτια. Το πρώτο κομμάτι αφορά την υλοποίηση ενός *Procedural Content Generation* συστήματος όπως αυτά που περιγράψαμε στο προηγούμενο κεφάλαιο, και σε ένα *Machine Learning Content Generation* σύστημα. Σε αυτό το κεφάλαιο αναλύουμε την τεχνική υλοποίηση του PCG.

Από το στάδιο της αρχικής ιδέας μέχρι την έναρξη της υλοποίησης μπορούμε να διακρίνουμε ένα ενδιάμεσο στάδιο, του σχεδιασμού και της έρευνας σχετικά με τις τεχνολογίες, τις δομές δεδομένων και τους αλγορίθμους που θα κληθούμε να υλοποιήσουμε. Αυτή η ιεραρχία από την ιδέα έως την υλοποίηση, ισχύει για όλες τις εφαρμογές της επιστήμης της πληροφορικής και ειδικότερα του game development *gamedevprocess*, όπου έχουμε πολλά υποσυστήματα που θα πρέπει να λειτουργήσουν παράλληλα και να συνεργαστούν με το σύστημα του PCG. Η σωστή οργάνωση και σχεδίαση της αρχιτεκτονικής του PCG είναι ένα πολύ κρίσιμο κομμάτι.

Κατά την υλοποίηση και ιδιαίτερα κατά το στάδιο της αξιολόγησης όπως είδαμε μπορεί να προκύψουν προβλήματα όπως μη αποδεκτά αποτελέσματα, γεγονός που μπορεί να οδηγήσει στον ανασχεδιασμό και την επιλογή διαφορετικών αλγορίθμων ή παραμέτρων. Αυτή η διαδικασία μπορεί να επαναληφθεί πολλές φορές καθώς, όπως αναλύθηκε, τα *καλά* συστήματα PCG βγαίνουν μέσα από δοκιμές και λάθη (*trial and error*).

Η υλοποίηση ενός τέτοιου συστήματος μπορεί να θεωρηθεί, και είναι και η

προσωπική εμπειρία της συγγραφέας, ως μια πολύ δημιουργική εργασία, ειδικά σε σύγκριση με την διαδικασία ανάπτυξης άλλων ειδών λογισμικού. Προβλήματα που απαιτούν σκέψη *outside of the box* και δημιουργικότητα, σε συνδυασμό με πολύ καλή γνώση της θεωρίας και της τεχνολογίας είναι ένα από τα χαρακτηριστικά που κάνουν αυτόν τον τομέα της πληροφορικής, τόσο ενδιαφέρον και ιδιαίτερο.

2.1 Game Engines

Οι Μηχανές Δημιουργίας Ηλεκτρονικών Παιχνιδιών (*Game Engines*) είναι σχεδιαστικά περιβάλλοντα υλοποίησης παιχνιδιών, όπως περιγράφει και το όνομα τους. Είναι πολύπλοκα υπολογιστικά συστήματα που χρησιμοποιούνται από εταιρείες και ομάδες ανθρώπων σε ολόκληρο τον κόσμο για την υλοποίηση ηλεκτρονικών παιχνιδιών. Συνήθως περιλαμβάνουν συστήματα που βοηθάνε τους προγραμματιστές και σχεδιαστές, όπως το σύστημα προσομοίωσης νόμων της φυσικής όπως κίνηση αντικειμένων, βαρύτητα κ.ά. (Physics Engine). [26]

Συστήματα που έχουν τα περισσότερα Game Engines είναι το σύστημα για την καταγραφή και επεξεργασία των εισόδων του χρήστης όπως το πάτημα ενός κουμπιού, η κίνηση του ποντικιού κ.ά (Input Engine), σύστημα για την εμφάνιση γραφικών (*textures*, *sprites*) (Graphics Engine), σύστημα για την κίνηση αντικειμένων με γραφικά (*animations*)(Animation Engine) και πολλά ακόμη.

Υπάρχουν Game Engines που εξειδικεύονται στην δημιουργία ενός συγκεκριμένου είδους παιχνιδιού, όπως για παράδειγμα το *Hero Engine* εξειδικεύεται στην δημιουργία παιχνιδιών που παίζονται μέσω Internet (*Online video games*). Αυτό σημαίνει ότι το Hero Engine έχει πολύ καλά ανεπτυγμένα συστήματα για την επικοινωνία *client-server* εφαρμογών σε πραγματικό χρόνο καθώς και την διαχείριση δεδομένων που αποθηκεύονται κεντρικά σε κάποιο *servers*.

Επίσης υπάρχουν και τα Game Engines που δεν φαίνεται να έχουν κάποια εξειδίκευση σε κανένα είδος, αλλά υποστηρίζουν το κάθε είδους game που θέλουμε να αναπτύξουμε. Αυτά τα Game Engine έχουν γνωρίσει μεγάλη δημοτικότητα τα τελευταία χρόνια, σε συνδυασμό με το γεγονός ότι είναι ελεύθερα διαθέσιμα, με αποτέλεσμα να διευρυνθεί η χρήση τους με μεγάλη επιτυχία από ομάδες όλων των μεγεθών (μικρά independent studio και developers μέχρι μεγάλες εταιρείες). Τέτοια Game Engines είναι το Unity Engine [23], το Unreal Engine [24], το Godot Engine (το οποίο είναι OpenSource) [25] κ.ά.

Η εκμάθηση ενός τέτοιου προγράμματος απαιτεί χρόνο και μελέτη, καθώς το κάθε Game Engine έχει δικές του υλοποιήσεις για το κάθε σύστημα, διαφορετικό

περιβάλλον αλληλεπίδρασης με τον χρήστη (*editor*) καθώς και διαφορετική γλώσσα προγραμματισμού που πρέπει να χρησιμοποιήσει ο προγραμματιστής. Υπάρχουν πολλά κριτήρια για την επιλογή ποιο Game Engine θα χρησιμοποιηθεί για την υλοποίηση ενός παιχνιδιού ή συστήματος, όπως στη δική μας περίπτωση. Τα πιο σημαντικά είναι:

- Οι γνώσεις της ομάδας (Game Engine, γλώσσα προγραμματισμού)
- Οι δυνατότητες που προσφέρει το Game Engine για τον συγκεκριμένο τύπο παιχνιδιού
- Το κόστος ανάπτυξης στο συγκεκριμένο Game Engine

• **Οι γνώσεις της ομάδας** Εάν έχουν προηγούμενη εμπειρία με κάποιο από τα διαθέσιμα Game Engines και με τις υποστηριζόμενες γλώσσες προγραμματισμού

• **Οι δυνατότητες του Game Engine** Εάν ο στόχος είναι η υλοποίησης μιας εφαρμογής για κινητές συσκευές, πρέπει να επιλεχθεί ένα Game Engine που να υποστηρίζει αυτή την πλατφόρμα.

• **Το κόστος του Game Engine** Πολλά Game Engines δεν παρέχονται ελεύθερα ή για να έχουμε πρόσβαση σε κάποιες λειτουργίες τους χρειάζεται η καταβολή κάποιου ποσού.

Σε αυτή την υλοποίηση επιλέχθηκε το **Unity Game Engine** καθώς η συγγραφέας το γνωρίζει όπως και την γλώσσα προγραμματισμού **C#** που χρησιμοποιήθηκε. Επιπλέον παρέχετε δωρεάν υπό προϋποθέσεις.

2.2 Unity Game Engine

2.2.1 Τεχνικά χαρακτηριστικά

Για την υλοποίηση χρησιμοποιήθηκε **Unity 2019.2.8f1** με **C#** και **API compatibility Level .Net Standard 2.0**.

2.2.2 Βιβλιοθήκες Unity

Χρησιμοποιήθηκαν κάποιες έτοιμες βιβλιοθήκες και συστήματα που παρέχει το Unity Game Engine και η γλώσσα **C#**. Συγκεκριμένα έγινε χρήση των:

- **Tilemap System** Είναι ένα από τις πιο πρόσφατες προσθήκες στα διαθέσιμα συστήματα του Unity. Παρέχει μεθόδους για την τοποθέτηση, προβολή και επεξεργασία δισδιάστατων γραφικών (*sprites*) ως κομμάτια ενός ορισμένου δισδιάστατου χώρου. [28]

- **Canvas System** Ελέγχει την τοποθέτηση και λειτουργικότητα των *controls* με τα οποία ο χρήστης αλληλεπιδρά με την εφαρμογή. [29]

2.3 Σχεδιασμός συστήματος PCG

Πριν από το στάδιο του σχεδιασμού ολοκληρώθηκε μια έρευνα πάνω στο αντικείμενο του *Procedural Content Generation* [30] [31] [32], [33] και του *Machine Learning Content Generation [34]* και ειδικά στις εφαρμογές που σχετίζονταν με το *Dungeon Generation* και το *Level Generation* [17], [35], [36]. Διαπιστώθηκε ότι παρόλο που η βιβλιογραφία πάνω στους αλγόριθμους που μπορούν να εφαρμοστούν είναι ιδιαίτερα εκτενής και αναλυτική, τα διαθέσιμα *dataset* που ενδείκνυνται για αυτό το σκοπό είναι ελάχιστα.

Αποφασίσαμε να επεκτείνουμε την υλοποίηση ώστε να περιλαμβάνει και ένα σύστημα *Procedural Content Generation* για *2D Levels* ώστε να είμαστε σε θέση να δημιουργήσουμε εμείς το *dataset* που θα χρησιμοποιήσουμε για την εκπαίδευση των *Machine Learning* μοντέλων. Με αυτό τον τρόπο μπορούμε να ελέγξουμε απόλυτα το μέγεθος, το είδος αναπαράστασης και τα δεδομένα που περιέχει το *dataset* μας, ώστε να πετύχουμε το καλύτερο δυνατό αποτέλεσμα με την εκπαίδευση.

Θεωρούμε ότι αυτή η προσέγγιση καλύπτει πιο σφαιρικά το πρόβλημα της δημιουργίας περιεχόμενου καθώς το προσεγγίζει όπως θα το προσέγγιζε και μια εταιρεία που ήθελε να κατασκευάσει ένα τέτοιο σύστημα. Για να εκπαιδεύσει τα μοντέλα Μηχανικής Μάθησης να παράγουν επίπεδα που θα μπορούσε να χρησιμοποιήσει σε κάποιο παιχνίδι, θα δημιουργούσε το απαραίτητο dataset. Η δημιουργία του dataset θα γινόταν είτε από τους σχεδιαστές, θα συγκεντρώνοντας παλιότερα assets ή μέσω ενός PCG συστήματος.

Ταυτόχρονα αυτό μας έδωσε την δυνατότητα να ερευνήσουμε σε ακόμα μεγαλύτερο

βάθος την θεωρία και τους αλγορίθμους του PCG. Η υλοποίηση αυτή αποτελεί από μία οπτική τον *dataset generator* του *Machine Learning* μοντέλου. Παράλληλα αναπτύξαμε και μηχανισμούς αξιολόγησης και προβολής των αποτελεσμάτων και των δύο συστημάτων (PCG, MLPG).

2.3.1 Περιγραφή Επιπέδου

Το επίπεδο που προσπαθούμε να δημιουργήσουμε όπως αναφέρθηκε είναι δισδιάστατο και αποτελείται από τετράγωνα (*tiles*). Το μέγεθος του μετριέται σε tiles ανά πλευρά του επιπέδου. Στην υλοποίηση του συστήματος PCG ο αριθμός των tiles ανά πλευρά είναι παράμετρος που μπορεί να αλλάξει αλλά για την εκπαίδευση του MLPG χρησιμοποιήθηκε dataset με σταθερό μέγεθος επιπέδου για όλα τα δείγματα.

Κάθε tile μπορεί να πάρει μία από τις παρακάτω τιμές:

- Τοίχος (*Wall*)
- Διάδρομος (*Corridor*)
- Δωμάτιο(*Room*)

Με αυτά τα τρία είδη tiles θέλουμε να κατασκευάσουμε επίπεδα που περιέχουν έναν αριθμό από δωμάτια, τα οποία περικλείονται από τοίχους και συνδέονται μεταξύ τους με διαδρόμους. Η τοποθέτηση αυτών των tiles σε σωστές θέσεις ώστε να σχηματίζονται επίπεδα που ταιριάζουν στην παραπάνω περιγραφή είναι ο γενικός στόχος των συστημάτων PCG και MLPG που προσπαθούμε να υλοποιήσουμε.

2.4 Υλοποίηση συστήματος PCG

2.4.1 Υποσυστήματα

Το σύστημα PCG αποτελείται από πολλά υποσυστήματα, τα οποία θα αναλύσουμε σε αυτό το κομμάτι. Το κάθε υποσύστημα έχει σχεδιαστεί με την λογική ότι αποτελεί ένα μεμονωμένο κομμάτι λογισμικού, παίρνει συγκεκριμένες εισόδους από άλλα υποσυστήματα ή τον χρήστη και παράγει εξόδους αντίστοιχα για τα άλλα υποσυστήματα και τον χρήστη. Ονομαστικά αυτά τα υποσυστήματα είναι:

- **Configuration System** Σύστημα αποθήκευσης και ανάγνωσης ρυθμίσεων.
- **Generation System** Περιέχει κλάσεις και μεθόδους για την δημιουργία ενός ή παραπάνω επιπέδων.
- **Data System** Περιέχει τις δομές δεδομένων, τις abstract κλάσεις και τα enumerations που αναπαριστούν όλες τις πληροφορίες που διαχειρίζεται το σύστημα PCG.
- **IO System** Παρέχει μεθόδους για την αποθήκευση των δημιουργημένων επιπέδων σε διάφορες μορφές όπως σε μορφή γράφουν ή σε αρχείο. Επίσης παρέχει μεθόδους για την ανάγνωση επιπέδων και μετατροπή τους σε κάποια άλλη μορφή αναπαράστασης.
- **UI System** Περιέχει την λειτουργικότητα του συστήματος αλληλεπίδρασης με τον χρήστη.
- **Evaluation System** Είναι υπεύθυνο για την αξιολόγηση ενός επιπέδου με βάση συγκεκριμένους κανόνες που ορίζονται κατά την υλοποίηση.

2.4.2 Αλγόριθμος PCG

Ως βασικός αλγόριθμος για το σύστημα δημιουργίας επιπέδων 2D χρησιμοποιήθηκε ένας αλγόριθμος *Space Partitioning*. Ο αλγόριθμος παίρνει τις διαστάσεις του επιπέδου, οι οποίες όπως είπαμε είναι σταθερές για κάθε επίπεδο, και χωρίζει το δωμάτιο οριζόντια σε τυχαίο αριθμό από παράλληλα επίπεδα. Στη συνέχεια για κάθε παράλληλο επίπεδο το χωρίζει κάθετα σε τυχαία δωμάτια. Ο αριθμός των παράλληλων και των κάθετων διαχωρισμός επιλέγετε τυχαία με βάση το μέγεθος του επιπέδου.

Με αυτό τον αλγόριθμο μπορούμε να έχουμε επίπεδα που ακολουθούν ένα συγκεκριμένο μοτίβο αλλά διαφέρουν στον αριθμό και το μέγεθος των δωματίων που έχει το καθένα όπως θα δούμε και στα παραδείγματα των δημιουργημένων επιπέδων.

Όλη η διαδικασία δημιουργίας του επιπέδου γίνετε άμεσα χωρίς να υπάρχει κάποια εμφανή καθυστέρηση στον χρήστη. Για την δημιουργία του *dataset* υλοποιήθηκε η επιλογή ο αλγόριθμος PCG να τρέξει για έναν αριθμό επαναλήψεων, προκαθορισμένο από τον χρήστη ώστε να παράγει μαζικά πολλά επίπεδα.

2.4.3 Βήματα αλγορίθμου PCG

Τα βήματα του αλγορίθμου PCG αναλυτικά είναι:

1. Γέμισμα όλου του επιπέδου με ένα είδος tile. Αυτό το tile είναι του είδους **Room**.
2. Ανάθεση των tiles που είναι στις άκρες του επιπέδου με tiles είδους **Wall**
3. Ανάθεση των tiles που είναι στις άκρες του επιπέδου, δίπλα στα tiles με είδος **Wall** με tiles είδους **Corridor**.
4. Υπολογισμός δωματίων.
5. Τοποθέτηση δωματίων στο επίπεδο. Σε αυτό το βήμα ο τα tiles που δεν είναι είδους **Room** γίνονται tiles είδους **Corridor**.
6. Τοποθέτηση tiles είδους **Wall** γύρω από τα δωμάτια.
7. Αποθήκευση παραγομένου επιπέδου σε αρχείο.
8. Αξιολόγηση του επιπέδου. Προαιρετικό βήμα.

2.4.4 Κανόνες μετατροπής

Για τα βήματα *2*, *5* και *6* όπου υλοποιείται η μετατροπή *tiles* από ένα είδος σε ένα άλλο χρησιμοποιήθηκαν κανόνες μετατροπής. Οι κανόνες αυτοί παίρνουν ως είσοδο ένα tile και ανάλογα με το είδος του κανόνα το μετατρέπουν σε κάποιο άλλο είδος tile ή το αφήνουν ίδιο.

Υλοποιήθηκαν δύο κατηγορίες κανόνων. Η πρώτη κατηγορία περιέχει κανόνες όπως αυτοί που αναλύσαμε στο κομμάτι σχετικά με τα **Cellular Automata** που χρησιμοποιούν την *γειτονιά* του tile για την μετατροπή. Η δεύτερη κατηγορία χρησιμοποιεί όρια μιας περιοχής για να εφαρμόσει μετατροπές στο είδος των tiles αυτής της περιοχής.

Για την πρώτη κατηγορία, υλοποιήθηκε η λογική της *γειτονιάς* ενός *tile* (cell) και ορίστηκε σε μια γενική μορφή (*abstraction*) η έννοια του κανόνα. Ο κανόνας ως *abstract class* μπορεί να εφαρμοστεί σε ένα *tile* και ανάλογα με την υλοποίηση που έχει γίνει και την *γειτονιά* του tile να γίνει μετατροπή του tile από ένα είδος σε ένα άλλο. Για τις ανάγκες αυτής της υλοποίησης χρησιμοποιήθηκε η **Moore**

Neighborhood και η **Von Neumann Neighborhood** ανάλογα με τις ανάγκες κάθε κανόνα. Η γειτονιά μας σε κάθε περίπτωση είναι απόστασης $X = 1$.

Η ίδια γενική μορφή (*abstraction*) ακολουθήθηκε και για την υλοποίηση των κανόνων της δεύτερης κατηγορίας. Τα όρια μπορεί να είναι τα όρια του επιπέδου, σε αυτή την περίπτωση ο κανόνας που θα εφαρμοστεί να μετατρέψει μόνο τα tiles περιμετρικά του επιπέδου σε ότι είδος δώσουμε ως παράμετρο όταν εφαρμόσουμε τον κανόνα.

Οι κανόνες που υλοποιήθηκαν για αυτή την εφαρμογή είναι οι εξής:

- **WALL FROM BOUNDS** Κανόνας μετατροπής σε **Wall** των tiles που είναι στα όρια μιας περιοχής. Χρησιμοποιείται για να γεμίσει περιμετρικά το επίπεδο με **Wall** tiles.
- **FLOOR FROM BOUNDS** Κανόνας μετατροπής σε **Corridor** των tiles που είναι στα όρια μιας περιοχής. Χρησιμοποιείται για να γεμίσει με διαδρόμους τους χώρους γύρω από τα δωμάτια και τους περιμετρικούς τοίχους.
- **WALL FOR ROOM** Κανόνας μετατροπής σε **Wall** των tiles με βάση την **Von Neumann Neighborhood**. Αυτός ο κανόνας ελέγχει όλα τα tiles της γειτονιάς και αν έστω και ένα είναι είδους**Corridor** τότε μετατρέπει το tile σε **Wall**. Χρησιμοποιείται για να κλείσει τα δωμάτια με τοίχους.
- **WALL FROM ADJACENTS** Κανόνας μετατροπής σε **Wall** των tiles με βάση την **Moore Neighborhood**. Εάν τα μισά και παραπάνω tiles της γειτονιάς είναι είδους **Wall** τότε και το tile θα μετατραπεί σε **Wall**.
- **ROOM FROM ADJACENTS** Κανόνας μετατροπής σε **Room** των tiles με βάση την **Von Neumann Neighborhood**. Εάν τα μισά και παραπάνω tiles της γειτονιάς είναι είδους **Room** τότε και το tile θα μετατραπεί σε **Room**.

Οι δύο τελευταίοι κανόνες αν και υλοποιήθηκαν για τις αρχικές ανάγκες της εφαρμογής τελικά αντικαταστάθηκαν από τους άλλους κανόνες και δεν χρησιμοποιούνται στην τελική της έκδοση. Ο κώδικας παρέμεινε στην εφαρμογή και περιγράφετε και εδώ για θεωρητικούς λόγους.

2.5 Δημιουργία Συνόλου Δεδομένων (*Dataset*)

Με τον παραπάνω αλγόριθμο περιγράψαμε πως δημιουργείται ένα επίπεδο στο *Unity Game Engine*. Αυτό το επίπεδο πρέπει να αναπαρασταθεί και να αποθηκευτεί

σε μια μορφή που να μπορεί να διαβάσει και να επεξεργαστεί ο αλγόριθμος του *MLPG*. Αυτή η μετατροπή γίνετε από το *IO System* που αναπτύχθηκε επίσης στο *Unity Game Engine* με σκοπό την αποθήκευση και ανάγνωση επιπέδων με μια συγκεκριμένη αναπαράσταση.

Αρχικά έπρεπε να γίνει μια καταγραφή των δεδομένων που απαιτούνται από το σύστημα *MLPG* για την εκπαίδευση ώστε να αποθηκεύεται η ελάχιστη δυνατή πληροφορία κατά την δημιουργία του *dataset*. Αυτό μπορούμε να το δούμε ως μια συμπιεσμένη αναπαράσταση του επιπέδου που περιέχει μόνο τα απαραίτητα για την εκπαίδευση δεδομένα.

Το *IO System* επίσης θα πρέπει να μπορεί να δημιουργήσει το αρχικό επίπεδο από την αυτή τη συμπιεσμένη αναπαράσταση χωρίς να χάσει κάποιο κομμάτι του επιπέδου ή να αλλοιωθεί η μορφή του με τον οποιοδήποτε τρόπο. Επιλέχθηκε η αναπαράσταση του επιπέδου σε *csv* αρχείο με τα όπου η κάθε γραμμή στο αρχείο αντιστοιχεί σε ένα *tile* του επιπέδου με τα εξής πεδία:

- **Tile X** Αντιπροσωπεύει ακέραιο θετικό αριθμό ή μηδέν και δείχνει την συντεταγμένη *x* του *tile*
- **Tile Y** Αντιπροσωπεύει ακέραιο θετικό αριθμό ή μηδέν και δείχνει την συντεταγμένη *y* του *tile*
- **Tile Type** Μπορεί να πάρει μία από τις τιμές *[0, 1, 2]* όπου ο κάθε αριθμός αντιστοιχεί σε ένα από τα είδη των *tiles[0: CORRIDOR, 1: WALL, 2: ROOM]*.

Αυτή η απλή αναπαράσταση ήταν αρκετή για να αποτυπώσουμε όλη την πληροφορία που χρειάζεται να γνωρίζει το *MLPG* για να εκπαιδεύσει τα μοντέλα του. Κάθε επίπεδο αποθηκεύεται σε ξεχωριστό αρχείο και το *dataset* αποτελεί έναν φάκελο με όλα τα αρχεία-επίπεδα που περιέχει.

Αντίστοιχες μέθοδοι ανάγνωσης και εγγραφής δημιουργήθηκαν και στην υλοποίηση του *MLPG* όπως θα αναλυθούν στο αντίστοιχο κεφάλαιο.

Κεφάλαιο 3

Machine Learning

Η Μηχανική Μάθηση είναι ένας τομέας της Τεχνητής Νοημοσύνης που έχει ως στόχο την εκπαίδευση συστημάτων για την λήψη αποφάσεων, εκτέλεση ενεργειών και δημιουργία δεδομένων χωρίς την παρέμβαση κάποιου ανθρώπινου παράγοντα. Στο κέντρο της θεωρίας της Μηχανικής Μάθησης βρίσκονται τα μοντέλα (*models*) τα οποία είναι μαθηματικές αναπαραστάσεις που δημιουργούνται από τα δεδομένα (*training sample*) του συστήματος. Με βάση αυτά τα μοντέλα, το σύστημα καλείται να "πράξει" σε καινούργια και άγνωστα δεδομένα.

Ως επιστήμη χρησιμοποιεί μαθηματικά μοντέλα και στατιστική για να φτιάξει μια αναπαράσταση των δεδομένων, να βρει *patterns* και να τα κατηγοριοποιήσει με στόχο να εκτελέσει ενέργειες χωρίς να χρειαστεί οδηγίες από κάποιον άνθρωπο ή άλλο σύστημα.

Ως τομέας, η Μηχανική Μάθηση δημιουργήθηκε από την έρευνα με στόχο την δημιουργία τεχνητής νοημοσύνης στα υπολογιστικά συστήματα. Η ιδέα πίσω από την Μηχανική Μάθηση είναι ιδιαίτερα απλή, η εκμάθηση μέσα από δεδομένα, περιγράφει πολύ γενικά το πως μαθαίνουμε και εμείς οι άνθρωποι αλλά και όλοι οι έμβιοι οργανισμοί. Στην υλοποίηση της είναι ένα πολύπλοκο πρόβλημα με χρόνια μελέτης και έρευνας που παραμένει άλυτο στο μεγαλύτερο του βαθμό.

Έχει γνωρίσει τεράστια ανάπτυξη και υιοθέτηση τα τελευταία χρόνια χάρη στην ραγδαία ανάπτυξη του *hardware* των υπολογιστών το οποίο οδήγησε σε μεγάλα βήματα τον τομέα των νευρωνικών δικτύων και συγκεκριμένα των *Deep Neural Networks*. Έχει ξεκινήσει να εφαρμόζετε σε κάθε τομέα του σύγχρονου κόσμου, όπως στην ιατρική, την φαρμακευτική, την οικονομία και την διοίκηση με αποτελέσματα που ήταν ανέφικτα πριν μερικά χρόνια. [37]

3.1 Machine Learning & Games

Η Μηχανική Μάθηση και τα παιχνίδια έχουν μια στενή σχέση από την στιγμή της δημιουργίας του πεδίου μέχρι και σήμερα. Μερικές από τις πρώτες έρευνες και εφαρμογές της ήταν στην εκμάθηση παιχνιδιών όπως το σκάκι και η ντάμα, με στόχο να είναι σε θέση να ανταγωνιστεί ανθρώπινους αντιπάλους, χωρίς την ανάγκη να προγραμματιστούν κανόνες και κινήσεις για το παιχνίδι.

Ένα από τα επιτεύγματα που έδωσαν πολύ δημοσιότητα στα *Deep Neural Networks* είναι η δημιουργία του συστήματος *AlphaGo* ένα *Deep Neural Network* το οποίο κατάφερε να νικήσει πρωταθλητές του επιτραπέζιου παιχνιδιού *Go* χωρίς *handicap* σε ταμπλό 19x19 [39]. Ως επίτευγμα είναι ιδιαίτερα εντυπωσιακό καθώς ο αριθμός κινήσεων και στρατηγικών αυτού του παιχνιδιού ήταν, και παραμένει, υπολογιστικά ασύλληπτα μεγάλος για να προγραμματιστεί πρόγραμμα που να παίζει αποτελεσματικά. Συνεπώς η επίτευξη του με την χρήση *Deep Neural Networks* ήταν μεγάλο σημείο για την επιστημονική κοινότητα για να αναγνωρίσει και να αρχίσει να εφαρμόζει Μηχανική Μάθηση και σε άλλα, υπολογιστικά άλυτα προβλήματα.

Εκτός από την εκπαίδευση ενός συστήματος για να παίζει ένα παιχνίδι, η Μηχανική Μάθηση μπορεί να εφαρμοστεί, όπως και έγινε σε αυτή την εργασία, για την δημιουργία κομματιών ενός παιχνιδιού. Υπάρχουν υλοποιήσεις που βασίζονται σε διάφορους αλγορίθμους της Μηχανικής Μάθησης για την δημιουργία περιεχομένου όπως το έχουμε περιγράψει παραπάνω.

Επιπλέον μεγάλη εξάπλωση έχει παρατηρηθεί και στην χρήση Μηχανικής Μάθησης για το *player modeling*. Την κατηγοριοποίηση δηλαδή του παίκτη με βάση τον τρόπο που παίζει για την εξαγωγή συμπερασμάτων, αν ο παίκτης παίζει τίμια για παράδειγμα, και την δημιουργία περιεχομένου που ταιριάζει σε αυτόν τον παίκτη, με σκοπό την μεγιστοποίηση της διασκέδασης και απόλαυσης που του παρέχει το παιχνίδι.

3.2 Machine Learning στα Video Games

Αν και πολλοί τομείς της Τεχνητής Νοημοσύνης έχουν χρησιμοποιηθεί και συνεχίζουν να χρησιμοποιούνται με μεγάλη επιτυχία στην βιομηχανία των *Video Games*, η Μηχανική Μάθηση δεν έχει την χρήση και δημοτικότητα που θα περιμέναμε απ ότι είναι γνωστό. Υπάρχουν πολλές θεωρίες σχετικά με αυτό το θέμα όπως θα δούμε.

Λόγω της μεγάλης ανταγωνιστικότητας και για την προστασία της τεχνολογικής γνώσης, οι εταιρείες του *Game Development* δεν συνηθίζουν να περιγράφουν ή να ανακοινώνουν τα συστήματα τους και με τους αλγορίθμους που υλοποιήθηκαν. Συνεπώς είναι πολύ δύσκολο να γνωρίζουμε με σιγουριά αν και που χρησιμοποιείται Μηχανική Μάθηση στον τομέα του *Game Development.*

Γνωρίζουμε όμως ότι τα παιχνίδια που έχουν ή χρησιμοποιούν Μηχανική Μάθηση είναι ένα πολύ μικρό υποσύνολο των παιχνιδιών που υπάρχουν , συνεπώς επικρατεί η θεωρία ότι ο τομέας της Μηχανικής Μάθησης δεν έχει μεγάλη δημοτικότητα, στον κόσμο του *Game Development.* Αυτό μπορεί να οφείλεται και στο γεγονός ότι οι μέθοδοι της Μηχανικής Μάθησης δεν μπορούν να εγγυηθούν για τα αποτελέσματα τους ή ότι θα υπάρχουν αποτελέσματα που θα είναι αξιοποιήσιμα στο τελικό προϊόν. Αν και υπάρχουν μεθοδολογίες και αλγόριθμοι που είναι αποτελεσματικοί σε ένα μεγάλο εύρος προβλημάτων, η διαδικασία υλοποίησης ενός αλγορίθμου Μηχανικής Μάθησης περιλαμβάνει πολύ πειραματισμό σε κάθε περίπτωση.

Αυτά τα στοιχεία αποθαρρύνουν τις εταιρείες από το να επενδύσουν χρόνο και ανθρώπους σε ένα κομμάτι του παιχνιδιού που μπορεί τελικά να μην είναι αξιοποιήσιμο. Αυτά τα προβλήματα θα λύνονται όσο προχωράει ο τομέας της έρευνας πάνω στο θέμα των εφαρμογών της Μηχανικής Μάθησης σε *Video Games.* [38]

3.2.1 Video Games που χρησιμοποιούν Μηχανική Μάθηση

Θα αναλύσουμε μερικά παιχνίδια που έχουν χρησιμοποιήσει μεθόδους Μηχανικής Μάθησης με επιτυχία σε διάφορους τομείς τους.

- **The Legend of Zelda (1986)** Ένα από τα πιο γνωστά παιχνίδια παγκοσμίως που έγινε σειρά παιχνιδιών είναι το *The Legend of Zelda.* Για το συγκεκριμένο παιχνίδι έγιναν προσπάθειες να συνδυαστούν οι μέθοδοι *Bayesian Network* και *PCA* για να δημιουργηθούν επίπεδα που να μοιάζουν με τα υπάρχον επίπεδα του παιχνιδιού. [40]

- **Black & White (2001)** Το Black & White είναι ένα *Strategy* παιχνίδι που δημοσιεύτηκε το 2001, και έγινε αμέσως επιτυχία χάρη στο πρωτοποριακό AI του για το οποίο βραβεύτηκε παγκοσμίως. Το παιχνίδι χρησιμοποιεί *Decision Trees* και *Reinforcement Learning*, δύο τεχνικές Μηχανικής Μάθησης για να δώσει την δυνατότητα τον παίκτη να εκπαιδεύσει την συμπεριφορά του *NPC* χαρακτήρα, που έχει μορφή ενός γιγάντιου ζώου. [42]

- **Galactic Arms Race (2010)** Το Galactic Arms Race (GAR) είναι ένα *Space Shooter* και *Action RPG* παιχνίδι. Χρησιμοποιεί μεθόδους *Neuroevolution* σε συνεργασία με μεθόδους *PCG* για την δημιουργία μοναδικών όπλων για τον κάθε χρήστη ανάλογα με το πως παίζει. [42]

3.3 Ορολογία Μηχανικής Μάθησης

Θα ορίσουμε μερικές έννοιες από την ορολογία της Μηχανικής Μάθησης για την καλύτερη κατανόηση της υλοποίησης αυτής της εργασίας.Αυτές οι έννοιες θα χρησιμοποιηθούν στην συνέχεια για να περιγράψουμε το κομμάτι του *MLCG*.

- **Sample** Ένα δείγμα εκπαίδευσης. Αποτελεί ένα μεμονωμένο δείγμα του σύνολου των πληροφοριών που έχουμε για την εκπαίδευση. Μπορεί να είναι μια σειρά σε ένα αρχείο με δεδομένα, μια εικόνα, ένα βίντεο, ένα ολόκληρο κείμενο ή οτιδήποτε άλλο αναπαριστά ένα μεμονωμένο δείγμα στην υλοποίηση μας.

- **Dataset** Όλα τα δεδομένα που έχουμε στη διάθεση μας για την διαδικασία της εκπαίδευσης και του ελέγχου. Το *dataset* συνηθίζετε να χωρίζετε σε *training samples* και *testing samples*.

- **Training samples** Ένα υποσύνολο του *Dataset*. Αυτά είναι τα *samples* που χρησιμοποιούνται για την εκπαίδευση του μοντέλου.

- **Testing samples** Ένα υποσύνολο του *Dataset*. Αυτά είναι τα *samples* που χρησιμοποιούνται για τον έλεγχο και την αξιολόγηση του εκπαιδευμένου μοντέλου.

- **Μοντέλο** Είναι μια μέθοδος μηχανικής μάθησης που εκπαιδεύουμε χρησιμοποιώντας το *training sample* ώστε να εφαρμοστεί σε άγνωστα δεδομένα (*testing sample*). Αναφέρονται και ως μοντέλα πρόβλεψης καθώς μετά την εκπαίδευση, προβλέπουν το αποτέλεσμα με βάση τα δεδομένα εισόδου.

3.4 Κατηγορίες Μηχανικής Μάθησης

Τις μεθόδους της Μηχανικής Μάθησης μπορούμε να τις κατηγοριοποιήσουμε ανάλογα με το είδος της εκπαίδευσης και τις μεθοδολογίες σε συγκεκριμένες κατηγορίες. Η κάθε κατηγορία αποτελεί από μόνη της ένα ολόκληρο επιστημονικό πεδίο, με τις ιδιαιτερότητες και τα προβλήματα στα οποία εξειδικεύεται να το

χαρακτηρίζουν. Θα περιγράψουμε μερικές από τις πιο μεγάλες και σημαντικές κατηγορίες της Μηχανικής Μάθησης.

- **Supervised learning** Οι μέθοδοι αυτής της κατηγορίας δημιουργούν μοντέλα που αντιστοιχούν σε μαθηματικές αναπαραστάσεις των δεδομένων. Η εκπαίδευση γίνετε με δεδομένα που γνωρίζουμε το αποτέλεσμα που επιθυμούμε να μας δώσουν. Μέσα από διάφορες μεθόδους, η Μηχανική Μάθηση προσπαθεί να βρει μια συνάρτηση που να εκφράζει καλύτερα τα δεδομένα εκπαίδευσης. Έχοντας αυτή την συνάρτηση μπορεί να πάρει το επιθυμητό αποτέλεσμα για άγνωστα δεδομένα.

- **Unsupervised learning** Στην κατηγορία του *Unsupervised learning* δεν γνωρίζουμε το αποτέλεσμα ή χαρακτηριστικό όπως στο *Supervised learning*. Σε αυτή την κατηγορία προσπαθούμε να βρούμε μοτίβα και κοινά χαρακτηριστικά μεταξύ των δειγμάτων μας ώστε να τα κατηγοριοποιήσουμε σε ομάδες. Στη συνέχεια εφαρμόζοντας την ίδια κατηγοριοποίηση σε άγνωστα δεδομένα να βρούμε στοιχεία από τις ομάδες στις οποίες ανήκουν.

- **Reinforcement learning** Το *Reinforcement learning* έχει μεγάλη συσχέτιση με τον τρόπο που μαθαίνουν οι άνθρωποι. Χρησιμοποιεί μεθόδους ανταμοιβής και τιμωρίας για να αποδώσουν ή να αφαιρέσουν βαθμούς από το μοντέλο. Σε αυτές τις κατηγορίες το μοντέλο έχει ως στόχο να μεγιστοποιήσει την ανταμοιβή του ή να ελαχιστοποιήσει την τιμωρία του, οπότε εφαρμόζει *search algorithms* για να αλληλεπιδράσει με το περιβάλλον/σύστημα και να βρει για κάθε περίπτωση ποια είναι η καλύτερη δυνατή στρατηγική ώστε να πετύχει τον στόχο του.

Η κατηγορία που χρησιμοποιήθηκε στη συγκεκριμένη εργασία ανήκει στο *Supervised learning* καθώς για κάθε δεδομένα εκπαίδευσης υπήρχε το αντίστοιχο δεδομένο-αποτέλεσμα που επιθυμούσαμε να επιστρέψει το σύστημα.

3.4.1 Μέθοδοι Supervised learning

Θα αναλύσουμε μερικές μεθόδους της κατηγορίας *Supervised learning* καθώς αυτή χρησιμοποιήθηκε για την υλοποίηση αυτής της εργασίας.

- **Decision trees** Τα *Decision Trees* όπως περιγράφει και το όνομα τους είναι μοντέλα πρόβλεψης με δενδρική αναπαράσταση. Χρησιμοποιούν τα χαρακτηριστικά των δεδομένων εισόδου για να περιηγηθούν μέσα στο δέντρο μέχρι να φτάσουν

σε κάποιο αποτέλεσμα-ρίζα. Κατά την εκπαίδευση του μοντέλου τα χαρακτηριστικά των δεδομένων εκπαίδευσης χωρίζονται με συγκεκριμένες μεθόδους, ανάλογα τον αλγόριθμο του *Decision Tree* για να δημιουργήσουν το τελικό μοντέλο-δέντρο.

- **Support vector machines** Ένα *SVM* στην ποιο απλή του μορφή, αποτελεί μια μαθηματική μέθοδο διαχωρισμού των δεδομένων σε δύο ομάδες. Χρησιμοποιεί μεθόδους διχοτόμησης του χώρου των δεδομένων ώστε τα στοιχεία της κάθε ομάδας να είναι στον ίδιο χώρο. Έχει τη δυνατότητα να ανάγει τις μεθόδους της σε δεδομένα πολυδιάστατων χώρων, ένα πολύ χρήσιμο χαρακτηριστικό ειδικά σε δύσκολα προβλήματα κατηγοριοποίησης.

- **Artificial neural networks** Τα *Artificial neural networks* είναι συστήματα που έχουν φτιαχτεί με έμπνευση τον ανθρώπινο εγκέφαλο και τον τρόπο λειτουργίας του. Αποτελούνται από *Artificial Neurons* οι οποίοι παίρνουν μια είσοδο, την επεξεργάζονται και βγάζουν ένα αποτέλεσμα που μπορεί να είναι είσοδος για τον επόμενο *Artificial Neuron.* Αυτά τα στοιχεία είναι οργανωμένα σε επίπεδα (*layers*) και η έξοδος του κάθε επιπέδου αποτελεί είσοδο για το επόμενο. *Artificial neural networks* μιας αρχιτεκτονικής που ονομάζετε *Generative Adversarial Networks* χρησιμοποιήθηκε για την υλοποίηση του MLCG αυτής της εργασίας.

3.5 Artificial Neural Networks (ANN)

Τα ANNs έχουν γνωρίσει μεγάλη δημοσιότητα τα τελευταία χρόνια χάρη στα πρωτοποριακά αποτελέσματα που παρουσιάζουν σε πολλά προβλήματα τεχνητής νοημοσύνης, όπως την επεξεργασία και ανάλυση εικόνας και βίντεο. Η επιτυχία τους οφείλετε σε μεγάλο βαθμό στην πρόοδο του hardware των ηλεκτρονικών υπολογιστών το οποίο έδωσε τους απαραίτητους πόρους για την ανάπτυξη των λεγόμενων Deep Neural Networks. Αν και η θεωρία και η έρευνα πάνω στα ANNs ξεκίνησε το 1940, οι δυνατότητες τους ξεκίνησαν να γίνονται αντιληπτές στο διάστημα 2009 με 2012 όπου υλοποιήσεις με ANNs ξεκίνησαν ξεπερνάνε τα μέχρι τότε καλύτερα μοντέλα μηχανικής μάθησης σε διάφορους τομείς, και να προσεγγίζουν τα ανθρώπινα όρια.

3.5.1 Χαρακτηριστικά ενός ANN

Θα αναλύσουμε μερικά από τα πιο βασικά χαρακτηριστικά ενός ANN τα οποία περιέχονται σε κάθε διαφορετική υλοποίηση των ANNs

- **Neuron** Ένα ANN αποτελείται από Artificial Neurons (Τεχνητοί Νευρώνες) σε αντιστοιχία με έναν ανθρώπινο εγκέφαλο. Ο κάθε νευρώνας δέχεται μια ή περισσότερες εισόδους, οι οποίες περνάνε από την συνάρτηση ενεργοποίησης του (activation function) μαζί με ένα βάρος. Η έξοδος της συνάρτησης ενεργοποίησης περνάει στην συνάρτηση εξόδου που είναι και το αποτέλεσμα που επιστρέφει ο νευρώνας ως έξοδο.

- **Activation function** Η συνάρτηση ενεργοποίησης έχει ως σκοπό να εναρμονίσει την έξοδο του νευρώνα με την είσοδο που δέχεται, ώστε μικρές αλλαγές στην είσοδο να επιφέρουν και μικρές αλλαγές στην έξοδο. Λειτουργεί και ως διακόπτης του νευρώνα, ο οποίος για συγκεκριμένες τιμές παράγει μηδενική έξοδο.

- **Weights** Τα βάρη είναι τιμές που αποδίδονται σε κάθε είσοδο ενός νευρώνα. Λαμβάνονται υπόψιν μαζί με την τιμή της εισόδου για να παράξουν την επιθυμητή τιμή εξόδου. Τα βάρη είναι από τα πιο σημαντικά κομμάτια ενός νευρωνικού δικτύου καθώς αυτές οι τιμές είναι που αλλάζουν κατά την εκπαίδευση για να φτιαχτεί ένα νευρωνικό που να παράγει τα αποτελέσματα που θέλουμε.

- **Connections** Οι συνδέσεις μεταξύ νευρώνων είναι επίσης πολύ σημαντικές και είναι ένα από τα κομμάτια που καθορίζουν την αρχιτεκτονική του δικτύου. Οι συνδέσεις δεν αλλάζουν κατά την εκπαίδευση αλλά μπορούν να "αποκοπούν" προσωρινά για να βελτιωθεί η εκπαίδευση.

- **Layers** Τα ANNs είναι οργανωμένα σε επίπεδα, όπου το κάθε επίπεδο έχει ένα συγκεκριμένο αριθμό από νευρώνες, δέχεται συγκεκριμένο αριθμό από εισόδους σε αυτούς τους νευρώνες και βγάζει συγκεκριμένο αριθμό εξόδων για το επόμενο επίπεδο ή ως τελικό αποτέλεσμα. Το πρώτο επίπεδο ενός ANN ονομάζετε επίπεδο εισόδου (input layer) ενώ αντίστοιχα το τελευταίο επίπεδο ονομάζετε επίπεδο εξόδου (output layer). Υπάρχουν πολλών ειδών επίπεδα ανάλογα με τον αριθμό των νευρώνων και τις συνδέσεις τους με προηγούμενα και επόμενα επίπεδα.

3.5.2 Επίπεδα

Θα αναλύσουμε μερικά από τα πιο γνωστά επίπεδα καθώς και αυτά που χρησιμοποιήθηκαν σε αυτή την υλοποίηση. Η σχεδίαση της αρχιτεκτονικής ενός ANN περιλαμβάνει την επιλογή των επιπέδων που θα περιέχει, την σύνδεση τους και την ιεραρχία τους καθώς και τον αριθμό των εισόδων τους. Υπάρχουν μεθοδολογίες για την επιλογή επιπέδων και σχεδίαση αρχιτεκτονικών αλλά τα περισσότερα

μοντέλα δημιουργούνται με συνεχείς επαναλήψεις δοκιμών και αναδιοργάνωσης μέχρι να βρεθεί η αρχιτεκτονική με τα καλύτερα αποτελέσματα.

- **ReLU** Rectified Linear Unit είναι ολόκληρο το όνομα αυτού του επιπέδου, για συντομία ReLU. Είναι ένα γραμμικό επίπεδο με συνάρτηση ενεργοποίησης και εξόδου την $f(x) = max(x, 0)$. Επίπεδο για τον μηδενισμό όλων των αρνητικών τιμών.

- **BatchNormalization** Όπως περιέχεται και στο όνομα, αυτό το επίπεδο εφαρμόζει μια συνάρτηση κανονικοποίησης στα δεδομένα ώστε να διατηρούνται οι τιμές από το ένα επίπεδο στο άλλον μέσα σε ένα συγκεκριμένο εύρος. Το επίπεδο BatchNormalization μπορεί να εφαρμοστεί στα δεδομένα εισόδου ή εξόδου ανάμεσα σε άλλα επίπεδα, και κρατάει τη μέση τιμή ενεργοποίησης κοντά στο 0 και την τυπική απόκλιση κοντά στο 1. [52]

- **Dropout** Το Dropout επίπεδο χρησιμοποιείται μόνο κατά την διάρκεια της εκπαίδευσης. Αυτό το επίπεδο αλλάζει τυχαία με βάση κάποια παράμετρο, κάποιες από τις τιμές εισόδου σε 0. Αυτό έχει ως αποτέλεσμα την "απενεργοποίηση" των νευρώνων που λαμβάνουν αυτές τις εισόδους. Ο αριθμός των εισόδων που θα γίνουν 0 ορίζεται ως παράμετρος που επιπέδου, για παράδειγμα με παράμετρο 0.2, το 20% των εισόδων θα γίνει 0. Το επίπεδο Dropout χρησιμοποιείται για την αποφυγή του overfitting. [43]

- **Dense** Το Dense είναι ένα μη γραμμικό, βαθύ επίπεδο που μπορεί εν δυνάμει να αναπαραστήσει οποιαδήποτε μαθηματική συνάρτηση. Είναι από τα πιο δημοφιλή επίπεδα των Deep ANN. Έχουν για συνάρτηση εξόδου $f(x) = activation((x \bullet kernel) + bias)$. Όπου ο kernel αντιστοιχεί στα weights της εισόδου τα οποία είναι σε διανυσματική μορφή, το x είναι η είσοδος του νευρώνα επίσης σε διανυσματική μορφή, το bias είναι μια σταθερή τιμή και η activation είναι η συνάρτηση ενεργοποίησης. [50]

- **Conv2D** Από τα πιο ευρέως διαδεδομένα επίπεδα είναι τα Convolution. Με την χρήση ενός convolutional matrix εφαρμόζει συνέλιξη στα δεδομένα εισόδου για να παράξει την έξοδο. Τα δεδομένα εισόδου πρέπει να είναι σε διανυσματική μορφή.

3.5.3 Backpropagation

Ένας από τους πιο σημαντικούς αλγορίθμους για την λειτουργία των Deep ANNs είναι ο αλγόριθμος του Backpropagation. Όπως δηλώνει και το όνομα του, ο Backpropagation είναι η μέθοδος που "αναθέτει" το σφάλμα από ένα πέρασμα

του δικτύου, στα βάρη των νευρώνων ώστε να είναι ισοκατανεμημένο ανάλογα με την συμμετοχή του κάθε νευρώνα στην τελική έξοδο. Στη συνέχεια τα βάρη διορθώνονται με βάση το σφάλμα που τους αναλογεί και η εκπαίδευση συνεχίζετε. Με κάθε είσοδο ή σε batches εισόδων, επαναλαμβάνετε αυτή η διαδικασία μέχρι τα βάρη να μην έχουν μεγάλες μεταβολές σε κάθε ενημέρωση. [46]

Η μέθοδος του Backpropagation είναι αυτή που έκανε δυνατή την ύπαρξη των Deep ANNs καθώς αντιμετωπίζει το πρόβλημα των vanishing gradient problem όπου το λάθος που επιστρεφόταν από τα τελικά επίπεδα προς τα αρχικά γινόταν 0 μετά από ορισμένο αριθμό επιπέδων και το δίκτυο δεν μπορούσε να εκπαιδευτεί.

3.6 Generative Adversarial Networks (GANs)

Τα Generative Adversarial Networks αποτελούν μια κατηγορία των ANNs η οποία δημιουργήθηκε το 2014 από τον Ian Goodfellow και τους συνεργάτες του. Η βασική ιδέα πίσω από τα GANs περιλαμβάνει δύο ANN τα οποία λειτουργούν ανταγωνιστικά το ένα με το άλλο. Λέγονται Generative επειδή ως στόχο τα GANs έχουν την δημιουργία περιεχομένου. Αρχικά τα GANs είχαν προταθεί ως μέθοδοι για Unsupervised εκπαίδευση αλλά στη συνέχεια επεκτάθηκαν και υπάρχουν υλοποιήσεις τους για Supervised, Semi-supervised και Reinforcement Μηχανική Μάθηση.

Βασικός στόχος ενός GAN είναι η παραγωγή περιεχομένου που να μοιάζει όσο το δυνατόν πιο αληθινό. Για την εκπαίδευση του χρειάζεται ένα dataset με πραγματικά παραδείγματα από περιεχόμενο όπως αυτό που θέλουμε να παράγει το δίκτυο, όπως εικόνες ανθρώπων ή επίπεδα ενός παιχνιδιού (όπως συμβαίνει στην συγκεκριμένη υλοποίηση). [45]

3.6.1 Μοντέλο GAN

Όπως αναφέρθηκε, ένα GAN αποτελείται από δύο ANNs, το Generative Network και το Discriminative Network. Το κάθε ANN έχει την δική του αρχιτεκτονική και παραμέτρους. Τα δύο δίκτυα εκπαιδεύονται μέσω ανταγωνισμού καθώς η έξοδος του ενός αποτελεί την είσοδο του άλλου και συνεχώς προσπαθούν να ξεπεράσουν το ένα το άλλο.

- **Generative Network** Το Generative Network είναι υπεύθυνο για την παραγωγή του επιθυμητού περιεχομένου με βάση την τιμή της εισόδου που του δίνουμε. Αυτό το περιεχόμενα που θα παραχθεί θα δοθεί ως είσοδο στο Discriminative Network. Η έξοδος του Discriminative θα δοθεί πάλι στο Generative ως feedback για να γίνει διόρθωση, αν χρειάζεται των weights του.

- **Discriminative Network** Το Discriminative Network έχει ως στόχο των διαχωρισμό των πραγματικών δειγμάτων από αυτά που παράγει ο Generator. Λαμβάνει ως είσοδο ένα δείγμα και "αποφασίζει" εάν είναι πραγματικό δείγμα ή παραγόμενο από τον Generator. Αυτή την απόφαση την δίνει ως feedback στον Generator και αυτό λαμβάνει ως feedback εάν η πρόβλεψη του ήταν σωστή ή όχι για να διορθώσει τα weights του.

3.6.2 Εκπαίδευση

Η διαδικασία της εκπαίδευσης περιλαμβάνει την συνεχή ανατροφοδότηση των δύο δικτύων όπου σταδιακά γίνονται καλύτερα, το καθένα στον σκοπό του, καθώς ανταγωνίζονται το ένα το άλλο. Για την εκπαίδευση απαιτείται ένα dataset πραγματικών παραδειγμάτων που δίνονται στο κάθε δίκτυο με διαφορετική επισήμανση ανάλογα με τον σκοπό τους.

Στον Generator δίνετε μια είσοδος και η έξοδος που θα έπρεπε να παράξει από το σύνολο των πραγματικών δειγμάτων. Στον Discriminator δίνετε αντίστοιχα ένα δείγμα, (πραγματικό ή παραγόμενο), και η επισήμανση του αντίστοιχα, εάν είναι πραγματικό ή όχι. Η ανατροφοδότηση μεταξύ των δύο δικτύων βελτιώνει συνεχώς τις αποδόσεις και των δύο.

3.6.3 Αποτελέσματα και αποδόσεις

Τα αποτελέσματα ενός GAN όπως και όλων των ANN σχετίζονται άμεσα με την αρχιτεκτονική τους. Ο σχεδιασμός και η παραμετροποίηση ενός ANN αποτελεί ένα πολύ δύσκολο έργο το οποίο περιλαμβάνει συνεχόμενες επαναλήψεις και δοκιμές. Αντίστοιχα η σχεδίαση και η εκπαίδευση δύο δικτύων, όπου το ένα αλληλεπιδρά με το άλλο αποτελεί ένα έργο διπλάσιας δυσκολίας με διπλάσιες απαιτήσεις σε πόρους και δοκιμές.

Τα αποτελέσματα των GANs δικαιολογούν το κόστος τους καθώς παράγουν ιδιαίτερα μοναδικό και αξιόπιστο περιεχόμενο, δύο χαρακτηριστικά που απαιτούνται στον τομέα του Content Generation όπως αναλύθηκε στο κεφάλαιο 1.

Κεφάλαιο 4

Δημιουργία περιεχομένου με τη χρήση Μηχανικής Μάθησης

Αυτό το κεφάλαιο περιγράφει την διαδικασία σχεδιασμού και υλοποίησης των μοντέλων που αναπτύχθηκαν με σκοπό την παραγωγή επιπέδων, παρόμοιων χαρακτηριστικών με τα επίπεδα που αναλύθηκαν στον κεφάλαιο 2. Η υλοποίηση είναι μια εφαρμογή Machine Learning Content Generation (MLCG). Αυτή η υλοποίηση αποτελεί μια απόδειξη, από την θεωρία στην πράξη ότι Deep Neural Networks και συγκεκριμένα GANs μπορούν να χρησιμοποιηθούν για την κατασκευή περιεχομένου.

Η υλοποίηση περιλάμβανε την ολοκλήρωση πολλών επιμέρους κομματιών, κάποια από τα οποία δεν χρησιμοποιήθηκαν τελικά στην εκπαίδευση του τελικού μοντέλου. Κάτι το οποίο είναι συχνό και αναμενόμενο όταν υλοποιούνται εφαρμογές τέτοιας πολυπλοκότητας. Για παράδειγμα υλοποιήθηκαν μέθοδοι που πρόσθεταν θόρυβο στις διακριτές τιμές της εισόδου ώστε να μετατρέπονται σε συνεχείς τιμές με κανονική κατανομή. Αυτό το κομμάτι θεωρήθηκε ότι τελικά δεν προσφέρει στην απόδοση του συστήματος οπότε δεν χρησιμοποιείται στα παραδείγματα που θα αναλυθούν. Παραμένει κομμάτι της υλοποίησης χωρίς όμως να συνεισφέρει στο τελικό αποτέλεσμα.

4.1 Τεχνική περιγραφή

Το σύστημα MLCG αναπτύχθηκε στην γλώσσα*Python*, μια από τις πιο διαδεδομένες γλώσσες για υλοποιήσεις Μηχανικής Μάθησης. Η έκδοση που χρησιμοποιήθηκε είναι η 3.7 και είναι ιδιαίτερα σημαντική καθώς αυτή είναι η πιο πρόσφατη έκδοση

που υποστηρίζει μερικές από τις βιβλιοθήκες που χρησιμοποιήθηκαν. Μερικές από τις πιο σημαντικές βιβλιοθήκες και εργαλεία που χρησιμοποιήθηκαν:

- **Numpy** Βιβλιοθήκη για επιστημονικές εφαρμογές και υπολογισμούς.
- **Tensorflow** Αποτελεί ένα ολοκληρωμένο σύστημα για εφαρμογές Machine Learning με υποστήριξη για πολλές γλώσσες προγραμματισμού.
- **Keras** Αποτελεί την βιβλιοθήκη σε Python για την χρησιμοποίηση του συστήματος Tensorflow σε εφαρμογές ανεπτυγμένες σε αυτή την γλώσσα.

4.2 Δεδομένα εκπαίδευσης

Όπως αναλύθηκε και στο κεφάλαιο σχετικά με την υλοποίηση του PCG συστήματος σε Python, το dataset αποτελείται από αρχεία κειμένου, όπου το κάθε αρχείο αντιστοιχεί σε ένα επίπεδο. Για την εκπαίδευση του μοντέλου, δημιουργήθηκαν εκατοντάδες τέτοια αρχεία και αυτά αντιστοιχούν στο dataset μας.

Όπως σε όλες τις εφαρμογές Μηχανικής Μάθησης, τα δεδομένα του dataset δεν δίνονται κατευθείαν στο μοντέλο για εκπαίδευση αλλά περνάνε από διάφορες μεθόδους προεργασίας ώστε να εξασφαλίσουμε καλύτερα και πιο αξιόπιστα αποτελέσματα.

4.2.1 Χαρακτηριστικά του dataset

Το κάθε επίπεδο είναι ένας πίνακας 900 στοιχείων, όπου το κάθε στοιχείο αντιστοιχεί σε ένα κελί στην 30x30 grid απεικόνιση του επιπέδου. Μόλις το PCG σύστημα δημιουργεί ένα επίπεδο, το μετατρέπει στην αναπαράσταση του αρχείου και το αποθηκεύει ώστε να μπορεί να διαβαστεί από το σύστημα Μηχανικής Μάθησης.

Απόσπασμα από αρχείο του dataset:

```
30, 30
Tile X,Tile Y,Tile Type
0,0,1
0,1,1
0,2,1
```

0 , 3 , 1
0 , 4 , 1
0 , 5 , 1

Στο παραπάνω παράδειγμα βλέπουμε τις πρώτες γραμμές από ένα αρχείο επιπέδου. Η πρώτη σειρά δηλώνει το μέγεθος του επιπέδου, κάτι το οποίο προστέθηκε για πληροφοριακούς λόγους αλλά δεν χρησιμοποιείται από το MLCG. Η δεύτερη σειρά αναγράφει το είδος των στοιχείων που αντιστοιχούν στην κάθε κολόνα αντίστοιχα.

- Η πρώτη κολόνα, Tile X δηλώνει την x συντεταγμένη του στοιχείου μέσα στο επίπεδο.
- Η δεύτερη κολόνα, Tile Y δηλώνει την y συντεταγμένη του στοιχείου μέσα στο επίπεδο.
- Η τρίτη κολόνα, Tile Type, δηλώνει το είδος του στοιχείου (Corridor, Wall, Room) με ένα διακριτό ακέραιο αριθμό αντίστοιχα.

Η αρίθμηση των στοιχείων μέσα στο grid ξεκινάει από την κάτω αριστερή γωνία, αυτό είναι το στοιχείο με συντεταγμένες 0,0 και προχωράει πρώτα στον κάθετο άξονα (Y) και στη συνέχεια στον οριζόντιο άξονα (X). Το τελευταίο στοιχείο που αναγράφετε με συντεταγμένες 29,29 αντιστοιχεί στην πάνω δεξιά γωνία του grid.

4.2.2 Προ επεξεργασία Δεδομένων

Πριν από το στάδιο της εκπαίδευσης, τα δεδομένα που θα εισάγουμε στο σύστημα πρέπει να περάσουν από το στάδιο της προ επεξεργασίας ώστε να πληρούν κάποιες προϋποθέσεις.

Αρχικά γίνετε μια αλλαγή της διακριτής τιμής του κάθε στοιχείου για να τα μεταφέρουμε από το εύρος [0, 2] στο εύρος [-1, 1] το οποίο είναι το εύρος που θέλουμε να παραμείνουν οι τιμές μας και κατά την εκπαίδευση μέσα στους νευρώνες του δικτύου. Οπότε εφαρμόζουμε μια αλλαγή τιμής:

- Η τιμή 0 μετατρέπεται σε -1
- Η τιμή 1 μετατρέπεται σε 0

- Η τιμή 2 μετατρέπεται σε 1

Το δεύτερο κομμάτι αφορά το είδος του μοντέλου που θέλουμε να εκπαιδεύσουμε. Στην υλοποίηση της εργασίας αναπτύχθηκαν δύο διαφορετικά μοντέλα GANs, το Dense και το Convolutional. Το κάθε μοντέλο λόγω των επιπέδων που έχει, δέχεται την είσοδο σε διαφορετική μορφή. Το Dense μοντέλο δέχεται ως είσοδο ένα διάνυσμα (900, 1). Επειδή το Dense μοντέλο ήταν το πρώτο που σχεδιάστηκε και αναπτύχθηκε, τα δεδομένα έχουν αυτή την μορφή όταν διαβάζονται από τα αρχεία του dataset. Το Convolutional μοντέλο δέχεται ως είσοδο έναν πίνακα (30, 30) διαστάσεων.

Εάν η εκπαίδευση μας θα γίνει με το Convolutional μοντέλο, τα δεδομένα από διανυσματική μορφή (900, 1) μετατρέπονται σε πίνακα (30, 30) διαστάσεων.

Υπάρχει η δυνατότητα προσθήκης θορύβου στα δεδομένα ώστε να έχουν μια καλύτερη κατανομή στον χώρο ανάμεσα στις διακριτές τιμές. Επειδή αυτό το στάδιο προσθέτει πολυπλοκότητα χωρίς να παρουσιάζει κάποια εμφανή διαφορά στα αποτελέσματα των μοντέλων, δεν χρησιμοποιήθηκε στις τελικές εκπαιδεύσεις. Παρόλαυτα υπάρχει η δυνατότητα να ενεργοποιηθεί οποιαδήποτε στιγμή από μια ρύθμιση στην υλοποίηση.

4.3 Επεξεργασία Αποτελεσμάτων

Η αντίθετη διαδικασία πρέπει να εκτελεστεί για τα αποτελέσματα των μοντέλων GAN. Η έξοδος του κάθε μοντέλου σε διαστάσεις είναι αντίστοιχη της εισόδου που δέχεται, συνεπώς το Dense μοντέλο παράγει διανύσματα (900, 1) και το Convolutional μοντέλο, πίνακα (30, 30).

Τα αποτελέσματα του Convolutional μοντέλου πρέπει να μετατραπούν σε διάνυσμα (900, 1) για να αποθηκευτούν με την μορφή που έχει όλο το dataset.

Επίσης τα αποτελέσματα των μοντέλων είναι σε συνεχείς τιμές και πρέπει να γίνει η μετατροπή τους στην κοντινότερη διακριτή τιμή από τις διαθέσιμες [-1, 0, 1] και στην συνέχεια να γίνει η αλλαγή τιμής όπως έγινε στην προ επεξεργασία:

- Η τιμή -1 μετατρέπεται σε 0
- Η τιμή 0 μετατρέπεται σε 1
- Η τιμή 1 μετατρέπεται σε 2

4.3.1 Δειγματοληψία κατά την εκπαίδευση

Για να παρακολουθούμε την πρόοδο της εκπαίδευσης ανά τακτές χρονικές περιόδους, ορίστηκε μια μεταβλητή που δηλώνει ανά πόσες εποχές, θα γίνετε δειγματοληψία από το μοντέλο. Επίσης μια δειγματοληψία γίνετε μόλις τελειώσει η εκπαίδευση για να έχουμε δείγμα από τον τελικό μοντέλο.

Σε κάθε δειγματοληψία, το μοντέλο εκτελείται με μια είσοδο εκπαίδευσης και με βάση τα βάρη που έχει εκείνη την στιγμή. Το αποτέλεσμα περνάει από την επεξεργασία που αναλύθηκε παραπάνω και αποθηκεύεται σε αρχείο με όνομα που να εκφράζει το στάδιο της εκπαίδευσης στο οποίο βρισκόταν το νευρωνικό όταν πάρθηκε αυτό το δείγμα.

Για παράδειγμα το αρχείο `combined_1000_09-03-2020_21-11-56FGAE.csv` είναι ένα δείγμα που δημιουργήθηκε από τον Generator σε συνεργασία με τον Discriminator (combined) στην εποχή 1000 της εκπαίδευσης. Τα υπόλοιπα στοιχεία δηλώνουν την ημερομηνία που εκτελέστηκε η συγκεκριμένη εκπαίδευση και ένα τυχαίο μοναδικό αλφαριθμητικό.

Το μοντέλο που παρήγαγε αυτό το δείγμα αναφέρετε στον φάκελο που είναι αποθηκευμένο `cnn_gan-09_03_2020_21_08_34`, είναι το Convolutional μοντέλο (cnn) καθώς και η ημερομηνία και ώρα που ξεκίνησε η εκπαίδευση του συγκεκριμένου μοντέλου.

4.3.2 Αποθήκευση μοντέλου

Το κάθε μοντέλο που εκπαιδεύεται αποθηκεύεται η αρχιτεκτονική του και τα βάρη του σε ειδικά αρχεία ώστε να μπορούμε οποιαδήποτε στιγμή να τα φορτώσουμε και να τα εκτελέσουμε. Η αποθήκευση γίνετε μέσα στον φάκελο του μοντέλου, που θα δημιουργηθεί μόλις ξεκινήσει η εκπαίδευση, θα αναγράφει το είδος του μοντέλου και την ημερομηνία εκπαίδευσης.

Μέσα στον φάκελο του μοντέλου θα αποθηκευτούν:

- **Χαρακτηριστικά του μοντέλου** Όλη η αρχιτεκτονική του μοντέλου, δηλαδή τα επίπεδα, ο αριθμός και το είδος των παραμέτρων του κάθε επιπέδου, για τον Generator και τον Discriminator. Αυτά αποθηκεύονται μέσα στον φάκελο `model_data` στα αρχεία *generator.json* και *discriminator.json* αντίστοιχα.
- **Τα βάρη του μοντέλου** Σε αντίστοιχα αρχεία *generator.h5* και *discriminator.h5*

αποθηκεύονται τα βάρη του κάθε δικτύου.

- **Μετρικές και αποτελέσματα** Στο αρχείο *results.txt* αποθηκεύονται οι τελικές μετρήσεις και αποτελέσματα της απόδοσης του κάθε μοντέλου μαζί με διάφορες παραμέτρους που έχουμε ορίσει για την εκπαίδευση και τις τιμές τους. Με αυτό τον τρόπο μπορούμε εύκολα να συγκρίνουμε τα διάφορα μοντέλα.
- **Επίπεδα που παράχθηκαν** Στον φάκελο *Results* αποθηκεύονται τα επίπεδα που δημιουργήθηκαν με την δειγματοληψία που αναφέραμε παραπάνω.

4.4 Μοντέλο Dense GAN

Το μοντέλο GAN που αναφέρετε ως Dense είναι το πρώτο μοντέλο GAN που αναπτύχθηκε και εκπαιδεύτηκε στα πλαίσια αυτής της εργασίας. Αναφέρετε σε διάφορα παραδείγματα υλοποιήσεων GAN [49] [47] [48] και είναι πιο εύκολο στην κατανόηση και υλοποίηση. Αναφέρετε ως μοντέλο Dense επειδή η επεξεργασία των εισόδων γίνετε στα Dense επίπεδα του δικτύου.

4.4.1 Αρχιτεκτονική Generator

Το μοντέλο του Generator δικτύου, αποτελείται από 22 επίπεδα συνολικά, από τα οποία τα πρώτα 21 μπορούν να χωριστούν σε τριάδες καθώς ακολουθούν μια κοινή αρχιτεκτονική και το τελευταίο επίπεδο δημιουργεί την έξοδο με τις διαστάσεις του επιπέδου όπως τις έχουμε ορίσει. Κάθε τριάδα επιπέδου ξεκινάει με ένα Dense επίπεδο, στην συνέχεια υπάρχει ένα LeakyReLU επίπεδο [51] και το τρίτο επίπεδο εναλλάσσετε από BatchNormalization επίπεδο σε Dropout.

- **Dense επίπεδο** Το κάθε Dense επίπεδο, πέρα από το αρχικό που λαμβάνει ως είσοδο το επίπεδο ως διάνυσμα (900, 1) στοιχείων, παίρνει ως είσοδο, την έξοδο του προηγούμενου Dropout επιπέδου. Για κάθε Dense επίπεδο ορίζουμε τον αριθμό των *units* που θα έχει ως έξοδο. Για παράδειγμα για το επίπεδο εισόδου ορίζουμε:

  ```
  Dense(units=256, input_dim=self.dungeon_dimension)
  ```

  Το παραπάνω επίπεδο έχει είσοδο ένα διάνυσμα (900, 1) και ως έξοδο ένα διάνυσμα (256, 1). Το μέγεθος της εισόδου πρέπει να καθοριστεί μόνο στο αρχικό επίπεδο του δικτύου, τα υπόλοιπα γνωρίζουν τι είσοδο δέχονται με βάση την έξοδο του προηγούμενου επιπέδου. [50]

- **LeakyReLU** Μετά από κάθε Dense επίπεδο στην κάθε τριάδα, υπάρχει ένα επίπεδο *LeakyReLU*. Αυτό το επίπεδο είναι μια παραλλαγή του επιπέδου *ReLU* που αναλύσαμε με την διαφορά ότι στην περίπτωση μη ενεργοποίησης του επιπέδου από την είσοδο, η έξοδος δεν είναι μηδενική όπως στο επίπεδο της *ReLU* αλλά επιστρέφει ως έξοδο μια μικρή τιμή. Η συνάρτηση ενεργοποίησης της LeakyReLU ορίζετε ως:

  $f(x) = alpha * x, \;\; if \;\; x < 0$

  $f(x) = x, \;\; if \;\; x >= 0$

  Το alpha είναι μια παράμετρος που μπορεί να οριστεί σε κάθε επίπεδο LeakyReLU. Στην συγκεκριμένη υλοποίηση ορίστηκε ως εξής:

  ```
  LeakyReLU(alpha=0.2)
  ```

  Ο ορισμός του alpha με την τιμή 0.2 έγινε μετά από έρευνα σε αντίστοιχες υλοποιήσεις και παρατηρήσεις κατά την εκπαίδευση. [47] [48] [49]

- **BatchNormalization** Το τελευταίο επίπεδο σε κάθε τριάδα είναι είτε το επίπεδο του BatchNormalization, ή το επίπεδο Dropout. Το επίπεδο του BatchNormalization όπως αναφέρει και το όνομα κανονικοποιεί την είσοδο που δέχεται. Το επίπεδο Dropout απενεργοποιεί τυχαία κάποιες από τις εξόδους μετατρέποντας τις τιμές σε 0.

  ```
  BatchNormalization(momentum=0.8)
  Dropout(rate=0.3)
  ```

  Η παράμετρος momentum χρησιμοποιείται με την τιμή 0.8 για να βελτιώσει την απόδοση του νευρωνικού και να αυξήσει την ταχύτητα της εκπαίδευσης [47].

  Η παράμετρος rate δηλώνει το ποσοστό επί τις εκατό των τιμών που θα γίνουν 0 στο επίπεδο Dropout.

- **Dense επίπεδο εξόδου** Το τελευταίο επίπεδο έχει ως έξοδο το διάνυσμα του παραγομένου επιπέδου στις διαστάσεις που το έχουμε ορίσει:

  ```
  Dense(np.prod(self.dungeon_shape), activation="tanh")
  ```

  Σε αντίθεση με τα υπόλοιπα επίπεδα Dense που έχουν την γραμμική συνάρτηση ως συνάρτηση ενεργοποίησης, το συγκεκριμένο επίπεδο έχει την υπερβολική εφαπτομένη για να πάρουμε έξοδο στο πεδίο τιμών [-1, 1] που είναι το πεδίο τιμών που έχουμε ορίσει για τις τιμές του επιπέδου κατά την περίοδο της εκπαίδευσης.

```
Dense(units=256, input_dim=self.dungeon_dimension)
LeakyReLU(alpha=0.2)
BatchNormalization(momentum=0.8)

Dense(units=512)
LeakyReLU(alpha=0.2)
Dropout(rate=0.3)

Dense(units=1024)
LeakyReLU(alpha=0.2)
BatchNormalization(momentum=0.8)

Dense(units=1024)
LeakyReLU(alpha=0.2)
Dropout(rate=0.3)

Dense(units=1024)
LeakyReLU(alpha=0.2)
BatchNormalization(momentum=0.8)

Dense(units=1024)
LeakyReLU(alpha=0.2)
Dropout(rate=0.3)

Dense(units=1024)
LeakyReLU(alpha=0.2)
BatchNormalization(momentum=0.8)

Dense(np.prod(self.dungeon_shape), activation="tanh")
```

4.4.2 Αρχιτεκτονική Discriminator

Το Dense μοντέλο του Discriminator δικτύου, είναι πολύ πιο μικρό από το μοντέλο του Discriminator. Αποτελείται από μόλις 5 επίπεδα στην τελική του μορφή, έγιναν και κάποιες δοκιμές με δύο επιπλέον επίπεδα Dropout αλλά στη συνέχεια αφαιρέθηκαν. Η είσοδος του δικτύου είναι ένα επίπεδο διανυσματικής μορφής με τις διαστάσεις και τις τιμές όπως τις έχουμε αναλύσει και έχει ως έξοδο μία τιμή στο συνεχές διάστημα [0, 1].

Τα επίπεδα του Discriminator όπως ορίζονται στην υλοποίηση:

```
Dense(units=512, input_dim=self.dungeon_dimension)
LeakyReLU(alpha=0.2)
Dense(units=256)
LeakyReLU(alpha=0.2)
Dense(1, input_dim=self.dungeon_dimension, activation="sigmoid")
```

Μπορούμε να διακρίνουμε μια παρόμοια αρχιτεκτονική με τον Generator, έχουν αρχικά ένα Dense επίπεδο και στη συνέχεια ένα επίπεδο LeakyReLU. Στον Discriminator παρόλαυτα δεν υπάρχει το επίπεδο του BatchNormalization καθώς δεν χρειάζεται να κρατήσουμε τις τιμές των εξόδων σε ένα συγκεκριμένο διάστημα.

Το επίπεδο της εξόδου παράγει μία τιμή με συνάρτηση ενεργοποίησης την σιγμοειδή συνάρτηση, ώστε να έχουμε το αποτέλεσμα μεταξύ 0 και 1. Όπου το 0 και τιμές κοντά στο 0 δηλώνουν ότι το επίπεδο αποτελεί παράγωγο του Generator και όχι κάποιο από τα δείγματα εκπαίδευσης και το 1 δηλώνει ότι το επίπεδο είναι από τα δείγματα εκπαίδευσης ή μοιάζει τόσο πολύ που δεν μπορεί να το ξεχωρίσει ο Discriminator.

4.5 Μοντέλο CNN GAN

Το μοντέλο GAN που αναφέρετε ως CNN είναι το δεύτερο μοντέλο GAN που αναπτύχθηκε και εκπαιδεύτηκε στα πλαίσια αυτής της εργασίας. Η σχεδίαση και ανάπτυξη του βασίστηκε στα Auxiliary Classifier GAN [54] τα οποία έχουν αναπτυχθεί για την παραγωγή εικόνων με συγκεκριμένο είδος περιεχομένου το οποίο ορίζετε από διακριτές κλάσεις. Ο Discriminator ενός Auxiliary Classifier GAN έχει ως στόχο να αναγνωρίσει την κλάση στην οποία ανήκει η εικόνα που παράχθηκε από τον Generator.

Αν και δεν είναι αυτός ο στόχος της εργασίας, δημιουργήσαμε το μοντέλο του CNN GAN με βάση την αρχιτεκτονική των Auxiliary Classifier GANs με μερικές αλλαγές για να ταιριάζει στον σκοπό αυτής της εργασίας. Όπως αναφέρθηκε τα αποτελέσματα του CNN GAN θεωρήθηκαν ως καλύτερης ποιότητας και πρωτοτυπίας από αυτά του Dense GAN.

4.5.1 Αρχιτεκτονική Generator

Το μοντέλο του Generator δικτύου, αποτελείται επίσης από 22 επίπεδα συνολικά αλλά σε αυτό το μοντέλο υπάρχουν εφτά διαφορετικά είδη επιπέδων. Μπορούμε

και πάλι να διακρίνουμε ομάδες επιπέδων να επαναλαμβάνονται μέσα στο δίκτυο, συγκεκριμένα την τετράδα των επιπέδων Conv2D, LeakyReLU, BatchNormalization και Dropout, την βλέπουμε τέσσερις φορές. Επιπλέον όμως βλέπουμε και άλλα επίπεδα στην αρχή και στο τέλος του δικτύου.

Τα δύο αρχικά επίπεδα, Dense και Reshape έχουν ως στόχο την αλλαγή της εισόδου στο κατάλληλο μέγεθος και διαστάσεις για τα επίπεδα Conv2D. Επίσης το ίδιο ισχύει και για το επίπεδο εξόδου Reshape που έχει ως στόχο να μετατρέψει την έξοδο στις διαστάσεις του επιπέδου που περιμένουμε να παραχθεί από το δίκτυο.

```
Dense(units=900, input_shape=(self.latent_size,))
Reshape(target_shape=self.dungeon_shape)

Conv2D(32, 3, padding='same', strides=2)
LeakyReLU(alpha=0.2)
BatchNormalization(momentum=0.8)
Dropout(0.3)

Conv2D(64, 3, padding='same', strides=1)
LeakyReLU(alpha=0.2)
BatchNormalization(momentum=0.8)
Dropout(0.3)

Conv2D(128, 3, padding='same', strides=2)
LeakyReLU(alpha=0.2)
BatchNormalization(momentum=0.8)
Dropout(0.3)

Conv2D(256, 3, padding='same', strides=1)
LeakyReLU(alpha=0.2)
BatchNormalization(momentum=0.8)
Dropout(0.3)

Flatten()
Dense(units=900)
BatchNormalization(momentum=0.8)

Reshape(target_shape=self.dungeon_shape)
```

• **Reshape** Το επίπεδο Reshape δεν εφαρμόζει καμία συνάρτηση στα δεδομένα

αλλά αλλάζει την διάταξη τους ώστε να πάρουν άλλο σχήμα και διαστάσεις. Είναι πολύ χρήσιμο επίπεδο για περιπτώσεις όπως σε αυτή την υλοποίηση που τα δεδομένα δεν έχουν τις σωστές διαστάσεις για να δοθούν ως είσοδος σε ένα επίπεδο Conv2D.

- **Conv2D** Το επίπεδο που επεξεργάζεται τις τιμές των δεδομένων μέσα από την εφαρμογή μιας συνάρτησης συνέλιξης των δεδομένων με έναν πυρήνα συνέλιξης. [53]

  ```
  Conv2D(32, 3, padding='same', strides=2)
  Dropout(rate=0.3)
  ```

  Η πρώτη παράμετρος ορίζει τον αριθμό των διαστάσεων της εξόδου.

  Η δεύτερη παράμετρος ορίζει το μέγεθος του πυρήνα με τον οποίο θα γίνει η συνέλιξη, για παράδειγμα ο αριθμός 3 ορίζει έναν πυρήνα 3x3 διαστάσεων.

  Η παράμετρος padding ορίζει εάν θα προστεθούν επιπλέον στοιχεία για να μην μειωθεί το μέγεθος των δεδομένων λόγω της συνέλιξης, με τιμή 'same' ορίζουμε ότι δεν θέλουμε να μειωθεί το μέγεθος των δεδομένων.

  Η παράμετρος strides ορίζει πόσα "βήματα" θα κάνει κάθε φορά ο πυρήνας της συνέλιξης πάνω στα δεδομένα.

4.5.2 Αρχιτεκτονική Discriminator

Αντίστοιχα με το Discriminator μοντέλο του Dense δικτύου, έτσι και του CNN είναι πολύ πιο μικρό σε σχέση με τον Generator. Παρατηρούμε ομοιότητες με το μοντέλο του Generator του CNN αλλά και με το Discriminator του Dense.

```
Conv2D(32, 3, padding='same', strides=2, input_shape=self.dungeon_
LeakyReLU(0.2)
Dropout(0.3)

Conv2D(64, 3, padding='same', strides=1)
LeakyReLU(0.2)
Dropout(0.3)

Flatten()
Dense(1, activation="sigmoid")
```

4.6 Εκπαίδευση GAN

Για την εκπαίδευση των δύο GAN χρησιμοποιήθηκε το ίδιο dataset όπως αυτό δημιουργήθηκε από την υλοποίηση του PCG. Το dataset αποτελείται από 114 αρχεία όπου το κάθε αρχείο αντιστοιχεί σε ένα επίπεδο. Πριν ξεκινήσει η διαδικασία την εκπαίδευσης, φορτώνονται όλα τα επίπεδα στην μνήμη και περνάνε από το στάδιο της προ επεξεργασίας για να έρθουν σε μορφή που να μπορεί να επεξεργαστούν τα δύο μοντέλα GAN.

Η διαδικασία της εκπαίδευσης είναι κοινή για τα δύο μοντέλα.

4.6.1 Μεταβλητές εκπαίδευσης

Κατά την δημιουργία του μοντέλου ορίζονται μεταβλητές που θα χρησιμοποιηθούν στο στάδιο της εκπαίδευσης. Όλες οι μεταβλητές έχουν προ ρυθμισμένες τιμές οι οποίες βγήκαν μέσα από δοκιμές. Μερικές από τις πιο σημαντικές είναι:

- **epochs** Θετικός ακέραιος αριθμός που ορίζει τον αριθμό των εποχών που θα επαναληφθεί η εκπαίδευση. Η προτεινόμενη τιμή του για αυτή την υλοποίηση είναι 50000.

- **batch_size** Θετικός ακέραιος αριθμός που ορίζει τον αριθμό των δειγμάτων που θα δίνονται σε κάθε εποχή για εκπαίδευση. Εάν το batch_size είναι μικρότερο του μεγέθους του dataset, τα δείγματα που θα συμμετέχουν στην εκπαίδευση επιλέγονται τυχαία με δειγματοληψία σε κάθε εποχή και χωρίς επανατοποθέτηση. Στην περίπτωση που το batch_size είναι μεγαλύτερο του μεγέθους του dataset, ορίζετε ως batch_size το μέγεθος του dataset. Η προτεινόμενη τιμή του για αυτή την υλοποίηση είναι 64.

- **sample** Θετικός ακέραιος αριθμός που ορίζει το διάστημα ανάμεσα στις εποχές που λαμβάνουμε ένα δείγμα επιπέδου από το μοντέλο σε εκείνο το στάδιο της εκπαίδευσης. Αυτό ορίστηκε για να μπορούμε να παρακολουθούμε την πρόοδο του δικτύου όσο εκπαιδεύεται. Η προτεινόμενη τιμή του για αυτή την υλοποίηση είναι 1000.

- **train_discriminator** Μία δυαδική μεταβλητή που ορίζει εάν θα πρέπει να εκπαιδεύσουμε και το δίκτυο του Discriminator. Σε περίπτωση που οριστεί να μην εκπαιδευτεί, το δίκτυο του Discriminator δεν θα πραγματοποιεί διόρθωση βαρών ανά τις εποχές της εκπαίδευσης. Παρατηρήθηκαν καλύτερα

αποτελέσματα όταν εκπαιδεύαμε και το δίκτυο του Discriminator. Η προτεινόμενη τιμή του για αυτή την υλοποίηση είναι *True*.

4.6.2 Είσοδος εκπαίδευσης Generator

Για την εκπαίδευση του Generator δίνουμε ως είσοδο στο δίκτυο, δύο πίνακες που περιέχουν ίδιο αριθμό δειγμάτων όπου το κάθε δείγμα έχει την μορφή προ επεξεργασμένου επιπέδου. Ο αριθμός των δειγμάτων ορίζετε από το batch_size. Ο ένας πίνακας περιέχει δείγματα από το dataset ενώ ο άλλος πίνακας περιέχει *θόρυβο*, τυχαία δημιουργημένα δείγματα που έχουν τιμές στο διάστημα που ορίσαμε για τα δείγματα των επιπέδων.

Τα στοιχεία των δύο πινάκων αντιστοιχίζονται ένα προς ένα καθώς ως στόχο το δίκτυο έχει να μετατρέψει τα δείγματα του θορύβου στα αντίστοιχα δείγματα του dataset μέσα από τις μετατροπές που θα εφαρμόσει με την εισαγωγή του θορύβου και το πέρασμα του από τα επίπεδα του δικτύου. Το τελικό αποτέλεσμα συγκρίνετε με το επιθυμητό αποτέλεσμα, το επίπεδο από το dataset, και διορθώνονται τα βάρη των επιπέδων αντίστοιχα.

4.6.3 Λειτουργία Generator

Για την λειτουργία του Generator εκτός του σταδίου της εκπαίδευσης, πρέπει να του δώσουμε ως είσοδο ένα δείγμα θορύβου για να μας δημιουργήσει ένα δείγμα επιπέδου.

4.6.4 Είσοδος εκπαίδευσης Discriminator

Για την εκπαίδευση του Discriminator δίνουμε ως είσοδο έναν πίνακα και ένα διάνυσμα. Ο πίνακας περιέχει τα δείγματα που θέλουμε να αξιολογήσει ως τεχνητά από τον Generator ή από το dataset και το διάνυσμα έχει δυαδικές τιμές που δηλώνουν την πραγματική προέλευση του κάθε δείγματος με τις τιμές 0 και 1.

Για να εκπαιδευτεί ο Discriminator παίρνει ως είσοδο τα επίπεδα και ελέγχει την έξοδο του εάν ταιριάζει με την αντίστοιχη τιμή για αυτό το επίπεδο με το διάνυσμα των πραγματικών τιμών. Ομοίως διορθώνει τα βάρη του στις περιπτώσεις λάθους εάν έχουμε ενεργοποιήσει την μεταβλητή train_discriminato.

4.7 Αποτελέσματα

Κεφάλαιο 5

Βιβλιογραφία

[1] Togelius, J. Kastbjerg, E. Schedl, D. Yannakakis, G.N. *What is procedural content generation?: Mario on the borderline.* In: Proceedings of the 2nd Workshop on Procedural Content Generation in Games (2011).

[2] Will Wright. *Spore & procedural generation.* In: Game Developers Conference (2005)
https://www.gdcvault.com/play/1019981/The-Future-of-Content

[3] Noor Shaker, J. Togelius, M. J. Nelson. *Procedural Content Generation in Games.* Chapter 1.1:2-3 (2016)

[4] Rogue. Epyx (1986)
http://pc.gamespy.com/pc/ftl-faster-than-light/1227287p1.html

[5] Spore. Maxis, Electronic Arts (2008)

[6] Chris Baker. "'No Man's Sky': How Games Are Building Themselves".In: Rolling Stone. (2016)

[7] Noor Shaker, J. Togelius, M. J. Nelson. *Procedural Content Generation in Games.* Chapter 1. 6-7 (2016)

[8] Togelius, J. Champandard, A.J. Lanzi, P. L. Mateas, M. Paiva, A. Preuss, M. Stanley. (2013). Procedural content generation: goals, challenges and actionable steps. Dagstuhl Follow-Ups. 6. 61-75.

[9] Noor Shaker, J. Togelius, M. J. Nelson. *Procedural Content Generation in Games.* Chapter 1. 2-3 (2016)

[10] Mount & Blade II Bannerlord. TaleWorlds Entertainment (2020)

[11] Oxygen Not Included. Klei Entertainment (2019)

[12] Noor Shaker, J. Togelius, M. J. Nelson. *Procedural Content Generation in Games*. Chapter 3.2. 33-38 (2016)

[13] Roland van der Linden, R. Lopes, R. Bidarra. *Procedural generation of dungeons*. In: IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES (2014)

[14] Noor Shaker, J. Togelius, M. J. Nelson. *Procedural Content Generation in Games*. Chapter 3. 31-55 (2016)

[15] Fuchs, Henry, Kedem, Zvi. M, Naylor, Bruce F.*On Visible Surface Generation by A Priori Tree Structures*. In: '80 Proceedings of the 7th annual conference on Computer graphics and interactive techniques. ACM, New York. pp. 124–133. (1980)

[16] Aluru, S. *Quadtrees and octrees*. In: D. Mehta and S. Sahni (ed.). Handbook of Data Structures and Applications. Chapman and Hall/CRC. pp. 19-1 – 19-26. (2004)

[17] Lawrence Johnson, Georgios N. Yannakakis, Julian Togelius. *Cellular automata for real-time generationof infinite cave levels*. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games. (2010)

[18] Noor Shaker, j. Togelius, M. J. Nelson. *Procedural Content Generation in Games*. Chapter 3.3. 38-42 (2016)

[19] Rozenberg, G. *Handbook of Graph Grammars and Computing by Graph Transformation*. Volume I. World Scientific (1997)

[20] Noor Shaker, J. Togelius, M. J. Nelson. *Procedural Content Generation in Games*. Chapter 12. 215-224 (2016)

[21] Antonios Liapis, Georgios N. Yannakakis, Julian Togelius. *Towards a Generic Method of Evaluating Game Levels*. In: Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.(2013)

[22] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge. *A Neural Algorithm of Artistic Style*.(2015)

[23] Unity Game Engine
https://unity.com/

[24] Unreal Game Engine
https://www.unrealengine.com/en-US/

[25] Godot
https://godotengine.org/

[26] Saiqa Aleem, Luiz Fernando Capretz, Jason Gregory. *Game Engine Architecture*. Third Edition. Section II. 415-995.(2019)

[27] Aleem, S., Capretz, L.F. & Ahmed, F. *Game development software engineering process life cycle: a systematic review*. J Softw Eng Res Dev 4, 6 (2016). https://doi.org/10.1186/s40411-016-0032-7

[28] Unity Tilemap.
https://docs.unity3d.com/Manual/class-Tilemap.html

[29] Unity Canvas.
https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/class-Canvas.html

[30] Adam M. Smith, Michael Mateas. *Answer Set Programming for Procedural Content Generation: A Design Space Approach*. In: IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 3, NO. 3. (2011)

[31] Daniele Gravina, Antonios Liapis, Georgios N. Yannakakis. *Constrained surprise search for content generation*. In: IEEE Conference on Computational Intelligence and Games (CIG). (2016)

[32] D. Karavolos, A. Liapis and G. N. Yannakakis. *A Multi-Faceted Surrogate Model for Search-based Procedural Content Generation*. In: IEEE Transactions on Games, doi: 10.1109/TG.2019.2931044. (2018)

[33] Cook Michael, Simon Colton *A Rogue Dream: Automatically Generating Meaningful Content For Games*. (2014)

[34] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, Julian Togelius *Procedural Content Generation via Machine Learning (PCGML)*. (2017)

[35] J. Dormans, S. Bakkes. *Generating Missions and Spaces for Adaptable Play Experiences*. In: IEEE Transactions on Computational Intelligence and AI in Games, vol. 3, no. 3, pp. 216-228, doi: 10.1109/TCIAIG.2011.2149523. (2011)

[36] Shaker, Noor, Yannakakis, Georgios, Togelius, Julian. *Towards Automatic Personalized Content Generation for Platform Games*. In: Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE. (2010)

[37] Osvaldo Simeone. *A Very Brief Introduction to Machine LearningWith Applications to Communication Systems* (2018)

[38] M. Tim Jones. *Machine learning and gaming*. In: IBM Developer (2019) https://developer.ibm.com/technologies/artificial-intelligence/articles/machine-learning-and-gaming/

[39] Silver, D., Huang, A., Maddison, C. et al. *Mastering the game of Go with deep neural networks and tree search*. In: Nature 529, 484–489 (2016) https://doi.org/10.1038/nature16961

[40] Summerville, James. *Sampling Hyrule: Multi-Technique Probabilistic Level Generation for Action Role Playing Games*. In: Experimental AI in Games: Papers from the AIIDE 2015 Workshop (2015)

[41] Wexler, James. *Artificial Intelligence in Games : A look at the smarts behind Lionhead Studio ' s " Black and White " and where it can and will go in the future.*

[42] Hastings, Erin J.; Guha, Ratan K.; Stanley, Kenneth O. *Evolving content in the Galactic Arms Race video game*. In: IEEE Symposium on Computational Intelligence and Games. IEEE: 241–248. (2009)

[43] Hinton, G.E., Krizhevsky, A., Srivastava, N., Sutskever, I., & Salakhutdinov, R. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. In: Journal of Machine Learning Research, 15, 1929-1958. (2014)

[44] Sergey Ioffe, Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. In: ICML. (2015)

[45] Bengio, Y., Courville, A.C., Goodfellow, I.J., Mirza, M., Ozair, S., Pouget-Abadie, J., Warde-Farley, D., & Xu, B. *Generative Adversarial Nets*. In: NIPS. (2014)

[46] Goodfellow, Ian, Bengio, Yoshua, Courville, Aaron. *Back-Propagation and Other Differentiation Algorithms*. In: Deep Learning. MIT Press. pp. 200–220. (2016)

[47] David Gündisch. Writing your first Generative Adversarial Network with Keras. https://towardsdatascience.com/writing-your-first-generative-adversarial-network-with-keras-2d16fd8d48891

[48] Renu Khandelwal. Generative Adversarial Network (GAN) using Keras. https://medium.com/datadriveninvestor/generative-adversarial-network-gan-using-keras-ce1c05cfdfd3

[49] Collection of Keras implementations of Generative Adversarial Networks (GANs). https://github.com/eriklindernoren/Keras-GAN

[50] Keras Dense Layer. https://keras.io/api/layers/core_layers/dense

[51] Keras LeakyReLU Layer. https://keras.io/api/layers/activation_layers/leaky_relu

[52] Keras BatchNormalization Layer. https://keras.io/api/layers/normalization_layers/batch_normalization

[53] Keras Conv2D Layer. https://keras.io/api/layers/convolution_layers/convolution2d/

[54] How to Develop an Auxiliary Classifier GAN. https://machinelearningmastery.com/how-to-develop-an-auxiliary-classifier-gan-ac-gan-from-scratch-with-keras/