ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

UNIVERSITY OF PIRAEUS
DEPARTMENT OF DIGITAL SYSTEMS

Δυναμική διαμόρφωση επιπέδου δυσκολίας παιγνίων υπολογιστών σε διαδικαστικά παραγόμενα περιβάλλοντα

Άγγελος Γεωργιάδης - Ε12029

Επιβλέπων καθηγητής - Απόστολος Μηλιώνης

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ

1.1. Αντικείμενο της εργασίας

Αντικείμενο της παρούσας εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός μοντέλου δυναμικής διαμόρφωσης επιπέδου δυσκολίας παιχνιδιών, με σκοπό την διατήρηση της εμπειρίας του παιχνιδιού όσο πιο διασκεδαστική και δίκαιη για τον παίχτη, ανάλογα με τις ικανότητές του, σε ένα όσο το δυνατόν περισσότερο χαοτικό και τυχαίο περιβάλλον. Η υλοποίηση της εργασίας είχε πιο πολύ εκπαιδευτικό παρά ερευνητικό σκοπό και στόχευε στην εξοικείωσή μου με το αντικείμενο της δημιουργίας παιχνιδιών.

1.2. Διάρθρωση της εργασίας

Στη παρούσα εργασία παρουσιάζεται ο τρόπος με τον οποίο λειτουργούν και αλληλεπιδρούν μεταξύ τους τα διάφορα συστήματα τα οποία χρησιμοποιούνται στο παιχνίδι που αναπτύχθηκε.
Αρχικά, στο Κεφάλαιο 2 γίνεται μια εισαγωγή σε συγκεκριμένες έννοιες οι οποίες χρησιμοποιούνται εκτενώς και αποτελούν βασικό κομμάτι του project. Εξηγούνται λεπτομερώς έννοιες όπως τι είναι ένα βιντεοπαιχνίδι και σε ποιές κατηγορίες κατατάσσονται,τι σημαίνει 2D Action Platformer και άλλες έννοιες. Στη συνέχεια, στο Κεφάλαιο 3 παρουσιάζονται σχετικά παιχνίδια τα οποία έχουν υλοποιημένες και κάνουν χρήση αυτές τις έννοιες. Έπειτα, στο Κεφάλαιο 4 γίνεται περιγραφή και ανάλυση του παιχνιδιού που αναπτύχθηκε, αιτιολογώντας γιατί υλοποιήθηκαν τα διάφορα συστήματα και με ποιον τρόπο αλληλεπιδρούν μεταξύ τους. Τέλος, στο Κεφάλαιο 5 γίνεται μία σύνοψη της εργασίας και παρουσιάζονται τα συμπεράσματα και οι προοπτικές για πιθανές μελλοντικές εργασίες πάνω στο συγκεκριμένο αντικείμενο.

1.3. Unity

Το Unity είναι μία μηχανή ανάπτυξης παιχνιδιών (Game engine) και χρησιμοποιείται κυρίως για την ανάπτυξη δισδιάστατων και τρισδιάστατων βιντεοπαιχνιδιών και προσομοιώσεων σε υπολογιστές, κονσόλες και κινητές συσκευές. Οι μηχανές ανάπτυξης παιχνιδιών συνήθως προσφέρουν πολλές λειτουργίες στον προγραμματιστή, όπως οι λειτουργίες φυσικής (Physics), ώστε να μη χρειάζεται να τα αναπτύξει ο ίδιος, και να μπορεί να επικεντρωθεί στην ανάπτυξη του παιχνιδιού και στους εσωτερικούς μηχανισμούς αυτού. Ωστόσο ο προγραμματιστής μπορεί να διαμορφώσει τις παραμέτρους αυτών των λειτουργιών, όπως για παράδειγμα τη βαρύτητα. Επίσης το Unity αποτελείται από κάποια βασικά εργαλεία που συμβάλουν στην ανάπτυξη των παιχνιδιών, όπως για παράδειγμα το Scene που δίνει στον προγραμματιστή τη δυνατότητα σκηνοθεσίας και τοποθέτησης αντικειμένων μέσα στην σκηνή του παιχνιδιού, τον Inspector μέσω του οποίου ο προγραμματιστής μπορεί να εισάγει ή να διαγράψει κομμάτια λειτουργιών(Components) από τα αντικείμενα της σκηνής, και να μετατρέψει βασικές παραμέτρους των κομματιών αυτών. Οι γλώσσες προγραμματισμού που υποστηρίζει είναι η Javascript και η πιο δημοφιλής C# και στο project που αναπτύχθηκε γίνεται χρήση της C#.

1.4. Γραφικά

Τα γραφικά που χρησιμοποιούνται στο παιχνίδι αποτελούν μέρος του Open Pixel Project, το οποίο είναι ένα project που παρέχει δωρεάν μία μεγάλη συλλογή γραφικών. Η συλλογή αποτελείται από δισδιάστατους χαρακτήρες κινούμενων σχεδίων, περιβάλλοντα, αντικείμενα και γραφικά διεπαφής. Η συγκεκριμένη συλλογή επιλέχθηκε λόγω της μεγάλης ποικιλίας που παρέχει. Κάποια γραφικά χρησιμοποιήθηκαν έτσι όπως ήταν και κάποια διαμορφώθηκαν ώστε να ταιριάζουν με τις ανάγκες του παιχνιδιού. Το project βρίσκεται στην σελίδα http://www.openpixelproject.com/

2. ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

2.1. Βιντεοπαιχνίδι

Με τον όρο βιντεοπαιχνίδι εννοείται οποιοδήποτε παιχνίδι πραγματοποιείται με την χρήση κάποιας ηλεκτρονικής συσκευής. Αυτή μπορεί να είναι ένας ηλεκτρονικός υπολογιστής, μια κονσόλα βιντεοπαιχνιδιών, ένα κινητό τηλέφωνο και άλλα.

Όλα τα ηλεκτρονικά παιχνίδια έχουν μια μορφή εισόδου δεδομένων από τον χρήστη: πληκτρολόγιο, ποντίκι, joystick, gamepad, οθόνη αφής κ.α., και μια μορφή εξόδου που ικανοποιεί τις αισθήσεις του παίχτη: οθόνη υπολογιστή ή τηλεόραση, ηχεία και απτική τεχνολογία, μεταξύ άλλων.

Τα βιντεοπαιχνίδια κατατάσσονται σε διάφορες κατηγορίες,οι οποίες με τη σειρά τους χωρίζονται σε υποκατηγορίες. Μερικά παραδείγματα κατηγοριών φαίνονται παρακάτω

- Δράσης(Action Games)
- Περιπέτειας(Adventure Games)
- Δράσης περιπέτειας(Action Adventure Games)
- Δράσης τρόμου(Action Horror Game)
- Ρόλων(Role Playing Games)
- Δράσης Ρόλων(Action Role Playing Games)
- Στρατηγικής(Strategy games)
- MMOG(Massive Multiplayer Online Games)
- MMORPG(Massive Multiplayer Online Role Playing Games)
- Πλατφόρμας(Platformer Games)
- Πάλης(Fighting Games)
- Beat'em up
- Shoot'em up
- Βολών πρώτου προσώπου(First Person Shooter)
- Βολών τρίτου προσώπου(Third Person Shooter)
- Εξομοίωσης(Simulation Games)
- Παζλ(Puzzle Games)
- Αγώνων(Racing Games)
- Αθλητισμού(Sports Games)
- Arcade games(Περιλαμβάνουν διάφορα είδη)

Επίσης τα βιντεοπαιχνίδια διαχωρίζονται σε δισδιάστατα(2D) και τρισδιάστατα(3D) ανάλογα με τον τύπο εικόνας που παράγεται.

2.2. 2D Platformer

Τα παιχνίδια κατατάσσονται σε διάφορες κατηγορίες ανάλογα με το σκοπό τους, τις διάφορες αλληλεπιδράσεις που παρέχουν, τα γραφικά και τη αφηγηματική κατεύθυνση τους(narrative direction).

Το παιχνίδι που σχεδιάστηκε και υλοποιήθηκε κατατάσσεται στην υποκατηγορία Platformer,η οποία ανήκει με τη σειρά της στην κατηγορία Action. Στη συγκεκριμένη κατηγορία ο παίχτης ελέγχει έναν χαρακτήρα και πρέπει να διασχίσει το περιβάλλον με διάφορους μηχανισμούς. Ο παίχτης ελέγχει τις κινήσεις ώστε να αποφύγει τον θάνατο του χαρακτήρα που ελέγχει. Ένας από τους πιο συχνά εμφανιζόμενους μηχανισμούς στη συγκεκριμένη κατηγορία παιχνιδιών είναι το άλμα.

Σκοπός των Action Platformer είναι να φτάσει ο χαρακτήρας από ένα αρχικό σημείο σε ένα τελικό αποφεύγοντας διάφορα εμπόδια, ώστε να προχωρήσει στο επόμενο επίπεδο(Level). Μόλις το καταφέρει αυτό, φορτώνεται το επόμενο επίπεδο που πιθανώς είναι δυσκολότερο από το προηγούμενο ή παρουσιάζει κάποιο νέο εμπόδιο στο παιχνίδι.

2.3. Procedural Content Generation (PCG)

2.3.1. Ορισμός

Η διαδικαστική παραγωγή περιεχομένου(Procedural Content Generation) είναι μία μέθοδος παραγωγής δεδομένων αλγοριθμικά αντί για χειρωνακτικά. Στα βιντεοπαιχνίδια χρησιμοποιείται για την δημιουργία τεράστιας ποσότητας περιεχομένου,όπως για παράδειγμα για τη δημιουργία κόσμων,επιπέδων,χαρακτήρων,γραφικών αλλά και ιστοριών. Τα πλεονεκτήματα της χρήσης αυτής της μεθόδου παραγωγής αποτελούν μικρότερα μεγέθη αρχείων,μεγαλύτερες ποσότητες περιεχομένου και τυχαιότητα για λιγότερο προβλέψιμες αλληλεπιδράσεις μέσα στο παιχνίδι. Συνήθως γίνεται χρήση μιας συνάρτησης ή διαδικασίας η οποία δεδομένης μίας τυχαίας τιμής σπόρου(seed) παράγει διαφορετικό αποτέλεσμα κάθε φορά που εκτελείται. Για τον υπολογισμό της τυχαίας τιμής αυτής συνήθως γίνεται χρήση της ημερομηνίας και ώρας που εκτελείται η συνάρτηση.

2.3.2. Κυτταρικά Αυτόματα

Ένα κυτταρικό ή κυψελικό αυτόματο είναι ένα υπολογιστικό μοντέλο συστημάτων με αναδυόμενη πολυπλοκότητα. Τα κυτταρικά αυτόματα μελετώνται στη θεωρία υπολογισμού, στη φυσική, στη θεωρητική βιολογία και αλλού. Επινοήθηκαν με σκοπό την τυπική περιγραφή των λειτουργιών του βιολογικού κυττάρου. Πρόκειται για μια «κοινωνία» ατόμων (ψηφιακών και όχι μόνο), που κινούνται καθώς περνά ο χρόνος σε ένα πλέγμα (είτε τετραγώνων είτε άλλων σχημάτων που είναι συνήθως δισδιάστατα), γεννούν και πεθαίνουν, ή ακόμα τρέφονται. Όσο πιο περίπλοκες λειτουργίες τούς δοθούν, τόσο περισσότερα μπορούν να κάνουν. Τέτοιου είδους πειράματα χρησιμεύουν κατ' αρχάς στην προσομοίωση της συμπεριφοράς ζωντανών βιολογικών οργανισμών. Αλλά έχουν επίσης και άλλες εφαρμογές. Για παράδειγμα, μελετάται έτσι η αυτοοργάνωση μιας κοινωνίας ανεξαρτήτων μελών, η εξέλιξή της, η δημιουργία εικόνας τάξης από μια τυχαία κατάσταση αταξίας, ακόμα και η μαθηματική τυχαιότητα σε μια κοινότητα. Παράλληλα όμως, μπορεί να είναι και ένα πολύ ενδιαφέρον ερευνητικό παιχνίδι, με εντελώς απρόβλεπτη και χαοτική συμπεριφορά των ατόμων του.

2.4. Dynamic Difficulty Adjustment (DDA)

Η δυναμική διαμόρφωση επιπέδων δυσκολίας(Dynamic Difficulty Adjustment) παιχνιδιών αποτελεί τη διαδικασία αυτόματης διαμόρφωσης παραμέτρων, σεναρίων και συμπεριφορών σε ένα βιντεοπαιχνίδι σε πραγματικό χρόνο, βασισμένο στης ικανότητες του παίχτη. Ο σκοπός του DDA είναι η διατήρηση του ενδιαφέροντος του παίχτη σε όλη τη διάρκεια του παιχνιδιού, προσφέροντας του τη κατάλληλη πρόκληση, ανάλογα με τις ικανότητες του. Συνήθως, το επίπεδο δυσκολίας στα παιχνίδια αυξάνεται σταθερά κατά τη διάρκεια του παιχνιδιού (είτε με γραμμικό τρόπο, είτε με βήματα που συμβολίζονται από τα επίπεδα). Οι τιμές των παραμέτρων μπορούν να επιλεγούν μόνο στην αρχή του παιχνιδιού επιλέγοντας ένα επίπεδο δυσκολίας, συνήθως της μορφής Easy, Medium, Hard. Όμως, αυτό μπορεί να οδηγήσει σε μία απογοητευτική εμπειρία για τον παίχτη, είτε είναι έμπειρος είτε άπειρος, διότι μπορεί η επιλογή του επιπέδου δυσκολίας που κάνει στην αρχή του παιχνιδιού να μην ταιριάζει στο επίπεδο εμπειρίας του.

Η δυναμική διαμόρφωση δυσκολίας επιχειρεί να διορθώσει αυτό το πρόβλημα δημιουργώντας μία εμπειρία φτιαγμένη στα μέτρα του κάθε παίχτη. Καθώς οι ικανότητες του χρήστη προοδεύουν (δηλαδή βελτιώνονται μαθαίνοντας τους διάφορους μηχανισμούς), οι προκλήσεις του παιχνιδιού θα πρέπει επίσης να αυξάνονται. Ωστόσο, ο σχεδιασμός τέτοιων χαρακτηριστικών αποτελεί δύσκολο κομμάτι για τους προγραμματιστές παιχνιδιών, και γι’αυτό το λόγο αυτή η μέθοδος δεν είναι αρκετά διαδεδομένη. Μερικά παραδείγματα παραμέτρων τα οποία μπορούν να χρησιμοποιηθούν σε ένα DDA σύστημα αποτελούν :

- Η ταχύτητα των εχθρών
- Η συχνότητα των εχθρών
- Οι δυνατότητες των εχθρών
- Οι δυνατότητες του παίχτη
- Το μέγεθος του χάρτη
- Η διάρκεια του παιχνιδιού

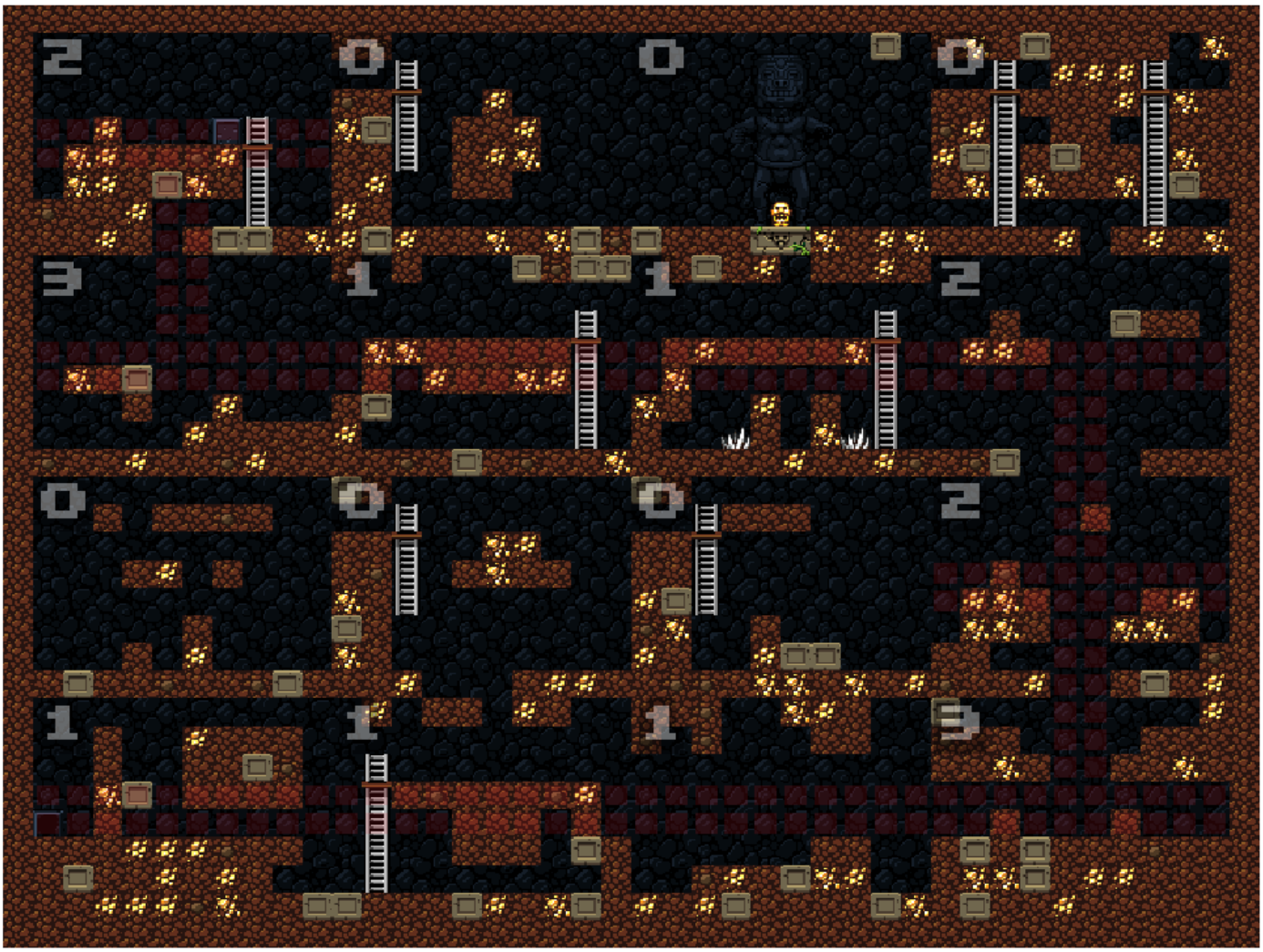
3. ΣΧΕΤΙΚΕΣ ΕΡΓΑΣΙΕΣ

3.1. Spelunky

Το Spelunky είναι ένα παιχνίδι πλατφόρμας(Platformer), κατασκευασμένο από τον Derek Yu, στο οποίο ο παίχτης ελέγχει έναν εξερευνητή ο οποίος διερευνά μία σειρά σπηλιών, ενώ παράλληλα μαζεύει θησαυρούς, σώζει ζωές, παλεύει αντιπάλους και αποφεύγει παγίδες. Οι σπηλιές είναι διαδικαστικά παραγόμενες, κάνοντας το κάθε παιχνίδι μοναδικό. Η διαδικασία με την οποία παράγεται μία σπηλιά εξηγείται παρακάτω.

Αρχικά, μία σπηλιά αποτελείται από 16 δωμάτια σε ένα 4x4 πλέγμα. Υπάρχουν 4 είδη βασικών δωματίων τα οποία είναι κατασκευασμένα με το χέρι :

- 0 : Δωμάτιο που δεν ανήκει στο μονοπάτι λύσης
- 1 : Δωμάτιο που σίγουρα έχει αριστερή και δεξιά έξοδο
- 2 : Δωμάτιο που σίγουρα έχει αριστερή,δεξιά και κάτω έξοδο.Αν υπάρχει δωμάτιο “2” από πάνω του, τότε σίγουρα θα έχει και πάνω έξοδο
- 3 : Δωμάτιο που σίγουρα έχει αριστερή,δεξιά και πάνω έξοδο

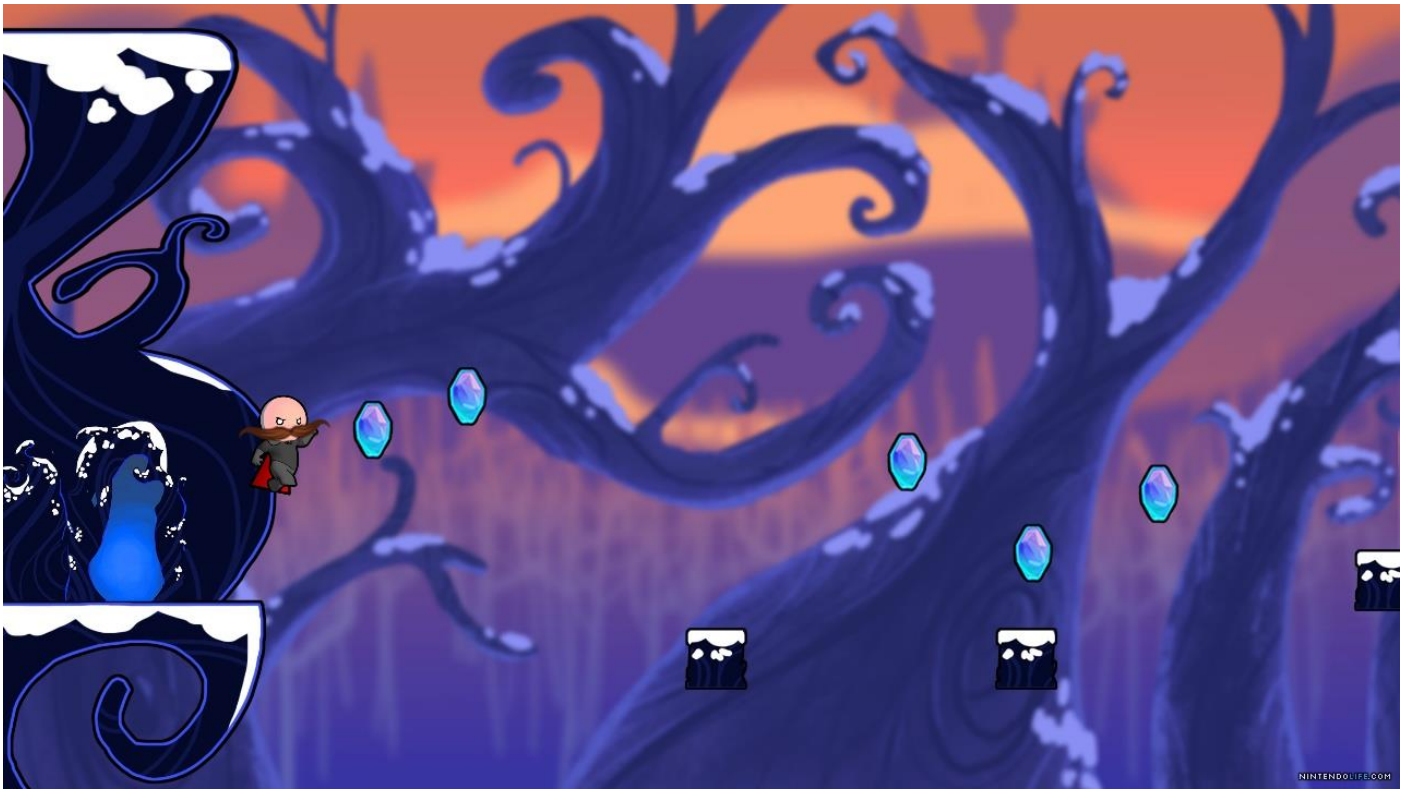


Η πρώτη ενέργεια της γεννήτριας είναι να τοποθετεί ένα αρχικό δωμάτιο στην πρώτη γραμμή. Ο τύπος δωματίου δεν επηρεάζει το αρχικό δωμάτιο, αλλά γενικά ένα αρχικό δωμάτιο είναι τύπου 1 ή 2. Κάθε φορά που ένα δωμάτιο τοποθετείται είναι τύπου 1. Έπειτα, αποφασίζει που θα πάει μετά. Επιλέγει τυχαία(Ομοιόμορφη κατανομή) έναν αριθμό από 1 ως 5. Αν ο αριθμός είναι 1 ή 2, το μονοπάτι πηγαίνει αριστερά.Αν ο αριθμός είναι 3 ή 4, το μονοπάτι πηγαίνει δεξιά. Αν είναι 5 το μονοπάτι πηγαίνει κάτω.(Αν το μονοπάτι βρίσκεται στα άκρα τότε το μονοπάτι αυτόματα πηγαίνει κάτω και αλλάζει την οριζόντια κατεύθυνσή του). Αν το μονοπάτι μετακινείται οριζόντια δεν έχουμε πρόβλημα επειδή τοποθετούνται δωμάτια που πάντα έχουν αριστερή και δεξιά έξοδο(τύπου 1). Όμως αν αποφασίσει να πάει κάτω, πρέπει να αλλάξει ο τύπος δωματίου σε 2 ώστε να έχει έξοδο. Μόλις προχωρήσει στο επόμενο δωμάτιο, ελέγχει αν το προηγούμενο δωμάτιο ήταν τύπου 2. Αν είναι αληθές, τότε το τρέχων δωμάτιο πρέπει να είναι τύπου 2 με όλες τις εξόδους ανοιχτές, ή τύπου 3. Εφόσον τα δωμάτια τύπου 2 και 3 έχουν αριστερή και δεξιά έξοδο, μπορούμε να ξανατρέξουμε τον αλγόριθμο. Αν είμαστε στη τελευταία γραμμή, αντί να κατέβουμε ένα επίπεδο τοποθετούμε το δωμάτιο στο οποίο ολοκληρώνεται το επίπεδο. Τέλος, τοποθετούνται τυχαία τα δωμάτια τύπου 0 τα οποία δεν έχουν σχέση με το τελικό μονοπάτι και μπορεί να μην συνδέονται.

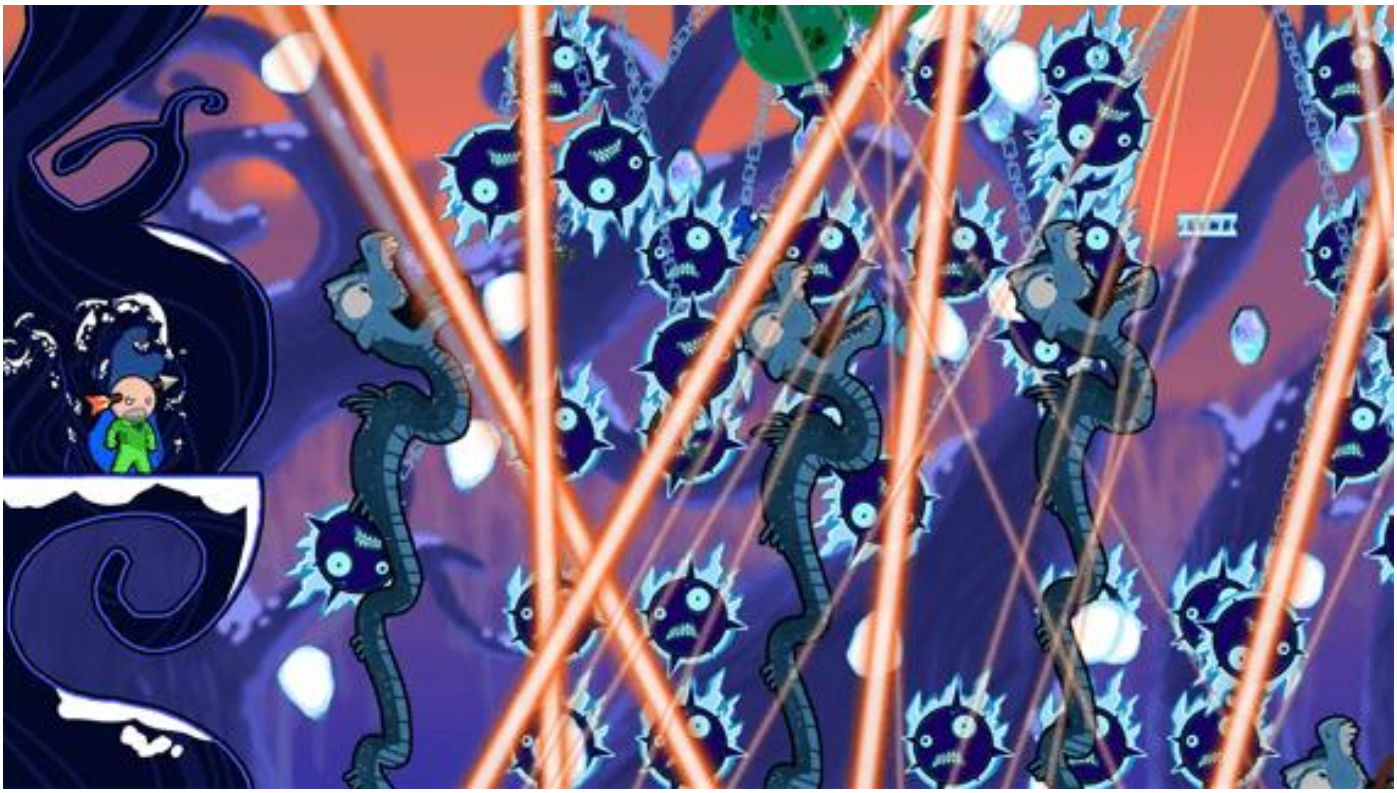
3.2. Cloudberry Kingdom

Το Cloudberry Kingdom είναι επίσης ένα παιχνίδι πλατφόρμας(Platformer), κατασκευασμένο από το Pwnee Studio. Το παιχνίδι χρησιμοποιεί ένα σετ αλγορίθμων οι οποίοι δημιουργούν διαδικαστικά παραγόμενα επίπεδα. Τα επίπεδα προσαρμόζονται στο επίπεδο ικανοτήτων του παίχτη, στις ικανότητες του χαρακτήρα του παιχνιδιού και στις αλλαγές των κανόνων φυσικής του παιχνιδιού. Το Cloudberry Kingdom δεν κάνει απότομες αλλαγές στο παιχνίδι που σημαίνει ότι ο παίχτης δεν έχει την αίσθηση ότι το παιχνίδι τον συγχωρεί ή τον τιμωρεί, το οποίο μπορεί να φανεί αρκετά αρκετά ενοχλητικό για πολλούς.

Επίσης, το Clourberry Kingdom παρουσιάζει ιδιαίτερο ενδιαφέρον όσον αφορά την τοποθέτηση των αντιπάλων, καθώς τα επίπεδα μπορεί να διαφέρουν σε μεγάλο βαθμό όσον αφορά το επίπεδο δυσκολίας τους. Η διαφορά αυτή μπορεί να φανεί στις δυο παρακάτω εικόνες όπου στη μία το επίπεδο δυσκολίας είναι χαμηλό και στην άλλη είναι υψηλό. Παρ'όλα αυτά, το παιχνίδι δεν δημιουργεί απότομες αλλαγές στην δυσκολία (difficulty spikes) μεταξύ διαδοχικών επιπέδων.



Εύκολο επίπεδο με ελάχιστες προκλήσεις



Δύσκολο επίπεδο με πολλές προκλήσεις

4. ΑΝΑΛΥΣΗ ΠΑΙΧΝΙΔΙΟΥ

4.1. Περιγραφή

Το παιχνίδι που αναπτύχθηκε είναι μία απόπειρα συνδυασμού των παιχνιδιών που παρουσιάστηκαν παραπάνω, χρησιμοποιώντας την παραγωγή επιπέδων γραμμικά για ευκολότερα επίπεδα δυσκολίας, αλλά και βασισμένη σε δωμάτια για δυσκολότερα επίπεδα δυσκολίας. Παρακάτω παρουσιάζονται οι οντότητες του παιχνιδιού και περιγράφονται με λεπτομέρειες τα χαρακτηριστικά και οι συμπεριφορές τους. Επίσης γίνεται ανάλυση της δομής των συστημάτων PCG και DDA που χρησιμοποιούνται, περιγράφοντας λεπτομερώς την διαδικασία εκτέλεσης του, τη σειρά εκτέλεσης τους καθώς και εναλλακτικές μέθοδοι που θα μπορούσαν να χρησιμοποιηθούν.

Ο παίχτης ελέγχει έναν χαρακτήρα ο οποίος ξεκινώντας από ένα αρχικό σημείο (starting point) πρέπει να φτάσει σε κάποιον στόχο(goal) ώστε να ολοκληρώσει το επίπεδο(level), χρησιμοποιώντας διάφορους μηχανισμούς, όπως για παράδειγμα η ικανότητα άλματος(jump). Μόλις το καταφέρει αυτό δημιουργείται το επόμενο επίπεδο που διαμορφώνεται όσον αφορά στο επίπεδο δυσκολίας, με βάση ενός χρονομέτρου.

Επίσης, επειδή η γραπτή περιγραφή δεν δίνει πλήρη εικόνα των διάφορων μηχανισμών, έχει αναπτυχθεί ένα Tutorial μέσα στο παιχνίδι, ώστε να προεκπαιδεύσει και να δώσει μία πρώτη εικόνα στον παίχτη για το παιχνίδι. Ο παίχτης δεν είναι αναγκασμένος να το ολοκληρώσει, αλλά εξηγούνται διάφοροι μηχανισμοί οι οποίοι μπορεί να του φανούν χρήσιμοι.

4.2. Οντότητες

Σε αυτή την ενότητα περιγράφονται οι οντότητες που σχεδιάστηκαν και τοποθετούνται μέσα στο επίπεδο και παρουσιάζονται τα κομμάτια(Components) από τα οποία αποτελούνται τα αντικείμενά τους

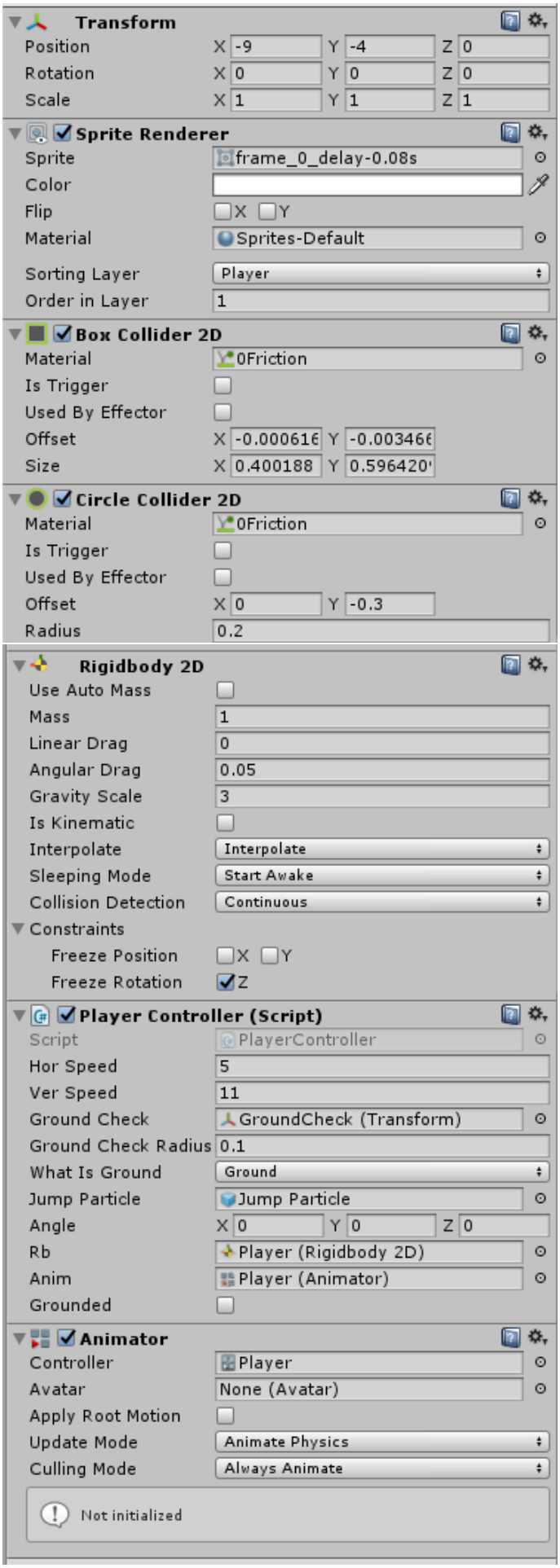
1. 

   Player - Ο χαρακτήρας που ελέγχει ο παίχτης.

   -Ικανότητες :

- Άλμα(Jump)
- Διπλό άλμα(Double-jump)
- Τρέξιμο(Run)
- Αλλαγή βαρύτητας(Gravity-flip)

Components :

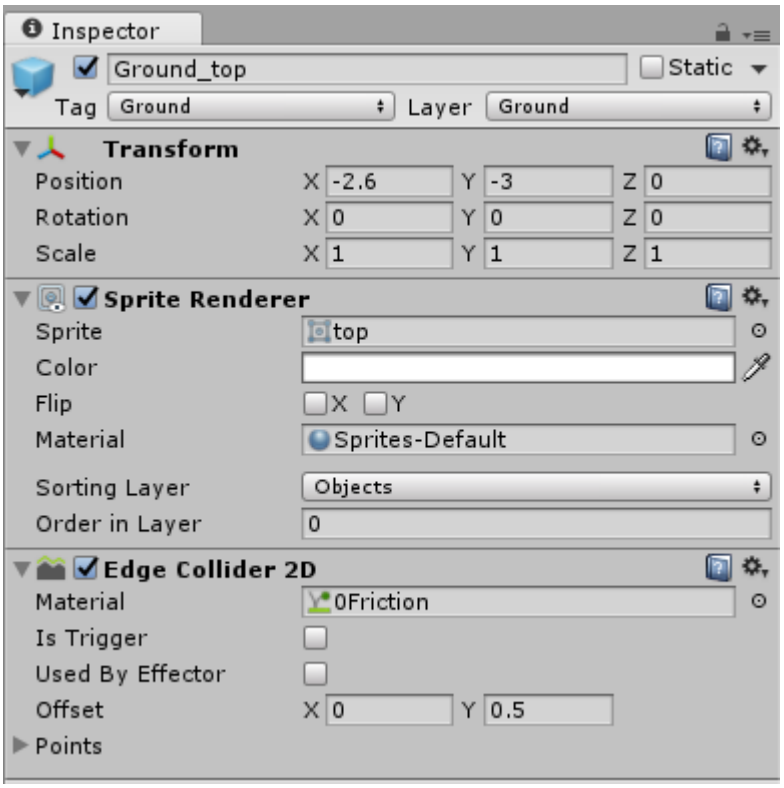

Transform
Position X -9 Y -4 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sprite Renderer
Sprite frame_0_delay-0.08s
Color
Flip X Y
Material Sprites-Default
Sorting Layer Player
Order in Layer 1
Box Collider 2D
Material 0Friction
Is Trigger
Used By Effector
Offset X -0.000616 Y -0.003466
Size X 0.400188 Y 0.596420
Circle Collider 2D
Material 0Friction
Is Trigger
Used By Effector
Offset X 0 Y -0.3
Radius 0.2
Rigidbody 2D
Use Auto Mass
Mass 1
Linear Drag 0
Angular Drag 0.05
Gravity Scale 3
Is Kinematic
Interpolate Interpolate
Sleeping Mode Start Awake
Collision Detection Continuous
Constraints
Freeze Position X Y
Freeze Rotation Z
Player Controller (Script)
Script PlayerController
Hor Speed 5
Ver Speed 11
Ground Check GroundCheck (Transform)
Ground Check Radius 0.1
What Is Ground Ground
Jump Particle Jump Particle
Angle X 0 Y 0 Z 0
Rb Player (Rigidbody 2D)
Anim Player (Animator)
Grounded
Animator
Controller Player
Avatar None (Avatar)
Apply Root Motion
Update Mode Animate Physics
Culling Mode Always Animate
Not initialized

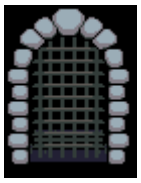
-Transform : Υπεύθυνο για την θέση, την περιστροφή και το μέγεθος του αντικειμένου
-Sprite Renderer : Υπεύθυνο για την αναπαραγωγή των γραφικών και την εμφάνιση των sprites στην οθόνη
-Box & Circle Collider 2D : Χρησιμοποιούνται για τον έλεγχο συγκρούσεων του παίχτη με άλλα αντικείμενα όπως το έδαφος και οι αντίπαλοι
-Rigidbody 2D : Υπεύθυνο για την εφαρμογή λειτουργιών φυσικής, όπως η βαρύτητα
-Player Controller : Το πρόγραμμα που είναι υπεύθυνο για τη κίνηση και τις ικανότητες του χαρακτήρα που ελέγχει ο παίχτης
-Animator : Υπεύθυνο για τη κίνηση των γραφικών του χαρακτήρα

2. 

Ground - Έδαφος το οποίο χρησιμοποιείται ως βάση για την τοποθέτηση των υπόλοιπων οντοτήτων

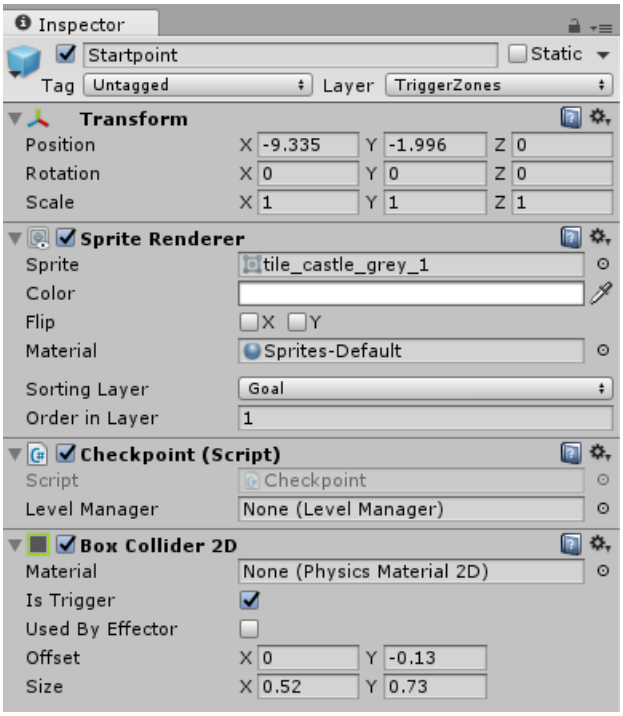
Components :



3. 

Starting Point - Κλειστή πόρτα η οποία συμβολίζει το σημείο στο οποίο ο παίχτης ξεκινάει. Ο χάρτης περιέχει πάντα ένα Starting Point.

Components :

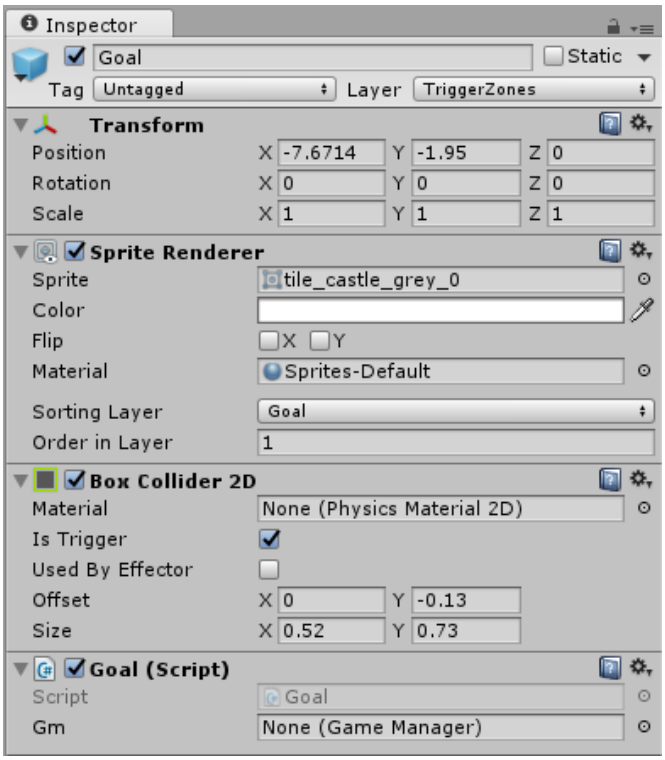


Checkpoint : Το πρόγραμμα που ελέγχει τον μηχανισμό του Checkpoint

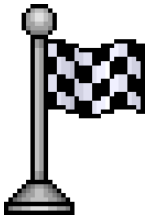
4. 

Goal - Ανοιχτή πόρτα που συμβολίζει το σημείο στο οποίο ο παίχτης πρέπει να φτάσει για να προχωρήσει στο επόμενο επίπεδο. Ο χάρτης περιέχει πάντα ένα Goal.

Components :

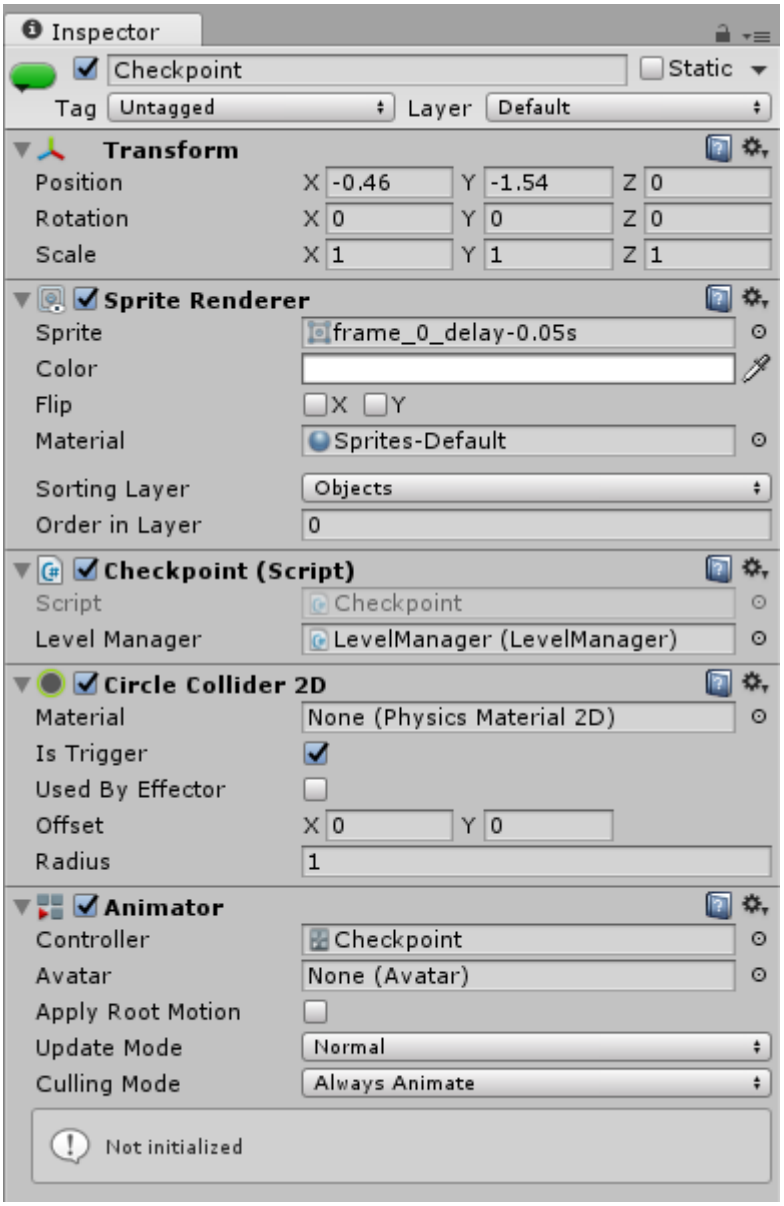


Goal : Το πρόγραμμα το οποίο ελέγχει την αύξηση του επιπέδου και του χρονομέτρου όταν ο παίχτης έρθει σε επαφή με το αντικείμενο

5. 

Checkpoint - Σημείο το οποίο δημιουργείται σε ενδιάμεσα σημεία του χάρτη και χρησιμοποιείται ώστε να διευκολύνει την πρόοδο του παίχτη. Αν ο παίχτης ακουμπήσει ένα Checkpoint και αργότερα πεθάνει, τότε θα αναστηθεί στο σημείο αυτό. Ο χάρτης περιέχει πάντα ένα ή περισσότερα Checkpoints.

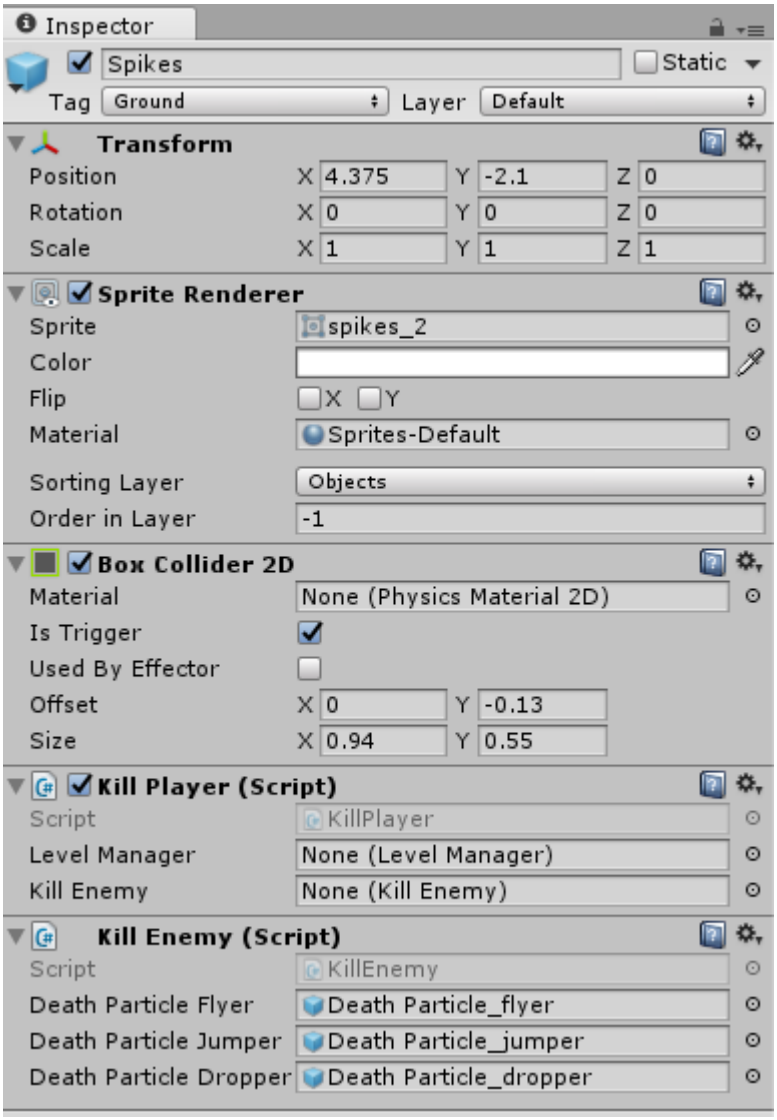
Components :



6. 

Spikes - Στατικό εμπόδιο το οποίο τοποθετείται ακριβώς πάνω από το έδαφος. Έχει την ικανότητα να σκοτώσει οποιονδήποτε χαρακτήρα έρθει σε επαφή μαζί του,όμως δεν μπορεί να μετακινηθεί.

Components :

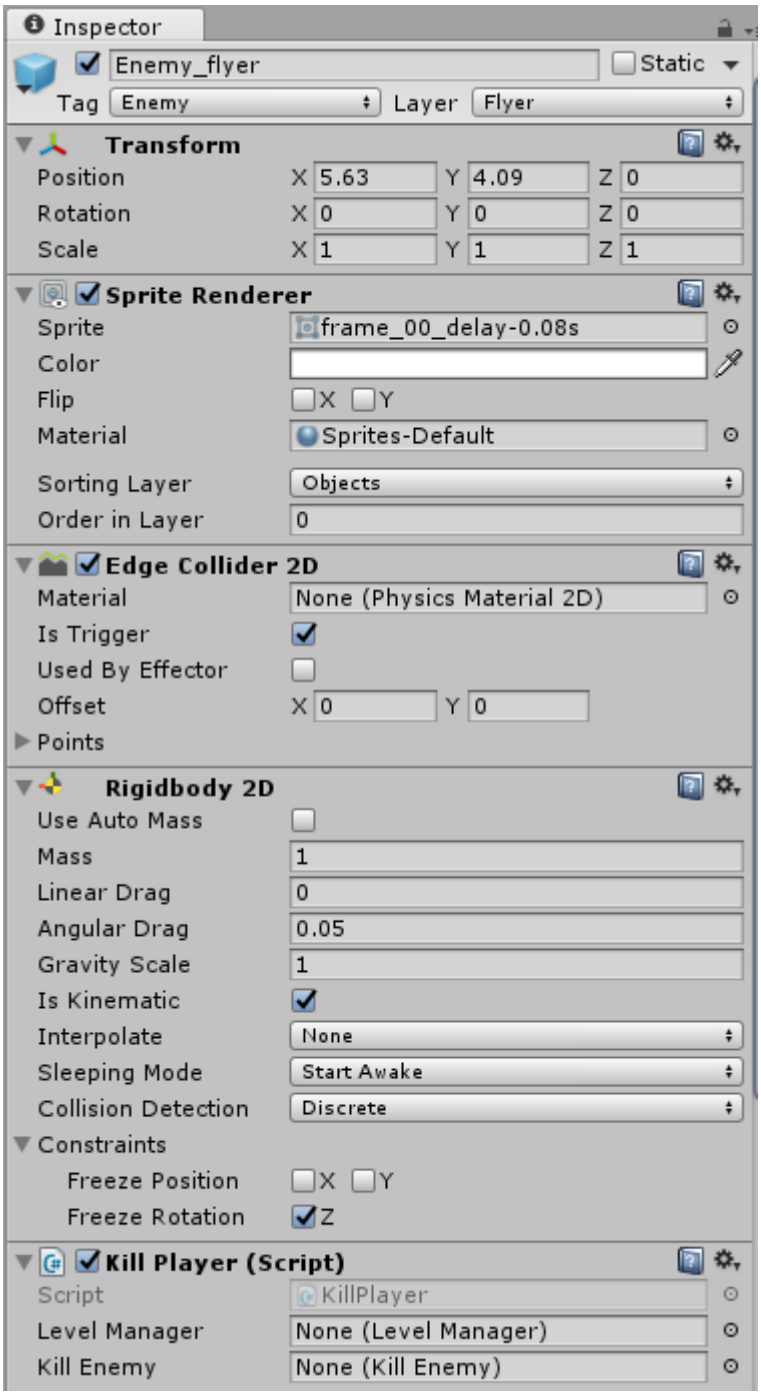


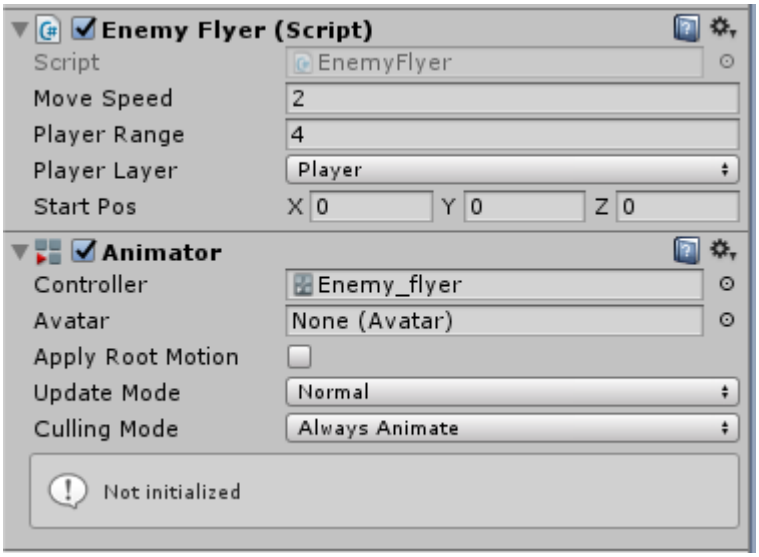
Kill Player : Με το πρόγραμμα αυτό το αντικείμενο μπορεί να σκοτώσει τον παίχτη
Kill Enemy : Με το πρόγραμμα αυτό το αντικείμενο μπορεί να σκοτώσει τους υπόλοιπους αντιπάλους

7.

Flyer - Αντίπαλος ο οποίος αιωρείται και ακολουθεί τον παίχτη μόλις αυτός εισέλθει στην εμβέλεια του.

Components :



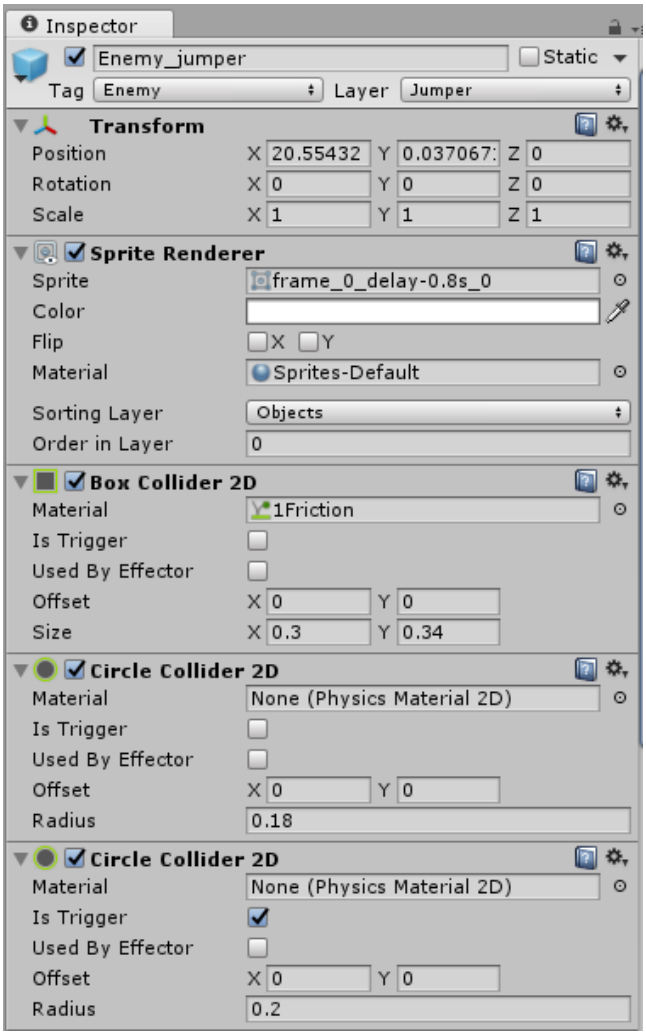


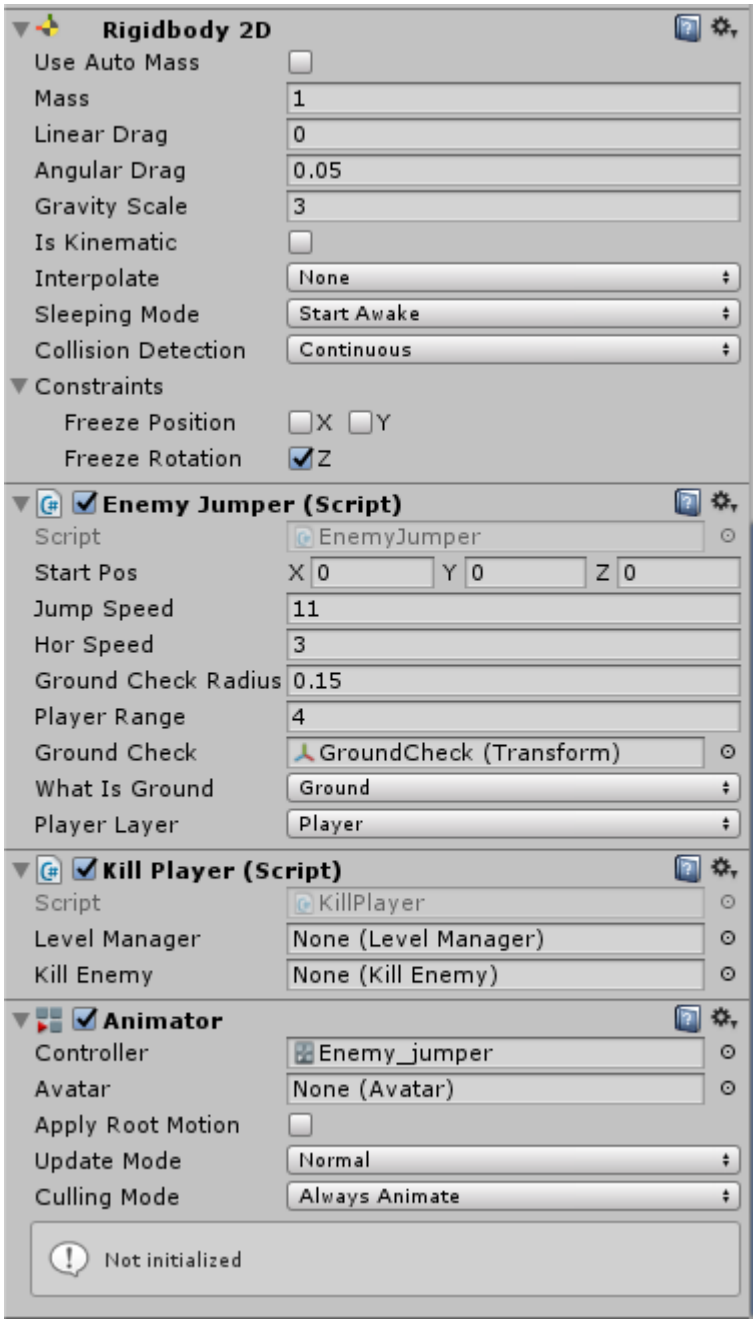
Enemy Flyer : Το πρόγραμμα που είναι υπεύθυνο για τη κίνηση του Flyer αντιπάλου και την ικανότητα του να κυνηγάει τον παίχτη

8. 

Jumper - Αντίπαλος ο οποίος έχει την ικανότητα άλματος ταυτόχρονα με τον παίχτη όταν αυτός εισέλθει στην εμβέλεια του.

Components :

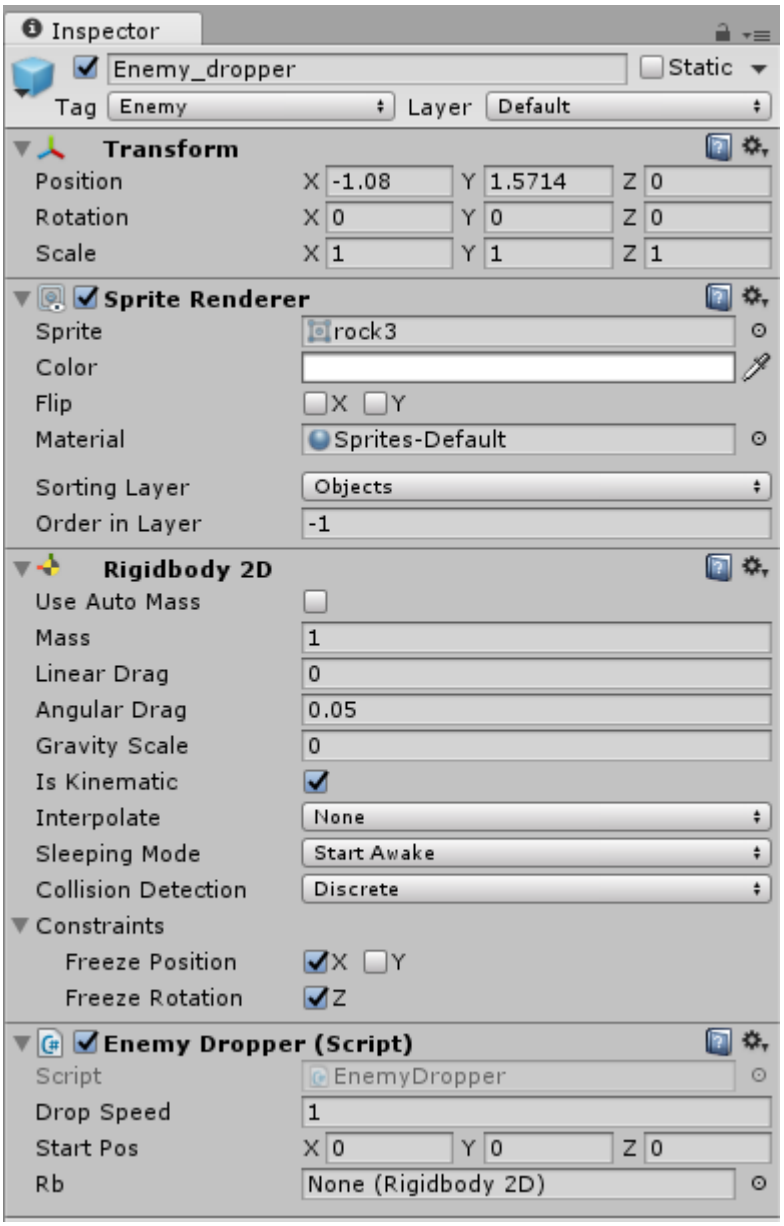



Rigidbody 2D
Use Auto Mass
Mass 1
Linear Drag 0
Angular Drag 0.05
Gravity Scale 3
Is Kinematic
Interpolate None
Sleeping Mode Start Awake
Collision Detection Continuous
Constraints
Freeze Position X Y
Freeze Rotation Z
Enemy Jumper (Script)
Script EnemyJumper
Start Pos X 0 Y 0 Z 0
Jump Speed 11
Hor Speed 3
Ground Check Radius 0.15
Player Range 4
Ground Check GroundCheck (Transform)
What Is Ground Ground
Player Layer Player
Kill Player (Script)
Script KillPlayer
Level Manager None (Level Manager)
Kill Enemy None (Kill Enemy)
Animator
Controller Enemy_jumper
Avatar None (Avatar)
Apply Root Motion
Update Mode Normal
Culling Mode Always Animate
Not initialized

9. 

Dropper - Εμπόδιο το οποίο πέφτει προς το έδαφος όταν ο παίχτης περάσει από κάτω του.

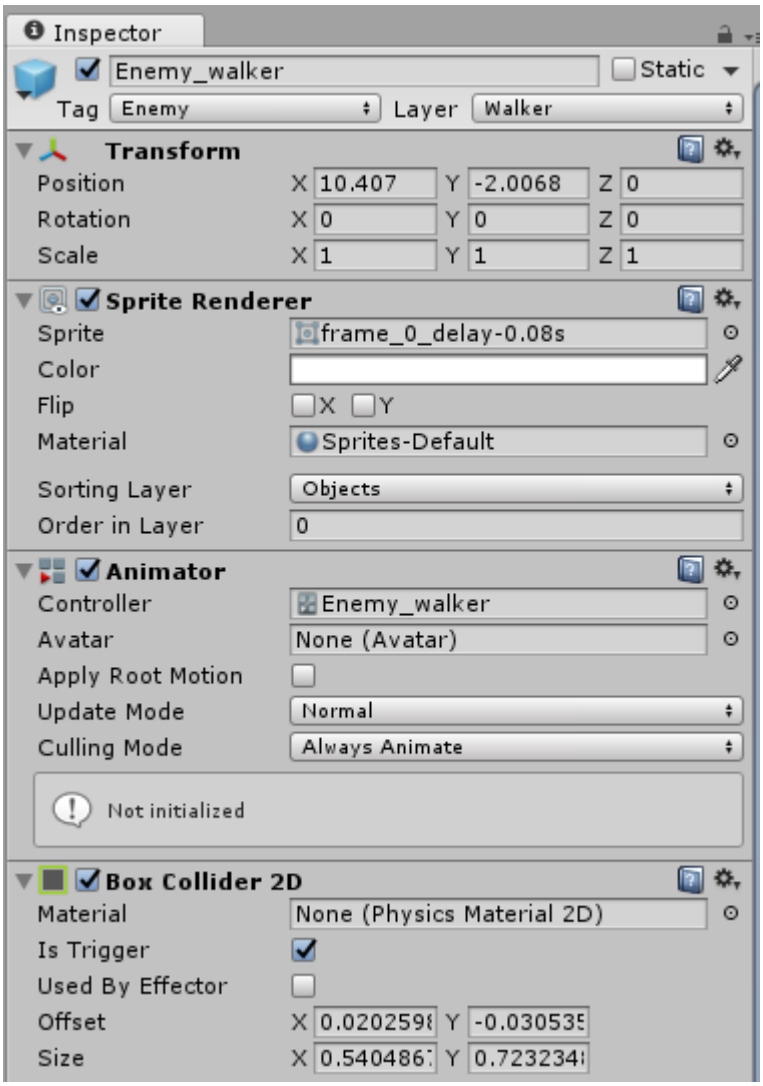
Components :

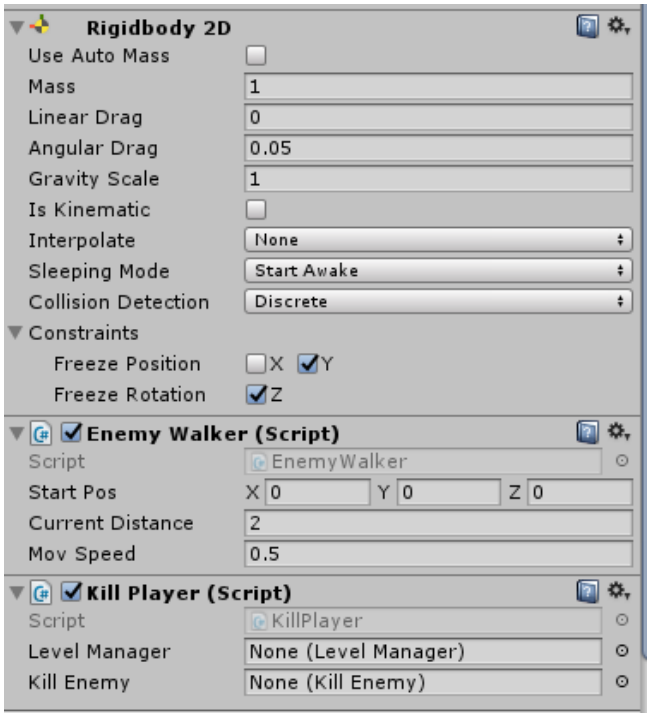


10. 

Walker - Αντίπαλος ο οποίος περπατάει δύο κελιά δεξιά και επιστρέφει στην αρχική του θέση.

Components :

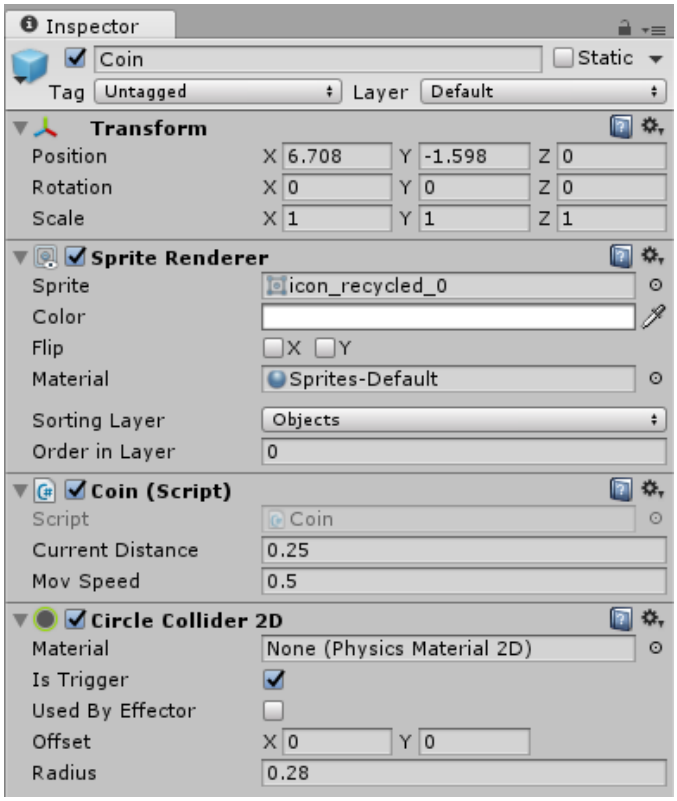




11. 

Coin - Κέρμα το οποίο μόλις το μαζέψει ο παίχτης προσθέτονται μερικά δευτερόλεπτα στον χρονομετρητή. Ο αριθμός τον δευτερολέπτων διαφέρει ανάλογα την αρχική δυσκολία που επιλέγει ο παίχτης

Components :



4.3. Κανόνες

Στην ενότητα αυτή παρουσιάζονται μερικοί κανόνες οι οποίοι υλοποιήθηκαν με σκοπό την εξισορρόπηση του παιχνιδιού.

Αρχικά, τα Spikes σκοτώνουν οποιαδήποτε οντότητα έρθει σε επαφή μαζί τους. Οι Droppers μπορούν να σκοτώσουν οποιαδήποτε οντότητα έρθει σε επαφή μαζί τους,εκτός από τα Spikes, και μόλις συμβεί αυτό, καταστρέφονται. Οι Flyers, Jumpers και Walkers μπορούν να σκοτώσουν μόνο τον Player και δεν μπορούν να σκοτώσουν ο ένας τον άλλον, δηλαδή όταν έρθουν σε επαφή δεν συμβαίνει τίποτα. Ο Player δεν έχει τη δυνατότητα να σκοτώσει κανέναν αντίπαλό του όπως μπορεί σε πολλά άλλα παιχνίδια. Μπορεί, ωστόσο, να χρησιμοποιήσει το περιβάλλον του, μέσω των διάφορων εμποδίων που έχουν αυτή τη δυνατότητα. Αν ένας αντίπαλος πεθάνει δεν επανεμφανίζεται στο αρχικό του σημείο. Όμως, επανεμφανίζονται όλοι οι ζωντανοί αντίπαλοι στην αρχική τους θέση αν ο παίχτης πεθάνει. Ένας αντίπαλος μπορεί να εξαφανιστεί αν ο παίχτης πεθάνει τρεις φορές από αυτόν και ο λόγος που υλοποιήθηκε αυτός ο μηχανισμός είναι να αποτρέψει σημεία τα οποία είναι πολύ δύσκολα να αντιμετωπίσει ο παίχτης.

Όσον αφορά τις ικανότητες του παίχτη, υπάρχουν κάποιοι περιορισμοί οι οποίοι υλοποιήθηκαν ώστε να αντικρούσουν διάφορες στρατηγικές που θα μπορούσε ο παίχτης να καταχραστεί. Καταρχάς, στην οντότητα του Player υπάρχει ένα αντικείμενο ονομαζόμενο “GroundCheck” το οποίο χρησιμοποιείται ως έλεγχος για τον περιορισμό των αλμάτων. Δηλαδή, ο παίχτης μπορεί να κάνει άλμα μόνο αν βρίσκεται πάνω στο έδαφος. Στη συνέχεια, μπορεί να κάνει δεύτερο άλμα μόνο αν έχει κάνει το πρώτο και βρίσκεται στον αέρα. Το GroundCheck υλοποιήθηκε ώστε ο παίχτης να μην μπορεί να κάνει άπειρα άλματα. Όσον αφορά την ικανότητα gravity-flip, αυτό που κάνει είναι να αντιστρέφει τη βαρύτητα του παίχτη και ουσιαστικά πέφτει προς τα πάνω, αλλά δεν αντιστρέφεται η ικανότητα άλματος, που σημαίνει ότι δεν μπορεί να κάνει άλμα όσο είναι αντεστραμμένος. Μπορεί όμως να κάνει άλμα στον αέρα πέφτοντας προς τα κάτω, αν επανέλθει στην αρχική του βαρύτητα. Τέλος, η ικανότητα του τρεξίματος δεν έχει κανέναν περιορισμό, διότι δεν υπάρχει κάποια στρατηγική που άμα τη καταχραστεί ο παίχτης του δίνει μεγάλο πλεονέκτημα.

4.4. Εφαρμογή του PCG

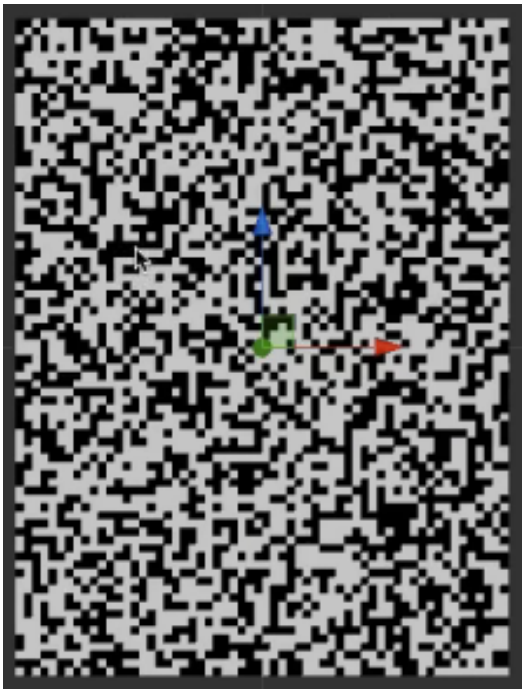
Στο σημείο αυτό παρουσιάζεται η διαδικασία δημιουργίας και παραγωγής του χάρτη καθώς και η τοποθέτηση των οντοτήτων μέσα στο χάρτη. Η παραγωγή του χάρτη αποτελεί μέρος ενός από τα επίσημα μαθήματα του Unity από τον Sebastian Lague στη σελίδα https://unity3d.com/learn/tutorials/s/procedural-cave-generation-tutorial

4.4.1. Map Generator

Στην αρχή, γίνεται το μαρκάρισμα του εδάφους και των κενών κελιών στο MapGenerator.cs script μέσω των κυτταρικών αυτομάτων. Ξεκινώντας δημιουργείται ένας δισδιάστατος πίνακας map με μεγέθη width και height. Τα χαρακτηριστικά αυτά καθορίζονται ανάλογα το επίπεδο δυσκολίας, δηλαδή στα χαμηλά επίπεδα δυσκολίας είναι πιο μικρά και στα υψηλά επίπεδα είναι μεγαλύτερα. Έπειτα, καλείται η μέθοδος RandomFillMap() η οποία διασχίζει όλο το χάρτη και δίνει τυχαία την τιμή 1 για κάθε κελί που είναι έδαφος και 0 για κάθε κελί που είναι κενό.

```
void RandomFillMap(){
	if (useRandomSeed) {
		seed = System.DateTime.Now.Ticks.ToString ();
	}
	System.Random pseudoRandom = new System.Random (seed.GetHashCode ());
	for (int x = 0; x < width; x++) {
		for (int y = 0; y < height; y++) {
			if(x==0||x==width-1||y==0||y==height-1){
				map[x,y]=1;
			}
			else{
				map[x,y]=(pseudoRandom.Next(0,100)<randomFillPercent)?1:0;
			}
		}
	}
}
```

Η πιθανότητα να γίνει 1 ή 0 καθορίζεται από τη μεταβλητή randomFillPercent η οποία επιλέγεται πάλι ανάλογα το επίπεδο δυσκολίας τυχαία. Το αποτέλεσμα της μεθόδου αυτής δίνει αποτελέσματα όπως φαίνεται παρακάτω

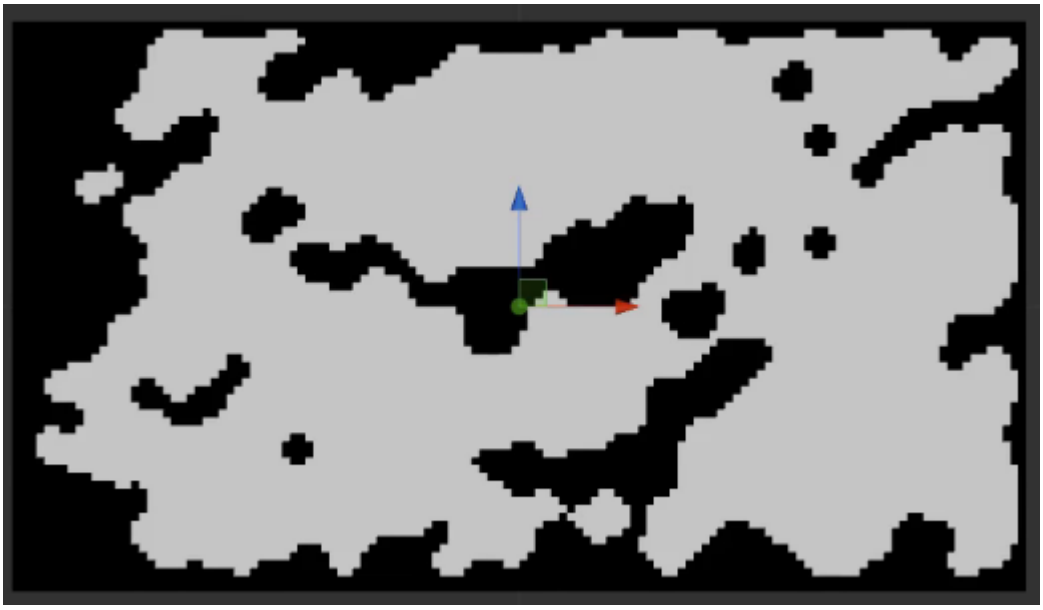


Όπως βλέπουμε τα μαύρα κελιά συμβολίζουν το έδαφος και τα άσπρα είναι κενά. Επειδή, όμως, η μορφή που παίρνει ο χάρτης δεν είναι ιδιαίτερα χρήσιμη, χρειάζεται να την απαλύνουμε. Για να το καταφέρουμε αυτό χρησιμοποιούμε τη μέθοδο SmoothMap() η οποία εκτελείται 5 φορές.

```
void SmoothMap(){
        for (int x = 0; x < width; x++) {
                for (int y = 0; y < height; y++) {
                        int neighbourWallTiles = GetSurroundingWallCount (x, y);
                        if (neighbourWallTiles > 4)
                                map [x, y] = 1;
                        else if (neighbourWallTiles < 4)
                                map [x, y] = 0;
                }
        }
}
int GetSurroundingWallCount(int gridX,int gridY){
        int wallCount = 0;
        for (int neighbourX = gridX - 1; neighbourX <= gridX + 1; neighbourX++) {
                for (int neighbourY = gridY - 1; neighbourY <= gridY + 1; neighbourY++) {
                        if (IsInMapRange(neighbourX,neighbourY)) {
                                if (neighbourX != gridX || neighbourY != gridY) {
                                        wallCount += map[neighbourX,neighbourY];
                                }
                        }
                        else {
                                wallCount ++;
                        }
                }
        }
        return wallCount;
}
```

Η μέθοδος αυτή ελέγχει για κάθε κελί τον αριθμό των γειτόνων του μέσω της μεθόδου GetSurroundingWallCount(), η οποία επιστρέφει τον αριθμό γειτόνων με βάση τη γειτονιά Moore(δηλαδή ελέγχει τα 8 κελιά γύρω από το τρέχων). Αν ο αριθμός των γειτόνων με τιμή 1 είναι μεγαλύτερος των 4, τότε το κελί παίρνει τη τιμή 1, αλλιώς παίρνει τη τιμή 0.

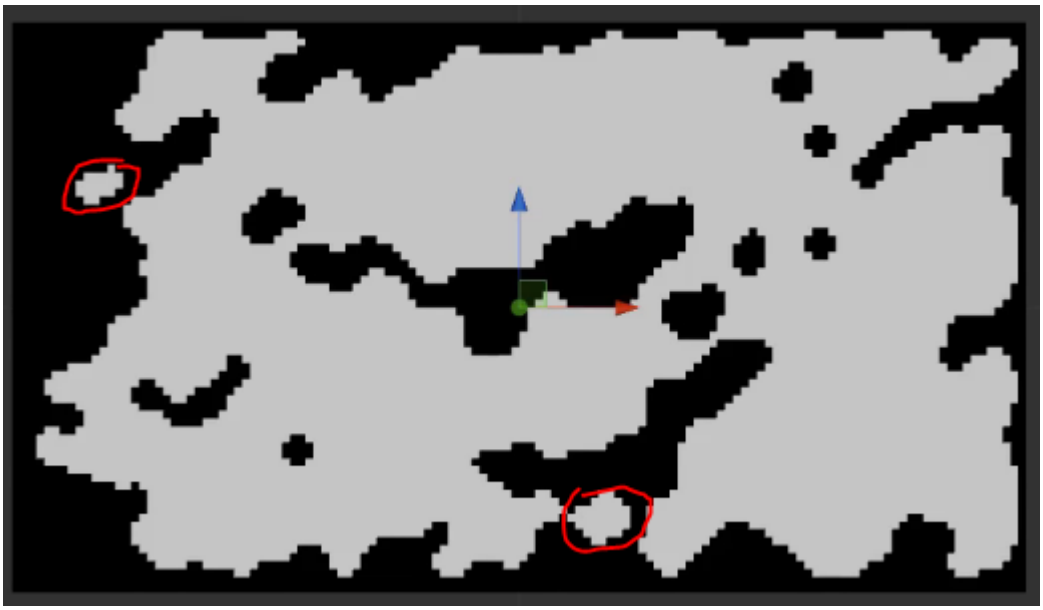
Το αποτέλεσμα της μεθόδου αυτής φαίνεται παρακάτω



Παρατηρούμε ότι η μορφή του χάρτη έχει αρχίσει να μοιάζει περισσότερο σε μια σπηλιά και θα μπορούσαμε να τη χρησιμοποιήσουμε ως χάρτη για το παιχνίδι.

Ωστόσο, παρουσιάζονται δύο νέα προβλήματα τα οποία θα πρέπει να αντιμετωπιστούν. Αρχικά, φαίνονται μερικές ομάδες κελιών εδάφους οι οποίες δεν είναι συνδεδεμένες με τα υπόλοιπα κελιά εδάφους.

Το δεύτερο πρόβλημα που θα πρέπει να λύσουμε είναι ορισμένες ομάδες κενών κελιών οι οποίες δεν είναι συνδεδεμένες όπως φαίνεται παρακάτω.



Επομένως, χρειαζόμαστε μία μέθοδο ProcessMap() η οποία διαχειρίζεται τέτοιου είδους καταστάσεις. Κατ'αρχάς θα πρέπει να εντοπιστούν αυτές οι ομάδες και αυτό θα το καταφέρουμε με τη χρήση των μεθόδων GetRegions() και GetRegionTiles().

```
public List<Coord> GetRegionTiles(int startX, int startY) {
        List<Coord> tiles = new List<Coord> ();
        int[,] mapFlags = new int[width,height];
        int tileType = map [startX, startY];
        Queue<Coord> queue = new Queue<Coord> ();
        queue.Enqueue (new Coord (startX, startY));
        mapFlags [startX, startY] = 1;
        while (queue.Count > 0) {
                Coord tile = queue.Dequeue();
                tiles.Add(tile);
                for (int x = tile.tileX - 1; x <= tile.tileX + 1; x++) {
                        for (int y = tile.tileY - 1; y <= tile.tileY + 1; y++) {
                                if (IsInMapRange(x,y) && (y == tile.tileY || x == tile.tileX)) {
                                        if (mapFlags[x,y] == 0 && map[x,y] == tileType) {
                                                mapFlags[x,y] = 1;
                                                queue.Enqueue(new Coord(x,y));
                                        }
                                }
                        }
                }
        }
        return tiles;
}
```

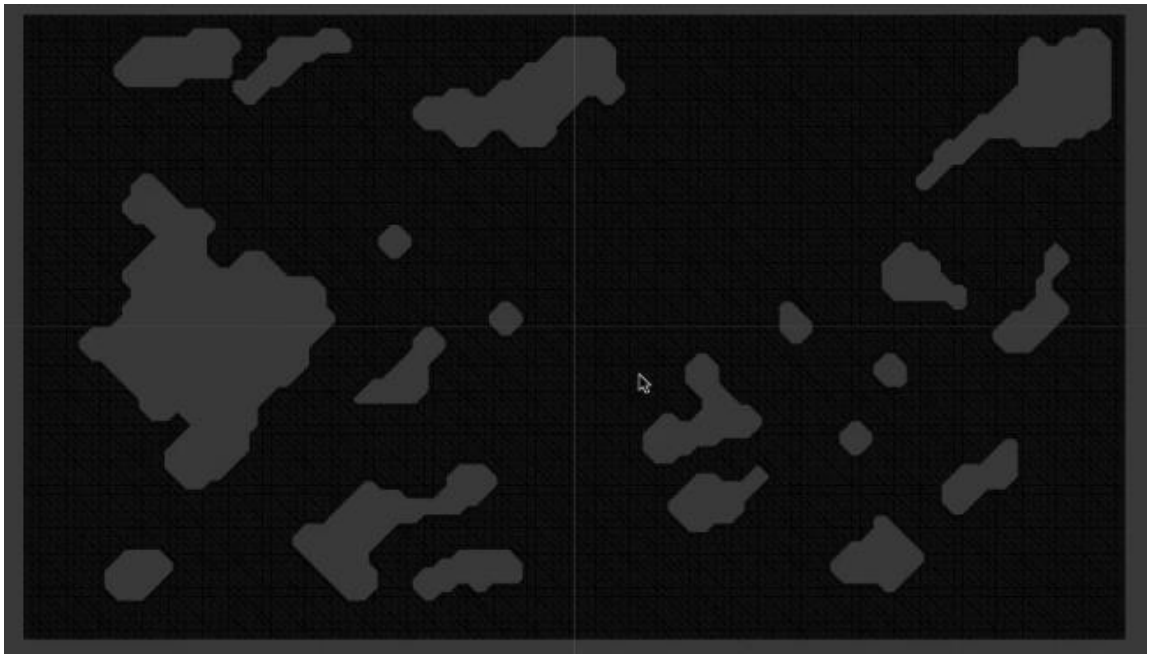
Η GetRegionTiles() ξεκινώντας από ένα κελί το οποίο προστίθεται στη λίστα tiles, τοποθετεί σε μία ουρά τους 4 γείτονες του(Von Neumann γειτονιά). Όσο η ουρά είναι γεμάτη προσθέτονται οι γείτονες που έχουν την τιμή tileType στη λίστα tiles, αφαιρούνται από την ουρά και προσθέτονται οι δικοί τους γείτονες. Ο πίνακας mapFlags χρησιμεύει για να γνωρίζουμε αν έχει ελεγχθεί το τρέχων κελί. Με τη μέθοδο αυτή επιστρέφεται η λίστα των κελιών της περιοχής ενός δεδομένου τύπου κελιού(εδάφους ή κενού).

```
List<List<Coord>> GetRegions(int tileType) {
	List<List<Coord>> regions = new List<List<Coord>> ();
	int[,] mapFlags = new int[width,height];
	for (int x = 0; x < width; x ++) {
		for (int y = 0; y < height; y ++) {
			if (mapFlags[x,y] == 0 && map[x,y] == tileType) {
				List<Coord> newRegion = GetRegionTiles(x,y);
				regions.Add(newRegion);
				foreach (Coord tile in newRegion) {
					mapFlags[tile.tileX, tile.tileY] = 1;
				}
			}
		}
	}
	return regions;
}
```

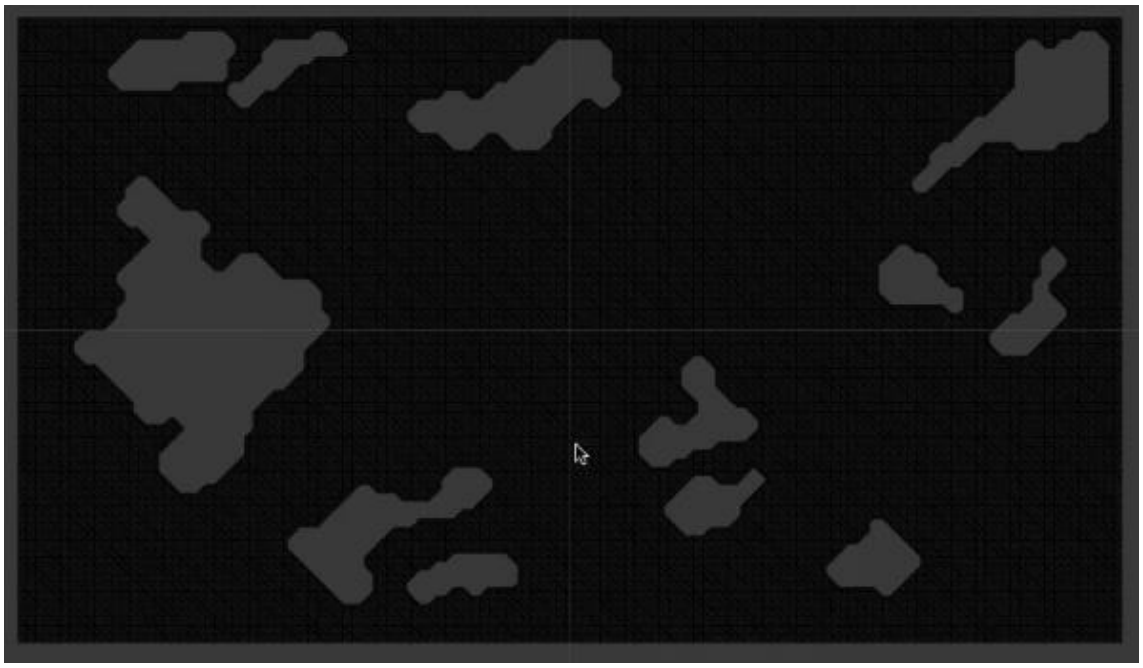
Η GetRegions() μέθοδος διατρέχοντας όλα τα κελιά του χάρτη επιστρέφει μία λίστα περιοχών δεδομένου ενός τύπου κελιού. Αν το κελί δεν έχει ελεγχθεί,δηλαδή η τιμή του στον πίνακα mapFlags είναι 0, και η τιμή του κελιού στον πίνακα map είναι τύπου tileType, τότε δημιουργείται νέα περιοχή και προστίθεται στη λίστα regions με τη μέθοδο GetRegionTiles(). Το κάθε κελί της περιοχής αυτής μαρκάρεται στον πίνακα mapFlags, ώστε να μην ξαναδημιουργήσουμε νέα περιοχή.

```
public List<Room> ProcessMap(){
	List<Room> survivingRooms = new List<Room> ();
	survivingPassageCoords = new List<Coord> ();
	List<List<Coord>> wallRegions = GetRegions (1);
	wallThresholdSize = 12;
	foreach (List<Coord> wallRegion in wallRegions) {
		if (wallRegion.Count < wallThresholdSize) {
			foreach (Coord tile in wallRegion) {
				map[tile.tileX,tile.tileY] = 0;
			}
		}
	}
	roomThresholdSize = 20;
	List<List<Coord>> roomRegions = GetRegions (0);
	foreach (List<Coord> roomRegion in roomRegions) {
		if (roomRegion.Count < roomThresholdSize) {
			foreach (Coord tile in roomRegion) {
				map [tile.tileX, tile.tileY] = 1;
			}
		} else {
			survivingRooms.Add (new Room (roomRegion, map));
		}
	}
	survivingRooms.Sort ();
	survivingRooms [0].isMainroom = true;
	survivingRooms [0].isAccessibleFromMainRoom = true;
	ConnectClosestRooms (survivingRooms);
	return survivingRooms;
}
```

Χρησιμοποιώντας, λοιπόν, τη μέθοδο GetRegions(), μπορούμε να υπολογίσουμε στην μέθοδο ProcessMap() τις περιοχές τοίχων(wallRegions) και τις περιοχές δωματίων(roomRegions) και να τις περιορίσουμε. Έτσι, για κάθε περιοχή τοίχων, αν ο αριθμός των κελιών είναι μικρότερος από το κατώφλι τοίχων(wallThresholdSize), η περιοχή αυτή γίνεται κενή. Όμοια, για κάθε περιοχή δωματίου, αν ο αριθμός των κελιών είναι μικρότερος από το κατώφλι δωματίου(roomThresholdSize), τότε το δωμάτιο γίνεται περιοχή τοίχων. Επίσης, ταξινομούμε τα δωμάτια με βάση το μέγεθος τους για να ορίσουμε το μεγαλύτερο δωμάτιο ως κύριο(MainRoom), και αποθηκεύουμε τα δωμάτια στη λίστα survivingRooms για τη τοποθέτηση οντοτήτων αργότερα.



Παρακάτω μπορούμε να παρατηρήσουμε ότι μικρές περιοχές δωματίων εξαφανίζονται.



Καταλήγουμε, λοιπόν, στο σημείο όπου ο χάρτης αποτελείται από έναν αριθμό δωματίων με τα κατάλληλα μεγέθη, τα οποία όμως δεν είναι συνδεδεμένα μεταξύ τους. Γι'αυτό χρειαζόμαστε τη μέθοδο ConnectClosestRooms() η οποία συνδέει τα κοντινότερα δωμάτια μεταξύ τους.

Δημιουργούμε λοιπόν μία κλάση Room η οποία αποτελείται από τα εσωτερικά και εξωτερικά κελιά του(tiles και edgeTiles αντίστοιχα) και υπολογίζονται στον constructor της. Η κλάση αυτή μας βοηθάει ώστε να μπορέσουμε να υπολογίσουμε τις κοντινότερες αποστάσεις μεταξύ των δωματίων αργότερα, στην ConnectClosestRooms().

```
public class Room : IComparable<Room>{
	public List<Coord> tiles;
	public List<Coord> edgeTiles;
	public List<Room> connectedRooms;
	public int roomSize;
	public bool isAccessibleFromMainRoom;
	public bool isMainroom;

	public Room(){

	}

	public Room(List<Coord> roomTiles,int[,] map){
		tiles=roomTiles;
		roomSize=tiles.Count;
		connectedRooms=new List<Room>();

		edgeTiles=new List<Coord>();
		foreach(Coord tile in tiles){
			for(int x=tile.tileX-1;x<=tile.tileX+1;x++){
				for(int y=tile.tileY-1;y<=tile.tileY+1;y++){
					if(x==tile.tileX||y==tile.tileY){
						if(map[x,y]==1){
							edgeTiles.Add(tile);
						}
					}
				}
			}
		}
	}
	public void SetAccessibleFromMainRoom(){
		if (!isAccessibleFromMainRoom) {
			isAccessibleFromMainRoom = true;
			foreach (Room connectedRoom in connectedRooms) {
				connectedRoom.SetAccessibleFromMainRoom ();
			}
		}
	}
```

```
	public static void ConnectRooms(Room roomA,Room roomB){
		if (roomA.isAccessibleFromMainRoom) {
			roomB.SetAccessibleFromMainRoom ();
		} else if (roomB.isAccessibleFromMainRoom) {
			roomA.SetAccessibleFromMainRoom ();
		}

		roomA.connectedRooms.Add (roomB);
		roomB.connectedRooms.Add (roomA);
	}
	public bool IsConnected(Room otherRoom){
		return connectedRooms.Contains (otherRoom);
	}
	public int CompareTo(Room otherRoom){
		return otherRoom.roomSize.CompareTo (roomSize);
	}
	public List<Coord> ReturnTiles(){
		return tiles;
	}

}
```

Αρχικά η ConnectClosestRooms() ελέγχει εάν τα δωμάτια είναι συνδεδεμένα με το κύριο δωμάτιο που ορίσαμε νωρίτερα στην ProcessMap(). Όλα τα δωμάτια που έχουν πρόσβαση στο κύριο δωμάτιο αποθηκεύονται στη λίστα roomListB και όλα τα δωμάτια που δεν έχουν πρόσβαση στο κύριο δωμάτιο αποθηκεύονται στη λίστα roomListA. Συνεχίζοντας, για κάθε δωμάτιο στη λίστα roomListA διατρέχουμε τα δωμάτια της λίστας roomListB αναζητώντας το κοντινότερο δωμάτιο με βάση την απόσταση. Μόλις βρεθεί το κοντινότερο δωμάτιο για όλα τα δωμάτια της λίστας roomListA, δημιουργείται το πέρασμα μεταξύ των δωματίων.

```
void ConnectClosestRooms(List<Room> allRooms,bool forceAccessibilityFromMainRoom=false){
	List<Room> roomListA = new List<Room> ();
	List<Room> roomListB = new List<Room> ();
	if (forceAccessibilityFromMainRoom) {
		foreach (Room room in allRooms) {
			if (room.isAccessibleFromMainRoom) {
				roomListB.Add (room);
			} else {
				roomListA.Add (room);
			}
		}
	} else {
		roomListA = allRooms;
		roomListB = allRooms;
	}
	int bestDistance = 0;
	Coord bestTileA = new Coord ();
	Coord bestTileB = new Coord ();
	Room bestRoomA = new Room ();
	Room bestRoomB = new Room ();
	bool possibleConnectionFound = false;
	foreach (Room roomA in roomListA) {
		if (!forceAccessibilityFromMainRoom) {
			possibleConnectionFound = false;
			if (roomA.connectedRooms.Count > 0) {
				continue;
			}
		}
		foreach (Room roomB in roomListB) {
```

```
                        if (roomA == roomB||roomA.IsConnected(roomB)) {
                                continue;
                        }
                        for (int tileIndexA = 0; tileIndexA < roomA.edgeTiles.Count; tileIndexA++) {
                                for (int tileIndexB = 0; tileIndexB < roomB.edgeTiles.Count; tileIndexB++) {
                                        Coord tileA = roomA.edgeTiles [tileIndexA];
                                        Coord tileB = roomB.edgeTiles [tileIndexB];
                                        int distanceBetweenRooms =(int)( Mathf.Pow (tileA.tileX –
                                                tileB.tileX, 2) + Mathf.Pow (tileA.tileY - tileB.tileY, 2));
                                        if (distanceBetweenRooms < bestDistance || !possibleConnectionFound) {
                                                bestDistance = distanceBetweenRooms;
                                                possibleConnectionFound = true;
                                                bestTileA = tileA;
                                                bestTileB = tileB;
                                                bestRoomA = roomA;
                                                bestRoomB = roomB;
                                        }
                                }
                        }
                }
                if (possibleConnectionFound&&!forceAccessibilityFromMainRoom) {
                        CreatePassage (bestRoomA, bestRoomB, bestTileA, bestTileB);
                }
        }
        if (possibleConnectionFound&&forceAccessibilityFromMainRoom) {
                CreatePassage (bestRoomA, bestRoomB, bestTileA, bestTileB);
                ConnectClosestRooms (allRooms, true);
        }
        if (!forceAccessibilityFromMainRoom) {
                ConnectClosestRooms (allRooms, true);
        }
}
```

Τέλος, εκτελείται η μέθοδος SmoothCells() διότι μέσα στο χάρτη εμφανίζονται κελιά τα οποία δεν μπορούν να δημιουργηθούν αργότερα στο Instantiator, παρ'όλα αυτά έχουν τη τιμή 1. Αυτό δημιουργεί πρόβλημα στη τοποθέτηση των αντικειμένων, διότι όπως θα δούμε συμβαίνει με βάση τη μορφή που έχει το έδαφος. Γι'αυτό το λόγο αφαιρούνται και μετατρέπονται σε κενά κελιά.

```
void SmoothCells(){
        //Smooth cells
        for (int i = 1; i < width-1; i++) {
                for (int j = 1; j < height-1; j++) {
                        if (map [i, j] == 1) {
                                if (map [i-1, j] == 1 && map [i+1, j] != 1 && map [i, j-1] != 1 && map [i, j+1] != 1)
{
                                        map [i, j] = 0;
                                }
                                if (map [i-1, j] != 1 && map [i+1, j] == 1 && map [i, j-1] != 1 && map [i, j+1] != 1)
{
                                        map [i, j] = 0;
                                }
                                if (map [i-1, j] != 1 && map [i+1, j] != 1 && map [i, j-1] != 1 && map [i, j+1] == 1)
{
                                        map [i, j] = 0;
                                }
                        }
                }
        }
}
```

4.4.2. Level Layout Handler

Στο LevelLayoutHandler script γίνεται το μαρκάρισμα των θέσεων των διαφόρων οντοτήτων. Η τοποθέτησή τους γίνεται είτε γραμμικά(Linear) είτε βασισμένο σε δωμάτια(Roombased) ανάλογα το αρχικό επίπεδο δυσκολίας και τον αριθμό των δωματίων. Στο πίνακα φαίνεται αναλυτικά για κάθε αρχικό επίπεδο δυσκολίας η αντίστοιχη προσέγγιση.

Επίπεδο Δυσκολίας	**Δημιουργία Checkpoint**	**Δημιουργία Αντικειμένων**
Very Easy	Linear	Linear
Easy	Linear	Linear
Medium	Linear(Vertical)/Roombased*	Roombased
Hard	Roombased	Roombased
Very Hard	Roombased	Roombased

*Σε αυτό το επίπεδο η δημιουργία εξαρτάται από τον αριθμό των δωματίων. Αν έχουμε ένα μόνο δωμάτιο τότε δημιουργείται με κάθετη γραμμική προσέγγιση, αλλιώς δημιουργείται με τη Roombased μέθοδο

Στο πίνακα φαίνονται αναλυτικά οι τιμές που μπορούν να πάρουν τα κελιά του πίνακα map και οι οντότητες που αντιστοιχούν σε αυτές.

Οντότητα	**Τιμή στον πίνακα map**
Empty	0
Ground	1
Starting Point	2
Goal	3
Spikes	4
Flyers	5
Jumpers	6
Walkers	7
Droppers	8
Checkpoint	9
Coin	10

Linear

```
public void CreateCheckPoints_Linear(List<MapGenerator.Room> survivingRooms){
	for (int i=0;i<mapGen.width-1;i++) {
		for (int j = 0; j < mapGen.height-1; j++) {
			//Create start point
			if (mapGen.map [i, j] == 0 && mapGen.IsInMapRange(i,j)&&startX==0&&startY==0) {
				startX = i;
				startY = j;
				mapGen.map [i, j] = 2;
				CreateArea (i, j);
				i = mapGen.width / 2;
				break;
			}
			//Create check point
			if (mapGen.map [i, j] == 0 && mapGen.IsInMapRange(i,j)&&checkX==0&&checkY==0) {
				checkX = i;
				checkY = j;
				mapGen.map [i, j] = 9;
				CreateArea (i, j);
				break;
			}
			//Create goal
			if (mapGen.map [i, j] == 0 && mapGen.IsInMapRange(i,j)&&
			i>10*mapGen.width/11 &&goalX==0&&goalY==0) {
				goalX = i;
				goalY = j;
				mapGen.map [i, j] = 3;
				CreateArea (i, j);
				break;
			}
		}
		if (startX != 0&& checkX != 0 && goalX != 0 ) {
			break;
		}
	}
}
```

Στη γραμμική(Linear) προσέγγιση ο χάρτης διασχίζεται από τα αριστερά προς τα δεξιά και τοποθετούνται τα Starting Point, Checkpoint και Goal με αυτή τη σειρά. Το Starting Point τοποθετείται στο πρώτο κενό σημείο το οποίο βρίσκεται πάνω από έδαφος. Έπειτα ο δείκτης i πηγαίνει στη μέση του χάρτη και ψάχνει το πρώτο σημείο το οποίο βρίσκεται πάνω από έδαφος, ώστε να τοποθετήσει το Checkpoint. Τέλος, το Goal τοποθετείται σε κάποιο σημείο μετά από τα 10/11 του μήκους του χάρτη, και αυτό το κάνουμε για να εξασφαλίσουμε ότι δε θα βρίσκεται δίπλα στο Starting Point ή στο Checkpoint.

Κάθε σημείο πρέπει να έχει τιμή 0, δηλαδή να είναι κενό, ώστε να μπορέσουμε να το μαρκάρουμε με τη τιμή που αντιστοιχεί. Επίσης δημιουργείται μία περιοχή γύρω από τα σημεία αυτά ώστε να μην τοποθετούνται αντίπαλοι σε κοντινά σημεία. Η κάθετη γραμμική(Linear Vertical) προσέγγιση διαφέρει μόνο στο ότι τα Checkpoint τοποθετούνται από πάνω προς τα κάτω, αντί να τοποθετούνται από αριστερά προς τα δεξιά όπως συμβαίνει με την απλή γραμμική.

```
public void CreateObjectPositions_Linear(List<MapGenerator.Room> survivingRooms){
        //SPIKES
        int rand = UnityEngine.Random.Range (mapGen.minSpikes, mapGen.maxSpikes);
        for(int i=mapGen.minSpikes;i<mapGen.width-mapGen.maxSpikes;i+=rand){
                for (int j=1;j<mapGen.height-1;j++) {
                        if (mapGen.map [i, j - 1] == 1 && mapGen.map [i, j] == 0 && mapGen.IsInMapRange (i, j))
{
                                mapGen.map [i, j] = 4;
                                spikesPos.Add (new MapGenerator.Coord(i,j));
                                rand = UnityEngine.Random.Range(mapGen.minSpikes,mapGen.maxSpikes);
                        }
                        if (i > mapGen.width - mapGen.maxSpikes) {
                                break;
                        }
                }
                if (i > mapGen.width - mapGen.maxSpikes) {
                        break;
                }
        }
        //WALKERS
        rand = UnityEngine.Random.Range (mapGen.minWalker, mapGen.maxWalker);
        for(int i=mapGen.minWalker;i<mapGen.width-mapGen.maxWalker;i+=rand){
                for (int j=1;j<mapGen.height-1;j++) {
                        if (mapGen.map [i, j - 1] == 1 && mapGen.map [i, j] == 0&&mapGen.map[i+1,j-1]==1 &&
                        mapGen.map[i+1,j]==0&&mapGen.map[i+2,j-1]==1&&mapGen.map[i+2,j]==0 &&
                        mapGen.IsInMapRange (i, j)) {
                                mapGen.map [i, j] = 7;
                                mapGen.map [i+1, j] = 7;
                                mapGen.map [i+2, j] = 7;
                                walkersPos.Add (new MapGenerator.Coord(i,j));
                                rand = UnityEngine.Random.Range (mapGen.minWalker, mapGen.maxWalker);
                        }
                        if (i > mapGen.width - mapGen.maxWalker) {
                                break;
                        }
                }
                if (i > mapGen.width - mapGen.maxWalker) {
                        break;
                }
        }
        //FLYERS
        rand = UnityEngine.Random.Range (mapGen.minFlyerX, mapGen.maxFlyerX);
        for (int i=mapGen.minFlyerX;i<mapGen.width-mapGen.maxFlyerX;i+=rand){
                if(i>=mapGen.width-mapGen.maxFlyerX){
                        break;
                }
                for (int j = 1; j <= mapGen.height-1 ; j++) {
                        if (mapGen.map [i, j] == 0 &&mapGen.map[i,j-1]==1&& mapGen.IsInMapRange(i,j)) {
                                rand = UnityEngine.Random.Range (mapGen.minFlyerY, mapGen.maxFlyerY);
                                if(mapGen.IsInMapRange(i,j+rand)&&mapGen.map[i,j+rand]==0){
                                        mapGen.map [i, j+rand] = 5;
                                        flyersPos.Add (new MapGenerator.Coord (i, j+rand));
                                        break;
                                }
                        }
                }
                rand = UnityEngine.Random.Range (mapGen.minFlyerX,mapGen.maxFlyerX);
        }
        //JUMPERS
        rand = UnityEngine.Random.Range (mapGen.minJumper, mapGen.maxJumper);
        for (int i = mapGen.minJumper; i < mapGen.width-mapGen.maxJumper; i+=rand) {
                for (int j = 1; j < mapGen.height-1; j++) {
                        if(mapGen.map[i,j-1]==1&&mapGen.map[i,j]==0&&mapGen.IsInMapRange(i,j)){
```

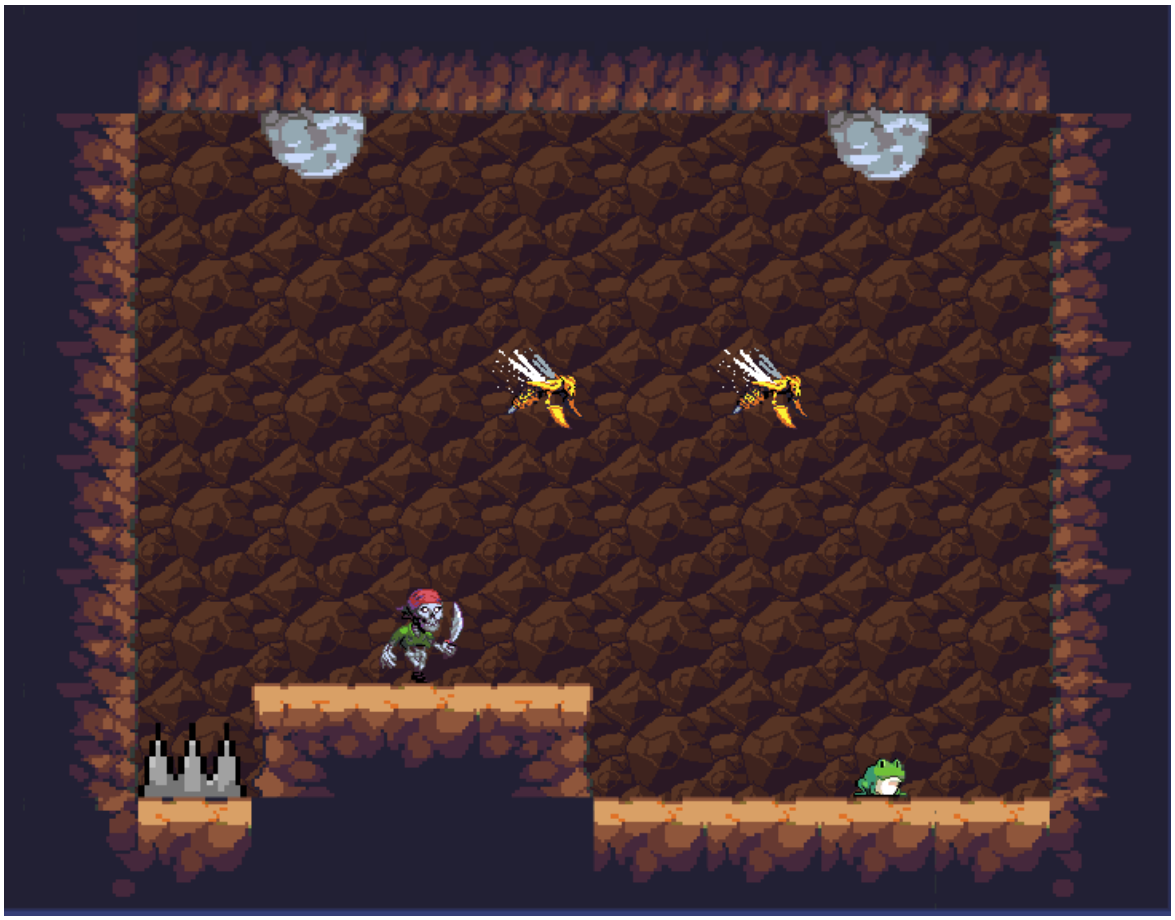
```
                                mapGen.map [i, j] = 6;
                                jumpersPos.Add (new MapGenerator.Coord (i, j));
                                rand = UnityEngine.Random.Range (mapGen.minJumper, mapGen.maxJumper);
                        }
                        if (i > mapGen.width-mapGen.maxJumper) {
                                break;
                        }
                }
                if (i > mapGen.width-mapGen.maxJumper) {
                        break;
                }
        }
        //DROPPERS
        rand = UnityEngine.Random.Range (mapGen.minDropper,mapGen.maxDropper);
        for (int i =mapGen.minDropper; i < mapGen.width-mapGen.maxDropper; i++) {
                for (int j=1;j<=mapGen.height-1;j++){
                        if(mapGen.map[i,j]==0&&mapGen.map[i,j+1]==1&&mapGen.IsInMapRange(i,j)){
                                mapGen.map [i, j] = 8;
                                droppersPos.Add (new MapGenerator.Coord (i, j));
                                rand = UnityEngine.Random.Range (mapGen.minDropper, mapGen.maxDropper);
                                i = i + rand;
                        }
                        if (i > mapGen.width-mapGen.maxDropper) {
                                break;
                        }
                }
                if (i >mapGen.width-mapGen.maxDropper) {
                        break;
                }
        }
        //COINS
        rand = UnityEngine.Random.Range (mapGen.minCoin,mapGen.maxCoin);
        for (int i =mapGen.minCoin; i < mapGen.width-mapGen.minCoin; i++) {
                for (int j=1;j<=mapGen.height-1;j++){
                        if(mapGen.map[i,j]==0&&mapGen.map[i,j-1]==0&&
                        mapGen.map[i,j-2]==1&&mapGen.IsInMapRange(i,j)){
                                mapGen.map [i, j] = 10;
                                coinsPos.Add (new MapGenerator.Coord (i, j));
                                rand = UnityEngine.Random.Range (mapGen.minCoin, mapGen.maxCoin);
                                i = i + rand;
                        }
                        if (i > mapGen.width-mapGen.maxCoin) {
                                break;
                        }
                }
                if (i >mapGen.width-mapGen.maxCoin) {
                        break;
                }
        }
}
```

Το μαρκάρισμα των θέσεων των αντιπάλων στη γραμμική μέθοδο CreateObjectPositions_Linear() γίνεται με τη χρήση της τυχαίας μεταβλητής rand, η οποία κάθε φορά παίρνει τιμή από ένα συγκεκριμένο πεδίο τιμών. Αρχικά, παίρνει τιμή μεταξύ των minSpikes και maxSpikes, όπου συμβολίζουν την ελάχιστη και μέγιστη απόσταση μεταξύ δύο διαδοχικών Spikes στον οριζόντιο άξονα. Χρησιμοποιώντας, λοιπόν, τη τιμή αυτή διασχίζουμε το χάρτη τυχαία στον οριζόντιο άξονα και τοποθετούμε το εμπόδιο Spikes κάθε φορά που βρίσκουμε τη κατάλληλη θέση, δηλαδή κενό κελί ακριβώς πάνω από έδαφος.

Με όμοιο τρόπο τοποθετούνται και οι Walkers, Jumpers, Droppers και Coins με τη διαφορά ότι χρησιμοποιούνται διαφορετικές ελάχιστες και μέγιστες αποστάσεις για το κάθε αντικείμενο (minWalker, maxWalker για τους Walkers, minJumper, maxJumper για τους Jumpers κ.ο.κ) και αλλάζουν τα pattern για τον έλεγχο τοποθέτησης τους. Επιπρόσθετα, οι αντίπαλοι Flyer τοποθετούνται πάλι με τυχαίο τρόπο στον άξονα x με τις αποστάσεις minFlyerX και maxFlyerX, και στον άξονα y τοποθετούνται μέσω των αποστάσεων minFlyerY και maxFlyerY τυχαία, κατά δύο έως έξι κελιά πάνω από το έδαφος.

Παρακάτω φαίνεται ένα παράδειγμα χάρτη σε μορφή πίνακα που θα μπορούσε να παραχθεί, ώστε να κατανοηθούν ευκολότερα τα pattern αυτά.

1	1	1	1	1	1	1	1	1	1
1	0	8	0	0	0	0	8	0	1
1	0	0	0	0	0	0	0	0	1
1	0	0	0	5	0	5	0	0	1
1	0	0	0	0	0	0	0	0	1
1	0	7	7	7	0	0	0	0	1
1	4	1	1	1	0	0	6	0	1
1	1	1	1	1	1	1	1	1	1



Empty	0
Ground	1
Spikes	4
Flyer	5
Jumper	6
Walker	7
Dropper	8

Roombased

```
public void CreateCheckPoints_Roombased(List<MapGenerator.Room> survivingRooms){
        startRoom=new MapGenerator.Room();
        goalRoom=new MapGenerator.Room();
        checkRoom = new MapGenerator.Room ();
        bool startC = false;
        bool goalC = false;
        int connRoomsCount;
        foreach (MapGenerator.Room room in survivingRooms) {
                connRoomsCount=room.connectedRooms.Count;
                //Create start point
                if (connRoomsCount == 1&&!startC) {
                        startRoom = room;
                        foreach (MapGenerator.Coord tile in startRoom.ReturnTiles()) {
                                if (mapGen.map [tile.tileX, tile.tileY - 1] == 1 &&
                                mapGen.IsInMapRange (tile.tileX, tile.tileY)) {
                                        startX = tile.tileX;
                                        startY = tile.tileY;
                                        mapGen.map [tile.tileX, tile.tileY] = 2;
                                        CreateArea (tile.tileX, tile.tileY);
                                        startC = true;
                                        break;
                                }
                        }
                }
                //Create goal
                if (connRoomsCount == 1 && room != startRoom&&!goalC) {
                        goalRoom = room;
                        foreach (MapGenerator.Coord tile in goalRoom.ReturnTiles()) {
                                if (mapGen.map [tile.tileX, tile.tileY - 1] == 1 &&
                                mapGen.IsInMapRange (tile.tileX, tile.tileY)) {
                                        goalX = tile.tileX;
                                        goalY = tile.tileY;
                                        mapGen.map [tile.tileX, tile.tileY] = 3;
                                        CreateArea (tile.tileX, tile.tileY);
                                        goalC = true;
                                        break;
                                }
                        }
                }
        }
        //Create checkpoints
        foreach (MapGenerator.Coord tile in mapGen.survivingPassageCoords) {
                if (previousCheckX==0&&previousCheckY==0) {
                        previousCheckX = tile.tileX;
                        previousCheckY = tile.tileY;
                }
                if(Mathf.Abs(tile.tileX-previousCheckX)>20&&Mathf.Abs(tile.tileY-previousCheckY)>20){
                        if (mapGen.map [tile.tileX, tile.tileY - 1] == 1&&mapGen.map [tile.tileX, tile.tileY] == 0
                                && mapGen.IsInMapRange (tile.tileX, tile.tileY)) {
```

```
                                checkpointsPos.Add (tile);
                                previousCheckX = tile.tileX;
                                previousCheckY = tile.tileY;
                                mapGen.map [tile.tileX, tile.tileY] = 9;
                                CreateArea (tile.tileX, tile.tileY);
                        }
                }
        }
}
```

Στη προσέγγιση δωματίων(Roombased) ο χάρτης διασχίζεται με βάση τα δωμάτια και τοποθετούνται τα Starting Point,Checkpoint και Goal.Το Starting Point τοποθετείται στο πρώτο κενό κελί πάνω από έδαφος μέσα σε κάποιο δωμάτιο. Έπειτα, ελέγχουμε τα υπόλοιπα δωμάτια και δεδομένου ότι το δωμάτιο που επιλέγεται δεν είναι αυτό που περιέχει το Starting point, τοποθετούμε το Goal στο κατάλληλο σημείο. Στη συνέχεια, γίνεται η τοποθέτηση των Checkpoint στα περάσματα των δωματίων. Το κάθε Checkpoint πρέπει να έχει απόσταση τουλάχιστον 20 κελιά οριζόντια και κάθετα από κάθε άλλο Checkpoint, ώστε να μην τοποθετούνται πολλά Checkpoint μαζί. Τέλος, όπως και προηγουμένως, δημιουργείται μία περιοχή γύρω από τα αντικείμενα αυτά ώστε να μην τοποθετούνται αντίπαλοι σε κοντινά σημεία.

```
public void CreateObjectPositions_Roombased(List<MapGenerator.Room> survivingRooms){
        foreach (MapGenerator.Room room in survivingRooms) {
                //SPIKES
                foreach (MapGenerator.Coord tile in room.ReturnTiles()) {
                        if (isOccupied (tile.tileX, tile.tileY)) {
                                continue;
                        }
                        if (mapGen.map [tile.tileX, tile.tileY-1] == 1 && mapGen.map [tile.tileX, tile.tileY] == 0) {
                                if (randomBoolean (mapGen.randomBoolValueS)) {
                                        mapGen.map [tile.tileX, tile.tileY] = 4;
                                        spikesPos.Add (tile);
                                }
                        }
                }
                //WALKERS
                foreach (MapGenerator.Coord tile in room.ReturnTiles()) {
                        if (isOccupied (tile.tileX, tile.tileY)) {
                                continue;
                        }
                        if (mapGen.map [tile.tileX, tile.tileY] == 0&&mapGen.map [tile.tileX+1, tile.tileY] == 0&&
                                mapGen.map [tile.tileX+2, tile.tileY] == 0 &&mapGen.map [tile.tileX, tile.tileY-1] == 1&&
                                mapGen.map [tile.tileX+1, tile.tileY-1] == 1&&mapGen.map [tile.tileX+2, tile.tileY-1]==1) {
                                if (randomBoolean (mapGen.randomBoolValueW)) {
                                        mapGen.map [tile.tileX, tile.tileY] = 7;
                                        mapGen.map [tile.tileX + 1, tile.tileY] = 7;
                                        mapGen.map [tile.tileX + 2, tile.tileY] = 7;
                                        walkersPos.Add (tile);
                                }
                        }
                }
                //FLYERS
                foreach (MapGenerator.Coord tile in room.ReturnTiles()) {
                        if (isOccupied (tile.tileX, tile.tileY)) {
                                continue;
                        }
                        if (mapGen.map [tile.tileX, tile.tileY]==0&&randomBoolean (mapGen.randomBoolValueF)) {
                                mapGen.map [tile.tileX, tile.tileY] = 5;
                                flyersPos.Add (tile);
```

```
                }
            }
            //JUMPERS
            foreach (MapGenerator.Coord tile in room.ReturnTiles()) {
                if (isOccupied (tile.tileX, tile.tileY)) {
                    continue;
                }
                if (mapGen.map [tile.tileX, tile.tileY-1] == 1 && mapGen.map [tile.tileX, tile.tileY] == 0) {
                    if (randomBoolean (mapGen.randomBoolValueJ)) {
                        mapGen.map [tile.tileX, tile.tileY] = 6;
                        jumpersPos.Add (tile);
                    }
                }
            }
            //DROPPERS
            foreach (MapGenerator.Coord tile in room.ReturnTiles()) {
                if (isOccupied (tile.tileX, tile.tileY)) {
                    continue;
                }
                if (mapGen.map [tile.tileX, tile.tileY+1] == 1 && mapGen.map [tile.tileX, tile.tileY] == 0) {
                    if (randomBoolean (mapGen.randomBoolValueD)) {
                        mapGen.map [tile.tileX, tile.tileY] = 8;
                        droppersPos.Add (tile);
                    }
                }
            }
            //COINS
            foreach (MapGenerator.Coord tile in room.ReturnTiles()) {
                if (isOccupied (tile.tileX, tile.tileY)) {
                    continue;
                }
                if (mapGen.IsInMapRange (tile.tileX, tile.tileY - 2) &&
                mapGen.IsInMapRange (tile.tileX, tile.tileY - 1) &&  mapGen.IsInMapRange (tile.tileX, tile.tileY)) {
                    if (mapGen.map [tile.tileX, tile.tileY - 2] == 1 &&
                    mapGen.map [tile.tileX, tile.tileY - 1] == 0 &&mapGen.map [tile.tileX, tile.tileY] == 0) {
                        if (randomBoolean (mapGen.randomBoolValueC)) {
                            mapGen.map [tile.tileX, tile.tileY] = 10;
                            coinsPos.Add (tile);
                        }
                    }
                }
            }
        }
        foreach (MapGenerator.Coord coord in mapGen.survivingPassageCoords) {
            if (isOccupied (coord.tileX, coord.tileY)) {
                continue;
            }
            //SPIKES
            if (mapGen.map [coord.tileX, coord.tileY-1] == 1 && mapGen.map [coord.tileX, coord.tileY] == 0) {
                if (randomBoolean (mapGen.randomBoolValueS)) {
                    mapGen.map [coord.tileX, coord.tileY] = 4;
                    spikesPos.Add (coord);
                }
            }
            //WALKERS
            if (mapGen.map [coord.tileX, coord.tileY] == 0 && mapGen.map [coord.tileX+1, coord.tileY] == 0&&
                mapGen.map [coord.tileX+2, coord.tileY] == 0 &&mapGen.map [coord.tileX, coord.tileY-1] == 1&&
                mapGen.map [coord.tileX+1, coord.tileY-1]==1&&mapGen.map [coord.tileX+2, coord.tileY-1]==1) {
                if (randomBoolean (mapGen.randomBoolValueW)) {
                    mapGen.map [coord.tileX, coord.tileY] = 7;
                    mapGen.map [coord.tileX + 1, coord.tileY] = 7;
                    mapGen.map [coord.tileX + 2, coord.tileY] = 7;
                    walkersPos.Add (coord);
                }
```

```
                }
                //FLYERS
                if (mapGen.map [coord.tileX, coord.tileY]==0&&randomBoolean (mapGen.randomBoolValueF)) {
                        mapGen.map [coord.tileX, coord.tileY] = 5;
                        flyersPos.Add (coord);
                }
                //JUMPERS
                if (mapGen.map [coord.tileX, coord.tileY-1] == 1 && mapGen.map [coord.tileX, coord.tileY] == 0) {
                        if (randomBoolean (mapGen.randomBoolValueJ)) {
                                mapGen.map [coord.tileX, coord.tileY] = 6;
                                jumpersPos.Add (coord);
                        }
                }
                //DROPPERS
                if (mapGen.map [coord.tileX, coord.tileY+1] == 1 && mapGen.map [coord.tileX, coord.tileY] == 0) {
                        if (randomBoolean (mapGen.randomBoolValueD)) {
                                mapGen.map [coord.tileX, coord.tileY] = 8;
                                droppersPos.Add (coord);
                        }
                }
                //COINS
                if (mapGen.map [coord.tileX, coord.tileY-2] == 1&&
                mapGen.map [coord.tileX, coord.tileY-1] == 0 && mapGen.map [coord.tileX, coord.tileY] == 0) {
                        if (randomBoolean (mapGen.randomBoolValueC)) {
                                mapGen.map [coord.tileX, coord.tileY] = 10;
                                coinsPos.Add (coord);
                        }
                }
        }
}
```

Το μαρκάρισμα των θέσεων των αντιπάλων στη CreateObjectPositions_Roombased() μέθοδο γίνεται με τη χρήση των μεταβλητών randomBoolValueS, randomBoolValueW, randomBoolValueF, randomBoolValueD, randomBoolValueJ και randomBoolValueC. Η τιμή randomBoolValueS συμβολίζει τη πιθανότητα να μην τοποθετηθούν τελικά Spikes σε κάποιο κελί δεδομένου ότι μπορούν να τοποθετηθούν Spikes, δηλαδή ικανοποιούν τις συνθήκες για τη τοποθέτηση τους. Συνεπώς, η πιθανότητα να τοποθετηθούν τελικά σε κάποιο κελί όπου μπορούν να τοποθετηθούν Spikes είναι 1-randomBoolValueS. Αντίστοιχα, η τιμή randomBoolValueW συμβολίζει τη πιθανότητα να μην τοποθετηθούν Walkers σε κάποιο κελί που μπορούν να τοποθετηθούν, η τιμή randomBoolValueF συμβολίζει τη πιθανότητα να μην τοποθετηθούν Flyers σε κάποιο κελί που μπορούν να τοποθετηθούν, κ.ο.κ.

Για κάθε κελί μέσα σε κάθε δωμάτιο ελέγχεται αν είναι κενό με την μέθοδο isOccupied(). Αν δεν είναι, συνεχίζεται η αναζήτηση στο επόμενο κελί, αλλιώς ελέγχεται αν το pattern της περιοχής του κελιού ταιριάζει με αυτό του αντικειμένου που τοποθετούμε. Εφόσον ταιριάζει τοποθετείται στο κελί αυτό το αντικείμενο με πιθανότητα 1-randomBoolValue και μαρκάρεται το σημείο αυτό με την αντίστοιχη τιμή στον πίνακα map.

Παρακάτω φαίνονται οι βοηθητικές μέθοδοι που χρησιμοποιούνται για το μαρκάρισμα των θέσεων των οντοτήτων

```
void CreateArea(int i,int j){
	for (int neighbourX = i - checkpointRadius; neighbourX <= i + checkpointRadius; neighbourX++) {
		for (int neighbourY = j - checkpointRadius; neighbourY <= j + checkpointRadius; neighbourY++) {
			if(!mapGen.IsInMapRange(neighbourX,neighbourY)){
				continue;
			}
			if (mapGen.map [neighbourX, neighbourY] == 1) {
				continue;
			}
			if (mapGen.map [neighbourX, neighbourY] == 0&&
			mapGen.IsInMapRange(neighbourX,neighbourY)) {
				mapGen.map [neighbourX, neighbourY] = 9;
			}
		}
	}
}

bool randomBoolean (float randomBoolValue){
	if (Random.value >= randomBoolValue)
	{
		return true;
	}
	return false;
}

bool isOccupied(int x,int y){
	if ((x == startX && y == startY) || (x == checkX && y == checkY) || (x == goalX && y == goalY)) {
		return true;
	}
	return false;
}
```

4.4.3. Instantiator

Μόλις γίνει το μαρκάρισμα των θέσεων στον πίνακα map και ολοκληρωθεί η αποθήκευση των θέσεων στις λίστες, ξεκινάει η εκτέλεση του Instantiator.cs script. Ο σκοπός του Instantiator είναι η τοποθέτηση των αντικειμένων μέσα στο παιχνίδι με βάση τις θέσεις τους. Για να το κάνει αυτό ο Instantiator χρειάζεται έναν φάκελο ο οποίος περιέχει όλα τα αντικείμενα που τοποθετούνται δυναμικά. Γι'αυτό ορισμένα αντικείμενα βρίσκονται στον φάκελο "Resources" και άλλα βρίσκονται στον φάκελο "Prefabs". Η σειρά της τοποθέτησης φαίνεται στην μέθοδο InstantiateGameObjects()

```
void InstantiateGameObjects(){
        SpawnBackground ();
        SpawnGround ();
        SpawnStartPoint ();
        SpawnCheckPoint ();
        SpawnGoal ();
        SpawnSpikes();
        SpawnWalkers ();
        SpawnFlyers();
        SpawnJumpers();
        SpawnDroppers();
        SpawnCoins ();
}

void SpawnBackground(){
        for (int i = 0; i < width; i++) {
                for (int j = 0; j < height; j++) {
                        Instantiate (Resources.Load ("Background"), new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                }
        }
}
```

Αρχικά, τοποθετείται το Background στη μέθοδο SpawnBackground(), διατρέχοντας ολόκληρο τον χάρτη και τοποθετώντας σε κάθε κελί ένα αντικείμενο Background.

```
void SpawnGround(){
        for (int i = 0; i < width; i++) {
                for (int j = 0; j < height; j++) {
                        //Debug.Log ("i: "+i+" j: "+j+" value : "+map [i, j]);
                        if (map [i, j] == 1) {
                                //Spawn ground outside of the map
                                if(i==0||j==0||i==width-1||j==height-1){
                                        //Left
                                        if (i == 0) {
                                                if (map [i+1, j] != 1) {
                                                        Instantiate (Resources.Load ("Ground_middle_right_1"),
                                                        new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                                                } else {
                                                        Instantiate (Resources.Load ("Ground_middle"),
                                                        new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                                                }
                                                for (int k = 1; k <= 10; k++) {
                                                        Instantiate (Resources.Load ("Ground_middle"),
                                                        new Vector3 (i - k, j, 0), Quaternion.Euler (0, 0, 0));
                                                }
                                        }
                                        //Bottom
                                        else if (j == 0) {
                                                if (map [i, j+1] != 1) {
                                                        Instantiate (Resources.Load ("Ground_top"),
                                                        new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                                                } else {
                                                        Instantiate (Resources.Load ("Ground_middle"),
                                                        new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                                                }
                                                for (int k = 1; k <= 10; k++) {
                                                        Instantiate (Resources.Load ("Ground_middle"),
                                                        new Vector3 (i, j - k, 0), Quaternion.Euler (0, 0, 0));
                                                }
                                        }
                                        //Right
                                        else if (i == width - 1) {
                                                if (map [i-1, j] != 1) {
                                                        Instantiate (Resources.Load ("Ground_middle_left_1"),
                                                        new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                                                } else {
                                                        Instantiate (Resources.Load ("Ground_middle"),
                                                        new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                                                }
                                                for (int k = 1; k <= 10; k++) {
                                                        Instantiate (Resources.Load ("Ground_middle"),
                                                        new Vector3 (i + k, j, 0), Quaternion.Euler (0, 0, 0));
                                                }
                                        }
                                        //Top
                                        else if (j == height - 1) {
                                                if (map [i, j-1] != 1) {
                                                        Instantiate (Resources.Load ("Ground_bottom"),
                                                        new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                                                } else {
                                                        Instantiate (Resources.Load ("Ground_middle"),
                                                        new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                                                }
                                                for (int k = 1; k <= 10; k++) {
                                                        Instantiate (Resources.Load ("Ground_middle"),
                                                        new Vector3 (i, j + k, 0), Quaternion.Euler (0, 0, 0));
                                                }
                                        }
                                        //Bottom-left
```

```
                if (i == 0 && j == 0) {
                        for (int k = 0; k <= 10; k++) {
                                for (int l = 0; l <= 10; l++) {
                                        Instantiate (Resources.Load ("Ground_middle"),
                                        new Vector3 (i - k, j - l, 0), Quaternion.Euler (0, 0, 0));
                                        }
                                }
                        }
                        //Top-left
                        if (i == 0 && j == height-1) {
                                for (int k = 0; k <= 10; k++) {
                                        for (int l = 0; l <= 10; l++) {
                                                Instantiate (Resources.Load ("Ground_middle"),
                                                new Vector3 (i - k, j + l, 0),
                                                Quaternion.Euler (0, 0, 0));
                                        }
                                }
                        }
                        //Bottom-right
                        if (i == width-1 && j == 0) {
                                for (int k = 0; k <= 10; k++) {
                                        for (int l = 0; l <= 10; l++) {
                                                Instantiate (Resources.Load ("Ground_middle"),
                                                new Vector3 (i + k, j - l, 0),
                                                Quaternion.Euler (0, 0, 0));
                                        }
                                }
                        }
                        //Top-right
                        if (i == width-1&& j == height-1) {
                                for (int k = 0; k <= 10; k++) {
                                        for (int l = 0; l <= 10; l++) {
                                                Instantiate (Resources.Load ("Ground_middle"),
                                                new Vector3 (i + k, j + l, 0),
                                                Quaternion.Euler (0, 0, 0));
                                        }
                                }
                        }
                continue;
        }
        //Spawn the ground with corresponding sprite
        if (map [i-1, j] != 1 && map [i+1, j] == 1  && map [i, j-1] == 1 && map [i, j+1] != 1) {
                Instantiate (Resources.Load ("Ground_top_left"),
                new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                continue;
        }
        if (map [i-1, j] == 1 && map [i+1, j] == 1 && map [i, j-1] == 1 && map [i, j+1] != 1) {
                Instantiate (Resources.Load ("Ground_top"),
                new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                continue;
        }
        if (map [i-1, j] != 1 && map [i+1, j] != 1 && map [i, j-1] == 1 && map [i, j+1] != 1) {
                Instantiate (Resources.Load ("Ground_top_2"),
                new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                continue;
        }
        if (map [i-1, j] == 1 && map [i+1, j] != 1 && map [i, j-1] == 1 && map [i, j+1] != 1) {
                Instantiate (Resources.Load ("Ground_top_right"),
                new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                continue;
        }
        if (map [i-1, j] != 1 && map [i+1, j] == 1 && map [i, j+1] == 1 && map [i, j-1] == 1) {
                Instantiate (Resources.Load ("Ground_middle_left_1"),
                new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
```

```
                    continue;
                }
                if (map [i-1, j] == 1 && map [i+1, j] == 1 && map [i, j+1] == 1 &&
                map [i, j-1] == 1&&map [i-1, j+1] != 1) {
                    Instantiate (Resources.Load ("Ground_middle_left_2"),
                    new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                    continue;
                }
                if (map [i-1, j] == 1 && map [i+1, j] != 1 && map [i, j+1] == 1 && map [i, j-1] == 1) {
                    Instantiate (Resources.Load ("Ground_middle_right_1"),
                    new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                    continue;
                }
                if (map [i-1, j] == 1 && map [i+1, j] == 1 && map [i, j+1] == 1 &&
                map [i, j-1] == 1&&map [i+1, j+1] != 1) {
                    Instantiate (Resources.Load ("Ground_middle_right_2"),
                    new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                    continue;
                }
                if (map [i-1, j] == 1 && map [i+1, j] == 1 && map [i, j+1] == 1 && map [i, j-1] == 1) {
                    Instantiate (Resources.Load ("Ground_middle"),
                    new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                    continue;
                }
                if (map [i-1, j] != 1 && map [i+1, j] == 1 && map [i, j+1] == 1 && map [i, j-1] != 1) {
                    Instantiate (Resources.Load ("Ground_bottom_left"),
                    new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                    continue;
                }
                if (map [i-1, j] == 1 && map [i+1, j] == 1 && map [i, j+1] == 1 && map [i, j-1] != 1) {
                    Instantiate (Resources.Load ("Ground_bottom"),
                    new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                    continue;
                }
                if (map [i-1, j] == 1 && map [i+1, j] != 1 && map [i, j+1] == 1 &&map [i, j-1] != 1) {
                    Instantiate (Resources.Load ("Ground_bottom_right"),
                    new Vector3 (i, j, 0), Quaternion.Euler (0, 0, 0));
                    continue;
                }
            }
        }
    }
}
```

Έπειτα εκτελείται η μέθοδος SpawnGround(). Στη μέθοδο αυτή γίνεται αντιστοίχιση του τύπου εδάφους στον φάκελο Resources, όπου υπάρχουν είδη εδαφών με διαφορετικά γραφικά, με τον τρόπο που διαμορφώθηκαν νωρίτερα στο χάρτη map. Το μέγεθος της μεθόδου οφείλεται στο γεγονός ότι για κάθε κελί εδάφους χρειάζεται να υπολογίσουμε το pattern στο οποίο αντιστοιχεί, και να του αναθέσουμε το κατάλληλο κελί.

Αρχικά ελέγχουμε αν το κελί βρίσκεται στις ακραίες τιμές του πίνακα, εντοπίζουμε τη πλευρά(αριστερά,δεξιά,πάνω,κάτω) και αντιστοιχούμε το κατάλληλο είδος εδάφους στο κελί αυτό. Επίσης επεκτείνουμε τις ακραίες τιμές κατά δέκα θέσεις έξω από τον χάρτη σε κάθε πλευρά για δημιουργήσουμε στο παίχτη την αίσθηση ότι βρίσκεται σε σπηλιά και δεν υπάρχει δεύτερη διέξοδος πέρα από το αντικείμενο Goal. Για κάθε κελί που βρίσκεται μέσα στο χάρτη ελέγχεται η γειτονιά του(Von Neumann) και με βάση αυτή τοποθετείται ο αντίστοιχος τύπος εδάφους.

```
void SpawnStartPoint(){
        player.transform.localPosition = new Vector2 (lvlHandler.startX, lvlHandler.startY);
        mainCamera.transform.localPosition = new Vector3 (lvlHandler.startX, lvlHandler.startY,-10);
        Instantiate (Resources.Load ("Startpoint"),
        new Vector3 (lvlHandler.startX, lvlHandler.startY, 0), Quaternion.Euler (0, 0, 0));
}
```

Μόλις τοποθετηθούν τα εδάφη μεταφέρεται ο χαρακτήρας του παίχτη και η κάμερα στη συντεταγμένη του αρχικού σημείου και δημιουργούμε το αρχικό σημείο.

```
void SpawnCheckPoint(){
        Instantiate (Resources.Load ("Checkpoint"),
        new Vector3 (lvlHandler.checkX+0.2f, lvlHandler.checkY+0.45f, 0), Quaternion.Euler (0, 0, 0));
        for (int k = 0; k < lvlHandler.checkpointsPos.Count; k++) {
                Instantiate (Resources.Load ("Checkpoint"),
                new Vector3 (lvlHandler.checkpointsPos[k].tileX+0.2f,
                lvlHandler.checkpointsPos[k].tileY+0.45f, 0), Quaternion.Euler (0, 0, 0));
        }
}
```

Στη συνέχεια εκτελείται η μέθοδος SpawnCheckPoint(), όπου τοποθετούνται τα Checkpoint με βάση των θέσεων τους στη λίστα checkpointsPos και γίνεται μετατόπιση των θέσεών τους κατά 0.2 στον άξονα x και 0.45 στον άξονα y για να ταιριάζουν περισσότερο στο περιβάλλον του χάρτη.

```
void SpawnGoal(){
        Instantiate (Resources.Load ("Goal"),
        new Vector3 (lvlHandler.goalX, lvlHandler.goalY, 0), Quaternion.Euler (0, 0, 0));
}
```

Στη μέθοδο SpawnGoal() τοποθετείται η πόρτα Goal στη θέση goalX,goalY που υπολογίστηκε νωρίτερα.

```
void SpawnSpikes(){
        for (int i = 0; i < lvlHandler.spikesPos.Count; i++) {
                Instantiate (Resources.Load ("Spikes"),
                new Vector3 (lvlHandler.spikesPos[i].tileX,  lvlHandler.spikesPos[i].tileY-0.1f, 0),
                Quaternion.Euler (0, 0, 0));
        }
}
void SpawnWalkers(){
        for (int i = 0; i < lvlHandler.walkersPos.Count; i++) {
                Instantiate (Resources.Load ("Enemy_walker"),
                new Vector3 (lvlHandler.walkersPos[i].tileX,  lvlHandler.walkersPos[i].tileY, 0),
                Quaternion.Euler (0, 0, 0));
        }
}

void SpawnFlyers(){
        for (int i = 0; i < lvlHandler.flyersPos.Count; i++) {
                Instantiate (Resources.Load ("Enemy_flyer"),
                new Vector3 (lvlHandler.flyersPos[i].tileX,  lvlHandler.flyersPos[i].tileY, 0),
                Quaternion.Euler (0, 0, 0));
        }
}
```

```
void SpawnJumpers(){
	for (int i = 0; i < lvlHandler.jumpersPos.Count; i++) {
		Instantiate (Resources.Load ("Enemy_jumper"),
		new Vector3 (lvlHandler.jumpersPos[i].tileX,  lvlHandler.jumpersPos[i].tileY, 0),
		Quaternion.Euler (0, 0, 0));
	}
}
void SpawnDroppers(){
	for (int i = 0; i < lvlHandler.droppersPos.Count; i++) {
		Instantiate (Resources.Load ("Enemy_dropper"),
		new Vector3 (lvlHandler.droppersPos[i].tileX,  lvlHandler.droppersPos[i].tileY+0.5f, 0),
		Quaternion.Euler (0, 0, 0));
	}
}
void SpawnCoins(){
	for (int i = 0; i < lvlHandler.coinsPos.Count; i++) {
		Instantiate (Resources.Load ("Coin"),
		new Vector3 (lvlHandler.coinsPos[i].tileX,  lvlHandler.coinsPos[i].tileY, 0),
		Quaternion.Euler (0, 0, 0));
	}
}
```

Τέλος γίνεται η τοποθέτηση των αντικειμένων στις αντίστοιχες θέσεις τους μέσω των λιστών συντεταγμένων spikesPos, walkersPos, flyersPos, jumpersPos, droppersPos και coinsPos. Οι λίστες αυτές δημιουργήθηκαν για να αποφύγουμε να διατρέξουμε για κάθε αντίπαλο το πίνακα map, με σκοπό να ελαχιστοποιήσουμε το χρόνο εκτέλεσης του Instantiator.

4.5. Εφαρμογή του DDA

4.5.1. Επίπεδα δυσκολίας

Η διαδικασία διαμόρφωσης του επιπέδου δυσκολίας συμβαίνει στο αντικείμενο GameManager το οποίο λειτουργεί σε όλη τη διάρκεια του παιχνιδιού. Για να το καταφέρω αυτό χρειάστηκε να υλοποιήσω Singleton το οποίο βρίσκεται στη μέθοδο Awake() του GameManager.cs και διατηρεί το αντικείμενο σε όλο το παιχνίδι.

Το παιχνίδι ξεκινάει από την επιλογή του επιπέδου δυσκολίας από τον παίχτη. Στο κάθε επίπεδο δυσκολίας(Very Easy, Easy, Medium, Hard και Very Hard) υπάρχουν ορισμένες μεταβλητές οι οποίες καθορίζονται στις αντίστοιχες μεθόδους (VeryEasyValues(), EasyValues(), MediumValues(), HardValues(), VeryHardValues()). Οι μεταβλητές αυτές αφορούν τα χαρακτηριστικά μεγέθη του χάρτη(width, height,randomFillPercent), την αρχική τιμή του χρονομέτρου(timeleft), καθώς και χαρακτηριστικά μεγέθη των αντικειμένων που τοποθετούνται μέσα σε αυτόν (οι ελάχιστες και μέγιστες αποστάσεις των αντιπάλων, οι ταχύτητές τους κ.ο.κ).

Επίσης, ανάλογα το αρχικό επίπεδο δυσκολίας ενεργοποιούνται διαφορετικοί αντίπαλοι στο ξεκίνημα του παιχνιδιού. Αν ο παίχτης επιλέξει το επίπεδο “Very Easy” όλοι οι αντίπαλοι είναι απενεργοποιημένοι εκτός από έναν ο οποίος επιλέγεται τυχαία. Αν ο παίχτης επιλέξει το επίπεδο “Easy” ενεργοποιούνται τα Spikes και Walkers. Αν επιλέξει το επίπεδο “Medium” ενεργοποιούνται οι Jumpers και Droppers, αν επιλέξει το επίπεδο “Hard” ενεργοποιούνται όλοι οι αντίπαλοι εκτός από τους Flyers, και αν επιλέξει το επίπεδο “Very Hard” ενεργοποιούνται όλοι οι αντίπαλοι.

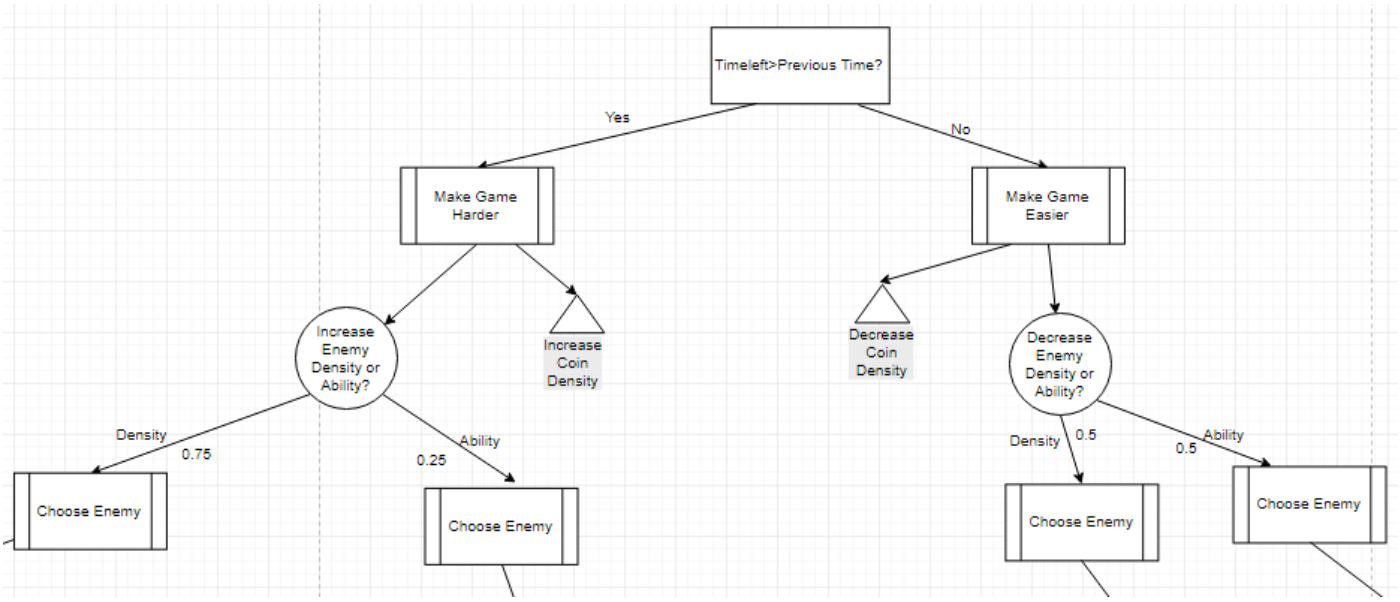
Τα αρχικά επίπεδα δυσκολίας δημιουργήθηκαν επειδή δεν γνωρίζουμε αρχικά αν ο παίχτης είναι έμπειρος ή άπειρος. Γι’αυτό του δίνουμε αρχικά την επιλογή να διαλέξει ένα από τα πέντε αυτά επίπεδα, και έπειτα προχωράμε στη διαδικασία προσαρμογής του παιχνιδιού με βάση τις ικανότητές του.

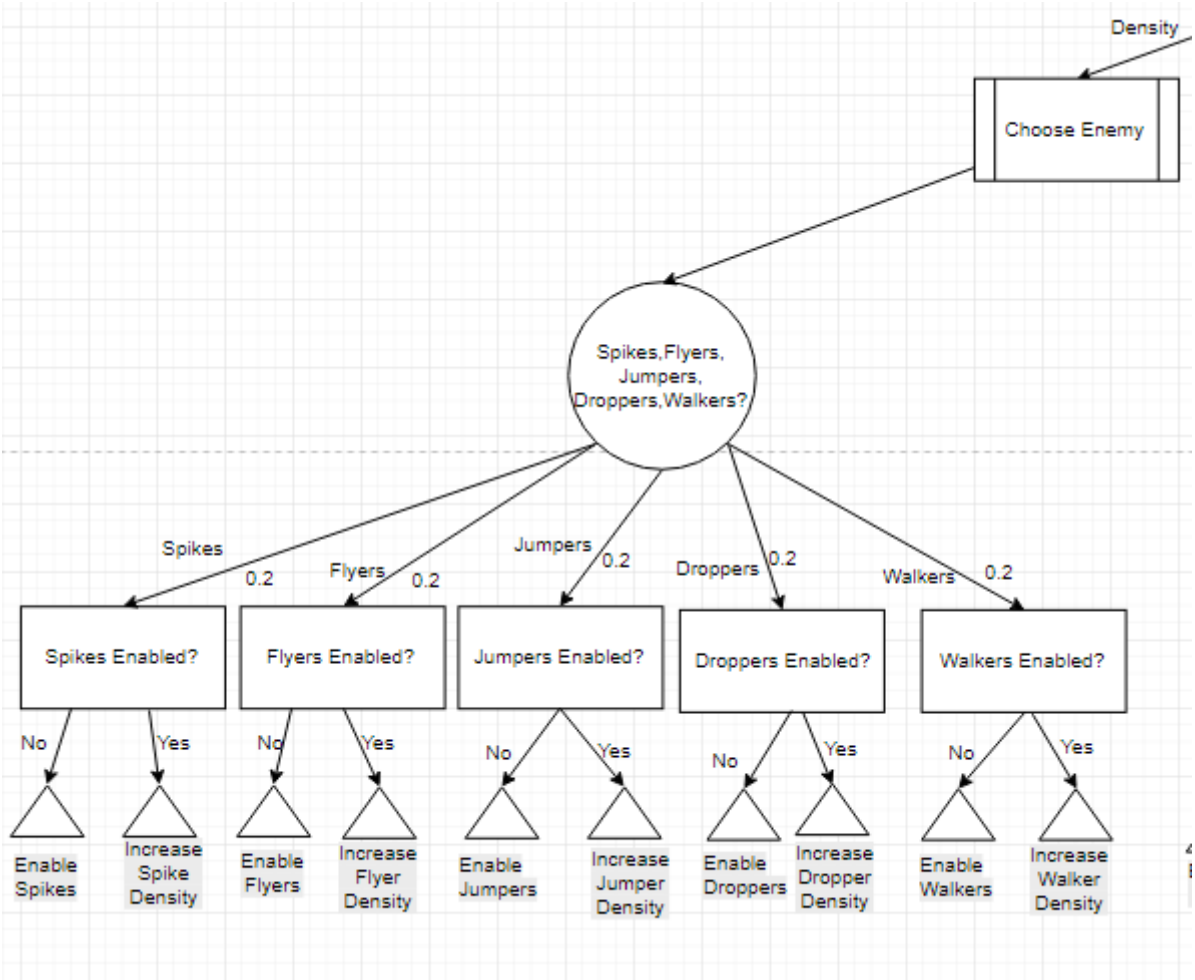
4.5.2. Διαμόρφωση δυσκολίας

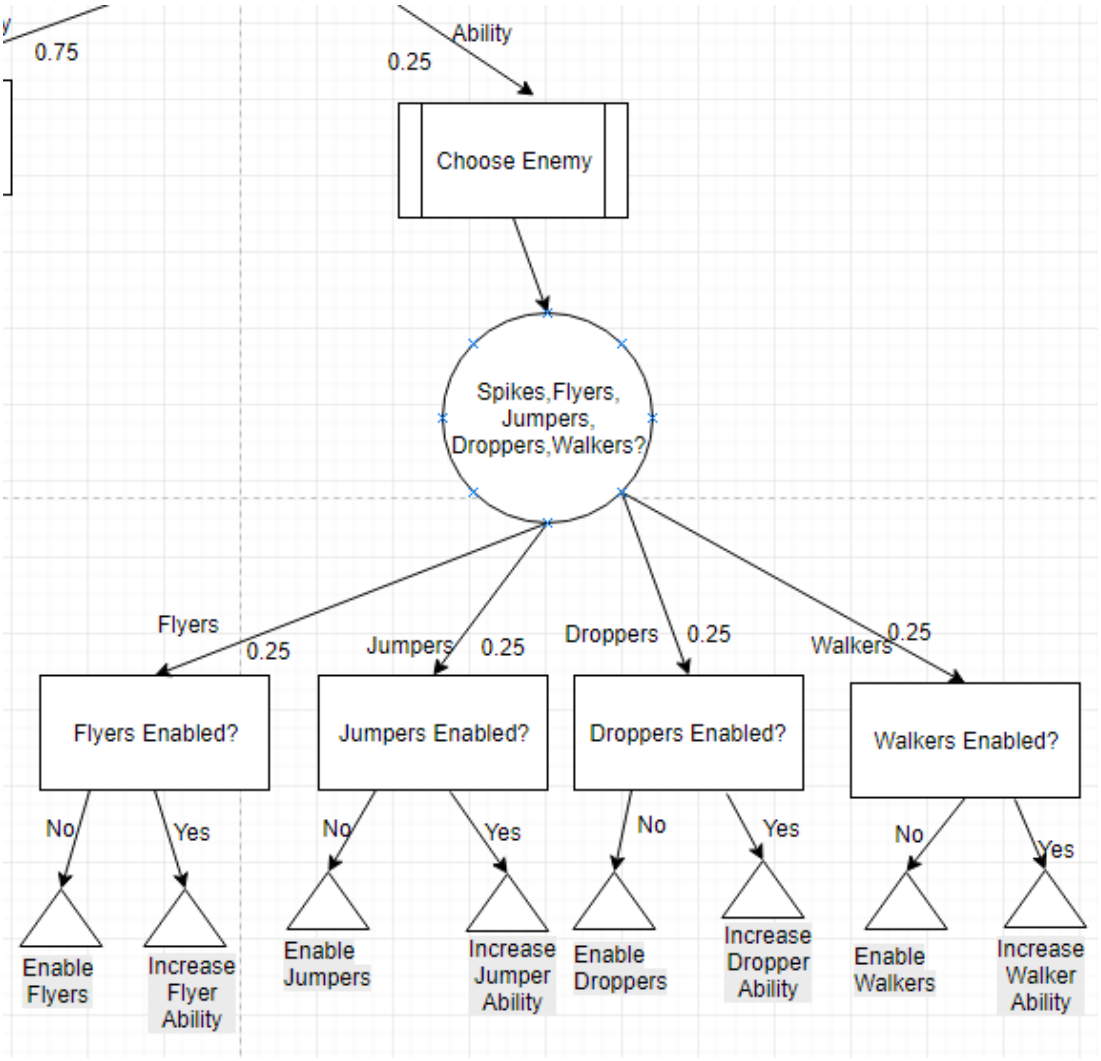
Η διαμόρφωση του επιπέδου δυσκολίας συμβαίνει στη μέθοδο DynamicDifficultyAdjustment() η οποία βρίσκεται στο GameManager.cs script. Επειδή το μέγεθος της μεθόδου είναι πολύ μεγάλο για να παρουσιαστεί στο παρόν έγγραφο, η μέθοδος αυτή παρουσιάζεται ως ένα δέντρο απόφασης (Decision Tree). Ένα δέντρο απόφασης είναι ένας γράφος από αποφάσεις όπου τα μονοπάτια οδηγούν σε συγκεκριμένες συνέπειες και αποτελεί ένα τρόπο απεικόνισης αλγορίθμων που αποτελούνται μόνο από εντολές επιλογών. Υπάρχουν τρία είδη κόμβων :

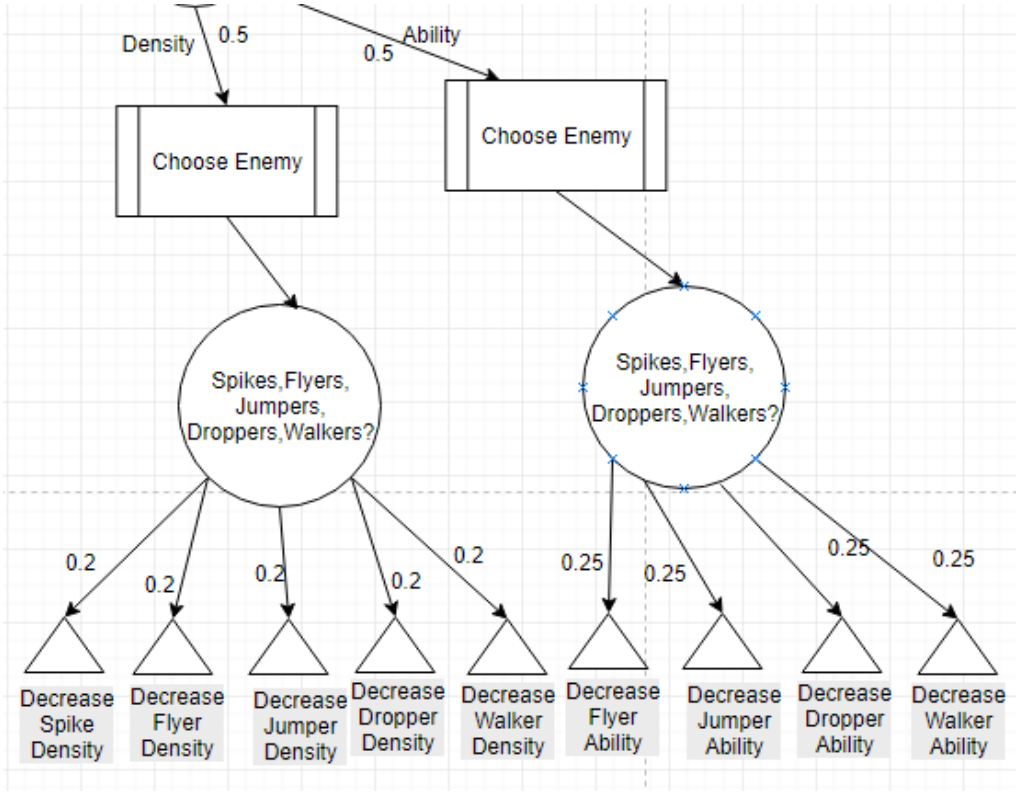
- Κόμβος απόφασης(Decision Node) - συμβολίζεται με τετράγωνο
- Κόμβος πιθανότητας(Chance Node) - συμβολίζεται με κύκλο
- Τελικός Κόμβος(End Node) - συμβολίζεται με τρίγωνο

Οι αριθμοί δίπλα από τα βέλη είναι η πιθανότητα επιλογής της συγκεκριμένης απόφασης


Timeleft>Previous Time?
Yes
No
Make Game Harder
Make Game Easier
Increase Enemy Density or Ability?
Increase Coin Density
Decrease Coin Density
Decrease Enemy Density or Ability?
Density
0.75
Ability
0.25
Density
0.5
Ability
0.5
Choose Enemy
Choose Enemy
Choose Enemy
Choose Enemy


Density
Choose Enemy
Spikes,Flyers, Jumpers, Droppers,Walkers?
Spikes
0.2
Flyers
0.2
Jumpers
0.2
Droppers
0.2
Walkers
0.2
Spikes Enabled?
Flyers Enabled?
Jumpers Enabled?
Droppers Enabled?
Walkers Enabled?
No
Yes
Enable Spikes
Increase Spike Density
Enable Flyers
Increase Flyer Density
Enable Jumpers
Increase Jumper Density
Enable Droppers
Increase Dropper Density
Enable Walkers
Increase Walker Density


0.75
Ability
0.25
Choose Enemy
Spikes,Flyers,
Jumpers,
Droppers,Walkers?
Flyers
0.25
Jumpers
0.25
Droppers
0.25
Walkers
0.25
Flyers Enabled?
Jumpers Enabled?
Droppers Enabled?
Walkers Enabled?
No
Yes
No
Yes
No
Yes
No
Yes
Enable
Flyers
Increase
Flyer
Ability
Enable
Jumpers
Increase
Jumper
Ability
Enable
Droppers
Increase
Dropper
Ability
Enable
Walkers
Increase
Walker
Ability


Density
0.5
Ability
0.5
Choose Enemy
Choose Enemy
Spikes,Flyers,
Jumpers,
Droppers,Walkers?
Spikes,Flyers,
Jumpers,
Droppers,Walkers?
0.2
0.2
0.2
0.2
0.2
0.25
0.25
0.25
0.25
Decrease
Spike
Density
Decrease
Flyer
Density
Decrease
Jumper
Density
Decrease
Dropper
Density
Decrease
Walker
Density
Decrease
Flyer
Ability
Decrease
Jumper
Ability
Decrease
Dropper
Ability
Decrease
Walker
Ability

Η αύξηση του επιπέδου δυσκολίας αποτελείται από δύο επιλογές, την αύξηση της πυκνότητας και την αύξηση των ικανοτήτων ενός αντιπάλου. Η πυκνότητα(Density) ορίζεται από τις τιμές που παίρνουν οι ελάχιστες και μέγιστες αποστάσεις (minSpikes,maxSpikes,minWalker,maxWalker) στα επίπεδα Very Easy και Easy, ενώ στα επίπεδα Medium, Hard και Very Hard ορίζεται από τις τυχαίες μεταβλητές randomBoolValue. Οι ικανότητες αποτελούνται συνήθως από τις ταχύτητες των αντιπάλων και των εμβελειών ανίχνευσης του παίχτη.

Αν η απόφαση είναι η αύξηση της πυκνότητας ενός αντιπάλου τότε μειώνουμε τις χαρακτηριστικές αυτές τιμές, δεδομένου ότι είναι ενεργοποιημένος αυτός ο αντίπαλος. Αν δεν είναι ενεργοποιημένος τότε ενεργοποιείται. Ομοίως, αν η απόφαση είναι η αύξηση της ικανότητας ενός αντιπάλου τότε αυξάνουμε την τιμή της ικανότητας του ή τον ενεργοποιούμε αν δεν είναι ενεργοποιημένος. Επειδή η ενεργοποίηση δημιουργεί ραγδαία αλλαγή στη δομή που παίρνει το παιχνίδι επαναφέρουμε όλες τις μεταβλητές στις αρχικές τιμές τους και αρχίζουμε πάλι τη διαμόρφωση.

Για οποιαδήποτε απόφαση, είτε αύξηση είτε μείωση του επιπέδου δυσκολίας, διαμορφώνουμε αντίστοιχα και την πυκνότητα των κερμάτων. Μεγαλύτερη πυκνότητα κερμάτων σημαίνει ότι ο παίχτης θα μπορέσει να αυξήσει το σκορ του με μεγαλύτερο ρυθμό. Επομένως, αυτό το κάνουμε για να δώσουμε μεγαλύτερες ανταμοιβές κατά τη διάρκεια του παιχνιδιού σε έμπειρους παίχτες και να τιμωρήσουμε τους άπειρους παίχτες που καθυστερούν. Επίσης τα κέρματα εκτός από την αύξηση του σκορ που προσφέρουν, προσθέτουν και μερικά δευτερόλεπτα στο χρονόμετρο του παιχνιδιού, το οποίο εξηγείται στην επόμενη ενότητα.

Χρονόμετρο

Το χρονόμετρο, όπως είδαμε στο διάγραμμα αποφάσεων, αποτελεί τον βασικό παράγοντα στην απόφαση αύξησης ή μείωσης της δυσκολίας. Σε κάθε αρχικό επίπεδο δυσκολίας το παιχνίδι ξεκινάει με ένα συγκεκριμένο αριθμό δευτερολέπτων στο χρονόμετρο. Ο λόγος που σχεδιάστηκε με αυτό το τρόπο είναι ότι στα μεγαλύτερα επίπεδα δυσκολίας ο παίχτης εκτός απ’την αποφυγή των αντιπάλων έχει να αντιμετωπίσει και το πρόβλημα της εξερεύνησης του χάρτη. Επομένως, δεν μπορούμε να χρησιμοποιήσουμε την ίδια αρχική τιμή χρονομέτρου για τα Very Easy και Very Hard παιχνίδια γιατί διαφέρουν και στην έκταση τους και στην δυσκολία αντιμετώπισής τους.

Ο παίχτης, μαζεύοντας όσα περισσότερα κέρματα μπορεί, πρέπει να φτάσει στο στόχο το γρηγορότερο δυνατό. Κάθε φορά που ο παίχτης ολοκληρώνει ένα επίπεδο προσθέτονται στο χρονόμετρο μερικά δευτερόλεπτα ανάλογα με την αρχική δυσκολία. Αν ο παίχτης καταφέρει να διασχίσει την απαραίτητη διαδρομή για να φτάσει στο στόχο αρκετά γρήγορα, τότε το χρονόμετρο έχει μεγαλύτερη τιμή από τη προηγούμενη τιμή του και επομένως αυξάνεται το επίπεδο δυσκολίας. Εναλλακτικά, αν ο παίχτης είναι αργός και το χρονόμετρο έχει μικρότερη τιμή από τη προηγούμενη, το παιχνίδι γίνεται πιο εύκολο.

Παρακάτω φαίνονται αναλυτικά οι αρχικές τιμές του χρονομέτρου και ο αριθμός των δευτερολέπτων που προστίθενται από τα κέρματα ανάλογα το επίπεδο δυσκολίας.

Επίπεδο δυσκολίας	**Αρχική τιμή χρονομέτρου**	**Δευτερόλεπτα που προσθέτονται από κάθε κέρμα**
VeryEasy	30	1
Easy	60	2
Medium	90	3
Hard	120	4
VeryHard	180	5

Εναλλακτικές μέθοδοι

Μία εναλλακτική μέθοδος απόφασης διαμόρφωσης δυσκολίας στην αρχή του project ήταν ο υπολογισμός ενός σκορ μέσω του συνδυασμού του αριθμού επιπέδου(Level) με τον αριθμό των θανάτων. Ο αριθμός του επιπέδου προσέφερε θετικά στο συνολικό σκορ ενώ ο αριθμός των θανάτων αφαιρούσε από το συνολικό σκορ όπως φαίνεται στην παρακάτω εντολή

```
score = score + lvlCount-deathCount;
```

Με την μέθοδο αυτή, όμως, υπάρχουν περιστατικά όπου ο παίχτης τιμωρείται με πολύ απότομο τρόπο και δεν μπορεί να επανέλθει με βάση την επίδοση του. Για παράδειγμα, αν σε κάποιο παιχνίδι ένα σημείο τον δυσκόλευε αρκετά και τον ανάγκασε να πεθάνει δέκα φορές, ενώ βρίσκεται στο δεύτερο επίπεδο, τότε για να επανέλθει στην αρχική του κατάσταση θα χρειαζόταν να διασχίσει τουλάχιστον 17 επίπεδα χωρίς να πεθάνει, δηλαδή θα έπρεπε να φτάσει στο επίπεδο 19. Αυτό φαίνεται αναλυτικά στον παρακάτω πίνακα

Score	**Level**	**Deaths**
0	1	0
1	2	10
-7	3	10
-14	4	10
-20	5	10
-25	6	10
-29	7	10
-32	8	10
-34	9	10
-35	10	10
-35	11	10
-34	12	10
-32	13	10
-29	14	10
-25	15	10
-20	16	10
-14	17	10
-7	18	10
1	19	10

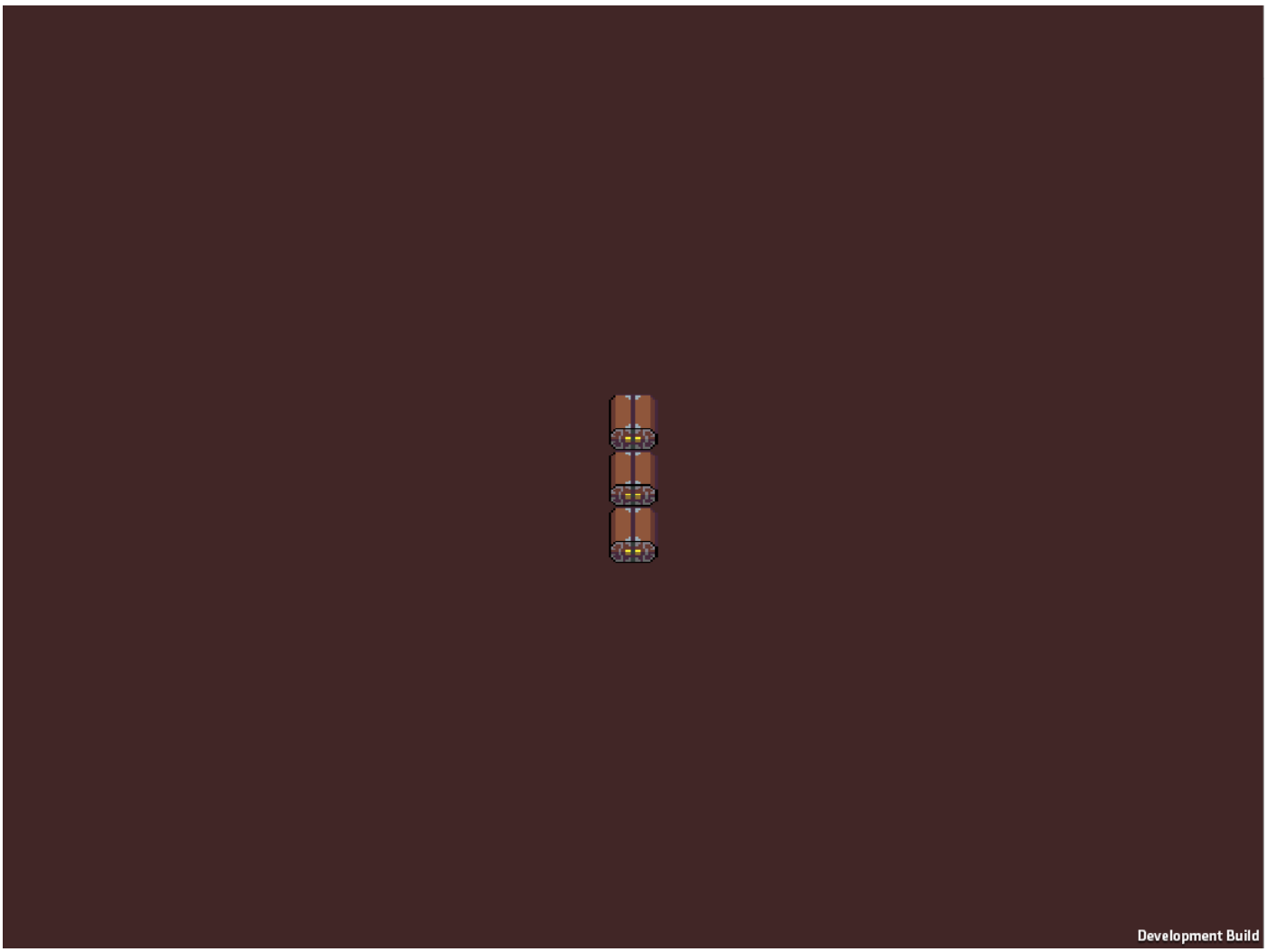
Έπειτα εξετάστηκε το ενδεχόμενο μηδενισμού του αριθμού των θανάτων σε κάθε επίπεδο, ώστε να ισορροπεί με τον αριθμό επιπέδου. Όμως, και η προσέγγιση αυτή κατέληξε να μην είναι τόσο χρήσιμη, καθώς το επίπεδο δυσκολίας εξαρτάται από τη τυχαία δόμηση του χάρτη πολύ περισσότερο από την επίδοση του παίχτη. Η παρουσία των αδιεξόδων στα μεγαλύτερα επίπεδα δυσκολίας σημαίνει ότι ο παίχτης θα χρειαστεί να διασχίσει μονοπάτια που δεν καταλήγουν στον στόχο, παρ'όλα αυτά όμως τον δυσκολεύουν και πιθανότατα προσθέτουν στον αριθμό των θανάτων του. Συνεπώς, ο παίχτης τιμωρείται για την εξερεύνηση του χάρτη, ενώ αυτή δεν αποτελεί μέρος των ικανοτήτων του. Συνεπώς, κατέληξα στο συμπέρασμα ότι ο αριθμός των θανάτων σε σύγκριση με τον αριθμό επιπέδου, δεν είναι κατάλληλος συνδυασμός για την διαμόρφωση επιπέδου δυσκολίας.

Μία δεύτερη εναλλακτική είναι η χρήση του ποσοστού των κερμάτων που κατάφερε να μαζέψει ο παίχτης μέσα στο επίπεδο. Με βάση τον συνολικό αριθμό κερμάτων, το επίπεδο δυσκολίας αυξάνεται αν ο παίχτης καταφέρει να μαζέψει παραπάνω από τα μισά κέρματα που υπάρχουν στον χάρτη, και μειώνεται αν μαζέψει λιγότερα από τα μισά. Το πρόβλημα όμως με την εναλλακτική αυτή είναι ότι δεν έχουμε κάποιον περιορισμό ώστε να ολοκληρώνεται το παιχνίδι, και ο παίχτης μπορεί να προχωράει με αργό ρυθμό χωρίς να χρειάζεται να ρισκάρει.

Αυτό που θα μπορούσαμε να κάνουμε για την επίλυση του παραπάνω προβλήματος είναι να συνδυάσουμε το ποσοστό κερμάτων με τη μέθοδο με το χρονόμετρο και να υπολογίζουμε ένα σκορ το οποίο καθορίζει τη διαμόρφωση της δυσκολίας.

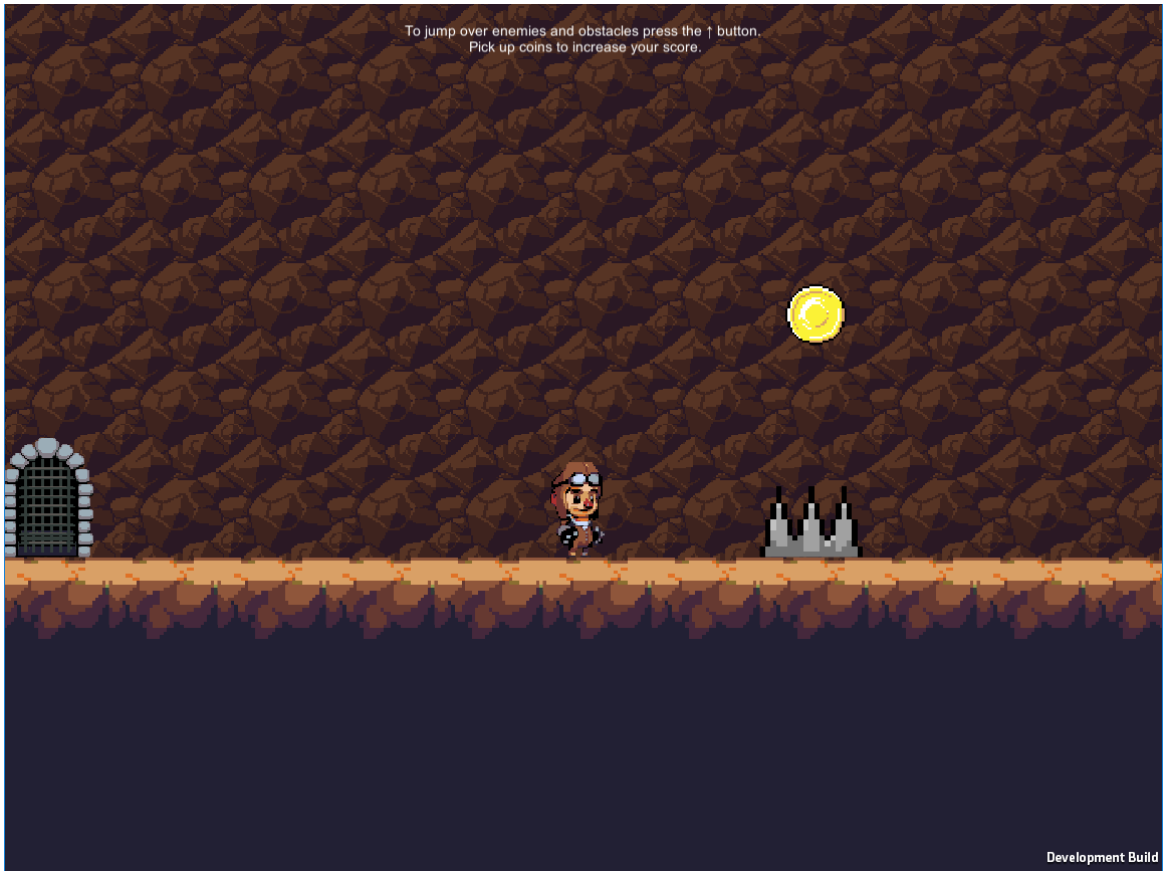
4.6. Λειτουργία(Εγχειρίδιο Χρήσης)

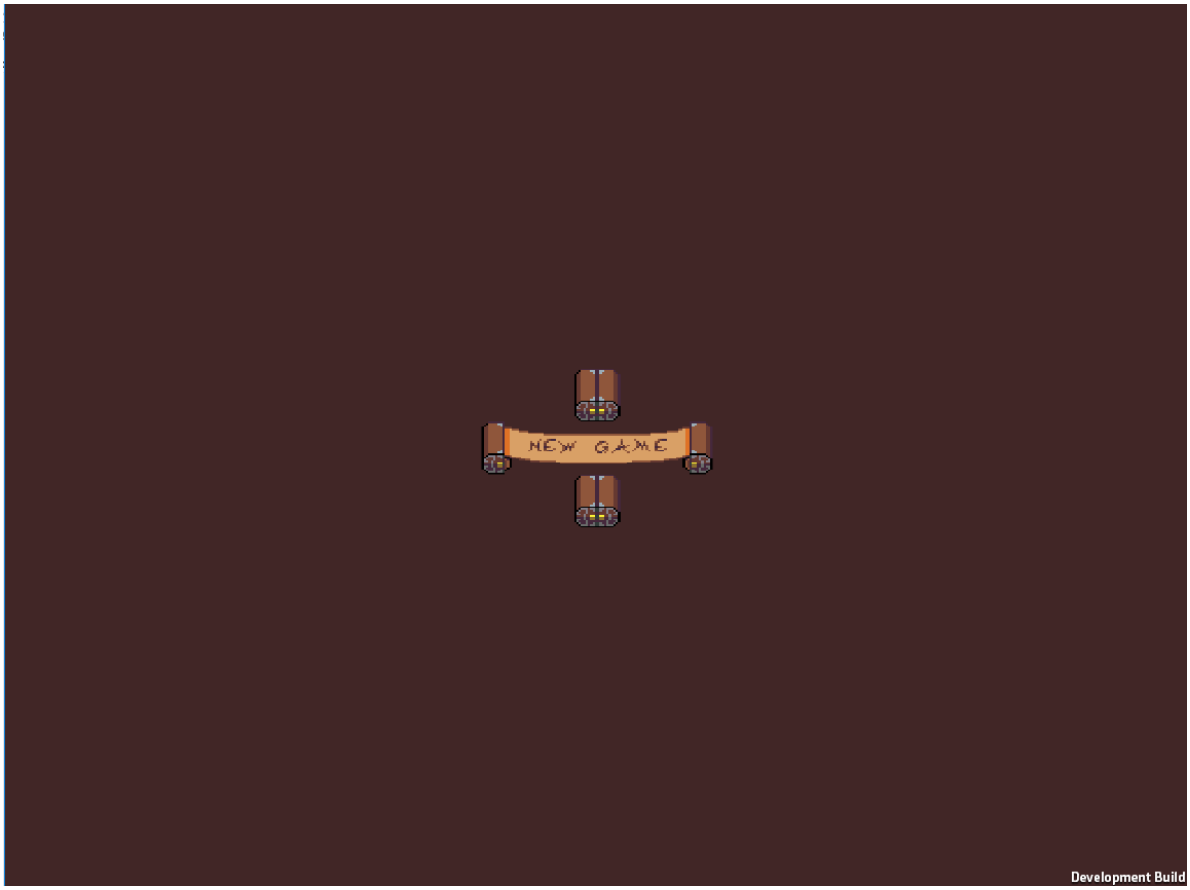
Ανοίγοντας το εκτελέσιμο αρχείο ξεκινάει το παιχνίδι και μας εμφανίζεται η αρχική οθόνη



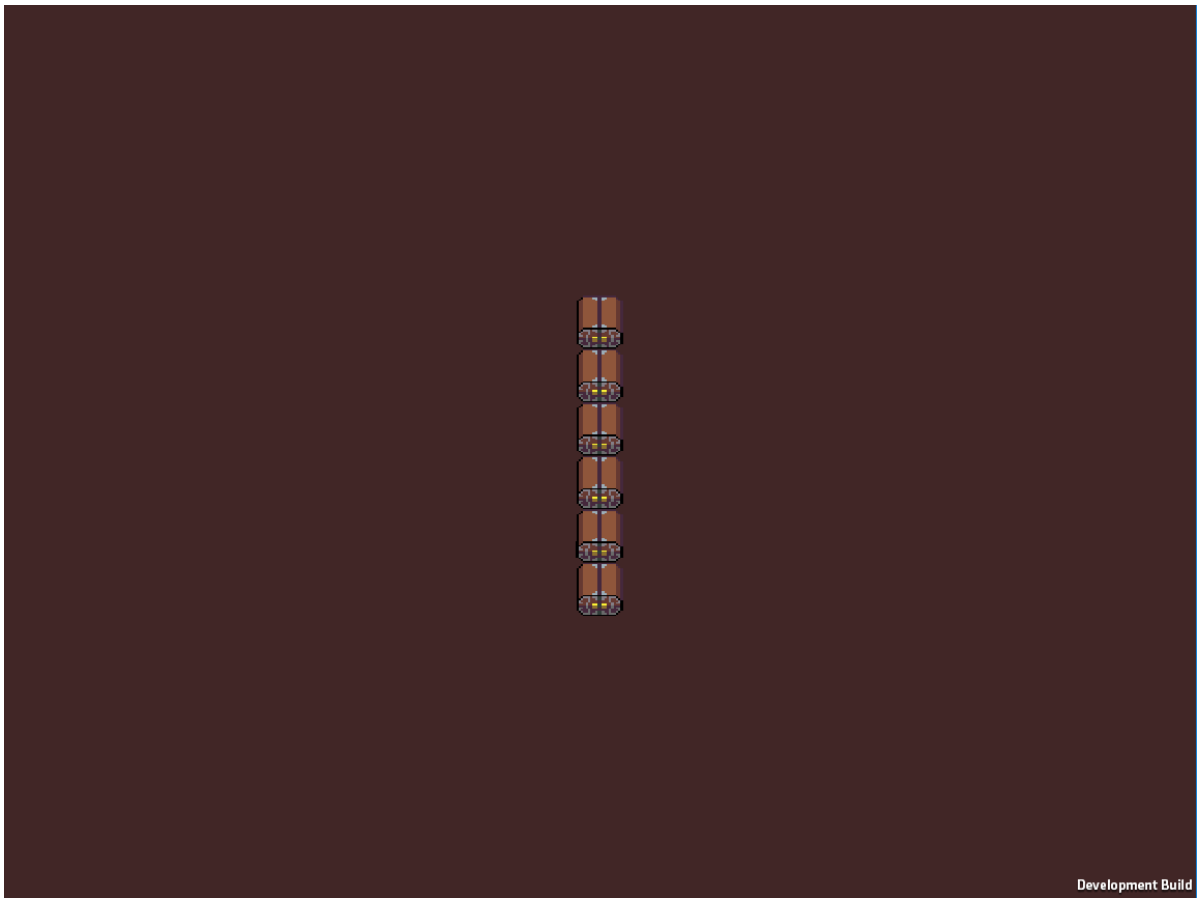
Όταν περνάμε το ποντίκι από πάνω ανοίγει ο πάπυρος και μας δείχνει την επιλογή

Το Tutorial αποτελεί μία σειρά προσχεδιασμένων επιπέδων τα οποία εξηγούν τις βασικές λειτουργίες και κανόνες του παιχνιδιού στον παίχτη.

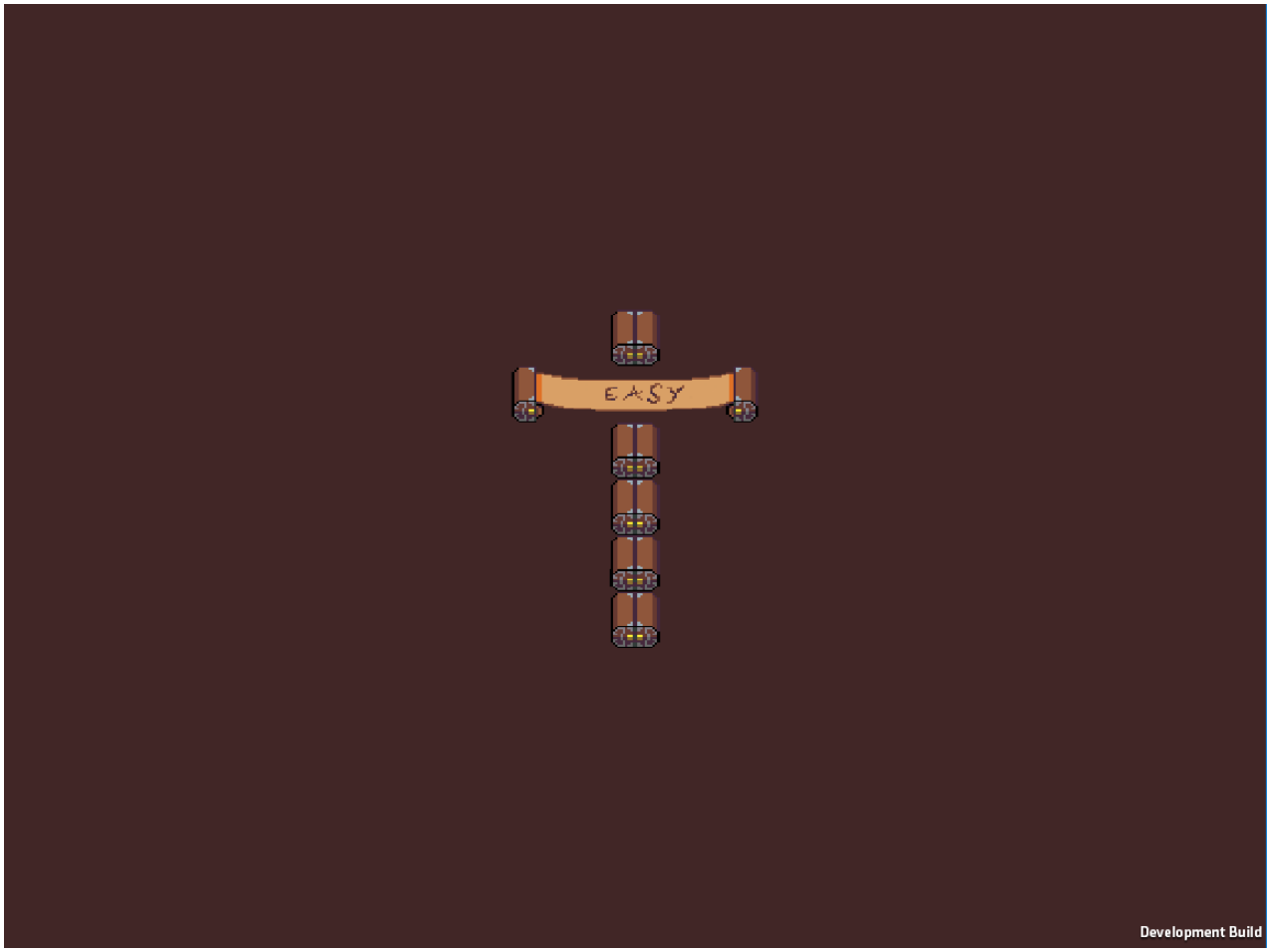




Μόλις πατάμε στο New Game μεταφερόμαστε στο μενού επιλογής επιπέδου δυσκολίας που φαίνεται παρακάτω

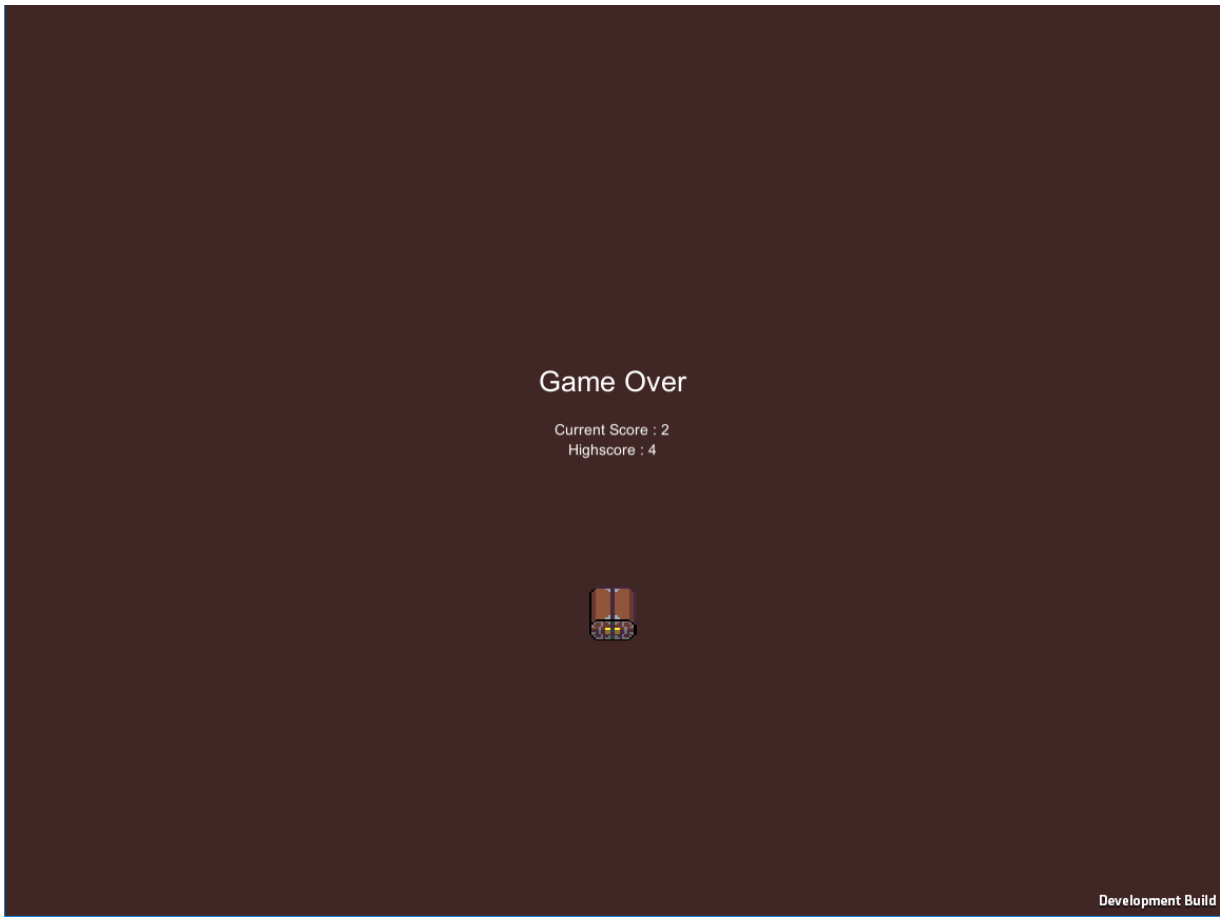


Επιλέγουμε δυσκολία και το παιχνίδι ξεκινάει στο αντίστοιχο επίπεδο δυσκολίας

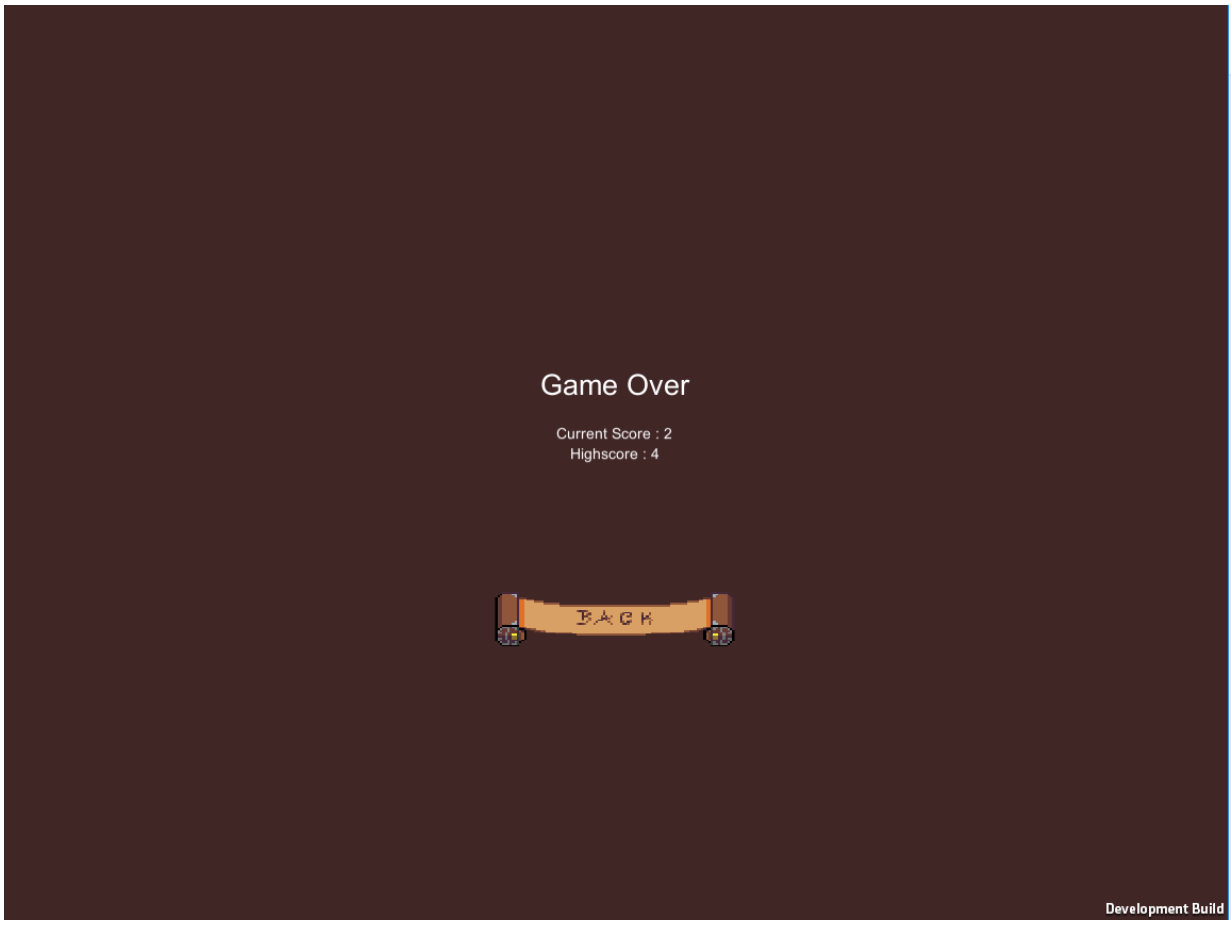

EASY
Development Build


Time Left : 36
0
Development Build

Μόλις ο χρόνος φτάσει στο 0 το παιχνίδι τελειώνει και εμφανίζεται η οθόνη Game Over



Πατάμε στο κουμπί Back και επιστρέφουμε στο αρχικό μενού



Τέλος για να κλείσει το παιχνίδι πατάμε στο κουμπί Exit

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

Στα προηγούμενα κεφάλαια παρουσιάστηκαν, εν συντομία, ο τρόπος με τον οποίο λειτουργεί το σύστημα δυναμικής διαμόρφωσης επιπέδου δυσκολίας σε συνδυασμό με ένα σύστημα διαδικαστικής παραγωγής περιεχομένου.

Με περισσότερες λεπτομέρειες, διατυπώθηκαν τα συστήματα σε θεωρητικό πλαίσιο ώστε να αντιληφθεί ο αναγνώστης το υπόβαθρο της παρούσας εργασίας. Έπειτα, παρουσιάστηκε το γενικό πλαίσιο του παιχνιδιού που συμπεριλαμβάνει τις διάφορες οντότητες και κανόνες που σχεδιάστηκαν σε αυτό και στη συνέχεια εμφανίστηκε το σύστημα διαδικαστικής παραγωγής περιεχομένου, όπου φάνηκαν οι μέθοδοι που χρησιμοποιήθηκαν και ο τρόπος με τον οποίο εκμεταλλεύτηκαν τα αποτελέσματα της κάθε μεθόδου. Στο τέλος, είδαμε πώς μπορούμε χρησιμοποιώντας τα δεδομένα του παιχνιδιού με τα δεδομένα της διαδικαστικής παραγωγής να δημιουργήσουμε ένα σύστημα προσαρμογής του επιπέδου δυσκολίας στο επίπεδο επίδοσης του παίχτη.

5.1. Στόχοι της εργασίας

Οι στόχοι της εργασίας δεν ήταν πλήρως ξεκάθαροι από την αρχή καθώς δεν γνώριζα τι θα χρειαστεί στην πορεία. Η εργασία ξεκίνησε με βασική προοπτική τον σχεδιασμό ενός συστήματος δυναμικής διαμόρφωσης δυσκολίας εφαρμοσμένο σε ένα σύστημα διαδικαστικής παραγωγής περιεχομένου σε ένα βιντεοπαιχνίδι. Για να συμβεί αυτό έπρεπε αρχικά να οριστεί η κατηγορία (genre) του παιχνιδιού και να σχεδιαστούν οι αντίστοιχοι μηχανισμοί, δηλαδή η κάμερα, ο χαρακτήρας παίχτη, οι αντίπαλοι και τα εμπόδια, οι συμπεριφορές τους, το έδαφος και οι αλληλεπιδράσεις αυτών. Εφόσον οι μηχανισμοί ήταν έτοιμοι θα έπρεπε να εφαρμοστεί το σύστημα παραγωγής περιεχομένου, να βρεθεί ένας τρόπος με τον οποίο θα τοποθετούνται μέσα στο επίπεδο τα αντικείμενα καθώς και να οριστούν οι παράμετροι που θα χρησιμοποιηθούν αργότερα στο σύστημα δυναμικής διαμόρφωσης δυσκολίας. Στο σημείο αυτό δημιουργήθηκε η διεπαφή χρήστη και το menu ώστε να υπάρχει μία δομή, καθώς και το Tutorial για την προπαίδευση του παίχτη. Τέλος, έπρεπε να σχεδιαστεί το σύστημα δυναμικής διαμόρφωσης δυσκολίας, ενώ παράλληλα να εφαρμόζονται τα γραφικά και τα κινούμενα σχέδια στους χαρακτήρες και πιθανότατα τα ηχητικά και οπτικά εφέ. Υπάρχουν διάφορες προσεγγίσεις διαμόρφωσης δυσκολίας που μπορούσαν να χρησιμοποιηθούν, όπως ο χειρισμός παραμέτρων(Parameter manipulation), η τεχνική reinforcement learning της μηχανικής μάθησης, τεχνικές γενετικών αλγόριθμων εφαρμοσμένες σε ευφυείς πράκτορες και τα νευρωνικά δίκτυα. Λόγω της έλλειψης εμπειρίας και γνώσεων μου στις περισσότερες τεχνικές από αυτές, ο καταλληλότερος τρόπος αποδείχθηκε ότι είναι ο χειρισμός παραμέτρων και χρησιμοποιήθηκε αυτός.

5.2. Αποτελέσματα και δυσκολίες που αντιμετωπίστηκαν

Πολλοί στόχοι της εργασίας επιτεύχθηκαν σε αρκετά ικανοποιητικό βαθμό, ωστόσο κάποιοι στόχοι έχουν χώρο βελτίωσης για ποικίλους λόγους. Αρχικά, σχεδιάστηκαν οι μηχανισμοί του παιχνιδιού, δηλαδή ο χαρακτήρας παίχτη, η κίνησή του, οι ικανότητές του, το έδαφος, το αρχικό και τελικό σημείο, το checkpoint, οι πέντε αντίπαλοι και οι συμπεριφορές τους. Στη συνέχεια εφαρμόστηκε το σύστημα διαδικαστικής παραγωγής, όπου χρειάστηκε να σχεδιαστούν η γεννήτρια του χάρτη MapGenerator, ο διαχειριστής επιπέδου LevelLayoutHandler και ο Instantiator ο οποίος δημιουργεί τα αντικείμενα στις αντίστοιχες θέσεις τους. Για να φτάσω στο σημείο αυτό πέρασα από διάφορα στάδια διαχωρισμού των κομματιών κώδικα του συστήματος διαδικαστικής παραγωγής, καθώς στην αρχή γίνονταν όλες οι διαδικασίες στο MapGenerator αντικείμενο. Αυτό όμως αποδείχθηκε χαοτικό όσον αφορά την οργάνωση του κώδικα και χρειάστηκε να διαχωριστεί σε τρία κομμάτια. Έπειτα δημιουργήθηκε ο διαχειριστής του παιχνιδιού GameManager ο οποίος θα ήταν υπεύθυνος για το σύστημα δυναμικής διαμόρφωσης του επιπέδου δυσκολίας και θα χρησιμοποιούσε τις παραμέτρους των προηγουμένων κομματιών για τη διαμόρφωση του παιχνιδιού. Στο σημείο αυτό αποφασίστηκε ότι για να γίνει η διαμόρφωση της δυσκολίας θα πρέπει με κάποιον τρόπο το παιχνίδι να ξεκινάει από μία αρχική κατάσταση. Για αυτόν τον λόγο σχεδιάστηκαν τα πέντε αρχικά επίπεδα δυσκολίας από τα οποία ο παίχτης χρειάζεται να επιλέξει ένα. Τέλος, εφόσον το project βρισκόταν στο τελικό στάδιο, είχα τη δυνατότητα να προσθέσω τα γραφικά και τα κινούμενα σχέδια των χαρακτήρων, διότι δεν επηρέαζαν τα υπόλοιπα συστήματα.

5.3. Συσχέτιση στόχων-αποτελεσμάτων

Στον παρακάτω πίνακα φαίνονται οι στόχοι στην αρχή και κατά τη διάρκεια του project σε σχέση με τα αποτελέσματα που επιτεύχθηκαν και βρίσκονται στον φάκελο Assets της εργασίας

Αρχικοί στόχοι	**Αποτελέσματα**
Game Mechanics	• Prefabs/LevelManager • Prefabs/Main Camera • Prefabs/Player • Prefabs/Second Camera • Resources/Checkpoint • Resources/Coin • Resources/Enemy_dropper • Resources/Enemy_flyer • Resources/Enemy_jumper • Resources/Enemy_walker • Resources/Goal • Resources/Ground_* • Resources/Spikes • Resources/Startpoint • Scenes/GameOver

	• Scenes/GameScene • Scenes/MainMenu • Scenes/MapOptions • Scenes/Tutorial • Scripts/CameraController • Scripts/Checkpoint • Scripts/Coin • Scripts/EnemyDropper • Scripts/EnemyDropper_DeathZone • Scripts/EnemyFlyer • Scripts/EnemyJumper • Scripts/EnemyWalker • Scripts/Goal • Scripts/KillEnemy • Scripts/KillPlayer • Scripts/LevelManager • Scripts/PlayerController • Scripts/TutorialGoal
Procedural Content Generation	• Prefabs/Instantiator • Resources/MapGenerator • Scripts/Instantiator • Scripts/LevelLayoutHandler • Scripts/MapGenerator
Dynamic Difficulty Adjustment	• Prefabs/GameManager • Scripts/GameManager
Graphics and Visual Effects	• Graphics/Animations • Graphics/Animators • Graphics/Sprites • Prefabs/Death Particle • Prefabs/Death Particle_dropper • Prefabs/Death Particle_flyer • Prefabs/Death Particle_jumper • Prefabs/Jump Particle • Prefabs/Respawn Particle • Resources/Background • Scripts/DestroyFinishedParticle
User Interface	• Scripts/UIAnimator • UI/EventSystem • UI/Game_Canvas • UI/GameOver_Canvas • UI/MainMenu_Canvas • UI/MapOptions_Canvas • UI/Tutorial_Canvas
Sounds and Sound Effects	-

*Τα αντικείμενα Ground_ είναι τα αντικείμενα εδάφους και συμπεριφέρονται με τον ίδιο τρόπο. Η μόνη διαφορά τους είναι τα γραφικά

5.4. Προοπτικές και αξιολόγηση εργασίας

Η εργασία έχει αρκετές προοπτικές βελτίωσης σε διάφορα χαρακτηριστικά. Καταρχάς, θα έπρεπε να σχεδιαστεί μία οθόνη φόρτωσης(loading screen) η οποία θα εμφανίζεται κάθε φορά που το παιχνίδι κάνει τους υπολογισμούς και φορτώνει τα αντικείμενα στη τρέχουσα σκηνή. Επίσης, ένα από τα χαρακτηριστικά με τα οποία δεν ασχολήθηκα καθόλου είναι ο σχεδιασμών ήχων και ηχητικών εφέ, διότι θεώρησα ότι δεν αποτελεί βασικό αντικείμενο της εργασίας και δεν έχω τις απαραίτητες γνώσεις για να τα σχεδιάσω. Ακόμα, το παιχνίδι θα γινόταν πιο διασκεδαστικό αν σχεδιάζονταν περισσότεροι αντίπαλοι με διαφορετικά χαρακτηριστικά και συμπεριφορές από τους προϋπάρχοντες, ωστόσο αυτό απαιτεί την ύπαρξη των κατάλληλων γραφικών. Επίσης, θα μπορούσε να αλλάξει ο τρόπος με τον οποίο ο παίχτης αποκτά τις ικανότητες που έχει, δηλαδή αντί να έχει εξ'αρχής όλες τις ικανότητές του να τις συλλέγει σιγά σιγά όσο προχωράνε τα επίπεδα. Ωστόσο, αυτό απαιτεί την εξισορρόπηση των ικανοτήτων με τα αντίστοιχα εμπόδια, διότι αν για παράδειγμα ο παίχτης δεν έχει την ικανότητα διπλού άλματος, δεν μπορεί να προσπεράσει τον αντίπαλο jumper, αφού αυτός μιμείται το άλμα του παίχτη. Ακόμα θα μπορούσαν να σχεδιαστούν περισσότερες ικανότητες όσον αφορά τον παίχτη ώστε να έχει περισσότερο ενδιαφέρον ο χειρισμός και οι κινήσεις του. Στη συνέχεια, το σύστημα δυναμικής διαμόρφωσης του επιπέδου δυσκολίας θα μπορούσε να υλοποιηθεί με διαφορετικές προσεγγίσεις (νευρωνικά δίκτυα, γενετικοί αλγόριθμοι,reinforcement learning) οι οποίες θα προσέφεραν εναλλακτικές λύσεις στο πρόβλημα αυτό. Ωστόσο, η χρήση της διαμόρφωσης παραμέτρων αποδείχθηκε απλή και αρκετά ικανοποιητική για τις ανάγκες της εργασίας. Ακόμα, το σύστημα δυναμικής διαμόρφωσης δυσκολίας θα ήταν πιο ισχυρό αν λάμβανε υπόψη και τα εμπόδια που δυσκολεύουν περισσότερο τον κάθε παίχτη σε κάθε επίπεδο. Συγκεκριμένα, το παιχνίδι στη κατάσταση που βρίσκεται επιλέγει μέσω μίας ισοπίθανης κατανομής τον αντίπαλο του οποίου θα αυξηθεί ή θα μειωθεί η πυκνότητά ή οι ικανότητες. Θα ήταν χρήσιμο, λοιπόν, αντί να έχουν οι αντίπαλοι την ίδια πιθανότητα να επιλεχθούν, να αλλάζουν οι πιθανότητες επιλογής τους με βάση το ποσοστό των θανάτων του παίχτη που προσέφεραν. Αυτό θα χρησίμευε διότι αν, για παράδειγμα, ένας παίχτης δυσκολεύεται με ένα συγκεκριμένο είδος αντιπάλων, τότε το παιχνίδι θα μείωνε πιο συχνά τη πυκνότητα του συγκεκριμένου είδους αντιπάλου και θα διαμορφωνόταν κατάλληλα. Τέλος, όπως είδαμε νωρίτερα ένα χαρακτηριστικό, που πιθανώς μπορεί να βελτιωθεί, είναι η αλλαγή του τρόπου όπου γίνεται η απόφαση διαμόρφωσης δυσκολίας σε μία μορφή συνδυασμού του χρονομέτρου με το ποσοστό των κερμάτων που μάζεψε ο παίχτης, όμως χρειάζεται περαιτέρω δοκιμές.

6. ΑΝΑΦΟΡΕΣ

[1] Adams, Ernest - *Fundamentals of Game Design* (3rd ed.)

[2] Thomas H. Apperley - *Genre and game studies: Toward a critical approach to video game genres*

[3] Shaker Noor, Togelius Julian, Nelson, Mark J. - *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*

[4] Andrew Adamatzky - *Game of Life Cellular Automata*

[5] Caetano Vieira Neto Segundo, Kennet Emerson Avelino Calixto, Rene Pereira de Gusmao - *Dynamic difficulty adjustment through parameter manipulation for Space Shooter game*

[6] Robin Hunicke, Vernell Chapman - *AI for Dynamic Difficulty Adjustment in Games*

[7] Robin Hunicke - *The case for dynamic difficulty adjustment in games*

[8] Yi-Na Li , Chi Yao , Dong-Jin Li , Kang Zhang - *Adaptive Difficulty Scales for Parkour Games*

[9] Aaron Burke - *Using Player Profiling to Enhance Dynamic Difficulty Adjustment in Video Games*

[10] Γραφικά Open Pixel Project http://www.openpixelproject.com/

[11] Video game https://en.wikipedia.org/wiki/Video_game

[12] Video game genres https://en.wikipedia.org/wiki/List_of_video_game_genres

[13] Platform game https://en.wikipedia.org/wiki/Platform_game

[14] Procedural Content Generation https://en.wikipedia.org/wiki/Procedural_generation

[15] Procedural Content Generation Wiki http://pcg.wikidot.com/

[16] Cellular Automaton https://en.wikipedia.org/wiki/Cellular_automaton

[17] Dynamic difficulty adjustment https://en.wikipedia.org/wiki/Dynamic_game_difficulty_balancing

[18] Spelunky https://en.wikipedia.org/wiki/Spelunky

[19] Spelunky Official Website http://www.spelunkyworld.com/

[20] Spelunky Map Generator http://tinysubversions.com/spelunkyGen/

[21] Cloudberry Kingdom https://en.wikipedia.org/wiki/Cloudberry_Kingdom

[22] Pwnee Studios Official Website http://www.pwnee.com/

[23] Unity Technologies. Unity 3D. http://unity3D.com/unity

[24] Unity Official Procedural Cave Generation Tutorial

https://unity3d.com/learn/tutorials/s/procedural-cave-generation-tutorial

[25] Singleton Pattern https://en.wikipedia.org/wiki/Singleton_pattern

[26] Decision Tree https://en.wikipedia.org/wiki/Decision_tree