

## Contents

<b>1</b>	<b>Τελική Εργασία: Τα σπήλαια του Χάους</b>	<b>2</b>
<b>2</b>	<b>Ιστορία, Rogue-like Games</b>	<b>3</b>
2.1	Περιγραφή του παιχνιδιού . . . . .	4
<b>3</b>	<b>Ο Χάρτης του παιχνιδιού</b>	<b>5</b>
3.1	Δημιουργία τυχαίου χάρτη . . . . .	5
3.1.1	Αντίπαλοι, Αντικείμενα και Παγίδες στα επίπεδα και δημιουργία επιπέδων . . . . .	7
3.2	Ορατότητα στον Χάρτη . . . . .	9
3.3	Αντικείμενα, παγίδες, παίκτης και αντίπαλοι στον χάρτη . . . . .	11
<b>4</b>	<b>Παίκτης, αντικείμενα και αντίπαλοι</b>	<b>11</b>
4.0.1	Πόντοι εμπειρίας / Experience points . . . . .	12
4.1	Χαρακτήρες . . . . .	12
4.1.1	Κίνηση . . . . .	12
4.1.2	”Ξεκούραση” . . . . .	12
4.1.3	Εξέλιξη χαρακτήρων . . . . .	13
4.1.4	Ειδικά για την Wizard class . . . . .	13
4.1.5	Σύνοψη χαρακτηριστικών . . . . .	14
4.2	Αντίπαλοι . . . . .	14
4.2.1	Συμπεριφορά αντιπάλων . . . . .	14
4.2.2	Παραδείγματα αντιπάλων . . . . .	15
4.2.3	”Ερπετό του Χάους” . . . . .	15
4.3	Αντικείμενα . . . . .	16
4.3.1	Consumables . . . . .	18
4.3.2	Όπλα . . . . .	18
4.3.3	Παγίδες . . . . .	18
4.4	Τοποθέτηση αντικειμένων στον χάρτη . . . . .	19
4.5	Τοποθέτηση αντιπάλων στον χάρτη . . . . .	19
4.5.1	Γεννήτριες αντιπάλων . . . . .	20
4.5.2	Αντικείμενα που ”ρίχνουν” οι αντίπαλοι . . . . .	20
4.6	Μάχες . . . . .	20
<b>5</b>	<b>Γραφικά, χειρισμός</b>	<b>21</b>
5.1	Εκτέλεση του παιχνιδιού . . . . .	21
5.2	User Interface . . . . .	22
5.2.1	Χάρτης . . . . .	22

5.2.2	Player Status . . . . .	22
5.2.3	Game Log . . . . .	22
5.3	Γραφικά . . . . .	24
5.3.1	Πλακίδια του χάρτη . . . . .	24
5.3.2	Χαρακτήρες του παίκτη και αντίπαλοι . . . . .	24
5.3.3	Αντικείμενα . . . . .	25
5.4	Χειρισμός . . . . .	25
<b>6</b>	<b>Γενικές συμβουλές</b>	<b>28</b>
6.1	Ενδεικτικό διάγραμμα κλάσεων / Οργάνωση του κώδικα . . . . .	28
<b>7</b>	<b>Αναφορές</b>	<b>29</b>
7.1	Procedural Dungeon Generation . . . . .	29
7.2	Line of Sight, Visibility . . . . .	30
7.3	AI . . . . .	30
7.3.1	Διάφορα . . . . .	30

## 1 Τελική Εργασία: Τα σπήλαια του Χάους

Διευκρινίσεις, όσον αφορά τις απαιτήσεις της εργασίας και τον τρόπο που θα βαθμολογηθεί:

1. Το πιο σημαντικό είναι η προσέγγιση του προβλήματος με μεθόδους του Αντικειμενοστραφούς Προγραμματισμού, δηλαδή, χρησιμοποιήστε όσες και όποιες τεχνικές είδαμε κατά τη διάρκεια της χρονιάς για να προσεγγίσετε την πολυπλοκότητα του προγράμματος (Abstract classes, interfaces, inheritance)
2. Η εργασία που θα παραδώσετε θα πρέπει να μπορεί να γίνει compile και να εκτελεστεί (ακόμα και αν έχει λάθη / bugs / ατέλειες) ώστε να βαθμολογηθεί. Μπορείτε να χρησιμοποιήσετε όποια έκδοση της Java επιθυμείτε. Ωστόσο, από την έκδοση 9 και μετά η γλώσσα παρέχει λειτουργικότητα που θα σας διευκολύνει σημαντικά. Παραδώστε την εργασία, ακόμα και αν δεν έχετε υλοποιήσει όλες τις λειτουργίες που περιγράφονται.
3. Δώστε προσοχή στην εκφώνηση (και ρωτήστε όπου έχετε απορία), ώστε να μη χρειαστεί να λύσετε ένα αρκετά πιο δύσκολο πρόβλημα από αυτό της εργασίας (π.χ. ο χρόνος, στην εργασία, είναι διακριτός και "ορίζεται" από τις επιλογές / δράσεις του παίκτη, αν προσπαθήσετε να προσεγγίσετε το πρόβλημα σε συνεχή χρόνο, είναι αρκετά πιο δύσκολο).
4. Χρησιμοποιήστε τόσο τα Java streams, όσο και τα utility methods στις συλλογές της Java, `List.of`, `Map.of`, ώστε να εξοικειωθείτε με αυτά και να ελαττώσετε

τον κώδικα που θα χρειαστεί να γράψετε. Για παράδειγμα `for` που επιλέγει κάποια στοιχεία από μια συλλογή, μπορεί να αντικατασταθεί με ένα `stream` της μορφής `aList.stream().filter(...).collect(Collectors.toList())` και ένα κατάλληλο `lambda expression` στη μέθοδο `filter`.

5. Αναλύστε το πρόβλημα πριν ξεκινήσετε να γράφετε, και αν στην πορεία διαπιστώσετε ότι κάποια κλάση / σχεδιαστική απόφαση σας δυσκολεύει, μη διστάσετε να την αλλάξετε. Σκοπός είναι, εκτός των άλλων, να εξασκηθείτε.
6. Για την εργασία δεν απαιτείται η χρήση `Threads`, αλλά ούτε και απαγορεύεται. Επίσης μπορείτε να χρησιμοποιήσετε `Design Patterns` (τα `Observer`, `Decorator`, `Composite` θα μπορούσαν να σας διευκολύνουν σε κάποιες περιπτώσεις). Ωστόσο, μην το θεωρήσετε υποχρεωτικό.
7. Χρησιμοποιήστε ένα καλό περιβάλλον προγραμματισμού, ώστε να μπορείτε να αυτοματοποιήσετε κάποιες εργασίες (δημιουργία `getters` και `setters`, δημιουργία των `hash` και `equals`, `toString` κλπ).
8. Σε πάρα πολλά σημεία προτείνονται κάποιες προσεγγίσεις, αλλά η επιλογή αφήνεται σε εσάς. Στα σημεία αυτά (αλλά και στο παιχνίδι, γενικότερα) μπορείτε να ακολουθήσετε όποια προσέγγιση θέλετε.
9. Τέλος, υπάρχουν πάρα πολλοί τρόποι με τους οποίους μπορείτε να μοντελοποιήσετε το παιχνίδι. Ένα αρκετά καλό (αλλά όχι και το μοναδικό) κριτήριο για να δείτε κατά πόσο η προσέγγιση που έχετε χρησιμοποιήσει είναι καλή, είναι να δείτε πόσο εύκολα μπορείτε να προσθέσετε νέες κατηγορίες παικτών, νέες δυνατότητες, νέους εχθρούς και αντικείμενα, κλπ.

Καλή επιτυχία!

## 2 Ιστορία, Rogue-like Games

Σαν παιχνίδια `rogue-like` χαρακτηρίζονται τα παιχνίδια ρόλων (`RPG games`), στα οποία ο παίκτης εξερευνάει επίπεδα ενός κόσμου (σπήλαιο, κάστρο, υπόγεια περάσματα, δάσος, συνδυασμοί αυτών, κλπ), τα οποία παράγονται τυχαία από τον υπολογιστή κατά το ξεκίνημα του παιχνιδιού. Ο παίκτης έχει μόνο μια "ζωή", δεν υπάρχει η δυνατότητα για αποθήκευση και επαναφορά και, γενικότερα, κάθε φορά, οι πίστες, ο κόσμος και οι αντίπαλοι διαφέρουν. Συνήθως, σκοπός είναι ο παίκτης να φτάσει σε κάποιο συγκεκριμένο επίπεδο και είτε να ανακτήσει κάποιο αντικείμενο, είτε να νικήσει κάποιον αντίπαλο, αν και σε πολλές περιπτώσεις χαρακτηρίζονται από πιο σύνθετο `gameplay` και σενάριο, το οποίο αποκαλύπτεται στον παίκτη καθώς

προχωράει στα επίπεδα. Η ονομασία roguelike προέρχεται από το παιχνίδι rogue (1980).

Τα κλασσικά roguelike παιχνίδια βασιζόντουσαν σε text / console interfaces, όπου οι παίκτες, αντίπαλοι, αντικείμενα, τοίχοι, κλπ αναπαρίστανται από γράμματα / χαρακτήρες ASCII (τα γραφικά εισήχθησαν αρκετά πιο μετά και στην αρχή ήταν απλά μια απευθείας "μετάφραση" των χαρακτήρων ASCII σε sprites). Ο τρόπος παιζίματος, ακολουθούσε (και ακολουθεί ακόμα, σε πολλές περιπτώσεις) τους περιορισμούς ενός τέτοιου interface, δηλαδή ο κόσμος του παιχνιδιού εξελίσσεται γύρω από τα actions του παίκτη. Το παιχνίδι περιμένει μέχρι ο παίκτης να αναλάβει κάποιο action, και μόνο τότε προχωράει και ο κόσμος. Θα μπορούσαμε να πούμε ότι το "ρολόι" του παιχνιδιού είναι το input του χρήστη / παίκτη.

## 2.1 Περιγραφή του παιχνιδιού

Οι υπήκοοι της αυλής του Χάους δεν αποκτούν ποτέ τις πλήρεις δυνάμεις τους, παρα μόνο όταν περιηγηθούν στον δαιδαλώδη λαβύρινθο σπηλαίων του "Logrus"<sup>1</sup> και αντιμετωπίσουν το ερπετό του Χάους, μια δοκιμασία που απαιτεί τεράστια προετοιμασία και ψυχικό σθένος. Σκοπός σας, ως ευγενής της αυλής του Χάους, είναι να περάσετε τη δοκιμασία, να κατακτήσετε τον Λαβύρινθο και το ερπετό του Χάους και να αγγίξετε το "Πετράδι της Κρίσης", το οποίο σύμφωνα με κάποιους είναι ένα από τα μάτια του ερπετού του Χάους.

Ο κόσμος του παιχνιδιού αποτελείται, λοιπόν, από σπήλαια τα οποία καταλαμβάνουν 10 επίπεδα. Κάθε σπήλαιο είναι στην πραγματικότητα ένας δισδιάστατος χώρος από πλακίδια, κάθε ένα από τα οποία μπορεί να είναι είτε "τοίχος" του σπηλαίου, είτε "πάτωμα", είτε, τέλος η έξοδος για το επόμενο ή για το προηγούμενο επίπεδο. Ο παίκτης και οι αντίπαλοι καταλαμβάνουν ένα πλακίδιο, και η κίνησή τους γίνεται μόνο μεταξύ γειτονικών πλακιδίων. Σαν γειτονιά ενός πλακιδίου ορίζονται οι 4 βασικές κατευθύνσεις (A, Δ, Β, Ν). Στα πλακίδια του σπηλαίου μπορεί να υπάρχουν **αντικείμενα** τα οποία άφησαν οι προηγούμενοι ευγενείς που προσπάθησαν να περάσουν τη δοκιμασία του Logrus ή και παγίδες που δημιουργούνται από τον Λαβύρινθο στην προσπάθειά του να δει αν είστε άξιοι των δυνάμεων του Χάους. Στο 10ο επίπεδο του λαβυρίνθου ο μοναδικός αντίπαλος είναι το "Serpent of Chaos", το οποίο και όταν ηττηθεί αφήνει στο πάτωμα το μοναδικό αντικείμενο "Jewel of Judgement", το οποίο είναι και ο τελικός σας στόχος.

Ο χρόνος στο παιχνίδι, όπως και τα κλασσικά roguelikes, εξελίσσεται γύρω από τις "δράσεις" (actions) του παίκτη. Δηλαδή, η κατάσταση του κόσμου αλλάζει μόνο όταν ο παίκτης εκτελεί κάποιο action, και μάλιστα, ο παίκτης έχει προτεραιότητα σε σχέση με οτιδήποτε άλλο στο παιχνίδι, δηλαδή πρώτα εκτελείται η δράση του

<sup>1</sup>[https://en.wikipedia.org/wiki/The\\_Pattern\\_and\\_the\\_Logrus](https://en.wikipedia.org/wiki/The_Pattern_and_the_Logrus)

παίκτη και μετά όλων των άλλων οντοτήτων.

Δυνατά actions του παίκτη είναι τα ακόλουθα (οι περισσότερες από τις παρακάτω έννοιες θα εξηγηθούν στη συνέχεια):

- Κίνηση (A/Δ/B/N): μπορείτε να χρησιμοποιήσετε τα πλήκτρα W, A, S, D
- Επίθεση στον κοντινότερο / γειτονικό αντίπαλο (βλ. σχετική παράγραφο): μπορείτε να χρησιμοποιήσετε το Space
- Χρήση κάποιου αντικειμένου: H για health potions, M για mana potions
- Ξεκούραση: R, κατά την οποία ο παίκτης ανακτά ένα ποσοστό των hit και mana points
- Αλλαγή όπλου: P. Αν ο παίκτης βρίσκεται σε ένα πλακίδιο, στο οποίο υπάρχει κάποιο όπλο, τότε ανταλλάζει το όπλο του με αυτό του πλακιδίου. Το όπλο που είχε ο παίκτης προηγουμένως στην κατοχή του παραμένει στο πλακίδιο.

### 3 Ο Χάρτης του παιχνιδιού

Στα rogue-like games ο χάρτης του παιχνιδιού δημιουργείται κατά τη διάρκεια του παιχνιδιού και είναι διαφορετικός κάθε φορά. Γενικότερα, οι χάρτες έχουν διάφορες θεματολογίες και περιλαμβάνουν από σπήλαια και υπόγεια κάστρων, δάση, πόλεις, μέχρι και ολόκληρους πλανήτες με πολλαπλά οικοσυστήματα / διαπλάσεις. Στο παιχνίδι που θα κατασκευάσουμε, θεωρούμε ότι ο παίκτης κινείται σε ένα σύμπλεγμα σπηλαίων / λαβυρίνθων και κατεβαίνει διαδοχικά επίπεδα μέχρι να γίνει αρκετά δυνατός ώστε να μπορέσει να νικήσει το ερπετό του Χάους και να αποκτήσει το πετράδι της Κρίσης.

#### 3.1 Δημιουργία τυχαίου χάρτη

Υπάρχει μια πληθώρα αλγορίθμων για τη δημιουργία ενός τυχαίου χάρτη. Γενικά η διαδικασία δημιουργίας game assets κατά την εκτέλεση του παιχνιδιού λέγεται Procedural Content Generation (PCG) και ποικίλει από τη δημιουργία περιοχών παιχνιδιού, μέχρι τη δημιουργία χαρακτήρων, γραφικών, μουσικής και στοιχείων σεναρίου. Οι αντίστοιχες παράγραφοι στις αναφορές δίνουν αρκετά Links για όποιον ενδιαφέρεται να δει κάποια πράγματα παραπάνω.

Στην περίπτωση του παιχνιδιού μας θα χρησιμοποιήσουμε έναν από τους πιο απλούς (στην υλοποίηση) αλγορίθμους για τη δημιουργία dungeons ενός rogue-like παιχνιδιού, και πιο συγκεκριμένα τον αλγόριθμο της "τυχαίας περιήγησης" (random-walk). Ο συγκεκριμένος αλγόριθμος είναι κατάλληλος για την κατασκευή

”σπηλαίων” και έχει χρησιμοποιηθεί σε πάρα πολλά rogue-like παιχνίδια. Έστω  $N \times M$  το πλέγμα του παιχνιδιού (στο οποίο κινείται ο παίκτης και οι αντίπαλοι) και  $percentage < 1$  το ποσοστό του χάρτη, το οποίο θέλουμε να είναι διαθέσιμο στον παίκτη. Ξεκινώντας από ένα τυχαίο ή προκαθορισμένο σημείο, ο αλγόριθμος εκτελεί μια τυχαία διαδρομή, μέχρι να καλύψει το ποσοστό αυτό. Πιο αναλυτικά, σε ψευδοκώδικά:

```
def makeMap(M, N, percentage):
    gameMap = array[M][N]
    # Γέμισμα του χάρτη με "τοιίχους"
    for xi in range(M):
    for yi in range(N):
        gameMap[xi][yi] = 'wall'

    curX = startX = randomPoint(N)
    curY = startY = randomPoint(M)

    filled = 1;

    gameMap[curX][curY] = 'start'

    while (filled / float(M*N)) < percentage:
    curX, curY = selectPointInRandomDirection()
    if gameMap[curX][curY] == 'wall':
        gameMap[curX][curY] = 'floor'
        filled += 1

    return gameMap, startX, startY
```

Για να είναι ολοκληρωμένος ο χάρτης, θα πρέπει να προσθέσουμε και την ”έξοδο”, την οποία, τυπικά, ονομάζουμε ”stairs” (σκάλες για το επόμενο επίπεδο), καθώς και αντιπάλους και αντικείμενα. Όσον αφορά την έξοδο, θα χρησιμοποιήσουμε την απλούστερη προσέγγιση, δηλαδή θα την τοποθετήσουμε στο πλακίδιο εδάφους που βρίσκεται μακρύτερα από το σημείο έναρξης.

Επειδή θέλουμε ο παίκτης να κινείται ελεύθερα μεταξύ των επιπέδων, το πλακίδιο ”εισόδου” για όλα τα επίπεδα (εκτός από το πρώτο), θα τον μεταφέρει στο προηγούμενο επίπεδο. Όταν δηλαδή το αποτέλεσμα της κίνησης του παίκτη τον φέρει στο πλακίδιο εισόδου, μεταφέρεται στο προηγούμενο επίπεδο. Αντίστοιχα, το πλακίδιο ”εξόδου” θα τον μεταφέρει στο επόμενο (εκτός αν βρίσκεται ήδη στο 10ο επίπεδο).

```
def place_exit(gameMap, startX, startY):
```

```

exitX = startX
exitY = startY
dist = 0

for xi in range(N):
for yi in range(M):
    new_dist = abs(xi - startX) + abs(yi - startY)
    if gameMap[xi][yi] == 'floor' and new_dist > dist:
        exitX = xi
        exitY = yi
        dist = new_dist

gameMap[xi][yi] = 'stairs'
return gameMap, exitX, exitY

```

Υπάρχουν, και πάλι πάρα πολλές προσεγγίσεις, ωστόσο, η συγκεκριμένη είναι από τις απλούστερες.



Figure 1: Περιοχή 80x40 με ποσοστό κάλυψης 0.45

### 3.1.1 Αντίπαλοι, Αντικείμενα και Παγίδες στα επίπεδα και δημιουργία επιπέδων

Όσον αφορά την τοποθέτηση των αντιπάλων και των αντικειμένων, θα τα δούμε στη συνέχεια. Τα βασικά πράγματα που πρέπει να αναφέρουμε είναι ότι:



Figure 2: Περιοχή 80x40 με ποσοστό κάλυψης 0.30 και εκκίνηση από το (4,4)

1. Όλα τα 10 επίπεδα δημιουργούνται στην αρχή του παιχνιδιού (και είτε διατηρούνται σε όποια δομή επιθυμείτε, είτε έχουν "αναφορά" (reference) το ένα στο άλλο (η δεύτερη προσέγγιση είναι μάλλον πιο συναφής με τον OOP, αλλά δεν είναι απαραίτητο να την ακολουθήσετε)
2. Κάθε φορά που φεύγει από ένα επίπεδο ο παίκτης οι αντίπαλοι (αλλά όχι και τα αντικείμενα) εξαφανίζονται. Αντίστοιχα, όταν μπαίνει σε ένα επίπεδο, οι αντίπαλοι στο επίπεδο αρχικοποιούνται εκ' νέου (με όποια από τις προσεγγίσεις που θα δούμε στη συνέχεια επιλέξετε).
3. Τα αντικείμενα παραμένουν στα επίπεδα καθ' όλη τη διάρκεια του παιχνιδιού
4. Το ερπετό του Χάους, δημιουργείται στην αρχή του παιχνιδιού και τοποθετείται στο 10ο επίπεδο. Σε αντίθεση με τους υπόλοιπους εχθρούς παραμένει εκεί καθ' όλη τη διάρκεια του παιχνιδιού. Όταν ο παίκτης φεύγει από το επίπεδο, τοποθετείται σε ένα τυχαίο πλακίδιο (εκτός από την είσοδο).
5. Το 10ο επίπεδο του λαβυρίνθου δεν έχει κάποια έξοδο. Ωστόσο, μόλις ο παίκτης νικήσει το ερπετό και "συλλέξει" το "Πετράδι της Κρίσης", θεωρούμε ότι ο παίκτης νίκησε.
6. Οι παγίδες, εφ' όσον δεν έχουν "ενεργοποιηθεί", παραμένουν στον χάρτη, όπως και τα αντικείμενα (μια παγίδα θα μπορούσε να είναι ένα αντικείμενο)



### 3.2 Ορατότητα στον Χάρτη

Παραδοσιακά, στα rogue-like games, ο χάρτης του παιχνιδιού δεν είναι, τις περισσότερες φορές, γνωστός εκ των προτέρων στον παίκτη και εμφανίζεται καθώς ο παίκτης εξερευνεί το περιβάλλον. Επιπλέον, η ορατότητα του παίκτη περιορίζεται, ανά πάσα στιγμή, από το περιβάλλον, έτσι ώστε να μπορεί να βλέπει μόνο προς την κατεύθυνση που είναι στραμμένος, και το "οπτικό του πεδίο" να περιορίζεται από τοίχους, εμπόδια, κλπ. Για μια γενική περιγραφή τεχνικών και αλγορίθμων, δείτε το 1, καθώς και το 2, το οποίο περιγράφει τους βασικότερους αλγορίθμους για υπολογισμό του visibility (ray-casting, shadow-casting, κλπ).

Στην περίπτωση μας, θα απλοποιήσουμε τις παραπάνω προσεγγίσεις και θα θεωρήσουμε τα ακόλουθα:

- Τα πλακίδια του χάρτη μπορούν να είναι
  - unknown, δηλαδή δεν έχουν ανοίξει ακόμα, ο παίκτης δεν έχει επισκεφτεί ακόμα το συγκεκριμένο σημείο του χάρτη
  - fogged, δηλαδή ο παίκτης έχει επισκεφτεί την περιοχή, αλλά το πλακίδιο βρίσκεται εκτός του οπτικού του πεδίου,
  - visible, βρίσκεται εντός του οπτικού του πεδίου.
- Ο παίκτης μπορεί να δει σε απόσταση 6 τετραγώνων από τη θέση που βρίσκεται, ανεξάρτητα από τοίχους, εμπόδια, κλπ. Ο υπολογισμός των πλακιδίων του χάρτη, τα οποία είναι ορατά, γίνεται με βάση την επόμενη συνάρτηση:

```
VISIBILITY_RADIUS = 6
```

```
def tile_visible(playerX, playerY, tileX, tileY):  
    return abs(playerX-tileX) + \  
           abs(playerY - tileY) < VISIBILITY_RADIUS
```

Εναλλακτικά μπορείτε να χρησιμοποιήσετε και Ευκλείδια απόσταση για τον υπολογισμό των ορατών πλακιδίων 4

Αρχικά, όλα τα πλακίδια είναι unknown, εκτός από την άμεση ορατή περιοχή στον χάρτη.

**Παρατήρηση:** η συγκεκριμένη προσέγγιση, συνεπάγεται ότι ο παίκτης θα μπορεί να βλέπει και πίσω από τοίχους. Αν θέλετε, μπορείτε να υλοποιήσετε κάποιον αλγόριθμο field of view – ωστόσο δεν είναι υποχρεωτικό.

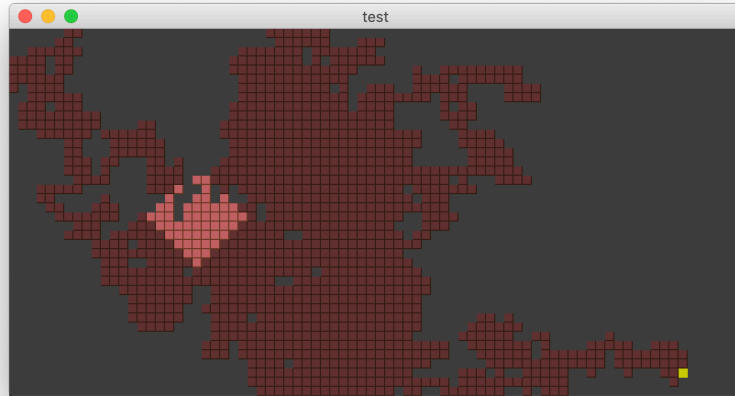


Figure 3: Παράδειγμα υπολογισμού ορατότητας με  $L_1$

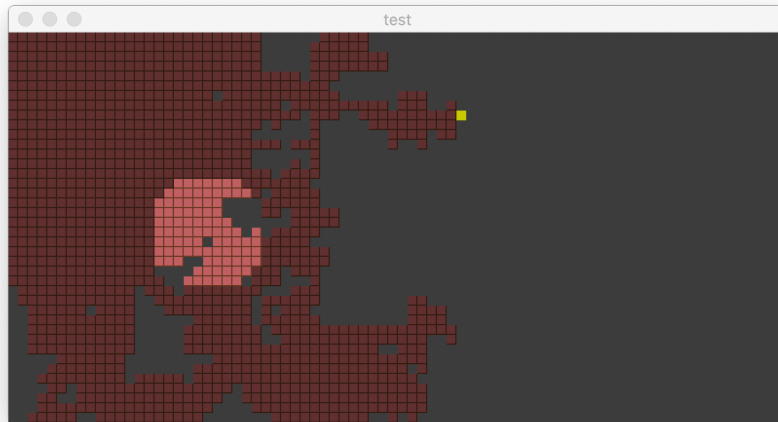


Figure 4: Παράδειγμα υπολογισμού ορατότητας με Ευκλείδεια απόσταση

### 3.3 Αντικείμενα, παγίδες, παίκτης και αντίπαλοι στον χάρτη

Ο παίκτης και κάθε αντίπαλος καταλαμβάνει αυστηρά ένα πλακίδιο του χάρτη από τα πλακίδια τύπου "εδάφους" / floor. Επειδή ο παίκτης, όπως θα δούμε, έχει προτεραιότητα στην κίνηση, απλά τον μετακινούμε στο πλακίδιο στην διεύθυνση που επέλεξε, αν αυτό, βέβαια είναι ελεύθερο και τύπου "έδαφος". Όσον αφορά τους αντιπάλους, η κίνηση είναι λίγο πιο σύνθετη, ιδίως αν είναι σε γειτονικά πλακίδια στην ίδια διεύθυνση και θέλουν να κινηθούν προς την διεύθυνση αυτή (αφού η τελική κατάσταση εξαρτάται από σειρά που θα τους κινήσουμε). Γενικά, μια καλή προσέγγιση είναι "πρώτα κινώ τους αντιπάλους που βρίσκονται πιο κοντά στον παίκτη".

Τα αντικείμενα του παιχνιδιού είναι τοποθετημένα στα πλακίδια, χωρίς όμως να τα καταλαμβάνουν, δηλαδή και ο παίκτης και οι αντίπαλοι μπορούν να μετακινηθούν σε πλακίδια που έχουν αντικείμενα. Επιπλέον, μόνο ο παίκτης μπορεί να αλληλεπιδράσει με τα αντικείμενα και τις παγίδες.

## 4 Παίκτης, αντικείμενα και αντίπαλοι

(Για αυτό το κομμάτι της εργασίας μπορείτε να δείτε τις εργασίες των προηγούμενων ετών που περιγράφουν κλάσεις και interfaces, ώστε να δείτε).

Οι χαρακτήρες που αναλαμβάνει να χειριστεί ο παίκτης στα παιχνίδια ρόλων περιγράφονται από ένα σύνολο αριθμών, οι οποίοι περιγράφουν είτε τα φυσικά χαρακτηριστικά του χαρακτήρα, είτε τη δυσκολία ή ευκολία με την οποία ο χαρακτήρας αυτός μπορεί να πραγματοποιήσει κάτι στο παιχνίδι. Το πλήθος και η πολυπλοκότητα των χαρακτηριστικών αυτών ποικίλλει ανάλογα με το είδος και την πολυπλοκότητα του παιχνιδιού. Για παράδειγμα, ένα τέτοιο σύνολο χαρακτηριστικών είναι τα strength, constitution, dexterity, intellect, wisdom και charisma. Στην περίπτωση μας θα χρησιμοποιήσουμε μόνο τα ακόλουθα από τα παραπάνω, και πάλι μόνο με τον απλούστερο τρόπο, πιο συγκεκριμένα:

- Hit Points: εκφράζουν το πόση "ζημια" (damage) μπορεί να αντέξει ο χαρακτήρας. Όταν φτάσουν στο 0, θα θεωρούμε ότι ο παίκτης έχασε.
- Strength: χαρακτηρίζει την δύναμη του χαρακτήρα και καθορίζει την "ζημιά" (damage) που κάνουν οι επιθέσεις του με όπλα
- Intellect: χαρακτηρίζει την νοητική δύναμη του παίκτη και καθορίζει το damage που κάνουν οι επιθέσεις του με τη χρήση μαγικών.

Στην πιο απλή περίπτωση το "βασικό" damage που κάνει ο χαρακτήρας του παίκτη είναι απ' ευθείας ανάλογο των χαρακτηριστικών αυτών και αυτή είναι και η προσέγγιση

που θα ακολουθήσουμε. Στο βασικό αυτό damage, προστίθενται επιπλέον πόντοι, οι οποίοι μπορεί να προκύπτουν από τα χαρακτηριστικά κάποιου αντικειμένου (βλ. παρακάτω).

Οι χαρακτήρες ενός τέτοιου παιχνιδιού κατατάσσονται σε κάποιες "κλάσεις", δηλαδή τύπου χαρακτήρων, οι οποίοι τύποι καθορίζουν και το πώς θα προσεγγίζει ο παίκτης / χαρακτήρας τις καταστάσεις του παιχνιδιού. Για παράδειγμα τυπικές κλάσεις είναι warrior, wizard, ranger, cleric, thief κλπ. Κάθε μια από τις κλάσεις αυτές επωφελείται διαφορετικά από τα παραπάνω χαρακτηριστικά και έχει και διαφορετικό τρόπο παιξίματος.

Το παιχνίδι θα υποστηρίξει δυο διαφορετικούς τύπους (κλάσεις) χαρακτήρων, με διαφορετικά χαρακτηριστικά. Έναν τύπο πολεμιστή (duelist) και έναν τύπο μάγου (caster). Ο πολεμιστής θα έχει περισσότερα hit points, δηλαδή θα μπορεί να δεχτεί περισσότερο damage, αλλά θα κάνει ταυτόχρονα λιγότερο damage στους αντιπάλους, ενώ ο μάγος θα έχει λιγότερα hit points και θα κάνει περισσότερο damage. Παράλληλα, ο μάγος θα έχει ένα επιπλέον resource, το "mana", το οποίο θα του επιτρέπει να κάνει μαγικά.

#### **4.0.1 Πόντοι εμπειρίας / Experience points**

Οι "πόντοι εμπειρίας" (experience points) είναι ένας τρόπος για να καταγράφεται η εξέλιξη ενός παίκτη στη διάρκεια του παιχνιδιού. Όσο ο παίκτης νικάει αντιπάλους και ξεπερνάει εμπόδια κερδίζει experience points (XP), και όταν συμπληρώνονται κάποια συγκεκριμένα XP προχωράει "επίπεδο", δηλαδή κερδίζει επιπλέον δυνατότητες και αυξάνει τα στατιστικά του. Θα χρησιμοποιήσουμε ένα απλοποιημένο μοντέλο, κατά το οποίο, όταν συμπληρώνονται κάποια XP και ο χαρακτήρας κερδίζει επίπεδο θα αυξάνονται τα στατιστικά του με έναν συγκεκριμένο τρόπο.

### **4.1 Χαρακτήρες**

#### **4.1.1 Κίνηση**

Οι χαρακτήρες μπορούν να κινηθούν στις 4 βασικές διευθύνσεις και μόνο σε τετράγωνα που είναι ελεύθερα.

#### **4.1.2 "Ξεκούραση"**

Οι χαρακτήρες μπορούν να χρησιμοποιήσουν το action "ξεκούραση" (rest), κατά το οποίο αναπληρώνουν ένα ποσο των χαμένων hit και mana points. Πιο συγκεκριμένα, για κάθε action κατά το οποίο κάνουν rest, αναπληρώνουν 5% των συνολικών τους HP και 5% των συνολικών MP. Για να μην είναι το παιχνίδι εξαιρετικά εύκολο (βλ. την παράγραφο "Αντίπαλοι" στη συνέχεια), μπορούμε (προαιρετικά πάντα) να

δώσουμε μια μικρή πιθανότητα (π.χ. 5%) για κάθε action τύπου rest, να εμφανιστεί ένας αντίπαλος εντός του οπτικού πεδίου του χαρακτήρα.

#### 4.1.3 Εξέλιξη χαρακτήρων

Οι πίνακες 1, 2, 3 περιγράφουν την εξέλιξη των χαρακτήρων, καθώς αυτοί αποκτούν εμπειρία και προχωράνε στο παιχνίδι.

Table 1: Σχέση XP / επιπέδων

Level	1	2	3	4	5	6
XP	0-299	300-899	900-2699	2700-6499	6500-13999	14000+

Table 2: Σχέση επιπέδων / στατιστικών για την κλάση "Duelist"

Level	Base HP	Base MP	Base Str	Base Int
1	30	-	10	-
2	60	-	20	-
3	80	-	25	-
4	90	-	30	-
5	100	-	35	-
6	140	-	45	-

Table 3: Σχέση επιπέδων / στατιστικών για την κλάση "Wizard"

Level	Base HP	Base MP	Base Str	Base Int
1	20	30	-	10
2	40	50	-	20
3	50	70	-	30
4	55	90	-	40
5	60	110	-	50
6	80	140	-	70

#### 4.1.4 Ειδικά για την Wizard class

Όπως είπαμε παραπάνω, η κλάση του μάγου έχει το resource "Mana", το οποίο χρησιμοποιεί για να εκτελεί μαγικά. Θα θεωρήσουμε ότι ο μάγος μας γνωρίζει ένα μόνο μαγικό "ξόρκι", το οποίο κοστίζει 5 Mana points, ανεξάρτητα επιπέδου. Μόλις τελειώσουν τα Mana Points, ο χαρακτήρας δεν μπορεί να εκτελέσει κάποιο

ξόρκι (δηλαδή να επιτεθεί), μέχρι να αναπληρώσει MP, είτε χρησιμοποιώντας κάποιο αντικείμενο, είτε με το να ξεκουραστεί.

#### 4.1.5 Σύνοψη χαρακτηριστικών

- Maximum Hit Points: αλλάζουν με το επίπεδο. Προκύπτουν σαν το άθροισμα των Base HP και τυχόν bonus από όπλα
- Hit Points: τα HP που έχει ανά πάσα στιγμή ο χαρακτήρας
- Maximum Mana Points: αντίστοιχα με τα Maximum Hit Points
- Mana Points: αντίστοιχα με τα Hit Points
- Strength, Intellect: αυξάνουν ανά επίπεδο, τροποποιούνται από αντικείμενα. Το damage είναι απ' ευθείας ανάλογο με αυτά.
- Damage: πόση ζημιά μπορεί να προκαλέσει ο χαρακτήρας

#### 4.2 Αντίπαλοι

Οι αντίπαλοι, μπορούν, εν δυνάμει, να έχουν όλα τα χαρακτηριστικά ενός παίκτη (strength, intellect, κλπ). Στην περίπτωση μας θα θεωρήσουμε ένα εξαιρετικά απλοποιημένο μοντέλο για τους αντιπάλους το οποίο θα αποτελείται από:

- όνομα
- Hit Points
- Damage
- πόσους πόντους εμπειρίας κερδίζει ο παίκτης όταν τους νικήσει
- ορατότητα: πόσο κοντά τους πρέπει να έρθει ο παίκτης ώστε να τον αντιληφθούν

##### 4.2.1 Συμπεριφορά αντιπάλων

Υπάρχουν πάρα πολλά άρθρα που περιγράφουν αλγορίθμους για τη συμπεριφορά των αντιπάλων σε ένα rogue-like (π.χ. περιπολίες, μετακίνηση μεταξύ σημείων ενδιαφέροντος, φύλαξη περιοχών, κλπ). Στην περίπτωση μας, κάθε τύπος αντιπάλου έχει μια ακτίνα (απόσταση) στην οποία θα αντιλαμβάνεται τον παίκτη. Μόλις ο παίκτης μπει εντός της ακτίνας αυτής, ο αντίπαλος θα κινείται προς τον παίκτη με σκοπό να του επιτεθεί (ακόμα και αν ο παίκτης δεν τον έχει δει ακόμα). Αν θέλετε, μπορείτε να τροποποιήσετε το παραπάνω, ώστε, π.χ. οι αντίπαλοι να κινούνται

τυχαία μέσα στο λαβύρινθο ή να φτιάχνετε ένα συγκεκριμένο path (ταυτόχρονα με τη δημιουργία του αντιπάλου), πάνω στο οποίο και θα μπορούν να κινηθούν.

- Όπως και στην περίπτωση του παίκτη, θα χρησιμοποιήσουμε μια απλοποιημένη προσέγγιση για την ορατότητα και επομένως οι αντίπαλοι αντιλαμβάνονται τον παίκτη ανεξάρτητα από το αν υπάρχουν τοίχοι ή άλλα εμπόδια στο ενδιάμεσο
- Η κίνηση είναι όπως του παίκτη, δηλαδή μπορούν να κινηθούν στις τέσσερις βασικές κατευθύνσεις
- Αν και υπάρχουν πολλοί αλγόριθμοι εύρεσης μονοπατιών (πως θα φτάσει ο αντίπαλος στον παίκτη), με γνωστότερο ίσως τον αλγόριθμο  $A^*$ , θα θεωρήσουμε ότι ο αντίπαλος κινείται έτσι ώστε να ελαχιστοποιήσει την απόσταση από τον παίκτη. Δεδομένου ότι η κίνηση γίνεται μόνο στις 4 βασικές διευθύνσεις, κάθε αντίπαλος θα πρέπει ανα πάσα στιγμή να επιλέγει ποιά απόσταση ( $\delta x$ ,  $\delta y$ ) θα ελαττώσει.
- Οι αντίπαλοι μπορούν να επιτεθούν στον παίκτη, μόνο όταν βρίσκονται σε γειτονικό τετράγωνο
- Το **DAMAGE** που κάνει κάθε αντίπαλος μπορεί να είναι είτε σταθερό, είτε να προκύπτει ρίχνοντας κάποια ζάρια και προσθέτοντας έναν αριθμό. Στον πίνακα 4 φαίνονται και οι δυο περιπτώσεις.
  - Όσον αφορά την περίπτωση με τα ζάρια, το "2d6+10" μεταφράζεται σαν "ρίχνω δυο ζάρια 6 πλευρών, προσθέτω το αποτέλεσμα και προσθέτω 10". Μπορείτε (προφανώς) να επαναχρησιμοποιήσετε την κλάση που δημιουργήσατε στην άσκηση 4.

#### 4.2.2 Παραδείγματα αντιπάλων

Οι κατηγορίες που φαίνονται στον πίνακα 4 είναι ενδεικτικές. Στην περίπτωση που θέλετε να προσθεσετε επιπλέον τύπους εχθρών, γενικά μια απλή και καλή τακτική είναι "λίγα HP, μεγάλο damage" ή "αρκετά HP, μικρό damage". Αν, τέλος, επιθυμείτε, μπορείτε να φτιάχνετε τους αντιπάλους τυχαία (βλ. αντίστοιχη παράγραφο για τα όπλα).

#### 4.2.3 "Ερπετό του Χάους"

Όπως είπαμε, στο τελευταίο επίπεδο κατοικεί η πεμπτουσία του λαβυρίνθου, το Ερπετό του Χάους, το οποίος και θα πρέπει να νικηθεί ώστε να τελειώσει το παιχνίδι. Σε αντίθεση με τους υπόλοιπους αντιπάλους, οι οποίοι εξαφανίζονται όταν αλλάζουμε

Table 4: Ενδεικτικός πίνακας αντιπάλων. Το "Player Level" αφορά τα επίπεδα του παίκτη κατά τα οποία εμφανίζεται ο συγκεκριμένος τύπος αντιπάλου

Player Level	Enemy Name	Hit Points	Damage	XP	Vis
1-2	Shadow Soldier	5	2	30	4
1-3	Ancient Guard	15	5	50	7
2-3	Fallen	30	8	80	2
3-4	Shade	40	10	100	6
3-5	Greater Shade	50	12	120	7
4-6	Chaos Knight	60	20	150	9
3-5	Shadow Serpent	20	4d6+10	100	4
5-6	Chaos Beast	80	2d6+10	200	5
5-6	Pattern Shade	50	3d6+10	400	10

επίπεδο, το Ερπετό παραμένει πάντα στο 10ο επίπεδο και μάλιστα διατηρεί τα χαρακτηριστικά του (Hit points), ώστε να μπορέσει ο ήρωας να τον νικήσει κάποια στιγμή (η τακτική είναι "επισκέπτομαι το 10ο επίπεδο, κάνω όσο damage μπορώ και φεύγω") Τα χαρακτηριστικά του τελικού αντιπάλου φαίνονται στον πίνακα 5. Όπως είπαμε και προηγουμένως, ενδεχεται να χρειαστούν παραπάνω από μια επισκέψεις στο 10ο επίπεδο μέχρι να νικηθεί το Ερπετό.

Στο επίπεδο μπορεί να εμφανιστούν και άλλοι αντίπαλοι, σύμφωνα με τους κανόνες που έχετε επιλέξει. Κάθε φορά που ο παίκτης φεύγει από το επίπεδο, οι υπόλοιποι αντίπαλοι εξαφανίζονται και ο Δράκος τοποθετείται σε τυχαίο σημείο.

Table 5: Χαρακτηριστικά του Ερπετού του Χάους (τελικός αντίπαλος)

Hit Points	Str	XP	Vis
1500	8d6+10	-	10

Όταν νικηθεί το Ερπετό του Χάους, αφήνει στο πλακίδιο όπου ηττήθηκε το "Πετράδι της Κρίσης". Το παιχνίδι ολοκληρώνεται όταν ο παίκτης πάρει στην κατοχή του το πετράδι, οπότε και μπορείτε να επιστρέψετε στην κονσόλα εκτυπώνοντας κάποιο μήνυμα.

### 4.3 Αντικείμενα

Τα αντικείμενα σε ένα παιχνίδι ρόλων (RPG) / roguelike μπορεί να καλύπτουν ένα τεράστιο εύρος λειτουργιών, ανάλογα με την πολυπλοκότητα του παιχνιδιού. Για παράδειγμα, κατηγορίες αντικειμένων μπορεί να είναι: Φαγητά (αν το παιχνίδι έχει παραμέτρους "πείνας"), Φάρμακα, αντίδοτα και φίλτρα (potions) για αναπλήρωση στατιστικών των χαρακτήρων, όπλα, πανοπλίες, κοσμήματα, εργαλεία επιδιόρθωσης,



μουσικά όργανα, εργαλεία επαγγελμάτων, κλπ. Ανεξάρτητα από την βασική τους χρήση, έχουν τη δυνατότητα, συνήθως, να τροποποιούν κάποια από τα βασικά χαρακτηριστικά / στατιστικά του χαρακτήρα, π.χ. strength, intellect, stamina, charisma, damage, hit points, mana points, κλπ. Αυτή η τροποποίηση μπορεί να διαρκεί όσο ο παίκτης χρησιμοποιεί ένα αντικείμενο (π.χ. όσο έχει εξοπλισμένο ένα σπαθί) είτε όταν καταναλώνει κάποιο αντικείμενο (π.χ. χρησιμοποιεί ένα potion). Τα μεν πρώτα ονομάζονται "equirables" τα δε δεύτερα consumables, ενώ, μπορεί να υπάρχουν και συνδυασμοί.

Το πως επηρεάζουν τα αντικείμενα τα χαρακτηριστικά του παίκτη έχει να κάνει με τα "item effects", τα οποία μπορούν να είναι διαφορετικών τύπων. Έτσι για παράδειγμα έχουμε effects, από αντικείμενα που μπορεί να φορέσει ή να χρησιμοποιήσει ο χαρακτήρας και effects που εφαρμόζονται μια φορά κατά τη χρήση ενός αντικειμένου (συνήθως consumable).

Θα έχουμε τριών ειδών αντικείμενα στο παιχνίδι, consumables, weapons και παγίδες. Όλα τα αντικείμενα θα φαίνονται στον χάρτη, εφ' όσον βρίσκονται σε περιοχή που έχει αποκαλυφθεί, σαν annotation στο πλακίδιο 5.

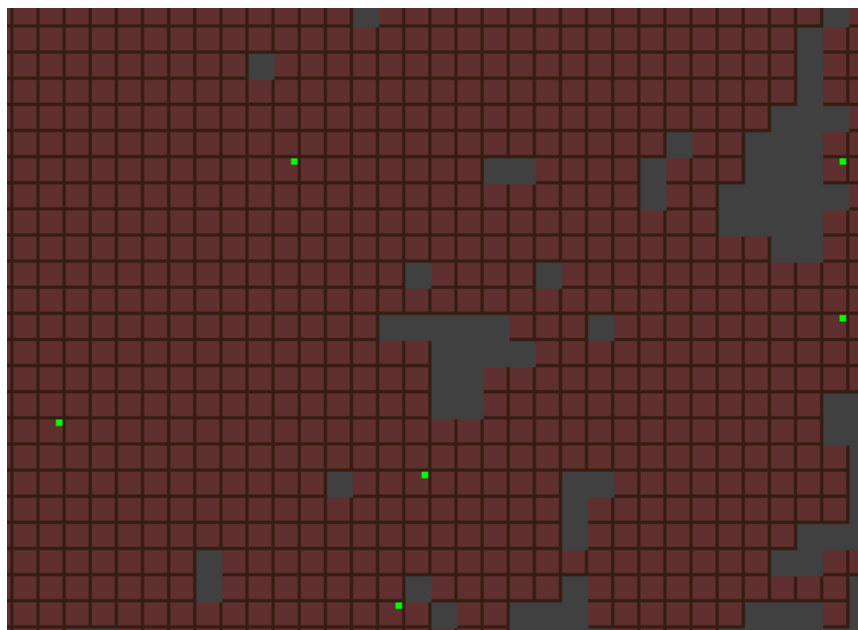


Figure 5: Περιοχή του χάρτη στην οποία φαίνονται πλακίδια με αντικείμενα

#### 4.3.1 Consumables

Consumables είναι τα αντικείμενα που χρησιμοποιεί ο παίκτης, τα οποία έχουν περιορισμένο αριθμό χρήσεων (σίγουρα περισσότερες από 1) και του αυξάνουν κάποιο από τα στατιστικά / χαρακτηριστικά αυτού. Σε σύνθετα rogue-like games υπάρχουν πολλών ειδών consumables και αντίστοιχα πολλών ειδών χαρακτηριστικά. Στην περίπτωση μας θα έχουμε 2 ειδών consumables:

- Health Potions: αυξάνουν το health του παίκτη κατά μια συγκεκριμένη ποσότητα, χωρίς να ξεπερνάει το Maximum Hit Points (προτείνεται 20 - 30HP ή κάποιο ποσοστό του συνολικού Health – αν θέλετε μπορείτε να έχετε και παραπάνω από έναν τύπο potion, π.χ. "Minor Health Potion", "Health Potion")
- Mana Potions: αυξάνουν τα mana points του παίκτη κατά ένα συγκεκριμένο ποσό (προτείνεται 15-25 MP, ισχύουν τα ίδια με τα Health Potions). Αν ο παίκτης δεν έχει mana points, τότε δεν έχουν κάποιο αποτέλεσμα.

Όταν ο παίκτης περάσει πάνω από πλακίδιο που περιέχει consumables, τότε τα μαζεύει όλα αυτόματα και τοποθετούνται στη λίστα με τα αντικείμενά του (inventory).

#### 4.3.2 Όπλα

Ο παίκτης μπορεί να έχει ένα weapon ανά πάσα στιγμή και όταν αλλάζει weapon (πατώντας το πλήκτρο P), τότε το όπλο που άφησε μένει στο πλακίδιο που βρίσκεται το προηγούμενο όπλο. Τα όπλα έχουν ενεργό item effect όσο τα χρησιμοποιεί ο χαρακτήρας. Τα effects αυτά αφορούν τα βασικά χαρακτηριστικά του παίκτη, δηλαδή Str, Int, HP, MP.

1. Πετράδι της Κρίσης Το πετράδι της κρίσης είναι equippable, δηλαδή κάτι που μπορεί να χρησιμοποιηθεί σαν όπλο. Δε μας ενδιαφέρουν τα στατιστικά του, αφού μόλις ο παίκτης το κάνει equip, το παιχνίδι ολοκληρώνεται.

#### 4.3.3 Παγίδες

Οι παγίδες είναι στην πραγματικότητα αντικείμενα (consumables), τα οποία γίνονται "use" μόλις ο παίκτης περάσει πάνω από το πλακίδιο και τα οποία απλά έχουν **αρνητικό** HP\_REPLENISH. Τη συγκεκριμένη συμπεριφορά μπορείτε να την μοντελοποιήσετε όπως θέλετε, είτε με έναν έλεγχο (για τον τύπο του αντικειμένου - instanceof) όταν το αποκτά ο χρήστης, είτε με κάποια ιδιότητα των αντικειμένων γενικότερα (π.χ. autoUse), είτε με οποιονδήποτε άλλο τρόπο.

#### 4.4 Τοποθέτηση αντικειμένων στον χάρτη

Όπως και τα περισσότερα πράγματα σε ένα roguelike, η τοποθέτηση αντικειμένων γίνεται τυχαία. Φροντίστε να έχετε κάποιον ικανοποιητικό αριθμό από health και mana potions σε κάθε επίπεδο, τα οποία να τοποθετούνται σε τυχαίες θέσεις στον χάρτη.

Όσον αφορά τα όπλα, μπορείτε είτε να έχετε έναν "προκατασκευασμένο" αριθμό όπλων και να επιλέγετε τυχαία ποιά και ποσά από αυτά θα τοποθετήσετε σε κάθε επίπεδο, είτε να δημιουργείτε τα όπλα τυχαία. Αν δυσκολευτείτε, κατασκευάστε απο πρίν έναν συγκεκριμένο αριθμό όπλων και τοποθετήστε τα στα επίπεδα.

Αν επιλέξετε την τυχαία δημιουργία όπλων, φτιάξτε μια απλή γεννήτρια, η οποία θα συνδυάζει ένα επίθετο για τον χαρακτηρισμό του όπλου (π.χ. "amazing", "ancient", "deadly", κλπ), ένα όνομα για τον τύπο του όπλου (εφ' όσον έχουμε 2 κλάσεις, θα μπορούσαν να είναι "Sword" και "Staff") και κάποια τυχαία bonus στα Strength, Intellect, Hit Points και Mana Points. Και πάλι κάποιοι καλοί κανόνες είναι "Strength + Hit Points = const", "Hit Points + Mana Points = const", κλπ, δηλαδή παίρνω 2 ή 3 χαρακτηριστικά, δίνω ένα συνολικό bonus (π.χ. 10) και μοιράζω το bonus στα χαρακτηριστικά αυτά. Καλό θα ήταν το bonus να αυξάνει όσο προχωράμε στο παιχνίδι (π.χ. συνολικό άθροισμα 10 για το πρώτο επίπεδο, 15 για το δεύτερο, κ.ο.κ.). Χρησιμοποιήστε όποια προσέγγιση σας φαίνεται πιο απλή.

Όπως αναφέραμε και στην αρχή, η δημιουργία των επιπέδων και η τοποθέτηση των αντικειμένων σε αυτά γίνεται μια φορά στην αρχή και τα αντικείμενα παραμένουν στις θέσεις τους, εκτός αν ο παίκτης ανταλλάξει το αντικείμενο που κρατάει με αυτό που βρίσκεται σε κάποιο πλακίδιο.

#### 4.5 Τοποθέτηση αντιπάλων στον χάρτη

Στις περισσότερες περιπτώσεις, η τοποθέτηση των αντιπάλων γίνεται κατά τη δημιουργία της πίστας. Το μοντέλο που θα ακολουθήσουμε είναι το εξής:

- Ανάλογα με τα HPs του παίκτη (όσο λιγότερα, τόσο μικρότερη η πιθανότητα) σε κάθε γύρο (κίνηση του παίκτη, όχι κατά τη διάρκεια της μάχης) τοποθετώ έναν αντίπαλο  $e$  στην θέση  $\max(vis(e), VISIBILITY\_RADIUS) + 1$ , δηλαδή ακριβώς έξω από την μέγιστη ακτίνα ορατότητας του παίκτη και του αντιπάλου
- φροντίζουμε να μην έχουμε παραπάνω από  $N$  αντιπάλους στην πίστα ταυτόχρονα (ο παίκτης μπορεί να κινηθεί με τέτοιο τρόπο, ώστε να μην δει ποτέ ο αντίπαλος τον παίκτη ή το αντίθετο)

Επιλέξτε όποιες τιμές / συναρτήσεις νομίζετε καλύτερες για την πιθανότητα εμφάνισης αντιπάλων σαν συνάρτηση με τα hit points του παίκτη, καθώς και για το μέγιστο

πλήθος των αντιπάλων στην πίστα. Γενικά μια συνάρτηση της μορφής  $p_0 e^{-4 \cdot hp / hp_{max}}$  με  $p_0 = [0.25, 0.10]$  φαίνεται αρκετά καλή. Ωστόσο, οποιαδήποτε προσέγγιση είναι δεκτή, π.χ. μπορείτε να χρησιμοποιήσετε μια απλή γραμμική σχέση της μορφής  $hp_{percentage} \cdot 0.2$  ώστε όταν ο παίκτης είναι στο 100% των hit points να εμφανίζονται εχθροί με πιθανότητα 0.2.

#### 4.5.1 Γεννήτριες αντιπάλων

Εναλλακτικά, μπορείτε να χρησιμοποιήσετε την προσέγγιση των "γεννητριών" αντιπάλων, οι οποίες όμως αλλάζουν δραστικά τον τρόπο παιχνιδιού. Πιο συγκεκριμένα, αντί να τοποθετούνται οι αντίπαλοι στην πίστα κατά τη δημιουργία της, ή με κάποια πιθανότητα σε κάθε action του χρήστη, όπως περιγράψαμε προηγουμένως, τοποθετούνται ειδικά πλακίδια, τα οποία γεννούν αντιπάλους. Κάθε τέτοιο πλακίδιο δικαιούται να έχει έναν συγκεκριμένο αριθμό αντιπάλων στην πίστα. Αν κάποιος από τους αντίπαλους ηττηθεί, τότε, σε κάθε γύρο, επανέρχεται στο παιχνίδι με πιθανότητα  $p$  (η οποία χαρακτηρίζει την γεννήτρια).

#### 4.5.2 Αντικείμενα που "ρίχνουν" οι αντίπαλοι

Κάθε αντίπαλος, όταν νικιέται, έχει πιθανότητα 25% να αφήσει ένα rotion στο πλακίδιο του χάρτη που βρισκόταν.

### 4.6 Μάχες

Η διαδικασία της μάχης θα είναι και αυτή αρκετά απλοποιημένη, τόσο για τους παίκτες, όσο και για τους αντιπάλους. Ένας αντίπαλος μπορεί να επιτεθεί σε έναν παίκτη όταν βρίσκονται σε γειτονικά (αριστερά, δεξιά, πάνω και κάτω) πλακίδια (δηλαδή, απόσταση με  $L_1$  ίση με το 1). Δεν μπορεί να επιτεθεί ενώ κινείται (κίνηση και επίθεση θεωρούνται σαν δυο διαφορετικά actions).

Όσον αφορά τον παίκτη, ο "πολεμιστής" επιτίθεται σε έναν από τους αντιπάλους που βρίσκονται σε γειτονικά πλακίδια. Μπορείτε, αν θέλετε, να προσθέσετε κάποια επιπλέον λογική, π.χ. επιτίθεται πάντα σε αυτόν που έχει τα λιγότερα health points.

Ο "μάγος" επιτίθεται στον αντίπαλο που είναι πιο κοντά σε αυτόν και είναι εντός του πεδίου ορατότητας. Οι επιθέσεις του μάγου αγνοούν τοίχους, εμπόδια κλπ, ώστε να είναι πιο απλοποιημένο το μοντέλο της μάχης.

Αν ο παίκτης αποφασίσει να κινηθεί (το game-play οδηγείται από τις "δράσεις" / actions του παίκτη) σε κάποιο πλακίδιο που δεν είναι γειτονικό, τότε ο αντίπαλος δεν μπορεί να επιτεθεί στον παίκτη, αλλά θα πρέπει στο επόμενο βήμα να τον ακολουθήσει (ένας τρόπος ώστε ο παίκτης να μπορεί να αποφεύγει τις μάχες).

Αν θέλετε, μπορείτε να υλοποιήσετε αντιπάλους που επιτίθενται από απόσταση, αυτό, ωστόσο, κάνει το gameplay αρκετά πιο δύσκολο.

## 5 Γραφικά, χειρισμός

Για τα γραφικά του παιχνιδιού χρησιμοποιήστε τις βιβλιοθήκες `javax.swing` και `java.awt`. Δεν χρειάζεται να χρησιμοποιήσετε περίπλοκα γραφικά (ούτως ή άλλως τα *rogue-like games* χαρακτηρίζονται από απλά, συνήθως `text`, γραφικά). Επίσης, καλό θα ήταν να μη βασιστείτε στην `JavaFX`, χωρίς ωστόσο να είναι κάτι που απαγορεύεται. Απλά, επειδή η συγκεκριμένη βιβλιοθήκη απαιτεί κάποια πράγματα να γίνουν με συγκεκριμένο τρόπο, ίσως να σας δυσκολέψει.

### 5.1 Εκτέλεση του παιχνιδιού

Δεν χρειάζεται να κατασκευάσετε διαφορετικές οθόνες, π.χ. αρχική οθόνη, οθόνη επιλογής χαρακτήρα, οθόνη τέλους, κλπ, παρά μόνο τη βασική οθόνη του παιχνιδιού με το `user interface` που περιγράφεται στη συνέχεια. Τυχόν παράμετροι του παιχνιδιού (π.χ. η κλάση του χαρακτήρα και το όνομά του) θα δίνονται σαν παράμετροι στην `main` μέθοδο της βασικής κλάσης του παιχνιδιού, π.χ.

```
$ java rogue.Game wizard Merlin
```

και αντίστοιχα η `main` θα έχει μια μορφή σαν την ακόλουθη:

```
// ...
public static void main(String[] args) {
    if(args.length < 2) {
        System.out.println("Missing arguments");
        System.out.println("Please run as: java rogue.Game " +
            " player-class player-name");
        System.out.println("   where player-class is either " +
            "\"wizard\" or \"duelist\"");
        System.out.println("   and player-name is the " +
            "character name");
        System.exit(0);
    }
    // Player class is in args[0]
    // Player name is in args[1]
}
```

Χρησιμοποιώντας τα `args[0]` και `args[1]` μπορείτε να αρχικοποιήσετε τον χαρακτήρα του παίκτη. Όταν ο παίκτης χάσει ή νικήσει, το παιχνίδι επιστρέφει στην κονσόλα (ή στο πρόγραμμα / IDE που το εκτέλεσε) και τυπώνει ένα αντίστοιχο μήνυμα για το αποτέλεσμα.

Αν θέλετε μπορείτε να φτιάξετε ενδιάμεσες οθόνες (επιλογή κλάσης, ονόματος, κλπ), δεν είναι ωστόσο απαραίτητο.

## 5.2 User Interface



Figure 6: Παράδειγμα του User Interface του παιχνιδιού

### 5.2.1 Χάρτης

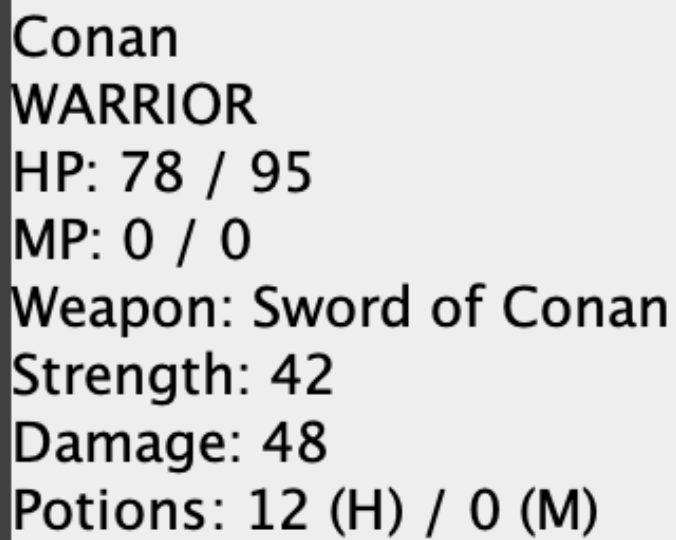
Το μεγαλύτερο και βασικότερο τμήμα της οθόνης το καταλαμβάνει ο χάρτης του παιχνιδιού, παραδείγματα του οποίου είδαμε παραπάνω.

### 5.2.2 Player Status

Το παράθυρο κατάστασης του παίκτη μας δίνει πληροφορίες για τα βασικά στοιχεία του χαρακτήρα. Στην πραγματικότητα αποτυπώνει απ' ευθείας τα βασικά χαρακτηριστικά του, δηλαδή hit points, mana points, κλπ. Δεν είναι ανάγκη να προχωρήσετε σε κάποιον ειδικό σχεδιασμό, με περίπλοκα γραφικά και `javax.swing` components. Αρκεί μια απλή μορφή με, π.χ. `JLabel`. Θα πρέπει ωστόσο, να ανανεώνεται σε κάθε action του παιχνιδιού, ώστε να δείχνει την τελευταία πληροφορία.

### 5.2.3 Game Log

Το Game Log είναι απλά ένα `JTextArea`, το οποίο καταγράφει συνεχώς τι συμβαίνει στο παιχνίδι. Για παράδειγμα, όταν ο παίκτης βρεθεί σε πλακίδιο που έχει Potions (τα οποία και μαζεύει αυτόματα) θα του γράψει ένα κείμενο της μορφής `you found N potions`. Αντίστοιχα, αν στο πλακίδιο υπάρχει κάποιο όπλο `you`



Conan  
WARRIOR  
HP: 78 / 95  
MP: 0 / 0  
Weapon: Sword of Conan  
Strength: 42  
Damage: 48  
Potions: 12 (H) / 0 (M)

Figure 7: Παράδειγμα πιθανής μορφής του Player Status (με χρήση JLabel)

see the .... Αν αλλάξει το όπλο που έχει με αυτό που είναι στο πλακίδιο, τότε αντίστοιχα you pick up the .. και you dropped the .... Τέλος, μπορεί να γράφει διάφορες πληροφορίες για τις μάχες (π.χ. damage, health, κλπ) καθώς επίσης και πληροφορίες οι οποίες να προσδίδουν στην ατμόσφαιρα του παιχνιδιού (π.χ. αν τοποθετηθεί ένα Pattern Shade κοντά στον παίκτη, και σύμφωνα με τα όσα είπαμε παραπάνω, μπορεί να γράφει κάτι της μορφής the Pattern senses your struggle and calls a Shade to stop you).

Μπορείτε να χρησιμοποιήσετε οποιαδήποτε μέθοδο για να ανανεώνονται τα παραπάνω συνεχώς, π.χ. Observer pattern, Events, κλπ.

### 5.3 Γραφικά

Τα παρακάτω είναι απλά προτάσεις, είσαστε ελεύθεροι να επιλέξετε οποιαδήποτε προσέγγιση επιθυμείτε για τον σχεδιασμό των παικτών, αντικειμένων, αντιπάλων, κλπ. Προσπαθήστε να μην αναλώσετε πάρα πολύ χρόνο στο συγκεκριμένο (εκτός αν φυσικά το επιθυμείτε)

#### 5.3.1 Πλακίδια του χάρτη

Τα πλακίδια του χάρτη είναι απλά rectangles με κάποιο συγκεκριμένο μήκος πλευράς (στην ενδεικτική λύση χρησιμοποιήθηκε μήκος πλευράς  $W = 8$ ). Βασικό ίσως είναι να φροντίσετε ο χάρτης να χωράει σε όλη την οθόνη, ώστε να μη χρειαστεί να υλοποιήσετε scrolling ή αντίστοιχες μεθόδους). Μπορείτε να προσδώσετε "βάθος" στα πλακίδια σχεδιάζοντας πρώτα ένα rectangle με πλευρά  $W$  στο χρώμα υποβάθρου και στη συνέχεια ένα rectangle με πλευρά  $W - 1$  στο χρώμα του πλακιδίου.

Με αυτόν τον τρόπο, και, δεδομένου ότι η πίστα αποτελείται από  $N \times M$  πλακίδια, αρκεί να σχεδιάσετε δυο τετράγωνα (fillRect) στις θέσεις  $x \cdot TILE\_SIZE, y \cdot TILE\_SIZE$  με διαφορετικά μήκη πλευρών, π.χ.

```
g.setColor(new Color(100, 60, 40));
g.fillRect(x * TILE_SIZE, y * TILE_SIZE,
    TILE_SIZE, TILE_SIZE);
g.setColor(new Color(180, 100, 100));
g.fillRect(x * TILE_SIZE, y * TILE_SIZE,
    TILE_SIZE - 1, TILE_SIZE - 1);
```

#### 5.3.2 Χαρακτήρες του παίκτη και αντίπαλοι

Για τον χαρακτήρες του παίκτη μπορείτε είτε να σχεδιάσετε έναν κύκλο με την μέθοδο fillArc, είτε ένα rectangle. Αυτό που θέλει προσοχή είναι το ότι η πραγματική περιοχή, αν χρησιμοποιήσετε tiles με βάθος όπως παραπάνω, είναι



η  $x \cdot TILE\_SIZE, y \cdot TILE\_SIZE, TILE\_SIZE - 1, TILE\_SIZE - 1$ .

Επομένως μπορείτε να σχεδιάσετε τη θέση του παίκτη με ένα, π.χ. `rectangle`

```
g.setColor(new Color(30, 200, 200));
g.fillRect(x*TILE_SIZE+1, y*TILE_SIZE+1,
    TILE_SIZE-3, TILE_SIZE-3);
```

ή έναν κύκλο

```
g.setColor(new Color(30, 200, 200));
g.fillArc(x*TILE_SIZE+1, y*TILE_SIZE+1,
    TILE_SIZE-3, TILE_SIZE-3, 0, 360);
```

Αντίστοιχα, και για τους αντιπάλους χρησιμοποιήστε τετράγωνα ή κύκλους, φροντίζοντας ώστε οι διαφορετικοί τύποι αντιπάλων να έχουν διαφορετικά χρώματα (και πάλι δεν είναι υποχρεωτικό)

### 5.3.3 Αντικείμενα

Όπως είδαμε και παραπάνω, μπορείτε απλά να κάνετε `annotate` τα πλακίδια στα οποία υπάρχουν αντικείμενα. Ο τρόπος με τον οποίο έγινε αυτό στην εικόνα 5 είναι ο εξής:

```
g.setColor(new Color(30, 250, 30));
g.fillRect(x*TILE_SIZE+TILE_SIZE-3, y*TILE_SIZE, 2, 2);
```

Μπορείτε αντίστοιχα να χρησιμοποιήσετε όποια άλλη προσέγγιση θέλετε.

## 5.4 Χειρισμός

Όπως είπαμε και παραπάνω, ο χειρισμός του παιχνιδιού γίνεται με τη χρήση πλήκτρων. Μπορείτε να χρησιμοποιήσετε έναν `KeyListener` για να διαβάσετε την είσοδο από το πληκτρολόγιο, π.χ.

```
this.addKeyListener(new KeyListener() {
    @Override
    public void keyTyped(KeyEvent e) {
        switch(e.getKeyChar()) {
            case 'a':
            case 'A':
                //
                gameState.movePlayer(-1, 0);
                // ή, π.χ.
```

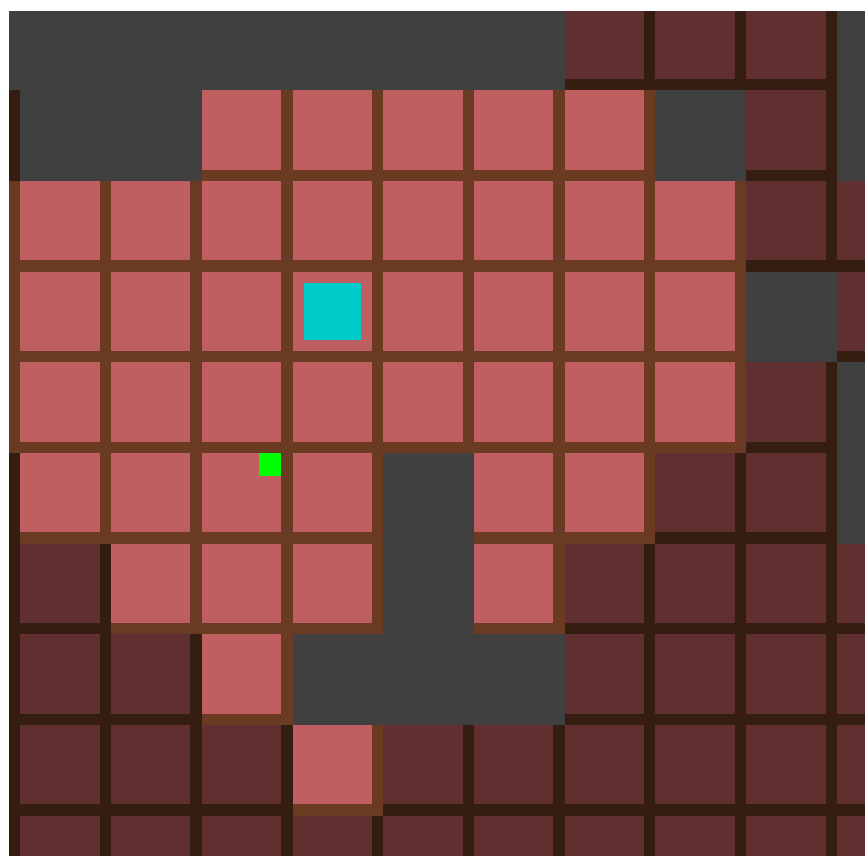


Figure 8: Ενδεικτική απεικόνιση του παίκτη

```

// gameState.movePlayerLeft()
break;
    case 'd':
    case 'D':
//
gameState.movePlayer(+1, 0);
// ή, π.χ.
// gameState.movePlayerRight()
break;
// .. other cases here ...
    default:
break;
    }
    // Notify that game state needs to be advanced
    // (i.e. update game state according to action)
    //
    // and repaint the map
    }
    // other overridden methods (not used)
    // ...
});

```

Το σημείο που θέλει προσοχή εδώ είναι ότι οι `keyListeners` δέχονται `events`, μόνο όταν το παράθυρο / component στο οποίο έχουν προστεθεί είναι `focused`, λαμβάνει δηλαδή τα `events` εισόδου. Υπάρχουν διάφορες προσεγγίσεις για να δέξεστε το `input` από οποιοδήποτε component, όπως π.χ. τα `keybindings`. Δεν είναι υποχρεωτικό να χρησιμοποιήσετε κάποια τέτοια προσέγγιση, και μπορείτε να θεωρήσετε ότι ο παίκτης θα δίνει το `focus` στο component που χρειάζεται, ή μπορείτε να το δίνετε εσείς και να το επιστρέφετε στην περίπτωση που χαθεί, π.χ.

```

// In main application window
frame.addWindowFocusListener(new WindowAdapter() {
    public void windowGainedFocus(WindowEvent e) {
        gameCanvas.requestFocusInWindow();
    }
});
//...

```

```

// In actual game canvas, or canvas container
GameCanvas canvas = this;

```

```

this.addFocusListener(new FocusListener() {
    @Override
    public void focusGained(FocusEvent e) {
        // Nothing here
    }

    @Override
    public void focusLost(FocusEvent e) {
        // return focus to canvas
        canvas.requestFocusInWindow();
    }
});

```

Για να διαβάσετε περισσότερα για τον χειρισμό του focus στο swing, δείτε το αντίστοιχο tutorial.

## 6 Γενικές συμβουλές

Δείτε τις παλαιότερες ασκήσεις και παραδείγματα, τα οποία περιλαμβάνουν αρκετές προσεγγίσεις για το πως να υλοποιήσετε / δομήσετε τις κλάσεις και τα interfaces.

Χρησιμοποιήστε περισσότερες από μια main μεθόδους, ώστε να μπορείτε να δοκιμάζετε ανεξάρτητα τμήματα του προγράμματος.

### 6.1 Ενδεικτικό διάγραμμα κλάσεων / Οργάνωση του κώδικα

Το παρακάτω παράδειγμα οργάνωσης του κώδικα είναι καθαρά ενδεικτικό, είστε ελεύθεροι να χρησιμοποιήσετε όποια προσέγγιση θέλετε. Επιπλέον, αφορά τις βασικές κλάσεις του παιχνιδιού και όχι π.χ. τις κλάσεις που εξυπηρετούν τόσο το UI και τα μηνύματα μεταξύ των κλάσεων (τις οποίες μπορείτε επίσης να προσεγγίσετε όπως θέλετε, είτε με κάποιο Design Pattern (composite / observer), είτε με κάποιον άλλο τρόπο).

- player (package)
  - AbstractPlayer (abstract)
  - Wizard
  - Duelist
- enemies (package)
  - Enemy (interface) (\*)

- κλάσεις για τους διάφορους εχθρούς (\*)
- SerpentOfChaos
- Μπορείτε να έχετε interfaces για το visibility / position των Enemies ή να τα έχετε απ' ευθείας στην Enemy class
- items (package)
  - ItemEffectType (enum) (\*)
  - ItemEffect (\*)
  - Item (interface) (\*)
  - Equippable (interface) (\*)
  - Consumable (interface) (\*)
  - WeaponType (enum)
  - Weapon (\*)
  - Διάφορες κλάσεις για Health / Mana potions (concrete consumables) και Weapons ή μια κλάση για WeaponMaker / PotionMaker (\*)
- Dice (\*)
- GameMap
- (GameWorld (συλλογή & διαχείριση GameMaps))
- Game (entry-point)

Τις κλάσεις που σημειώνονται με (\*) μπορείτε να τις πάρετε από κάποια από τις ασκήσεις των προηγούμενων ετών. Επίσης, μια από τις παραπάνω (π.χ. η Game ή κάποια άλλη) θα αποτελεί και το βασικό game state, το οποίο θα αλλάζει με τα actions του χρήστη.

## 7 Αναφορές

### 7.1 Procedural Dungeon Generation

1. <https://www.firespark.de/?id=article&article=ProceduralGenerationResources&general>
2. <https://dl.acm.org/doi/pdf/10.5555/3144605.3144638?download=true>

3. <https://github.com/marukrap/RoguelikeDevResources>
4. [https://www.firespark.de/resources/downloads/MA\\_Beyer.pdf](https://www.firespark.de/resources/downloads/MA_Beyer.pdf)
5. <https://www.gridsgames.com/blog/2014/06/procedural-map-generation/>
6. <https://www.gridsgames.com/blog/2016/03/generating-populating-caves>
7. [http://www.roguebasin.com/index.php?title=Random\\_Walk\\_Cave\\_Generation](http://www.roguebasin.com/index.php?title=Random_Walk_Cave_Generation)

## 7.2 Line of Sight, Visibility

1. [http://www.roguebasin.com/index.php?title=Articles#Line\\_of\\_sight.2C\\_field\\_of\\_vision](http://www.roguebasin.com/index.php?title=Articles#Line_of_sight.2C_field_of_vision)
2. [http://www.adammil.net/blog/v125\\_Roguelike\\_Vision\\_Algorithms.html](http://www.adammil.net/blog/v125_Roguelike_Vision_Algorithms.html)

## 7.3 AI

- [http://www.roguebasin.com/index.php?title=Roguelike\\_Intelligence](http://www.roguebasin.com/index.php?title=Roguelike_Intelligence)
- Pathfinding [https://www.reddit.com/r/roguelikedev/comments/3slu9c/faq\\_friday\\_25\\_pathfinding/](https://www.reddit.com/r/roguelikedev/comments/3slu9c/faq_friday_25_pathfinding/)
- A\* [https://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)
- A\* Pathfinding <https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding>

### 7.3.1 Διάφορα

- [http://repositori.uji.es/xmlui/bitstream/handle/10234/179991/MEMORIA\\_PinillaBermejoAndoniTFG.pdf](http://repositori.uji.es/xmlui/bitstream/handle/10234/179991/MEMORIA_PinillaBermejoAndoniTFG.pdf)