Περιεχόμενα

1 Τελική Εργασία: Απόδραση από το κάστρο του Βασιλιά Leoric

Μερικές 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 κινήσεις των μονάδων 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 τα Design Patterns πού είδαμε στις δύο τελευταίες εργασίες (το Observer Pattern προσφέρεται ιδιαίτερα, καθώς επίσης και το Composite). 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 μοντελοποίηση στην οποία καταλήξατε είναι καλή είναι να δείτε πόσο εύκολα μπορείτε να προσθέσετε νέους τύπους μονάδων και νέες δυνατότητες στις μονάδες αυτές.

Καλή επιτυχία!

2 Παιχνίδια τύπου Rogue-Like Dungeon Crawler

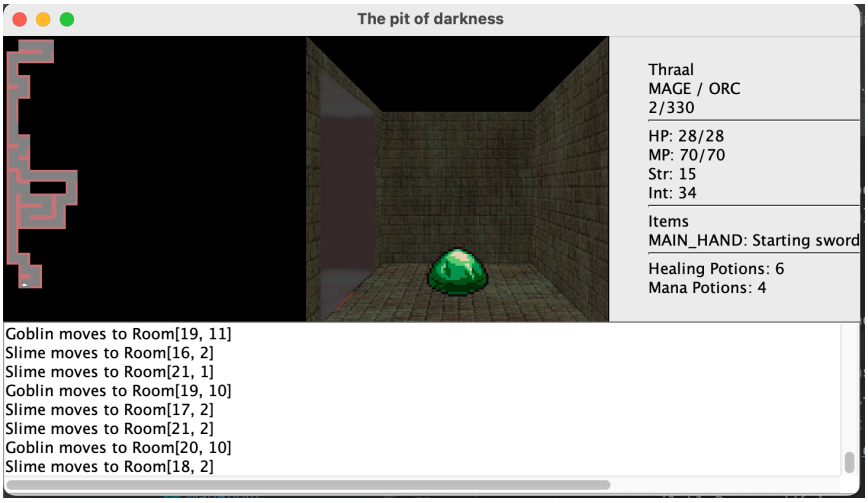
Σαν παιχνίδια rogue-like χαρακτηρίζονται τα παιχνίδια ρόλων (RPG games), στα οποία ο παίκτης εξερευνάει επίπεδα ενός κόσμου (σπήλαιο, κάστρο, υπόγεια περάσματα, δάσος, συνδυασμοί αυτών, κλπ), τα οποία παράγονται τυχαία από τον υπολογιστή κατά το ξεκίνημα του παιχνιδιού. Ο παίκτης έχει μόνο μια ”ζωή”, 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 τα πιο καινούργια rogue-like games χαρακτηρίζονται, συνήθως, από ιδιαίτερα σύνθετο gameplay, σενάριο και mechanics. Η ονομάσία `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 παιξίματος, 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

”Ο καταραμένος βασιλιάς Leoric και ο στρατός των απέθαντων έχει αιχμαλωτίσει τον χαρακτήρα του παιχνιδιού στα υπόγεια του κάστρου του. Ο παίκτης μας θα πρέπει να επιβιώσει, να εξοπλιστεί και να διασχίσει 4 επίπεδα με λαβυρίνθους πριν αντιμετωπίσει τον ίδιο τον βασιλιά Leoric στον τελικό λαβύρινθο. Όμως οι

σκοτεινές δυνάμεις του Leoric επαγρυπνούν. Για να τα καταφέρει θα πρέπει να αντιμετωπίσει ορδές από slimes, goblins, ghosts, skeletons, skeleton lords και skeleton demons πριν καταφέρει να φτάσει στον ίδιο τον Leoric."



Σχήμα 1: Εικόνα του gameplay

Ο κόσμος του παιχνιδιού αποτελείται από 5 διαφορετικά επίπεδα, τα οποία με τη σειρά τους αποτελούνται από τετράγωνα πλακίδια τα οποία επικοινωνούν μεταξύ τους δημιουργώντας, σε κάθε επίπεδο, έναν σύνθετο λαβύρινθο. Οι διαστάσεις των επιπέδων είναι 25 πλακίδια και ο λαβύρινθος του κάθε επιπέδου δημιουργείται τυχαία [1].

Κάθε δωμάτιο / πλακίδιο έχει 4 πλευρές οι οποίες αντιστοιχούν στις 4 διευθύνσεις του ορίζοντα, δηλαδή Α/Δ/Β/Ν (N/S/E/W). Τα δωμάτια / πλακίδια ενδέχεται να επικοινωνούν με ένα η περισσότερα γειτονικά, ανάλογα με τη διάταξη του εκάστοτε λαβυρίνθου. Αν δυο δωμάτια επικοινωνούν, τότε ο παίκτης και οι εχθροί μπορούν να κινηθούν ανάμεσα στα δυο δωμάτια, αλλιώς θεωρούμε ότι ανάμεσα στα πλακίδια υπάρχει τοίχος [2].

Σε κάθε επίπεδο, εκτός από το τελευταίο, υπάρχουν πάντα 3 εχθροί, οι οποίοι εμφανίζονται σε τυχαίο σημείο του λαβυρίνθου (αρκεί να μην καταλαμβάνεται από τον παίκτη ή κάποιον άλλο εχθρό) και κινούνται προς τον παίκτη. Για την εύρεση του μονοπατιού από τον εχθρό προς τον παίκτη μπορείτε να χρησιμοποιήσετε όποια προσέγγιση θέλετε (είτε breadth-first search, είτε `A*`, είτε κάποιον άλλο αλγόριθμο).

[1] είναι η πιο απλή προσέγγιση, αν θέλετε μπορείτε να φορτώνετε τα επίπεδα από κάποιο αρχείο

[2] οι όροι "δωμάτιο" και "πλακίδιο" χρησιμοποιούνται ισοδύναμα, κυρίως επειδή η 2η άσκηση αναφερόταν σε "δωμάτια"

Όταν κάποιος εχθρός πεθαίνει, αντικαθίσταται στο επόμενο βήμα του παιχνιδιού με έναν νέο σε κάποιο τυχαίο σημείο του λαβυρίνθου. Μπορείτε να έχετε το πλήθος των εχθρών σαν παράμετρο του επιπέδου του παίκτη και του επιπέδου στο οποίο βρίσκεται.

Ο παίκτης ξεκινάει από το πιο Βόρειοδυτικό δωμάτιο του χάρτη και η έξοδος κάθε επιπέδου βρίσκεται στο πιο Νοτιοανατολικό. Μόλις ο παίκτης βρεθεί στην έξοδο μεταφέρεται στο επόμενο επίπεδο.

Όταν πεθαίνει κάποιος εχθρός αφήνει στο δωμάτιο / πλακίδιο στο οποίο βρίσκεται αντικείμενα, είτε όπλα, είτε βοηθητικά αντικείμενα, είτε φίλτρα, τα οποία μπορεί να εξοπλισει και να χρησιμοποιήσει ο παίκτης.

2.1.1 Χρόνος στο παιχνίδι

Ο "χρόνος" στο παιχνίδι είναι διακριτός και οδηγείται από τις δράσεις του παίκτη. Κάποιες από τις δράσεις αυτές είναι "ελεύθερες χρόνου", δηλαδή δεν προχωράνε τη δράση (κατάσταση) του παιχνιδιού, παρά μόνο αλλάζουν την κατάσταση του χαρακτήρα του παίκτη, ενώ κατά τις περισσότερες δράσεις, πρώτα εκτελείται η "επιλογή" του χρήστη και στη συνέχεια εκτελείται η λογική του παιχνιδιού για όλο τον υπόλοιπο κόσμο.

2.1.2 Δράσεις του παίκτη

Οι δυνατές δράσεις (actions) του παίκτη είναι οι ακόλουθες:

- περιστροφή αριστερά και δεξιά (αλλάζει την κατεύθυνση που κοιτάει ο παίκτης). Πραγματοποιείται με τα πλήκτρα `a` και `d` και είναι "ελεύθερη δράση", δηλαδή ο παίκτης μπορεί να περιστραφεί και να κοιτάξει γύρω του, χωρίς να προχωρήσει το state του παιχνιδιού

- κίνηση προς τα μπροστά. Πραγματοποιείται με το πλήκτρο `w` και μετακινεί, αν αυτό είναι δυνατόν, τον παίκτη προς την κατεύθυνση που κοιτάει, δηλαδή σε ένα γειτονικό δωμάτιο, αν τα δωμάτια είναι συνδεδεμένα.

- επίθεση σε έναν εχθρό που βρίσκεται στο γειτονικό πλακίδιο προς το οποίο είναι στραμμένος ο παικτης. Προφανώς, αν τα πλακίδια δεν είναι συνδεδεμένα, τότε η επίθεση αποτυγχάνει. Πραγματοποιείται με το πλήκτρο `x`.

- "μαγική" επίθεση προς την κατεύθυνση που κοιτάζει ο παίκτης (αν είναι δυνατόν, με δεδομένη την κλάση του χαρακτήρα του παίκτη). Το "spell" κινείται προς την κατεύθυνση που κοιτάει ο παίκτης μέχρι να συναντήσει είτε κάποιον τοίχο, είτε κάποιον αντίπαλο. Πραγματοποιείται με το πλήκτρο `c`

- ξεκούραση για μια χρονική μονάδα του πειχνιδιού. Ο χαρακτήρας ανακτά κάποια από τα hitPoints του και κάποια από τα manaPoints (αν το επιτρέπει η κλάση του χαρακτήρα). Πραγματοποιείται με το πλήκτρο `r`.

- Ανάλογα με το πως θα επιλέξετε να υλοποιήσετε την λειτουργικότητα των `usable` items (φίλτρα, μαγικά ραβδιά, κλπ), μπορείτε είτε να έχετε 2 πλήκτρα, ένα για τη χρήση των φίλτρων που αναπληρώνουν τα hitPoints (`h`) και ένα για τη χρήση των φίλτρων που αναπληρώνουν τα manaPoints (`m`). Αν αντί για 2 ειδών φίλτρα επιλέξετε να υλοποιήσετε φίλτρα με σύνθετα αποτελέσματα, τότε δείτε ποια προσέγγιση ταιριάζει καλύτερα.

- Επίσης, ανάλογα με το πως θα επιλέξετε να υλοποιήσετε την πράξη "εξοπλίζω ένα Equippable item", μπορείτε είτε να διατηρείτε τα αντικείμενα που βρίσκονται[3] σε ένα πλακίδιο σε μια ουρά και με τη χρήση του πλήκτρου, π.χ. `e` να παίρνετε το πρώτο αντικείμενο από την ουρά, να το εξοπλίζετε στην κατάλληλη θέση και να βάζετε το αντικείμενο που είχε ο παίκτης στην θέση (αν υπάρχει) πίσω στην ουρά με τα αντικείμενα που διατηρεί το πλακίδιο (pick up and equip). Εναλλακτικά θα μπορούσατε, π.χ. να έχετε στο πλακίδιο ουρές ανάλογα με τη θέση (SlotType) στην οποία μπορει να τοπποθετηθεί κάθε αντικείμενο και με τα πλήκτρα `1` εως `4` να ανταλάσσετε αντικείμενα από τον παίκτη στις ουρές, κλπ. Αν και θα το αναφέρουμε και παρακάτω, ο παίκτης έχει inventory **μόνο** για τα usable items.

3 Ο Χάρτης του παιχνιδιού

Όπως αναφέραμε και προηγουμένως, ο χάρτης του κάθε επιπέδου αποτελείται από 25×25 δωμάτια έτσι ώστε κάθε δωμάτιο να συνδέεται με κάποια από τα γειτονικά του ώστε τελικά το κάθε επίπεδο να είναι ένας λαβύρινθός. Ο λαβύρινθος αυτός μπορεί να έχει ή να μην έχει κύκλους. Γενικά υπάρχουν πολλές προσεγγίσεις για τη δημιουργία απλών λαβυρίνθων (βλέπε `https://en.wikipedia.org/wiki/Maze_generation_algorithm`). Στην πιο απλή περίπτωση ένα randomized depth first search δίνει αρκετά καλά αποτελέσματα. Αν π.χ. θεωρήσουμε ότι έχουμε μια κλάση `Room` που περιγράφει ένα δωμάτιο, καθώς και τους γείτονές του στις διάφορες διευθύνσεις τότε, μια απλή υλοποίηση του randomized depth first search θα μπορούσε να είναι η ακόλουθη:

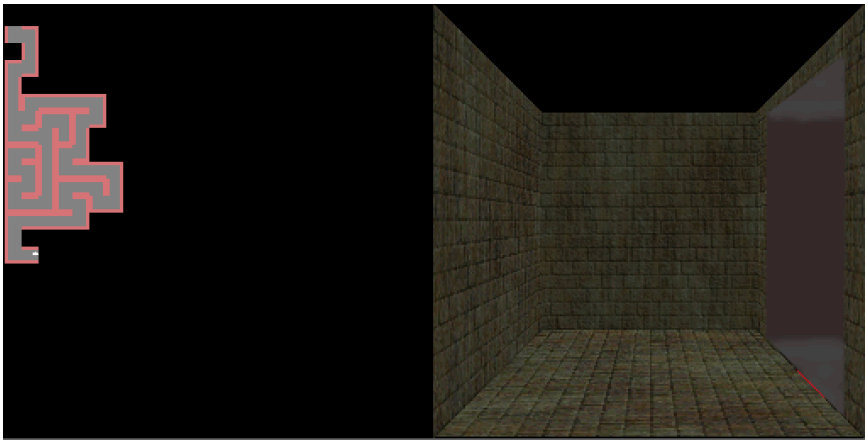
```
Room[][] generateMap(int width, int height) {
    map = new Room[height][width];
```

[3] αν σε ένα πλακίδιο έχουν πεθάνει παραπάνω από ένας εχθροι, ενδεχομένως να υπάρχουν πολλά equippable για το ίδιο slot

```
    Set<Room> availableRooms = new HashSet<>();
    for(int col = 0; col < width; col++) {
for(int row = 0; row < height; row++) {
    var r = new MapRoom(row, col);
    map[row][col] = r;
    availableRooms.add(r);
}
    }

    var startRoom = map[rng.nextInt(height)][rng.nextInt(width)];
    Set<Room> visited = new HashSet<>();
    LinkedList<Room> stack = new LinkedList<>();
    visited.add(startRoom);
    stack.push(startRoom);

    while(!stack.isEmpty()) {
var curRoom = stack.pop();

Set<Room> unvisitedNeighbors = getNeighbors(curRoom)
    .stream()
    .filter((x)-> {return ! visited.contains(x); })
    .collect(Collectors.toSet());

if(unvisitedNeighbors.size() > 0) {
    stack.push(curRoom);
    var newCurr = randRoom(unvisitedNeighbors);
    curRoom.connectRoom(newCurr);
    visited.add(newCurr);
    stack.push(newCurr);
}
    }
    System.out.println("TOTAL VISITED ROOMS: " + visited.size());
    return map;
}
```

όπου οι μέθοδος `getNeighbors` επιστρέφει τα γειτονικά δωμάτια (ανεξάρτητα από τον αν είναι συνδεδεμένα με το τρέχον), η `randRoom` επιλέγει ένα τυχαίο στοιχείο από μια λίστα με δωμάτια και η `connectRoom` συνδέει δυο γειτονικά δωμάτια, έτσι ώστε αν π.χ. βρίσκονται στην ίδια γραμμή και σε γειτονικές στήλες, να έχουν μια σχέση ”το δωμάτιο Χ είναι ανατολικά του Υ και το Υ δυτικά του Χ”.

Θα χρησιμοποιήσουμε τον λαβύρινθο που προκύπτει από τον παραπάνω αλγόριθμο για να σχεδιάσουμε δυο διαφορετικές προοπτικές του χάρτη, πιο συγκεκριμένα την προοπτική πρώτου προσώπου καθώς και την προοπτική της μικρογραφίας του λαβυρίνθου (”mini map”).



Σχήμα 2: Οι δύο διαφορετικές προβολές του λαβυρίνθου

Η προβολή πρώτου προσώπου απεικονίζει μέχρι και δυο δωμάτια στη διεύθυνση που κοιτάει ο παίκτης, ενώ η προβολή minimap δείχνει τη θέση του παίκτη, καθώς και το τμήμα του λαβυρίνθου το οποίο έχει επισκεπτεί.

Αν θέλετε μπορείτε να δείχνετε στο minimap τα δωμάτια στα οποία υπάρχουν αντικείμενα τα οποία έπεσαν από κάποιον εχθρό που νικήθηκε, καθώς, π.χ. και την θέση των κοντινών, στον παίκτη, εχθρών.

Τέλος, ο παίκτης και οι διάφοροι εχθροί που τον καταδιώκουν καταλαμβάνουν ένα πλακίδιο στον χάρτη, με τέτοιο τρόπο ώστε ούτε ο παίκτης ούτε οι εχθροί να μπορούν να κινηθούν σε πλακίδιο που είναι κατειλλημένο.

4 Παίκτης, αντικείμενα, αντίπαλοι και μηχανική των κανόνων

Οι χαρακτήρες στα παιχνίδια ρόλων περιγράφονται από ένα σύνολο αριθμών (στατιστικά του χαρακτήρα), οι οποίοι περιγράφουν τα χαρακτηριστικά τους καθώς και τη δυσκολία ή ευκολία με την οποία ο χαρακτήρας μπορεί να πραγματοποιήσει κάτι στο παιχνίδι. Το πλήθος και η πολυπλοκότητα των χαρακτηριστικών ποικίλλει ανάλογα με το παιχνίδι. Στην περίπτωσή μας θα χρησιμοποιήσουμε τα ακόλουθα στατιστικά:

- Hit Points: εκφράζουν το πόση ”ζημια” (damage) μπορεί να αντέξει ο χαρακτήρας.

Όταν φτάσουν στο 0, θα θεωρούμε ότι ο παίκτης έχασε.

- Mana Points: resource των κλάσεων που μπορούν να εκτελέσουν μαγικά. Η χρήση μαγικών καταναλώνει το resource και μπορεί να αναπληρωθεί είτε με ”Rest” είτε με χρήση κάποιου φίλτρου.
- Strength: χαρακτηρίζει την δύναμη του χαρακτήρα και επηρρεάζει την ”ζημιά” (damage) που κάνουν οι επιθέσεις του με όπλα
- Intellect: χαρακτηρίζει την νοητική δύναμη του παίκτη και επηρρεάζει το damage που κάνουν οι επιθέσεις του με τη χρήση μαγικών.
- Άμυνα στους διάφορους τύπους επίθεσης: εκφράζει πόσο ανθεκτικός είναι ο παίκτης και ελαττώνει το ποσό της επίθεσης που δέχεται από τους εχθρούς.

Οι εχθροί στο παιχνίδι περιγράφονται από ένα υποσύνολο αυτών των χαρακτηριστικών, πιο συγκεκριμένα, μπορούν να περιγραφούν μόνο από τα hit points και την άμυνά τους στους διαφορετικούς τρόπους επίθεσης.

4.1 Τύποι επίθεσης, άμυνας, όπλα

Η ζημιά που προκαλεί ο χαρακτήρας στους εχθρούς και αντίστοιχα οι εχθροί στον χαρακτήρα μπορεί να είναι τριών διαφορετικών τύπων:

- `SLASHING`
- `BLUNT`
- `MAGICAL`

Αντίστοιχα, τόσο ο χαρακτήρας του παίκτη όσο και οι εχθροί έχουν διαφορετική ανοχή στους τρείς αυτούς τύπους ζημιάς.

Οι επιθέσεις στο παιχνίδι μπορούν να γίνουν μόνο με τη χρήση κάποιου όπλου, δηλαδή, το στοιχείο του παιχνιδιού που ”παράγει” ζημιά είναι το όπλο, το οποίο με τη σειρά του είναι ένα αντικείμενο (βλ. στη συνέχεια). Τα όπλα μπορούν να παράγουν από έναν ως και τρεις διαφορετικούς τύπους ζημιάς, έτσι ώστε ο κάθε τύπος ζημιάς να έχει τα δικά του χαρακτηριστικά. Το ποσό του damage που παράγει ένα όπλο για κάθε κατηγορία καθορίζεται από ένα ζάρι (αντίστοιχο με αυτά που χρησιμοποιήθηκαν στην άσκηση 2). Αν θα θέλαμε να περιγράψουμε τη συμπεριφορά ενός όπλου, θα μπορούσαμε να πούμε ότι όταν το χρησιμοποιούμε για να επιτεθούμε, μας επιστρέφει μια ”λίστα” με το ποσό του damage για κάθε τύπο damage. Μια αφηρημένη περιγραφή θα μπορούσε να είναι:

```
public List<DamageTypeAmount> getDamage() { ... }
```

όπου το `DamageTypeAmount` περιγράφει το damage type και το damage amount (δηλαδή το αποτέλεσμα της ρίψης των ζαριών που περιγράφουν το συγκεκριμένο damage type). Προσοχή: τίποτα δεν μας απαγορεύει να έχουμε δύο ή περισσότερες φορές το ίδιο damage type στη λίστα. Όπως προκύπτει από τα παραπάνω, για κάθε όπλο, θα πρέπει να διατηρούμε τουλάχιστον στοιχεία για το damage type και το ζάρι το οποίο την παράγει.

Τέλος, τα όπλα του παίκτη είναι ταυτόχρονα και αντικείμενα (τα οποία καταλαμβάνουν μια από τις θέσεις στις οποίες μπορεί να εξοπλίσει αντικείμενα ο παίκτης), το οποίο σημαίνει ότι ενδέχεται να επηρρεάζουν τα υπόλοιπα στατιστικά του (π.χ. strength και intellect). Τα όπλα των εχθρών μπορούν να μοντελοποιηθούν αντίστοιχα, ωστόσο οι εχθροί δεν έχουν strength και intellect [4].

4.2 Χαρακτήρας του παίκτη, κλάση, φυλή και πόντοι εμπειρίας

Θεωρούμε ότι ο χαρακτήρας που ελέγχει ο παίκτης μπορεί να ανήκει σε κάποια από τις ακόλουθες φυλές: `ORC`, `HUMAN`, `ELF`, `TAUREN`. Κάθε φυλή έχει διαφορετικά χαρακτηριστικά, όσον αφορά τα στατιστικά του παίκτη, καθώς και κυρίως την ανοχή του στα διάφορα είδη damage.

Αντίστοιχα, ο χαρακτήρας που ελέγχει ο παίκτης μπορεί να ακολουθεί ένα από 3 διαφορετικά πρότυπα μαχητή (κλάση), πιο συγκεκριμένα `MAGE`, `WARRIOR` ή `BATTLEMAGE`.

Τα στοιχεία της φυλής και της κλάσης συνθέτουν τα βασικά στατιστικά στοιχεία του παίκτη. Επιπλέον, η κλάση του χαρακτήρα μας περιγράφει πως εξελίσσονται τα στατιστικά στοιχεία αυτά καθώς ο χαρακτήρας προχωράει στο παιχνίδι, νικάει αντιπάλους και συλλέγει εμπειρία.

4.2.1 Πόντοι εμπειρίας / Exprience 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 συμπληρωνονται 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 μέ έναν συγκεκριμένο τρόπο.

[4] καλό θα είναι τα όπλα του παίκτη που υπάρχουν στο παιχνίδι, είτε αυτά παράγονται τυχαία, είτε είναι φτιαγμένα από πριν, να έχουν τουλάχιστον 2 damage types, δεδομένου ότι κάποιοι αντίπαλοι έχουν πολύ μεγάλο defense σε ένα είδος ζημιάς

4.2.2 Εξέλιξη χαρακτήρων

Οι πίνακες 1, 2, 3, 4 περιγράφουν την εξέλιξη των χαρακτήρων, καθώς αυτοί αποκτούν εμπειρία και προχωράνε στο παιχνίδι.

Πίνακας 1: Σχέση XP / επιπέδων

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

Πίνακας 2: Σχέση επιπέδων / στατιστικών για την κλάση ”Πολεμιστή”

Level	Bonus HP	Bonus MP	Bonus Str	Bonus Int
1	+10	0	2	0
2	+15	0	4	0
3	+20	0	6	0
4	+25	0	8	0
5	+30	0	10	0
6	+40	0	12	0

Πίνακας 3: Σχέση επιπέδων / στατιστικών για την κλάση ”Μάγου”

Level	Bonus HP	Bonus MP	Bonus Str	Bonus Int
1	8	20	2	8
2	12	40	3	16
3	16	60	4	24
4	20	80	5	32
5	24	100	6	40
6	30	120	7	48

4.2.3 Στατιστικά φυλών

Τα στατιστικά αυτά προστίθενται στα στατιστικά της κλάσης και του επιπέδου του παίκτη. Τα ? στους πίνακες σημαίνει πρακτικά ότι οι τιμές εξαρτώνται από την εκάστοτε κλάση.

4.2.4 Τελικά στατιστικά του χαρακτήρα του παίκτη

Τα τελικά στατιστικά του χαρακτήρα του παίκτη προκύπτουν σαν το άθροισμα των παρακάτω στοιχείων:

Πίνακας 4: Σχέση επιπέδων / στατιστικών για την κλάση ”Μάγου-Πολεμιστή”

Level	Bonus HP	Bonus MP	Bonus Str	Bonus Int
1	10	10	5	5
2	13	20	7	9
3	16	30	9	13
4	20	40	11	17
5	28	50	13	21
6	40	700	15	25

Πίνακας 5: Αρχικά στατιστικά των ORCs

Str	Int	Def: Blunt	Def: Slashing	Def: Magic	HP	MP
10	8	1	1	0	?	?

Πίνακας 6: Αρχικά στατιστικά των TAURENs

Str	Int	Def: Blunt	Def: Slashing	Def: Magic	HP	MP
12	6	1	2	0	?	?

Πίνακας 7: Αρχικά στατιστικά των HUMANs

Str	Int	Def: Blunt	Def: Slashing	Def: Magic	HP	MP
9	9	1	1	0	?	?

Πίνακας 8: Αρχικά στατιστικά των ELFs

Str	Int	Def: Blunt	Def: Slashing	Def: Magic	HP	MP
6	12	0	1	2	?	?

- Στατιστικά Κλάσης + επιπλέον στατιστικά επιπέδου
- Στατιστικά φυλής
- Bonuses από όπλα και αντικείμενα

Συνοπτικά έχουμε:

- Maximum Hit Points: αλλάζουν με το επίπεδο. Προκύπτουν σαν το άθροισμα των Base HP και bonus από όπλα / ασπίδες
- 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.

4.2.5 Επιθέσεις με όπλα / Damage

- Για να υπολογίσουμε το αποτέλεσμα μιας επίθεσης αρχικά υπολογίζουμε το συνολικό strength και intellect του παίκτη.
- Στη συνέχεια παίρνουμε από το όπλο τη λίστα με τα διάφορα damage type και damage amount
  - Στην περίπτωση που έχουμε παραπάνω από μια φορές το ίδιο damage type, αθροίζουμε ως προς το damage type
- Προσθέτουμε το strength στα damage types `BLUNT` και `SLASHING` (μπορούμε είτε να το προσθέσουμε και στα δύο είτε μόνο στο ένα)
- Προσθέτουμε το intellect στο damage type `MAGICAL`
- Δημιουργούμε μια τελική λίστα με `DamageTypeAmount`, την οποία και χρησιμοποιούμε για να προκαλέσουμε Damage στον εχθρό

4.2.6 Επιθέσεις με spells / Casting

Οι κλάσεις Mage και Battle Mage έχουν το επιπλέον γνώρισμα ότι μπορούν να εκτελέσουν ”ξόρκια”. Κάθε ξόρκι χρησιμοποιεί ένα σταθερό ποσό από Mana Points (5) και έχει το πλεονέκτημα ότι δεν χρησιμοποιεί ζάρια για τον υπολογισμό του Damage. Για τον υπολογισμό του Damage που προκαλεί ένα ξόρκι:

- Υπολογίζω το συνολικό intellect του παίκτη

- Προσθέτω τα bonuses από όπλα και αντικείμενα
- Δημιουργώ ένα `DamageTypeAmount` με Damage type `MAGICAL` και amount ίσο με το παραπάνω άθροισμα.

4.2.7 Ξεκούραση

Αν ο παίκτης επιλέξει να ξεκουραστεί για έναν γύρο, τότε ανακτά 5 HP και 5 MP. Τόσο με την ξεκούραση, όσο και με τη χρήση κάποιου potion, τα hit points και mana points δεν μπορούν ποτέ να ξεπεράσουν τις μέγιστες τιμές που προκύπτουν από το άθροισμα "κλάση + φυλή + bonuses".

4.2.8 Αντικείμενα του παίκτη

Όπως θα δούμε και στη συνέχεια έχουμε 2 βασικούς τύπους αντικειμένων, αυτά που μπορεί να εξοπλίσει ο παίκτης και αυτά που μπορεί να χρησιμοποιήσει.

Τα αντικείμενα που μπορεί να εξοπλίσει μπαίνουν σε συγκεκριμένες θέσεις: `MAIN_HAND`, `OFF_HAND`, `FINGER`, `NECK`. Επειδή ο παίκτης πρέπει να μπορεί παντα να εκτελέσει μια επίθεση, θα πρέπει (τουλάχιστον) το αντικείμενο που εξοπλίζεται στο `MAIN_HAND` να είναι τύπου `Weapon`.

Τα αντικείμενα που μπορεί να χρησιμοποιήσει μπαίνουν απλά σε μια συλλογή (inventory). Ανάλογα με την προσέγγιση που θα ακολουθήσετε, μπορείτε είτε να υλοποιήσετε κάτι γενικό ("χρησιμοποίησε ένα healing potion", "χρησιμοποίησε ένα potion που έχει `HP_REPLENISH`") ή κάτι εξειδικευμένο ("χρησιμοποίησε το αντικείμενο στη θέση 5 του inventory" – συνεπάγεται ότι θα παρέχετε κάποιον τρόπο να μπορεί να δει ο παίκτης τα αντικείμενα και τις θέσεις τους). Όταν ένα αντικείμενο εξαντλήσει όλες τις χρήσεις του αφαιρείται αυτόματα από το inventory.

4.2.9 Στο παιχνίδι…

Μπορείτε να μοντελοποιήσετε τα παραπάνω με διάφορους τρόπους, είτε με ιεραρχίες κλάσεων (`AbstractPlayer` → `AbstractMage` → `OrcMage`) χρησιμοποιώντας abstract μεθόδους για να υλοποιήσετε τη λειτουργικότητα της φυλής και της κλάσης, είτε να χρησιμοποιήσετε `enums`, τα οποία να δίνουν τα βασικά στατιστικά για κάθε περίπτωση. Μπορείτε να χρησιμοποιήσετε όποια προσέγγιση θέλετε, στο τέλος όμως θα πρέπει να δημιουργήσετε (τουλάχιστον) τους ακόλουθους συνδυασμούς:

- Orc Mage
- Human BattleMage
- Tauren Warrior

- Elf Mage

4.3 Αντίπαλοι

Τα βασικά χαρακτηριστικά των αντιπάλων στο παιχνίδι είναι το όνομά τους, τα hit points τους, το damage που προκαλούν (το οποίο προέρχεται από το όπλο με το οποίο είναι εξοπλισμένοι), την ανοχή τους στους διάφορους τύπους damage καθώς και τα experience points που κερδίζει ο παίκτης όταν τους νικήσει.

Οι αντίπαλοι, όντας κάτοικοι του λαβυρίνθου, δημιουργούνται σε μια τυχαία ελεύθερη θέση και κινούνται προς τον παίκτη.

Όταν ηττηθούν, αφήνουν στο πλακίδιο που βρίσκονταν τη στιγμή που πέθαναν κάποια αντικείμενα (φίλτρα, όπλα και βοηθητικά αντικείμενα).

4.3.1 Παραδείγματα αντιπάλων

Οι παρακάτω αντίπαλοι έχουν υλοποιηθεί στην πρότυπη άσκηση και θα δοθούν αντίστοιχα sprites ώστε να μπορείτε να τους σχεδιάσετε:

Name	Hit Points	Weapon	Resistances	XP	Pl. Lvls
Slime	20	”Blob of Slime” (1d6+2 B)	Immune: Slashing	30	1
Goblin	30	”Crude Sword” (1d6+1 S)	Slashing: /2	60	2,3,4
Ghost	40	”Ghost claws” (1d6+2 M, 1d6+2 S)	Immune: Blunt	100	3,4,5
Skeleton	45	”Femur Bone” (2d6+4 B)	Slashing: /2	120	4,5,6
Sk. Demon	50	”Flaming Skull” (2d6+2 B, 1d6+1 M)	Slashing: /2	200	5,6
Sk. Lord	55	”Unholy Mace” (2d6+2 B, 1d6+2 S)	Slashing: /2	400	5,6
Leoric	120	”Mace of the Fallen Champion”	All: /2	30000	

Το όπλο του Leoric έχει τα εξής χαρακτηριστικά: `2d6+5 BLUNT`, `1d4+2 SLASHING`, `1d4+2 MAGICAL` και τα εξής bonuses: `STR: +10`, `INT: +10`, `MP: +10`. ΠΡΟΣΟΧΗ Ο Leoric είναι εχθρός, επομένως δεν επωφελείται από αυτά τα bonuses, απλά υπάρχουν στο όπλο. Μπορείτε, αν θέλετε, να κάνετε το συγκεκριμένο όπλο equippable (και να είναι το item που ρίχνει ο Leoric όταν χάσει) και σαν τελευταία δοκιμασία να προσθέσετε κάποιους αντιπάλους στο επίπεδο ώστε να εμποδίσετε τον παίκτη να φτάσει στην έξοδο (ή κάτι αντίστοιχο).

Όπως παρατηρείτε στον παραπάνω πίνακα, ο τύπος του εχθρού που θα εμφανιστεί στην πίστα εξαρτάται από το επίπεδο του παίκτη. Υπάρχουν διάφοροι τρόποι με τους οποίους θα μπορούσε να υλοποιηθεί κάτι τέτοιο. Στην πιο απλή περίπτωση, μπορείτε απλά να φτιάχνετε έναν πίνακα με όλους τους εχθρούς που είναι διαθέσιμοι στο επίπεδο και να επιλέγετε τυχαία έναν από αυτούς.

4.3.2 Παράδειγμα υλοποίησης breadth-first search

Το συγκεκριμένο παράδειγμα είναι αρκετά απλό και βασίζεται στις παραδοχές / μοντελοποίηση που έχουν γίνει για την υλοποίηση της πρότυπης εργασίας.

```
// Breadth first search from start to end
public List<Room> tracePathFrom(Room start, Room end) {
    // Inner class to hold the path
    class RoomPath {
public final RoomPath parent;
public final Room current;
public RoomPath(Room current, RoomPath parent) {
    this.current = current;
    this.parent = parent;
}
    }

    LinkedList<RoomPath> scanRooms = new LinkedList<>();
    Set<Room> visited = new HashSet<>();
    RoomPath currentRoom = new RoomPath(start, null);
    scanRooms.addFirst(currentRoom);
    visited.add(start);
    while(currentRoom.current != end) {
currentRoom = scanRooms.removeLast();
for(Room n: getNeighbors(currentRoom.current)) {
    if(! visited.contains(n)) {
visited.add(n);
scanRooms.addFirst(new RoomPath(n, currentRoom));
    }
}
    }
    LinkedList<Room> path = new LinkedList<>();
    while(currentRoom != null) {
path.addFirst(currentRoom.current);
currentRoom = currentRoom.parent;
    }
    return path;
}
```

4.4 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 ”equipables” 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). Στο παιχνίδι θα χρειαστεί (ενδεχομένως, ανάλογα με την προσέγγισή σας) να υλοποιήσετε δυο βασικούς τύπους αντικειμένων, equippable και usable.

Τα αντικείμενα που βρίσκονται πάνω σε ένα πλακίδιο δεν είναι υποχρεωτικό να φαίνονται στον χάρτη ή στο minimap (αν θέλετε, προσθέστε το σαν λειτουργικότητα). Ωστόσο, όταν ο παίκτης περνάει πάνω από κάποιο πλακίδιο που έχει αντικείμενα, το παιχνίδι θα πρέπει να τον ειδοποιεί (μέσω του Transcript, βλ. στη συνέχεια) για τα αντικείμενα που βρίσκονται στο πλακίδιο.

4.4.1 Τύποι αντικειμένων, Στατιστικά αντικειμένων

Όσον αφορά τους τύπους και τα στατιστικά των αντικειμένων, θεωρήστε παρόμοια προσέγγιση με την ’Ασκηση 4, δηλαδή έχουμε items, τα οποία χωρίζονται σε equippables και usables. Κάθε item έχει μια λίστα με effects τα οποία είτε αυξάνουν είτε αναπληρώνουν τα βασικά χαρακτηριστικά του παίκτη. Σαν βασικούς τύπους effects, μπορείτε να έχετε τα `HP_REPLENISH`, `MP_REPLENISH`, `HP_BOOST`, `MP_BOOST`, `STR_BOOST`, `INT_BOOST`.

4.4.2 Usables (potions)

Τα usables είναι τα αντικείμενα που όταν χρησιμοποιούνται παράγουν μια λίστα με item effects, συνήθως σχετικά με την αναπλήρωση των στατιστικών του χρήστη

(αν και ανάλογα με τα mechanics μπορούν να έχουν πιο ευρύ functionality, π.χ. timed-effects, κλπ). Ο αριθμός των χρήσεων μπορεί να είναι περιορισμένος.

Για τους σκοπούς του παιχνιδιού δημιουργήστε 2 τουλάχιστον είδη από usable items, ένα για την αναπλήρωση των hit points (Health Potion) και ένα για την αναπλήρωση των mana points (Mana Potion). Τα items μπορούν να έχουν όσες χρήσεις επιθυμείτε (π.χ. 2 ή 3). Ενδεικτικά, κάντε το Health Potion να αναπληρώνει 30 hit points και το Mana Potion 40 mana points. Αν θέλετε, μπορείτε να φτιάξετε και usable items με μικτά αποτελέσματα, ή ακόμα και generators για random potions.

Όσον αφορά τη χρήση των usables από τον χρήστη, μια ενδεικτική προσέγγιση θα ήταν:

- Καλώ μια μέθοδο ”use()” η οποία μου επιστρέφει τα item effects (ή άδεια λίστα αν έχουν εξαντληθεί οι χρήσεις)
- Εφαρμόζω τις αλλαγές που περιγράφουν τα effects στον χαρακτήρα.

Παρατήρηση: Σκεφτείτε πως θα μπορούσατε να φτιάξετε ”poison pools”, δηλαδή items που γίνονται ”auto-use” και προκαλούν damage στον χαρακτήρα.

Παρατήρηση - ”Αντικείμενα Παντού”: αν αναλογιστούμε ότι και τα damage types θα μπορουσαν να είναι item effects, μπορούμε να δούμε ότι ίσως τα πάντα θα μπορούσαν να είναι διάφοροι τύποι αντικειμένων, π.χ. τα όπλα, τα φίλτρα, τα spells (usables με άπειρο πλήθος χρήσεων). Δεν χρειάζεται (και δε συστήνεται) να χρησιμοποιήσετε αυτή την προσέγγιση, είναι ωστόσο κάτι για σκέψη.

4.4.3 Equipables

Ο χαρακτήρας που χειρίζεται ο παίκτης έχει 4 slots με διαφορετικούς τύπους για να εξοπλίζει αντικείμενα (ανεξάρτητα από κλάση ή φυλή)

- `MAIN_HAND` πρέπει να είναι ταυτόχρονα και όπλο
- `OFF_HAND` πρέπει να είναι ταυτόχρονα και όπλο
- `FINGER`
- `NECK`

Όταν κάνει equip ένα αντικείμενο που βρίσκεται στο πλακίδιο που είναι ο παίκτης, τότε το αντικείμενο του πλακιδίου γίνεται equip, και το αντικείμενο του slot μεταφέρεται στο πλακίδιο [5].

Καλό θα ήταν να έχετε τουλάχιστον 5-6 διαφορετικούς τύπους όπλων και 2-3 τύπους για κάθε slot. Αν θέλετε, μπορείτε να φτιάξετε μια γεννήτρια τυχαίων

[5] αρχή διατήρησης των αντικειμένων

αντικειμένων, ακολουθώντας, π.χ. τον παρακάτω κώδικα, ή χρησιμοποιώντας όποια προσέγγιση θέλετε.

```
public class ItemGenerator {
    // ..
    public enum Rarity {
NONE(0),
FEEBLE(1),
COMMON(4),
RARE(8),
SUPREME(12);
private final int totalBonus;
private Rarity(int totalBonus) {
    this.totalBonus = totalBonus;
}
    };


// ...
    static String[] namePrefix = {"laughing", "deadly", "eternal",
  "amphibious", "improbable",
  "reckless", "smiting", "boring",
  "mysterious"};

// ...

    static Map<SlotType, List<String>> slotsMap = Map.of(
      SlotType.NECK, List.of("Necklace", "Scarf", "Amulet",
"Necktie", "Bowtie"),
      SlotType.FINGER, List.of("Ring", "Mittens", "Glove"),
      SlotType.MAIN_HAND, List.of("Sword", "Katana", "Staff",
"Axe", "Mace", "Halberd", "Cutlass", "Morningstar"),
      SlotType.OFF_HAND, List.of("Short Sword", "Dagger", "Baton",
"Spiked Skull")
    );

// ...

    static String[] nameSuffix = {"of Doom", "of Pestilence",
  "of the Dead", "of Mythos",
```

```
  "of the clan McCloud",
  "of the Stars"};

    public static String nameGenerator(Rarity rarity, SlotType slotType) {
List<String> slotNames = slotsMap.get(slotType);
String slotName = slotNames.get(rng.nextInt(slotNames.size()));
String rarityName;
if(rarity == Rarity.NONE) {
    rarityName = "";
} else {
    rarityName = rarity.toString().toLowerCase() + " ";
}
return new StringBuilder(rarityName)
.append(namePrefix[rng.nextInt(namePrefix.length)])
.append(" ")
.append(slotName)
.append(" ")
.append(nameSuffix[rng.nextInt(nameSuffix.length)])
.toString();
    }

// ...

    public static Equippable randomEquippable() {
// None, Feeble, Common, Rare, Supreme
Rarity[] rarityProbabilities =
  {Rarity.NONE, Rarity.NONE, Rarity.NONE, Rarity.NONE,
   Rarity.NONE, Rarity.FEEBLE, Rarity.FEEBLE, Rarity.FEEBLE,
   Rarity.FEEBLE, Rarity.FEEBLE, Rarity.COMMON,
   Rarity.COMMON, Rarity.COMMON, Rarity.COMMON,
   Rarity.RARE, Rarity.RARE, Rarity.RARE,
   Rarity.SUPREME};

Rarity rarity = rarityProbabilities[rng.nextInt(rarityProbabilities
    .length)];
SlotType slotType = SlotType.values()[rng.nextInt(SlotType
    .values().length)];
boolean isWeapon = false;
if(slotType == SlotType.MAIN_HAND || slotType == SlotType.OFF_HAND) {
    isWeapon = true;
```

```
}
String itemName = nameGenerator(rarity, slotType);
// we have to split the bonus points of the item into 1-3 of the possible b

int totalBonus = rarity.totalBonus;
// HP_BONUS, MP_BONUS, STR_BONUS, INT_BONUS
int[] bonusDiv = {0, 0, 0, 0};
EffectType[] bonusItm = {EffectType.HP_BONUS, EffectType.MP_BONUS,
 EffectType.STR_BONUS, EffectType.INT_BONUS};

while(totalBonus > 0) {
    bonusDiv[rng.nextInt(bonusDiv.length)] += 1;
    totalBonus = totalBonus - 1;
}

List<ItemEffect> itemBonuses = new ArrayList<>();
for(int i = 0; i < bonusDiv.length; i++) {
    if(bonusDiv[i] > 0) {
itemBonuses.add(new ItemEffect(bonusItm[i], bonusDiv[i]));
    }
}

if(isWeapon) {
    // If this is a weapon, then we have the attack AND the bonus
    return new Weapon(itemName) {
// Implementation here
    };
} else {
    return new Equippable() {
// Implementation here
    };
}
    }
```

5 Γραφικά, χειρισμός

Το βασικό user interface του παιχνιδιού φαίνεται στην εικόνα 1. Το κύριο μέρος του user interface αποτελείται από 2 views του χάρτη του επιπέδου, έναν σε προοπτική πρώτου προσώπου και ένα mini map που δείχνει την τρέχουσα θέση του παίκτη

στον χάρτη και το τμήμα του χάρτη που έχει εξερυνήσει[6]. Και οι δύο αυτοί χάρτες έχουν διαστάσεις 256×256 pixels. Οι μέθοδοι για τον σχεδιασμό των χαρτών που δίνονται στη συνέχεια, καθώς και οι εικόνες που δίνονται σαν resources είναι προσαρμοσμένες για αυτές τις διαστάσεις. Μπορείτε ωστόσο να τροποποιήσετε τη σχεδίαση κατά πως επιθυμείτε.

5.1 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.

Αν θέλετε, επειδή το `JLabel` μπορεί να δεχτεί σαν κείμενο απλή `HTML` και να την μορφοποιήσει ανάλογα, μπορείτε να ακολουθήσετε αυτή την προσέγγιση, π.χ.

```
StringBuilder pstate =  new StringBuilder("<html>");
pstate.append(player.getName()).append("<br/>");
pstate.append(player.getPlayerClass()).append(" / " )
  .append(player.getPlayerRace()).append("<br/>");

pstate.append(player.getLevel()).append("/").
  append(player.getExperiencePoints()).append("<br/>");
pstate.append("<hr/>");

pstate.append("HP: ").append(player.getHitPoints())
    .append("/").append(player.getMaxHitPoints()).append("<br/>");

pstate.append("MP: ").append(player.getManaPoints())
  .append("/").append(player.getMaxManaPoints()).append("<br/>");
pstate.append("Str: ").append(player.getStrength()).append("<br/>");
pstate.append("Int: ").append(player.getIntellect()).append("<br/>");
pstate.append("<hr/>");

pstate.append("Items<br>");
for(Equippable e: player.getEquippedItems().values()) {
  pstate.append(e.getSlotType()).append(": ").append(e.getName()).append("
```

[6]σε παλιότερα games δεν υπήρχαν mini maps και οι παίκτες σχεδιάζαν τον χάρτη σε χαρτί "μιλλιμετρέ"

```
}
pstate.append("<hr/>");

Map<String, Integer> potionStats = new HashMap<>();
for(Usable u: player.getInventory()) {
  if(u instanceof HealingPotion) {
    potionStats.put("Healing", potionStats.getOrDefault("Healing", 0) + 1);
  } else if(u instanceof ManaPotion) {
    potionStats.put("Mana", potionStats.getOrDefault("Mana", 0) + 1);
  }
}

pstate.append("Healing Potions: ")
  .append(potionStats.getOrDefault("Healing", 0))
  .append("<br/>");

pstate.append("Mana Potions: ")
  .append(potionStats.getOrDefault("Mana", 0))
  .append("<br/>");

playerInfo.setText(pstate.toString());
```

Μπορείτε ακόμα και να χρησιμοποιήσετε ένα `TextArea`, όπως στο Game Log και να ανανεώνετε το κείμενο που εμφανίζεται.

Ο πιο απλός τρόπος για να είναι πάντα ενημερωμένη η κατάσταση του παίκτη είναι να χρησιμοποιήσετε ένα Observer patern με Subject τον Player και Observer το frame του παιχνιδιού. Για κάθε action του παίκτη μπορείτε να καλείτε την αντίστοιχη `notifyObservers`.

5.2 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 picked up a XYZ potion`. Αντίστοιχα, αν στο πλακίδιο υπάρχει κάποιο όπλο `you see ...`. 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 υπάρχει ένα slime σε απόσταση 5 πλακιδίων "`you hear bubbling sounds nearby`").

Μπορείτε να χρησιμοποιήσετε οποιαδήποτε μέθοδο για να ανανεώνονται τα παραπάνω συνεχώς, π.χ. Observer pattern, Events, κλπ. Μια απλή μέθοδος με δύο interfaces για το logging θα μπορούσε να είναι η ακόλουθη.

```
public interface TranscriptWriter {
  void setTranscriptReader(TranscriptReader r);
}

public interface TranscriptReader {
    void writeTranscript(String s);
}
```

Τέλος, για να προσθέσετε μια ScrollBar στο TextArea, μπορείτε να κάνετε το εξής:

```
statusText = new JTextArea(8, 80);
statusText.setEditable(false);
JScrollPane scrollPane = new JScrollPane(statusText);
```

ενώ για να μετακινείται η ScrollBar πάντα στο τέλος του κειμένου, προσθέστε τα ακόλουθα στο σημείο που προστίθεται κείμενο στο `statusText`.

```
JScrollBar jsb = scrollPane.getVerticalScrollBar();
jsb.setValue(jsb.getMaximum());
```

5.3 Χειρισμός

Ο χειρισμός του παιχνιδιού γίνεται με το πληκτρολόγιο. Το πιο απλό είναι να χρησιμοποιήσετε έναν `keyListener` και να αντιστοιχίσετε τα πλήκτρα σε actions του παίκτη, π.χ.

```
mp.addKeyListener(new KeyAdapter() {
@Override
public void keyTyped(KeyEvent e) {
    char c = e.getKeyChar();
    switch (c) {
    case 'a':
game.turnLeft();
break;
    case 'd':
game.turnRight();
break;
```

```
    case 'w':
game.move();
MapRoom r = (MapRoom) game.getPlayerRoom();
r.visitRoom();
break;
    case 'x':
game.attack();
break;
// ...

    default:
writeTranscript("I cannot do that, David");
    }
}
    });
```

Ένα πρόβλημα με την παραπάνω προσέγγιση είναι ότι οι `keyListeners` δέχονται events μόνο όταν το παράθυρο / component στο οποίο έχουν προστεθεί έχει το `focus`. Ένας τρόπος να το εξασφαλίσετε είναι ο ακόλουθος: Αν `mp` είναι το, π.χ. MapPanel το οποίο θέλουμε να δέχεται το input και `this` το JFrame

```
// στον constructor του JFrame

// αρχικό focus στο map panel
mp.requestFocus();

// listener, ώστε να το επαναφέρει -- έχει προβληματάκια, δηλαδη
// πρέπει να κάνετε click σε ένα άλλο παράθυρο και μετά στο Frame του
// παιχνιδιού, αλλά είναι η πιο απλή προσέγγιση
this.addWindowFocusListener(new WindowAdapter() {
public void windowGainedFocus(WindowEvent e) {
    mp.requestFocusInWindow();
}
    });
```

5.4 Σχεδίαση του mini map

Η σχεδίαση του minimap που ακολουθεί βασίζεται στην υπόθεση ότι υπάρχει μια κλάση GameMap, η οποία έχει μεθόδους που επιστρέφουν τις διαστάσεις του

λαβυρίνθου και μπορούν να μας επιστρέψουν ένα πλακίδιο / δωμάτιο που βρίσκεται σε μια συγκεκριμένη θέση.

Η μέθοδος θα μπορούσε να είναι σε μια class που κάνει extend το, π.χ. JPanel

```
public void paintMiniMap(GameMap map, Room curRoom, Direction d) {
    Graphics2D g = (Graphics2D) getGraphics();
    int offs = 3;
    g.setColor(Color.BLACK);
    g.fillRect(0, 0, width, height);
    Color vTile = new Color(130, 130, 130);
    Color vWall = new Color(200, 120, 120);
    Color dColor = new Color(255, 255, 255);
    for(int col = 0; col < map.getMapWidth(); col++) {
for(int row = 0; row < map.getMapHeight(); row++) {
    MapRoom room = (MapRoom)map.getRoomAt(row, col);
    if(room.isVisited()) {
g.setColor(vTile);
g.fillRect(offs + col*tileWidth, offs+row*tileHeight,
   tileWidth, tileHeight);
g.setColor(vWall);
if(room.getRoomAt(Direction.NORTH) == null) {
    g.fillRect(offs + col*tileWidth, offs+row*tileHeight,
       tileWidth, 2);
}
if(room.getRoomAt(Direction.SOUTH) == null) {
    g.fillRect(offs + col*tileWidth,
       offs+row*tileHeight + tileHeight - 2,
       tileWidth, 2);
}
if(room.getRoomAt(Direction.EAST) == null) {
    g.fillRect(offs + col*tileWidth+tileWidth-2,
       offs+row*tileHeight, 2, tileHeight);
}
if(room.getRoomAt(Direction.WEST) == null) {
    g.fillRect(offs + col*tileWidth, offs+row*tileHeight,
       2, tileHeight);
}
    }
    if(room == curRoom) {
g.setColor(dColor);
```

```
int ec = 4;
int xcent = 0, ycent = 0;
if(d == Direction.NORTH) {
    xcent = offs + col*tileWidth + tileWidth/2 - ec/2;
    ycent = offs + row*tileHeight;
} else if(d == Direction.SOUTH) {
    xcent = offs + col*tileWidth + tileWidth/2 - ec/2;
    ycent = offs + row*tileHeight + tileHeight - ec;
} else if(d == Direction.EAST) {
    xcent = offs + col*tileWidth + tileWidth - ec;
    ycent = offs + row*tileHeight + tileHeight/2 - ec/2;
} else if(d == Direction.WEST) {
    xcent = offs + col*tileWidth;
    ycent = offs + row*tileHeight + tileHeight/2 - ec/2;
}
g.fillArc(xcent, ycent, ec, ec, 0, 180);
    }
}
    }
}
```

5.5 Σχεδίαση της προοπτικής πρώτου προσώπου

Επισυνάπτεται ολόκληρη η κλάση, κυρίως για διευκόλυνση, όσον αφορά τα ονόματα και τις θέσεις των images. Η λειτουργία της κλάσης είναι απλή: Ξεκινώντας από το δωμάτιο που δίνεται και προχωρώντας προς τη διεύθυνση που κοιτάει ο χρήστης ζωγραφίζει τους τοίχους και τα πατώματα, ανάλογα με το τι βρίσκεται στις διαφορετικές διευθύνσεις για 2 διαδοχικά δωμάτια. Στη συνέχεια, ζωγραφίζει τους αντίπαλους που τυχόν βρίσκονται στα δωμάτια αυτά. Παρ' όλο που είναι γραμμένη με βάση τις υποθέσεις που έχουν γίνει για την ενδεικτική λύση της άσκησης, τα τμήματα της σχεδίασης μπορούν (λογικά) να μεταφερθούν σε οποιαδήποτε υλοποίηση.

```
public class MapPanel extends JPanel {
    // Images for drawing

    Map<String, BufferedImage> readImages() {
try {
    Map<String, BufferedImage> ret = new HashMap<>();
    ret.put("wall_near",
      ImageIO.read(new File("images/front_near256x256.png")));
```

```
        ret.put("wall_far",
          ImageIO.read(new File("images/front_near256x256.png")));
        ret.put("left_wall_near",
          ImageIO.read(new File("images/left_wall_near64x256.png")));
        ret.put("right_wall_near",
          ImageIO.read(new File("images/right_wall_near64x256.png")));
        ret.put("left_wall_far",
          ImageIO.read(new File("images/left_wall_far32x128.png")));
        ret.put("right_wall_far",
          ImageIO.read(new File("images/right_wall_far32x128.png")));
        ret.put("bottom_near",
          ImageIO.read(new File("images/bottom_near256x64.png")));
        ret.put("bottom_far",
          ImageIO.read(new File("images/bottom_far128x32.png")));
        ret.put("left_door_near",
          ImageIO.read(new File("images/left_door_near64x256.png")));
        ret.put("right_door_near",
          ImageIO.read(new File("images/right_door_near64x256.png")));
        ret.put("left_door_far",
          ImageIO.read(new File("images/left_door_far32x128.png")));
        ret.put("right_door_far",
          ImageIO.read(new File("images/right_door_far32x128.png")));

        return ret;
    } catch(IOException ioex) {
        System.out.println("could not read file");
        System.out.println(ioex);
        return null;
    }
        }


        final int width = 256;
        final int height = 256;

        Map<String, BufferedImage> wallImages;

        Room currentRoom;

        public MapPanel() {
```

```
setPreferredSize(new Dimension(width, height));
wallImages = readImages();
currentRoom = null;
    }

    public BufferedImage paintEnemyClose(BufferedImage i, Enemy e) {
Graphics2D g = (Graphics2D)i.getGraphics();
g.drawImage(e.getSprite(), 128-64, 256-128, 128, 128, null);
return i;
    }

    public BufferedImage paintEnemyFar(BufferedImage i, Enemy e) {
Graphics2D g = (Graphics2D)i.getGraphics();
g.drawImage(e.getSprite(), 128-32, 256-128, 64, 64, null);
return i;
    }

    public BufferedImage getImage(Room r, Direction d) {
BufferedImage b = new BufferedImage(width, height,
  BufferedImage.TYPE_4BYTE_ABGR);
Graphics2D g = (Graphics2D) b.getGraphics();
g.setColor(Color.BLACK);
g.fillRect(0, 0, width, height);

if(r.getRoomAt(d) == null) {
    g.drawImage(wallImages.get("wall_near"), 0, 0, width,height, null);
    return b;
}

// draw floor of next room as well as left and right
g.drawImage(wallImages.get("bottom_near"), 0, 191, null);

Room rN = r.getRoomAt(d);

if(rN.getRoomAt(d.left()) == null) {
    g.drawImage(wallImages.get("left_wall_near"), 0, 0, null);
} else {
    // draw a door
    g.drawImage(wallImages.get("left_door_near"), 0, 0, null);
}
```

```
if(rN.getRoomAt(d.right()) == null) {
    g.drawImage(wallImages.get("right_wall_near"), 191, 0, null);
} else {
    // draw a door
    g.drawImage(wallImages.get("right_door_near"), 191, 0, null);
}

// far room
Room rNN = rN.getRoomAt(d);
if(rNN == null) {
    g.drawImage(wallImages.get("wall_far"), 64, 64, null);
    return b;
}

// draw floor of next next room
g.drawImage(wallImages.get("bottom_far"), 63, 128+32-1, null);

if(rNN.getRoomAt(d.left()) == null) {
    g.drawImage(wallImages.get("left_wall_far"), 63, 63, null);
} else {
    // draw a door
    g.drawImage(wallImages.get("left_door_far"), 63, 63, null);
}

if(rNN.getRoomAt(d.right()) == null) {
    g.drawImage(wallImages.get("right_wall_far"), 128+32-1, 63, null);
} else {
    // draw a door
    g.drawImage(wallImages.get("right_door_far"), 128+32-1, 63, null);
}

if(rNN.getRoomAt(d) == null) {
    g.drawImage(wallImages.get("wall_far"), 95, 95, 64, 64, null);
}

return b;
    }

    void paintMap(Room r, Direction d, Map<Room, Enemy> enemyLocations) {
```

```
Graphics2D g = (Graphics2D) getGraphics();
BufferedImage b = getImage(r, d);
// b = paintEnemyFar(b, new Goblin());
Room rn = r.getRoomAt(d);
if(enemyLocations.get(rn) != null) {
    paintEnemyClose(b, enemyLocations.get(rn));
    Room rnn = rn.getRoomAt(d);
    if(enemyLocations.get(rnn) != null) {
paintEnemyFar(b, enemyLocations.get(rn));
    }

}
g.drawImage(b, 0, 0, null);
    }
}
```

5.5.1 Προσθήκη νέων sprites αντιπάλων

Αν θέλετε να προσθέσετε κάποιον νέο αντίπαλο και να του δώσετε ένα Sprite, το μόνο που πρέπει να προσέξετε είναι η εικόνα του sprite να είναι τετράγωνη (64×64 είναι ένα καλό μέγεθος) και το background να είναι transparent.

5.5.2 Γραφικά για το παιχνίδι

Μπορείτε να κατεβάσετε το `.zip` με τα sprites από το `https://nvlass.com/game2023-sprites.zip`