CS 246—Assignment 5, Group Project (Spring 2016)

R. Hackman B. Lushman

Due Date 1: Friday, July 15, 5pm
Due Date 2: Monday, July 25, 11:59pm

DO NOT EVER SUBMIT TO MARMOSET WITHOUT COMPILING AND TESTING FIRST. If your final submission doesn't compile, or otherwise doesn't work, you will have nothing to show during your demo. Resist the temptation to make last-minute changes. They probably aren't worth it.

This project is intended to be doable by three people in two weeks. Because the breadth of students' abilities in this course is quite wide, exactly what constitutes two weeks' worth of work for three students is difficult to nail down. Some groups will finish quickly; others won't finish at all. We will attempt to grade this assignment in a way that addresses both ends of the spectrum. You should be able to pass the assignment with only a modest portion of your program working. Then, if you want a higher mark, it will take more than a proportionally higher effort to achieve, the higher you go. A perfect score will require a complete implementation. If you finish the entire program early, you can add extra features for a few extra marks.

Above all, MAKE SURE YOUR SUBMITTED PROGRAM RUNS. The markers do not have time to examine source code and give partial correctness marks for non-working programs. So, no matter what, even if your program doesn't work properly, make sure it at least does something.

The Game of ChamberCrawler3000

In this project, you will produce the video game ChamberCrawler3000 (CC3k), which is a simplified rogue-like (a genre of video game based upon the game Rogue[1] - `http://en.wikipedia.org/wiki/Rogue_(video_game)`).

A game of CC3k, consists of a board 79 columns wide and 30 rows high (5 rows are reserved for displaying information). Game play is as follows: The player character moves through a dungeon and slays enemies and collects treasure until reaching the end of the dungeon (where the end of the dungeon is the 5th floor). A dungeon consists of different floors which consist of chambers connected with passages. In our simplification, each floor will always consist of the same 5 chambers connected in the exact same way.

CC3k differs from other rogue-likes in a significant way: it is does not update the terminal/window in real-time but rather redraws all elements every turn (e.g. after every command). Although, many early rogue-likes were programmed in a similar fashion to CC3k.

Some Definitions

It is understandable that this type of game may be new to readers of this document. Accordingly, some definitions are provided here to aid in the reading of this document.

Definition 1: A **character** is a person/animal/thing in the game of CC3k. This can be either the player character (PC), who is controlled by by the player of the game, or non-playable characters, who are strictly enemies in CC3k.

[1] Which is itself based on Dungeons and Dragons

Definition 2: An **item** is something the player character can pick up or use. In CC3k, this is either `gold` or `potions`. Potions offer potentially positive and negative effects to the player character.

Definition 3: A **chamber** is an individual room in the game of CC3k. Chambers are connected by **passages.**

Definition 4: A **floor** in CC3k is a predefined configuration of 5 chambers with connecting passageways. Figure 1 depicts an empty floor. Note that the configuration is the same for every floor in a game of CC3k.

Definition 5: **Health Points (HP)** is the representation of a character's health (both enemies and the player character). When a character's HP reaches 0, they are slain. For an enemy this means that they are removed from the floor and a tidy sum of gold is given to the player character. When the player character has 0 HP then the current game ends.

Definition 6: **Attack (Atk)** is the representation of a character's strength. This is how hard a character can hit another character. Though in CC3k conflict is solely between the player character and non-playable characters.

Definition 7: **Defense (Def)** is the representation of a character's toughness. This is how hard a character can be hit by another character.

Definition 8: A **cell** is either a wall, floor tile, doorway, or passage.

Definition 9: Something is **spawned** means that the particular something (an enemy, gold, etc) should be generated and placed on the board.

Definition 10: A **1 block radius** denotes the 8 adjacent cells to the character or item.

System Components

The major components of the system are as follows:

Player Character

By default, the player character is a human (who starts with 140 HP, 20 Atk, 20 Def). However, the player has the option of changing their race to something more fantastical (e.g. offer different attributes). The options include dwarf (100 HP, 20 Atk, 30 Def, gold is doubled in value), elves (140 HP, 30 Atk, 10 Def, negative potions have positive effect), and orc (180 HP, 30 Atk, 25 Def, gold is worth half value).

In our game board, the player character is always denoted by the '@' symbol.

Question. How could your design your system so that each race could be easily generated? Additionally, how difficult does such a solution make adding additional classes?

Enemies

Enemies are the mortal foes of the player character. In a traditional rogue-like, the enemy character would have some degree of artificial intelligence. However, for simplicity in CC3k enemies move one square randomly within the confines of the chamber they were spawned in. Except dragons, who always guard a treasure horde.

Enemies can be one of vampire(50 HP, 25 Atk, 25 Def), werewolf(120 HP, 30 Atk, 5 Def), troll(120 HP, 25 Atk, 15 Def), goblin (70 HP, 5 Atk, 10 Def), merchant (30 HP, 70 Atk, 5 Def), dragon (150 HP, 20 Atk, 20 Def, always guards a treasure horde), phoenix (50 HP, 35 Atk, 20 Def).

Merchants can be attacked and slain by the player character. Attacking or slaying a Merchant will cause every Merchant from that point forward to become hostile to the player character (and will attack them if they pass within a one block radius).

Dragons always spawn in a one block radius of its dragon horde pile of treasure (see Treasure). That is, if a dragon horde is spawned then a dragon is spawned.

Upon their demise, any enemy that is not a dragon or a merchant will drop 1 gold. This gold is immediately added to the player character's total.

Enemies (except dragons, who are stationary) move randomly 1 floor tile at a time, assuming the floor tile is unoccupied (see Section for floor tile description). An enemy can never leave the room it was spawned (created) in. Note that enemies should be moved in a line by line fashion. That is, starting at the leftmost enemy move all enemies on that row and then move to the next row starting with the leftmost. Any particular enemy should only be moved once per player action (e.g. moving to a line that has not been processed does not grant an extra move). However, should the player character be within a 1 block radius of an enemy then the enemy will always attack the player character.

Enemies are denoted on the map as follows: V(ampire), W(erewolf), N(goblin), M(erchant), D(ragon), X(Phoenix), T(roll), M(erchant).

Question. How does your system handle generating different enemies? Is it different from how you generate the player character? Why or why not?

Question. How could you implement special abilities for different enemies. For example, gold stealing for goblins, health regeneration for trolls, health stealing for vampires, etc.?

Items

Potions

In the game of CC3k, there is only one type of usable item: a potion. Potions are of two types: positive and negative. Potions can provide the player character with positive and negative bonuses as outlined below. Regardless of the potion itself, all potions are denoted on the map with a P. A player may not hold any potions. Accordingly, a potion cannot be used unless a player is standing within a 1 block radius of it.

The effects of a particular potion are not known until it is used for the first time, e.g. the player character will not know what a potion does until they use it for the first time in a session. However, they will only learn about the effects of that particular potion. Other potions will not have their effects revealed. The exception is for purchasing potions from the Merchant[2], which have their abilities displayed.

Positive Potions:

- `Restore health (RH)`: restore up to 10 HP (cannot exceed maximum prescribed by race)
- `Boost Atk (BA)`: increase ATK by 5
- `Boost Def (BD)`: increase Def by 5

Negative Potions:

- `Poison health (PH)`: lose up to 10 HP (cannot fall below 0 HP)
- `Wound Atk (WA)`: decrease Atk by 5
- `Wound Def (WD)`: decrease Def by 5

The effects of RH and PH are permanent while the effects of all other potions are limited to the floor they are used on. For example, using a BA potion will only boost the player character's Atk until the beginning of the next floor.

Note that the PC's Atk and Def can never drop below 0.

Question. What design pattern could you use to model the effects of temporary potions (Wound/Boost Atk/Def) so that you do not need to explicitly track which potions the player character has consumed on any particular floor?

[2]Only a concern for bonus purposes. The default game does not require Merchants to sell anything.

Treasure

Treasure in CC3k consists only of gold. Gold can be in several types of piles: normal (value 1), small horde (value 2), merchant hoard (value 4), and dragon hoard (value 6). Thus, a dragon must always protect a dragon hoard whenever it randomly spawns. A dragon horde can only be picked up once the dragon guarding it has been slain. Gold, regardless of type, is denoted by 'G' on the map.

A merchant horde is dropped upon the death of a merchant.

Question. How could you generate items so that the generation of Treasure and Potions reuses as much code as possible? That is, how would you structure your system so that the generation of a potion and then generation of treasure does not duplicate code?

Floors

Levels are generated to consist of the 5 chambers connected in the manner outlined in Figure 1. It would be more interesting to have randomly connected randomly generated chambers but that is more complicated than the time frame allows.

The player character should spawn randomly in a chamber (every chamber is equally likely) but it should never be the case that the player spawns in the chamber with the stairs going down to the next level. Stairs are denoted by '\'. Note that the stairway and player character may be spawned with equal probability on any floor tile in a chamber. That is, a larger chamber should be no more likely to spawn the PC/stairs than a smaller chamber, where any floor tile in the selected chamber is equally likely to spawn the PC/stairs.

Potions are spawned with equal probability on every level (e.g 1/6 chance to spawn a particular potion) and any chamber has a 1/5 chance to spawn a potion, such that any square in the room has an equal probability of spawning a potion. 10 potions are spawned on every floor.

We might like to have gold spawn more or less frequently as the game gets more difficult. However, to again simplify design the spawn rate of gold is 5/8 chance of normal, 1/8 dragon hoard, 1/4 small horde. Chambers are equally likely (as are floor tiles in any particular chamber) to spawn gold. 10 piles of gold are spawned on every floor.

With the exception of dragons, enemies have the following probability distribution of being spawned:

- Werewolf: 2/9
- Vampire: 3/18
- Goblin: 5/18
- Troll: 1/9
- Phoenix: 1/9
- Merchant: 1/9

20 enemies are spawned per floor. Every chamber is equally likely to spawn any particular monster (similarly for floor tiles).

We require that generation happens in the following order: player character location, stairway location, potions, gold, enemies. This is to allow us to more easily evaluate that your random generation is correctly implemented.

Note that multiple objects (enemies, gold, and potions) cannot occupy the same cell on the game board. That is, no two objects can ever occupy the same space. Except in the case of gold, which is picked up when the player character walks over it.

Additionally, when the PC would move onto the stairway down (e.g. the PC is in the block to the immediate east of the stairs and the player enters the command to move 1 block west) the next floor is generated and displayed to the player.

Additionally, items and enemies should only ever spawn on a floor tile and never in a doorway, passage, or the stairs leading down to the next floor.

Combat

By default, all enemies except for Merchants and Dragons are (by default) hostile to the player character. If the player character enters within a 1 block radius of any hostile enemy, they will attempt to attack the player character. Dragons are only hostile when the player is next to (read: in the 1 block radius of) its dragon horde and never hostile otherwise. Enemies never pursue the player character. If the player character is not within a 1 block radius of the enemy then it moves. Recall that Merchants can become hostile when one is attacked/slain by the player character.

Combat is resolved as follows: Enemies will auto-attack players given the previously specified criteria, however, there is a 50% chance their attack misses. The player character has the option of attacking in a 1 block radius around them[3]. Recall, that the PC has initiative and always attacks first.

Damage is calculated as follows: $Damage(Defender) = ceiling((100/(100+Def(Defender)))*Atk(Attacker))$, where Attacker specifies the attacking character (enemy or PC) and defender specifies the character being attacked. Thus, in a single round a character can be both an attacker and a defender.

Display

The display of CC3k is relatively simple, Figure 1 depicts an empty board. Walls are denoted by '|' and '−', doorways by '+', and passages by '#'. Floor tiles that can be walked upon are denoted by '.'. Chambers are denoted by the smaller polygons inside the larger rectangle. The player character can only ever occupy a passage block, doorway block, or a floor tile inside a chamber. The player character can see in all chambers simultaneously, e.g. through walls or doors[4]. Figures 2, 3, 4, 5, depict various board states. Note that Figure 1 represents a completely empty game board and is meant to act as a reference of what the game board would look like before any generation occurs.

Command Interpreter

Initially, the game will demand the player enter one of the specified races or quit. Play will then continue in the obvious way until the player restarts, reaches the end of floor 5, the PC dies, or the player quits. If the player reaches the end of the game or their character is slain, give them the option of playing again. Otherwise, do the appropriate thing (restart or quit).

The following commands can be supplied to your command interpreter:

- **no,so,ea,we,ne,nw,se,sw**: moves the player character one block in the appropriate cardinal direction
- **u <direction>**: uses the potion indicated by the direction (e.g. no, so, ea)
- **a <direction>**: attacks the enemy in the specified direction, if the monster is in the immediately specified block (e.g. must be one block north of the @)
- **h, e, d, o**: specifies the race the player wishes to be when starting a game
- **r**: restarts the game. All stats, inventory, and gold are reset. A new race should be selected.
- **q**: allows the player to admit defeat and exit the game.

Note that the board should be redrawn as appropriate every time a command is entered.

Ending the Game and Scoring

A game session ends in one of the following ways: the player character reaches the stairs on floor 5, the player character's health reaches 0, or the player restarts the game.

[3]The PC never misses.

[4]For the sake of simplicity, the player character has X-Ray vision.

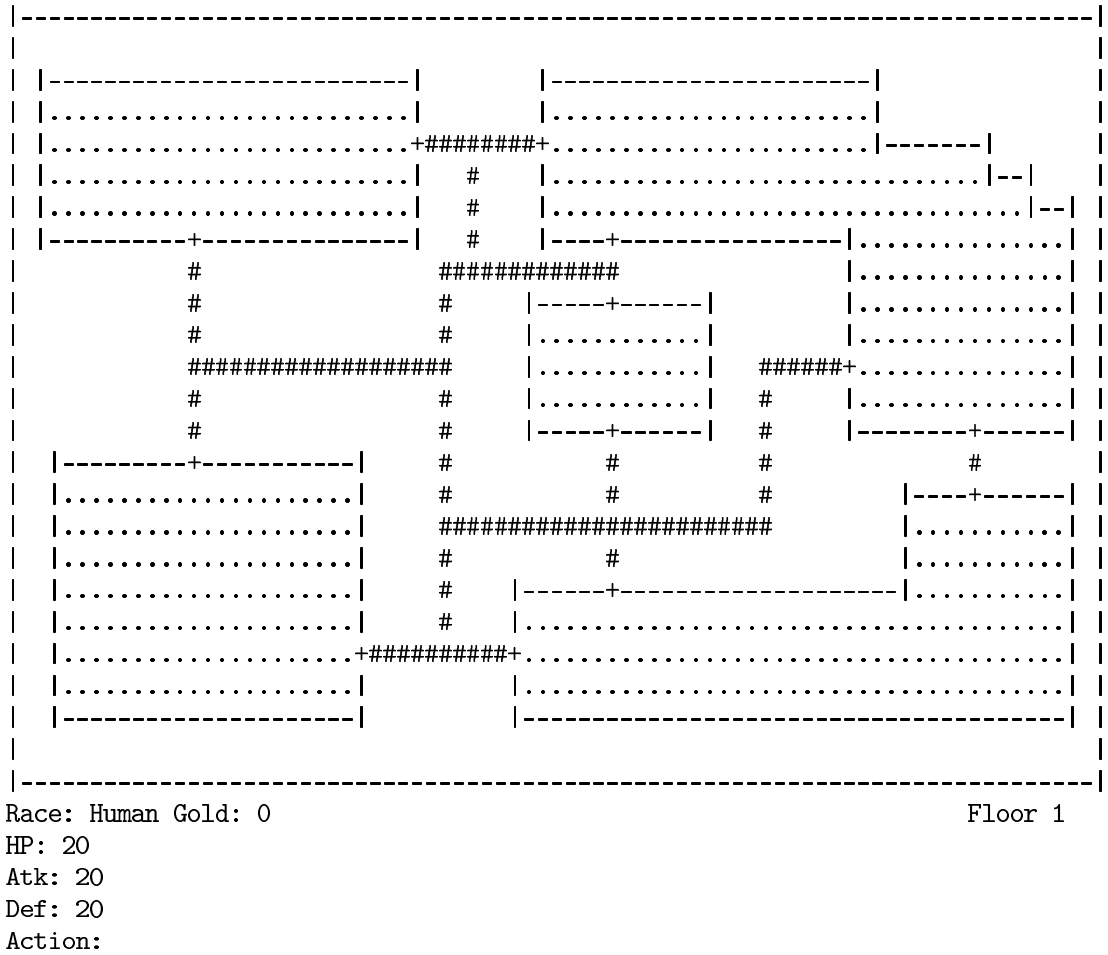

Figure 1: Empty board showing all passages between chambers.

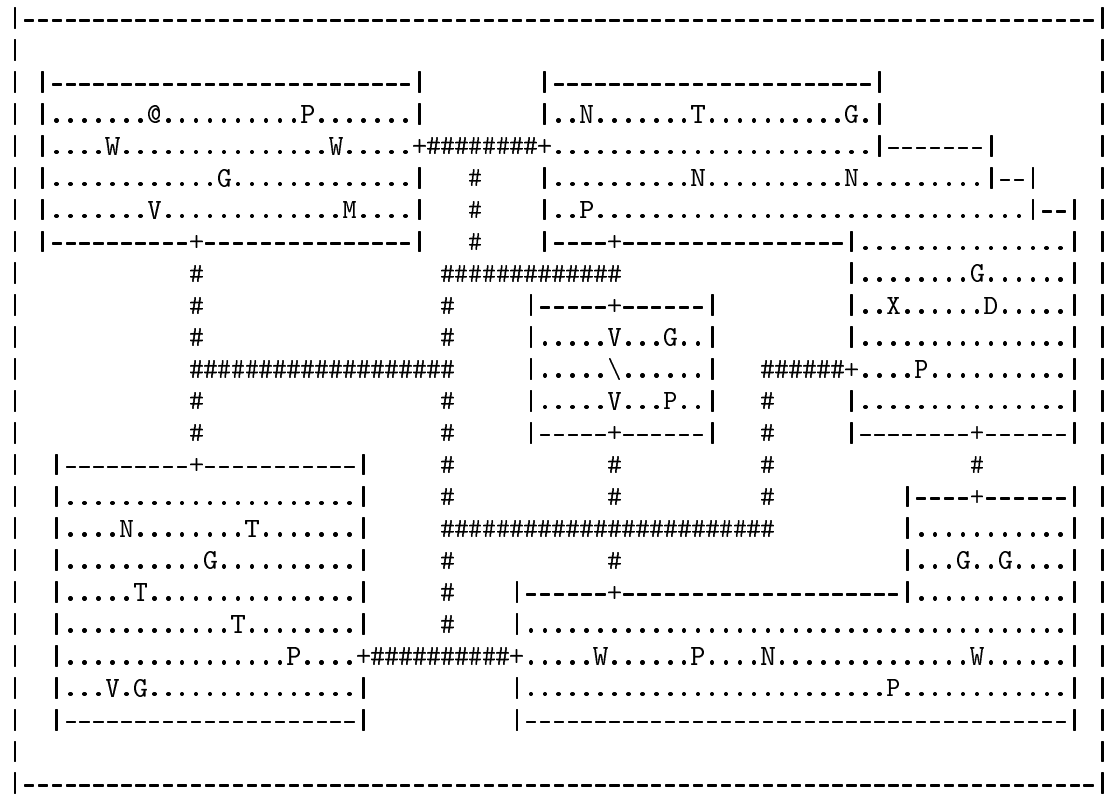

Figure 2: A board showing a (potentially) generated board.

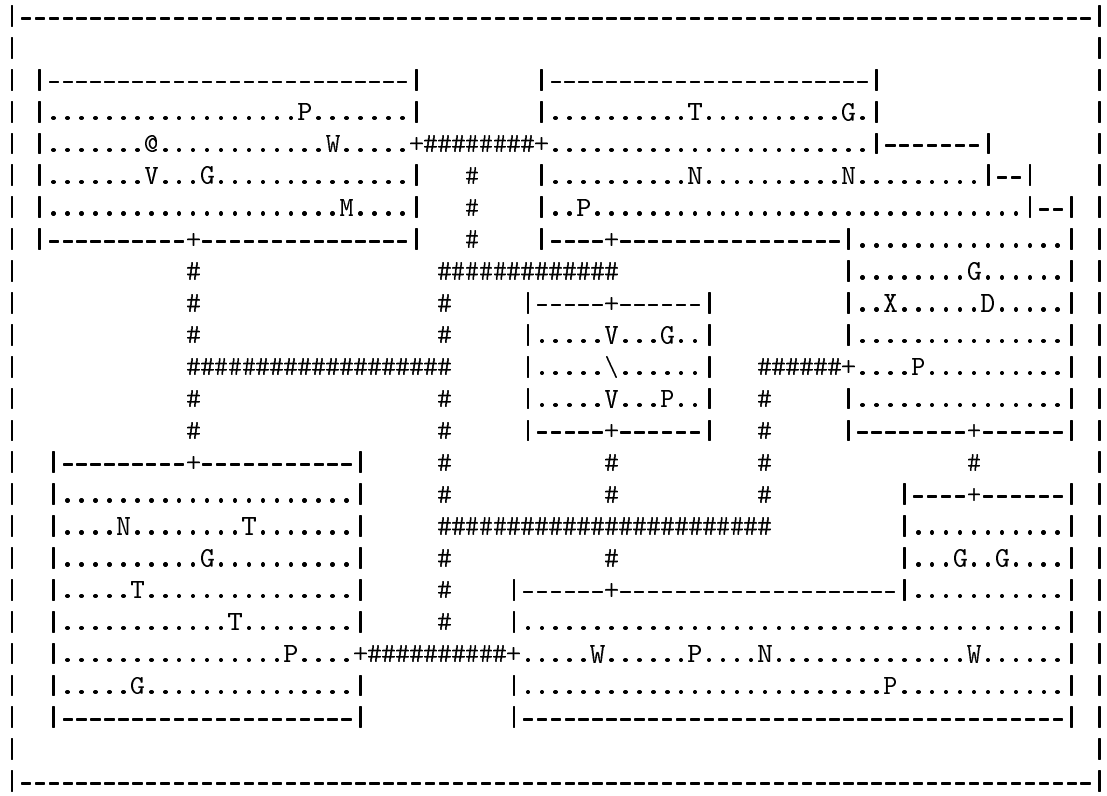

Figure 3: A board showing combat.

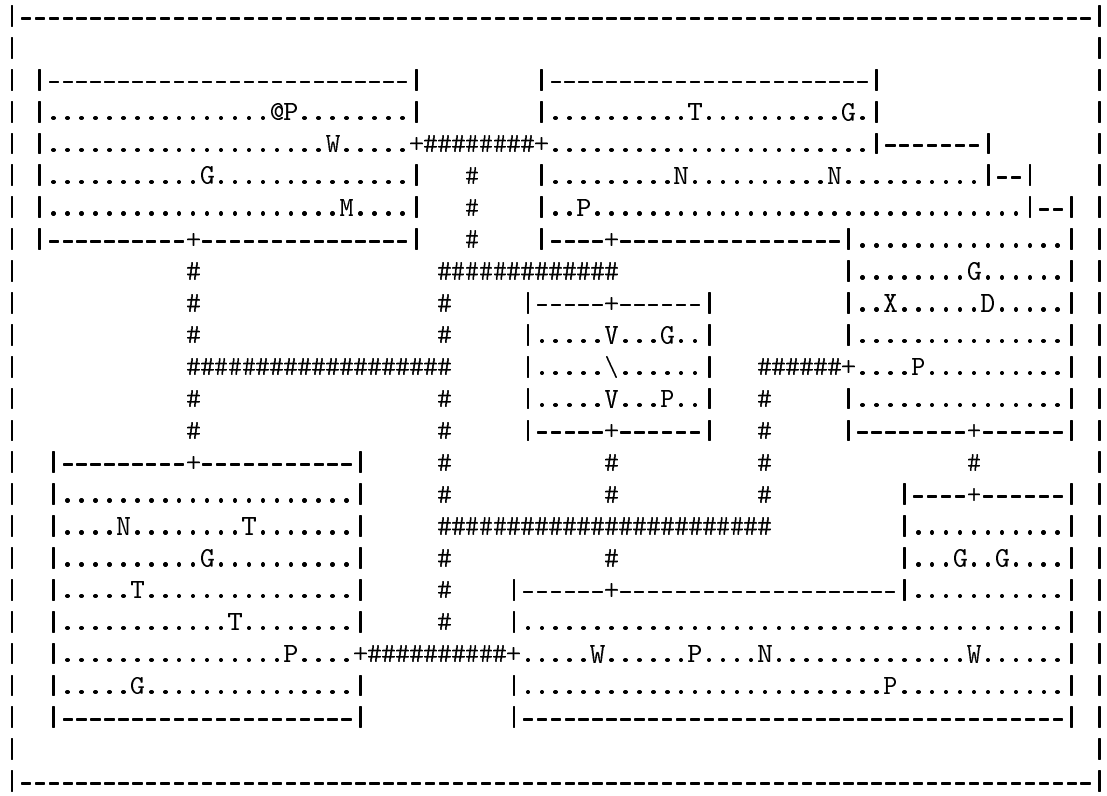

Figure 4: A board showing that potions are unknown until used.

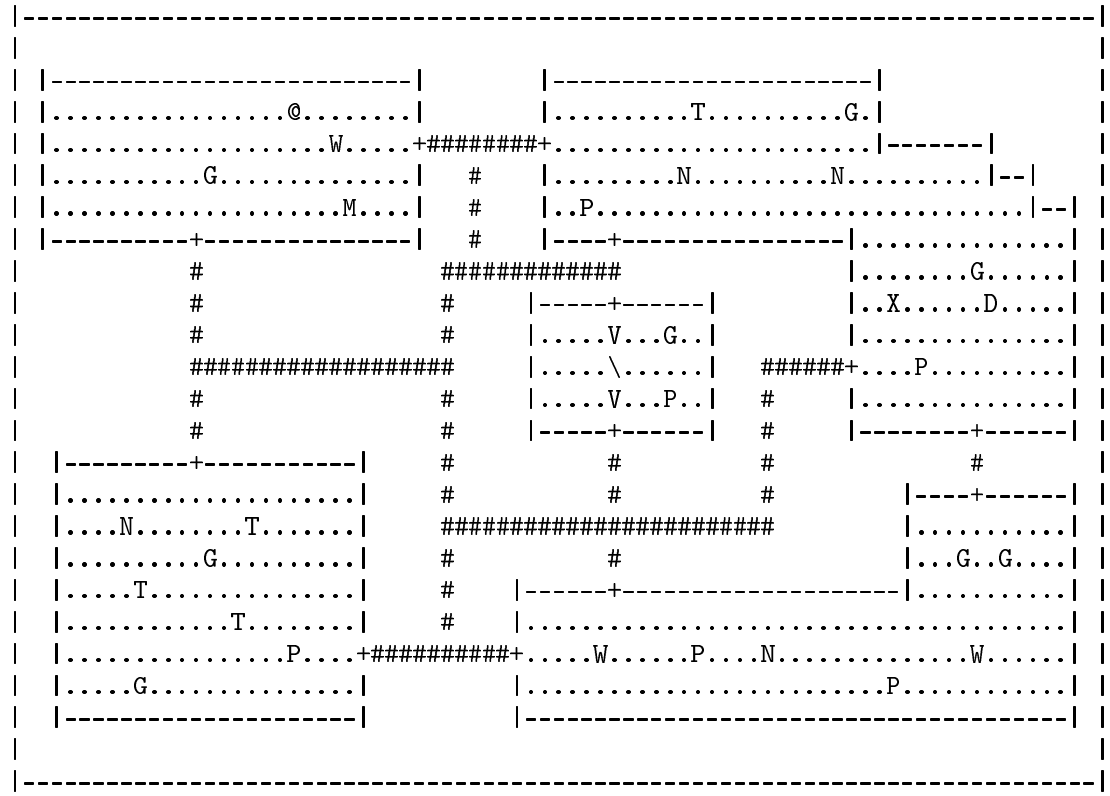

Figure 5: A board showing potion usage.

A player's score is only generated in the first of above two cases. Score is determined by the amount of gold they have collected in their current character's life time. Except for humans who have a 50% increase to their score.

Command Line options

Your program should have the ability to process an optional single command line argument, a file that specifies the layout of each of the 5 floors. This is specified by giving the exact floor layout as is presented to the player. Essentially, any diagram presented in this document (minus the text information in the last 5 rows). Note that potions and piles of gold will be denoted by numbers (outlined below) but should be converted to the appropriate character when displayed to the player.

The translation of numbers to items is as follows: 0 - RH, 1 - BA, 2 - BD, 3 - PH, 4 - WA, 5 - WD, 6 - normal gold pile, 7 - small horde, 8 - merchant horde, 9 - dragon horde.

However, you may find it useful to have a second optional argument that represents a seed for your random generation (so that you can have reproducible results). Although, this is **not** required.

Grading

Your project will be graded as follows:

Correctness and Completeness	60%	Does it work? Does it implement all of the requirements?
Documentation	20%	Plan of attack; Final design document.
Design	20%	UML; good use of separate compilation, good object-oriented practice; is it well-structured, or is it one giant function?

Even if your program doesn't work at all, you can still earn a lot of marks through good documentation and design, (in the latter case, there needs to be enough code present to make a reasonable assessment).

When all else fails...

This is a relatively complex project with many components and large amounts of random generation. Accordingly, if you find yourself running out of time or having trouble here's our suggestion for priorities:

- Reading in and printing a floor.
- Create general purpose player character, enemies, and items (e.g. No special types)
- Get movement and interaction working (combat, item use, etc) for these generalized versions.
- Introduce the different races and enemies.
- Introduce the different types of items.
- Random floor generation.

Accomplishing the first three points should reward you with at least 50% (assuming you have used object-oriented principles to get that far). That is, you will get a higher mark for submitting something that runs but not does implement all the requirements than submitting something that attempts to implement all the requirements but doesn't run.

If Things Go Well

If you complete the entire project, you can earn up to 10% extra credit for implementing extra features. The bonus enhancements for the CC3k project will be phrased in terms of free[5] Downloadable content (DLC).

[5]Contrary to the general trend in the gaming industry.

What this means to you is that you must provide some way of playing the game with and without your DLC (Note: recompilation with different flags is **not** permitted). This will allow project assessors to accurately determine that you met all of the base requirements.

The following is a list of possible DLC that you could develop. However, don't attempt to implement this until you've got the base game working.

- Use a curses[6] library to make this like an actual game with WASD controls. We offer no support for this though.
- Introduce an inventory system so that the player character can hold multiple potions
  - Expand the inventory and combat systems to make use of new types of treasure (e.g. weapons and armor)
- Give enemies some intelligence. Make them pursue the player character.
- Give the Merchant the ability to sell (and maybe buy from the player character).
- Add a class system to the game (e.g. Knight, Monk, etc). May require extensive reworking for magic users.
- Add more player character races
- Add more enemies
- Other stuff! This sky (or rather the chamber roof) is your limit.

To earn significant credit, enhancements must be algorithmically difficult, or solve an interesting problem in object-oriented design. Trivial enhancements will not earn a significant fraction of the available marks.

Submission Instructions

See **project_guidelines.pdf** for instructions about what should be included in your plan of attack and final design document.

A Note on Random Generation

To complete this project, you will require the random generation (or rather, pseudo-random) of numbers. You have two options available to you. In `<cstdlib>`, there are commands **rand** and **srand**, which generate a random number and seed the random generator respectively (typically, seeded with **time()** from `<ctime>`).

[6] `http://en.wikipedia.org/wiki/Ncurses`