Programming Assignment 3

Assignment Details

Assigned: October 28th, 2013. Due: November 23rd, 2013 at midnight.

Foreword

The requirements for this programming assignment are intentionally broad – the intent is that students have a large amount of freedom in terms of techniques to employ, as well as ample opportunity to showcase creativity.

The educational goal of this assignment is to give further experience with managing large(r) projects and reuse of code – once this assignment (and Programming Assignment 4, which will provide a more graphical rendering of the output of this program[1]) is complete, the student should have a greater understanding of the impact of object-oriented techniques on the reuse and maintenance of code.

The list of requirements is fairly long, but most of them are pretty loose in terms of what has to be implemented.

The Actual Assignment

PA3 requires one executable to be generated; this will allow the game we've been working on all semester to be played. At this point, it may not be terribly easy to play due to the drawing mechanics and the need to hit enter after each move (but, both will be fixed for PA4!)

Incorporating Previous Assignments

PA1 will be needed to read in data from XML files to set up the game world, and will need relatively few modifications (you are probably going to want to make some modifications to add more data to the world XML files, but this is optional.

PA2 will be used to generate dungeon levels for PA3. If your PA2 solution is unusable, a complete PA3 solution will be posted as part of the practicum assignment for November 6th. If you did not implement a Tile class for PA2, you will want to for PA3.

Positioning is the biggest change to the previous two assignments; you will need some mechanism to indicate position of Entities (whether Item or Creature, they need to be able to exist in the game world at a specific position). There are a few ways to do this:

1) Add coördinates to the Entity class. Keep in mind both that Items can be picked up by players (and thus may not exist on a dungeon level!), and that there are multiple dungeon

[1] And will be relatively simple, short, and most of the code will be provided in class. In short, PA3 *is* the big semester-end project, not PA4.

levels, so you will need some way to indicate that the item is being carried and have a dungeon level variable.

2) Add contents variables to the Tile class mentioned above.
3) Both of the above.

Requirements

- The game starts with the player on the upstairs of dungeon level 1; the level should be randomly generated
    - You may decide what (if anything!) the player starts with in their inventory
- You must keep track of a player; at minimum, the player must have:
    - Hit points
    - Score (which can be as easy as a duplication of experience, below)
    - Level and experience (or some way for the player to advance in skills as a result of defeating monsters)
    - Inventory (at minimum, this can consist of a single weapon and single piece of armor)
- The player must be capable of:
    - Moving (the player cannot move through walls or rock; only tunnels and room tiles)
    - Fighting (the player should attack a monster if the attempt is made to move into the tile that a monster is on)
    - Regenerating hit points over time
    - Using usable items; at minimum, a use command that uses (and destroys) a usable item on the same tile as the player – you are more than welcome to implement a more robust inventory, but just drinking a potion off the floor will suffice!
    - Gaining levels after a certain amount of experience; you may choose how much experience you need to gain a level, and how much experience killing monsters gives, but gaining levels must be possible and must give an advantage to the player.
    - Going up and down stairs
- You must keep track of multiple dungeon levels.
    - If a player returns to a previously visited dungeon level, it should be in the state it was in when the player left.
- If the player goes upstairs from the first level, the game is over. Print the player's score, and tell the player the dungeon was escaped.
- If the player's hit points are reduced to zero or less, the player dies – print score, tell them they're dead, and end the game.
- Each dungeon tile can accommodate multiple Items, but only one living thing, whether it be a Creature or the player.
- Each game turn should be handled as follows:
    - First, accept a command from the player; at minimum, these should be:
        - Movement. I suggest, at this stage, number pad keys – e.g., 4 for moving left, 6 for moving right, 8 for up, 2 for down.
            - If a player attempts to move on top of a monster, that causes an attack

        - Pick up. 'p' is fine. As stated above, at minimum you only need to worry about one weapon and one piece of armor, but you are welcome to do more.
            - Assuming you use the minimum inventory requirements, picking up a weapon or piece of armor should cause the weapon or armor (if any) the player is currently carrying to drop to the floor.
        - Use. 'u' is fine. At minimum, it should attempt to use an item on the floor.
    - Then all monsters on the dungeon level the player is currently on should act, at minimum:
        - They should attempt to move one space towards the player if possible; monsters can only move through room tiles or tunnels.
            - Ghosts, earth elementals, xorns or similar critters may be able to move through rocks and walls, though – these monsters, though, should be higher level.
            - IT IS NOT VALID TO HAVE EVERY MONSTER IN THE GAME HAVE THE ABILITY TO MOVE THROUGH ROCK AND WALLS.
                - If you're going for a haunted mine theme, at least add some kobolds…
        - If adjacent to the player, they should attack the player.
    - Then, the current dungeon level should be printed to the console – with player, item and monster positions displayed.
    - This can be modified for more interesting game situations (e.g., a haste spell for players or monsters may grant them extra movement/turns), and treated as a minimum level of complexity as opposed to a strict requirement for what a turn is.
- Attacks should happen, and be based on monster stats (you probably want to modify the Creature class to add some stats for attacks) or player level and weapon. The result of attacks should be printed to the console.
- Healing potions should work when used. Other consumable items are strictly optional.
- Monsters should be randomly generated when a new level is generated, and then periodically generated and dropped in the dungeon during play. At a minimum, monsters should be randomly generated with a level of twice the dungeon level or less.
    - How frequently this happens is up to you.
- Items should be randomly generated and put on the dungeon floor as well
- Monsters should have a chance to drop a random item on death
    - You may implement this as an inventory thing (e.g., the monster has an inventory, and drops its inventory on death), or just a chance at a random item generated each monster killed.
- Monsters should, at a minimum, attempt to move towards the player each turn
    - You do not need to implement particularly robust pathfinding; doing this *well* is beyond the scope of the class, but at minimum, monsters should try to move closer to the player in either the x or y direction; it's ok if they get stuck in the corners of rooms.

    - You may have immobile monsters, but again, it's not valid to have all monsters be immobile – fungus caves should have a few goblins running around.
- In general, a Tile should be drawn as its normal symbol if it's empty.
    - If there's a monster on the tile, draw the monster.
    - If there's an item and no monster, draw an item.

In general, though, if you have an idea for a game mechanic which would violate one of the above but be more interesting and not require any less effort, drop me an email – you'll probably be cleared to do it. This is, in large part, an exercise in creativity; while we need to end up with a game, the exact appearance of that game is quite flexible…

Display Requirement

Between processing the game turn and displaying the contents of the dungeon level to the screen, you must generate a two-dimensional vector of chars that represents the current state of the current dungeon level – note that this is **not** simply the current dungeon level (as, depending on how you have Entity positioning implemented, this may have more or less data than DungeonLevel itself contains), but, specifically, what character needs to be displayed at each position.

This should be, *specifically*, a `vector<vector<char>>`.

Then, this structure is printed to the console.

The reasoning for this should be a bit more apparent in PA4, when we have to use a library to have more interesting drawing, and are going to want to reuse this code with as few modifications as possible…

Documentation

One more requirement – you need to provide brief documentation for your project, which must include:

- What commands the user can use
    - A brief list of usable commands should be printed upon game startup
- A brief description of decisions made that are not mandated by the requirements; specifically, you should explain what method you chose for positioning and *why*.

Extra Credit

Ample extra credit will be given for going above and beyond the requirements above – remember, you're writing a game. More interesting games will get more points.

Notes

Code samples and XML data will be made available in the /CS216/info directory in source control. Homework and Practicum sessions will be geared towards supporting PA3 development.