ECE 39595 Java -- Rogue Project

This document describes the Java project. The project will be turned in in four steps, with dates given in the syllabus. As different steps are assigned, we will give more information for the steps. This document is basically an overview.

Documents associated with this project may be updated for clarity. If how something is done is not specified, you are free to do what you want. Feel free to ask clarifying questions during TA and Prof. Midkiff's office hours, and on Piazza.

We will be developing a rogue-like, character based game. A YouTube video demonstrating the classic 1980s version of rogue can be found at https://www.youtube.com/watch?v=vxF1osPkplA. We will do a character based stripped down version of the game.

The ascii display:

We will use the third-party *asciiPanel* code in Java to manage the physical, visible, game display. The ascii display only knows about characters, and displays them. It is driven by the *logical display* which knows about the objects and what characters should be displayed for them, and where they should be displayed. Example code will be provided showing how to use asciiPanel and curses to display characters.

The logical display:

Game objects are displayed on the logical display, which is called the *ObjectDisplayGrid* in my code. It manages the displayable objects at each location in the ascii display, and sends the character corresponding to each visible object to the ascii display. It is basically a 2D array of objects that represent all possible elements that can be shown in the game. It also receives keyboard input, and is implements the *Subject* role in the *Observer pattern*, which we will learn about.

The logical display has several areas.

A *top message area*, where the remaining hit points and the score are displayed. The number of lines in this top area is given in the XML file.

The game area of the display – this is the largest area, and immediately follows the top message area. The XML file gives its height.

The bottom message area, where the inventory is written and the *info* messages (discussed below) are written. About half of the bottom message area should be used for inventory, and half for *info* messages.

The top and bottom message areas only display text.

The width of the logical display is also given in the XML file.

The ascii display is made large enough to display all elements written to the logical display can be shown on the ascii display.

The Dungeon:

The dungeon will be defined in an XML file. The XML file will give the layout of the dungeon, where monsters, the player, weapons, armor, and scrolls can be found. It will also define the properties of weapons, monsters, armor and scrolls, i.e., how much damage they inflict or protect us from, and the actions of monsters and players when they are hit, move or die. By *parsing* the XML file, we will get the information needed to create the objects that represent the different parts of the dungeon. We will provide sample XML parsers in C++ and Java to help you in writing your parser.

The Dungeon also maintains lists of coordinates in the dungeon that are traversable.

The Dungeon also implements the *Observer* role in the *Observer pattern*, and receives notifications from the logical display when commands are entered. The Dungeon operates on three of those commands:

Help: '?': show the different commands in the *info* section of the display

Help 'H' <command>give more detailed information about the specified command in the *info* section of the display.

End game 'E' <Y | y>: end the game.

Creatures in the game:

There are two kinds of creatures in the game dungeon, monsters and the player. Different creature actions can be attached to the monsters and player to customize how they respond to being hit, when moving, and when dying. We will have 3 kinds of monsters and a player. Trolls are represented by a 'T', Snakes by an 'S' and Hobgoblins by an 'H'. The player is represented by an '@'.

Properties of creatures:

Creatures have health specified as hitpoints (i.e., how much more damage they can get before dying), maxHit, a maximum number of hitpoints that can be delivered when attacking a creature, a type, which is the character that represents the creature, a string that is the name of the creature, and lists of actions (see below) that are described in the XML file that will be performed in case of death or being hit. Players also have a score, how many moves have to be made to get a single hitpoint added to their health, and a pack that holds a list of items. Each creature also has an X and Y coordinate that indicates their position on the game display.

Built-in actions for creatures:

Creatures always have the following actions:

Initialize the display: put themselves on the display in the correct position at game startup.

Refresh the display: refresh the display during the play of the game.

Perform the actions specified in the XML file that are associated with being hit.

The damage delivered when hitting: this is a random number between 0 and maxHit, inclusive.

Perform the actions specified in the XML file as associated with dying.

Additional built-in actions for players:

Players have all of the built-in actions of a creature, plus the following additional actions.

Move (left, right, up, down): move to a new location. The move can only occur if the new location is traversable, i.e., is a dungeon floor or passageway. If the location to be moved to is occupied by another creature, the other creature is hit but the creature moving does not occupy that location. If the location to be moved to is occupied by an item (armor, a sword or a scroll) the creature moves to that location. A separate command is used to actually pick up the item, and only the player creature pics up items.

After the player moves, the location it leaves should show the dungeon floor or passage, if empty, or the item that is laying on that spot, if not empty.

As well, the XML file can specify move actions that are performed at each move.

Show or display the inventory 'I': show the contents of the pack, printing the *name* for each item in the pack. Each item is preceded by an integer 1 … *max items in the pack* that is used to refer to the item when removing or dopping items in the pack.

Change, or take off armor 'c': armor that is being worn is taken off and placed in the pack. If not armor is being worn a message should be shown in the *info* area of the game display.

Drop 'd' <integer>: drop item <integer>-1 from the pack. If the <integer>-1 does not refer to an item in the pack and informational message is printed on the game display in the *info* area.

Read an item 'r' <integer>: the item specified by the <integer>-1 must be a scroll that is in the pack. It causes the scroll to perform its actions.

Pick up an item from the dungeon floor 'p': pick up the visible item on the dungeon floor location that the player is standing on. If multiple items are in the location, only the top item is picked up.

Take out a weapon 'T' <integer>: take the sword identified by <integer>-1 from the pack. If the identified item is not a sword, or no such item exists, show a message in the *info* area of the game display.

Wear item 'w' <integer>: take the armor identified by <integer>-1 from the pack and make it the armor being worn. If the identified item is not armor, or no such item exists, show a message in the *info* area of the game display.

Actions that can be added to creatures:

Creatures will have the following actions attached to them:

Print: causes a message to be printed to the console (not the game display).

Teleport: causes the creature to go to a random place in the dungeon that is legal for a creature to be, i.e., within a room or a passageway between room. A creature should not end up outside of the traversable part of the dungeon.

UpdateDisplay: causes the display to be refreshed.

YouWin: This is typically executed by a monster that is killed. It updates the player's points.

In addition to these actions, Players also have the following actions:

DropPack: drop a single item from the pack. The dropped item should be placed on the floor and be available to be picked up at a later time.

EmptyPack: drop all items from the pack. The dropped items should be placed on the floor and be available to be picked up at a later time.

EndGame: typically executed when the player dies, it causes the game to be ended and all further input to be ignored.

Items in the game:

The game dungeon contains the following items: *armor*, *scrolls* and *swords*. Armor absorbs damage and protects the wearer (who is always the player in our game) from some of the damage inflicted by a monster. All items can be picked up by the player and stored in the pack. Armor can be worn, and swords can be wielded, by the player. Scrolls sit in the pack and are read by the player, causing them to take effect. Scrolls can only be read once.

All items have a string that is the name, and is what is written for the item when the player performs an inventory command.

The following characters are used to represent items: A weapon (i.e, a sword in our game) is "|", armor is "]", and a scroll is "?".

Actions for items:

The following items can be associated with items.

Bless/curse item: reduces the effectiveness of a sword or armor. Usually attached to a scroll, and blesses or curses when the scroll is read.

Hallucinate: it registers with the observer and when invoked causes each item within the dungeon to display a different character with each move of the player. This lasts for a limited number of moves, with the number of moves being set in the XML file.

Hallucinate implements the Observer interface of the Observer pattern, and registers with the Dungeon. It is notified of each input command, and keeps tracks of the number of moves so that hallucination stops after the specified number of moves.

When it is invoked it will print out a message in the info area of the game display that hallucinations will continue for some number of moves.

Structural elements of the dungeon:

The dungeon consists of Rooms and Passages. Rooms consist of walls (which are not traversable and are represented by an ‘X’) and floors (which are traversable and represented by a ‘.’). Passages consist of passage floors, which are represented by a ‘#’ and junctions, which is where the passage floor runs into a room wall, and are represented as a ‘+’. Passage floors and junctions are traversable. Items and creatures may be placed on room floors, passage floors and passage junctions. They may not be placed on room walls.

In the XML file defining a dungeon, items and creatures are always initially placed on a room floor. The coordinates of the item are relative to the upper left corner of the room, i.e., they are not absolute coordinates in the game display. Coordinates for rooms and passages are relative to the game area of the display, not the uppermost and leftmost corner of the ascii display.

Displaying messages in the message areas:

Because all objects placed on the logical display must have the same base type, a special type that holds a character must be used for displaying messages in the top and bottom message display areas.

A summary of commands:

These have been discussed above. This section exists to get all of the commands that are input by the user of the game in one section.

Commands processed by the dungeon:

?: help. Lists all commands in this section, but only gives the letter, not information about what the command does, e.g. "`h,l,k,j,i,?,H,c,d,p,R,T,w,E,0-9. H <cmd> for more info`"

H <next input character>: *<next input character>* is a command. H gives more detailed information about the command.

E <next input character>: If *<next input character>* is a 'Y' or 'y', the game is ended. If it is any other character, the game continues.

Move commands, processed by the Player and code needing to count moves.

Move commands are the vi/vim navigation keys.

h: move left 1 space.

l: move right 1 space. Note that this is a lower case el, *ℓ*, not an upper case i.

k: move up 1 space

j: move down 1 space

Move commands, processed by the Player

i: inventory -- show pack contents

c: take off/change armor

d: drop <item number> item from pack";

p: pick up item under player and put into the pack";

r <item in pack number>: read the scroll which is item number *<item in pack number>* in pack

t: take out weapon from pack

w: take out the armor which is item number *<item number in pack>* in the pack and put it on.

Structure of the XML file.

The XML file has the following high-level structure.

It starts by declaring the start of the dungeon, i.e.

```
<?xml version="1.0" encoding="UTF-8"?>
<Dungeon name="testDrawing" width="n" topHeight="n" gameHeight="n" bottomHeight="n">
<room and passage definitions>
</Dungeon>
```

Where **name** is the name string for an object, **`width`**, **`topHeight`**, **`gameHeight`** and **`bottomHeight`** are the logical display size parameters mentioned above in the logical display discussion, and ***n*** is an integer value for the parameter. These will always be string versions of the integer value.
Next Rooms and passages are defined. Rooms and passages can happen in any order. Creatures (monsters and the player) and items are always defined as part of a room within a room definition.

A group of rooms is defined as:

```
<Rooms>
      One or more room declarations
</Room>
```

Rooms are defined as:

```
<Room room="n">
<visible>n</visible>
<posX>n</posX>
<posY>n</posY>
<width>n</width>
<height>n</height>
<creature and item definitions>
</Room>
```

n is again an integer value. **`posX`** and **`posY`** are the coordinates within the game area of the upper left hand corner of the room. **`width`** is how wide the room is, and **`height`** is how tall it is. *creature and item definitions* are the definitions of creatures and items that at the start of the game are in the room.

There is no required order among **`visible`**, **`posX`**, **`posY, width, height`** and *<creature and item definitions>*.

Monster creatures are defined as:

```
<Monster name="cccc" room="n" serial="n">
<visible>n</visible>
<posX> n </posX>
<posY> n </posY>
<type>T</type>
<hp> n</hp>
<maxhit> n</maxhit>
<creature action definitions>
```

```
</Monster>
```

cccc is a string that gives the name of the monster, i.e., “Troll”, “Hobgoblin” or “Snake”. *n* is as above an integer. **`room`** is the number of the room that the monster is in, primarily for debugging purposes in the reference implementation of this, and serial is just another identifying number.

`visible` should be 0 if invisible and 1 if visible. In the current version of the game, all objects are visible. **`posX`** and **`posY`** are the coordinates of the monster at the start of the game relative to the upper left corner of the room the monster is in. hp is the number of hit points the monsters health starts at, and maxHit is the maximum number of hitpoints of damage a monster can deliver.

There is no required order among **`visible`**, **`posX`**, **`posY, type, hp`** and **`maxHit`**.

Player definitions.

Players are defined like Monsters, except the name is “Player” and the type is ‘@’.

Players can also have item definitions included in their definition, i.e. the line <*creature action definitions*> in the above should be line <*creature action and item definitions*>. The only items that will be defined in a Player are one sword and one armor.

Creature action definitions:

```
<CreatureAction name="cccc" type = "tttt">
<actionMessage>cccc</actionMessage>
<actionIntValue>n</actionIntValue>
<actionCharValue>c</actionCharValue>
</CreatureAction>
```

name specifies the name of the creature action, e.g., *YouWin*. Type is the type of the action, i.e., **`hit`**, **`death`**, **`move`**, specifying that it should be performed when the creature is hit, killed or moves. actionIntValue and actionCharValue specify parameters to the action, and their meaning is determined by the specific action type. *n* and *c* are an integer and character, respectively, and *cccc* and *tttt* are strings.

There is no required order among **`actionMessage`**, **`actionIntValue`** and **`actionCharValue.`**

Item definitions:

<*ItemType* **name="***cccc***" room="***n***" serial="** *n***">**
<visible>*n***</visible>**
<posX>*n***</posX>**
<posY>*n***</posY>**
<item action definitions>
</*ItemType***>**

ItemType is either Scroll, Armor or Sword, i.e., is the name of the kind of item this is. cccc and n are strings and integers, respectively. Room is the room the item is in at the start of the game and serial is a unique id for the item in the room. I included room and serial for debugging purposes.

`visible` should be 0 if invisible and 1 if visible. In the current version of the game, all objects are visible. **`posX`** and **`posY`** are the coordinates of the item relative to the coordinate of the upper left corner of the room. Item action definitions are defined below.

There is no required order among **`visible`**, **`posX`**, **`posY`** and *<item action definitions>*.

Item action definitions:

Item action definitions look like Creature action definitions.