**Ultima**

[This is the game of Ultima](https://youtu.be/DZQAIHpwunw) (watch the short video before beginning).

In this assignment, you will be adding to existing code to 1) implement a recursive "torch" lighting algorithm, and 2) use threads to implement multiple independent monsters moving about the map.

**Import the starter files to begin.** In the game of Ultima, there are monsters that you (the avatar) must go around and defeat. You attack monsters by running into them. Monsters attack you in the same way. You can also walk on lava if you want, but it costs you one hit point.

This project obviously has quite a few classes. Luckily, most (including the main game engine) have been completed already. Descriptions of the classes in the project:

|  |  |
| --- | --- |
| **Avatar** | The Avatar class represents you the player. Avatar has a number of hit points (life); once your hit points reach 0, the game is over and you lose. Avatar also has a damage amount, representing how many hit points of damage the hero causes when attacking a monster. If Avatar incurs damage (from a monster or from walking on dangerous terrain), the new hit point value is displayed in yellow text over the Avatar image. The hit point display stays visible for ~3 seconds and then disappears. You don't need to modify this class (initially). |
| **Tile** | The Tile class represents a single square in the Ultima game world. Tile objects can report the damage they cause if they are walked on. All tiles cause 0 damage except for lava which causes 1 point of damage. You don't need to modify this class. |
| **Monster** | The Monster class represents a monster that roams around the map randomly. A monster knows things like its location, its remaining hit points, the amount of damage it causes when it attacks, and the type of monster it is. Monster objects also keep a reference to a World object so they can call methods in World from their run method (namely the monsterMove method). Just like the Avatar, if a monster is damaged it displays its remaining hit points in red text over the monster's image for ~3 seconds. If a monster's hit points are 0 or less, the monster has been killed and no longer is shown.  Each monster is its own Java thread that, every X milliseconds, attempts to move itself around the World. They are not smart, they just choose north, south, east or west at random. If they can move in the randomly chosen direction they do, otherwise they skip their turn and remain in the same location.  If the monster randomly moves into your Avatar, you will lose hit points according to the damage attribute of the monster. Just like the Avatar, monsters cannot walk through walls, on water, or through mountains. Monsters can walk on lava, but it causes damage just as it does for the Avatar. |
| **World** | The World class holds the objects in the game world, including all the tiles, the avatar, and an ArrayList of the monsters. The constructor parses a configuration file with information about the Avatar as well as information about the monsters. The World object creates and keeps track of the Monster objects. Each monster operates on its own individual timer, so one thread is fired up per monster. |
| **Ultima** | The runner / application class. Handles the key press events and keeps the game running while the player is alive. You don't need to modify this class (initially). |
| **Stats** | A class that helps with timing of game events. You don't need to modify this class. |
| **StdDraw** | The drawing library (GUI) class. You don't need to modify this class. |

Probably the most difficult part of this assignment will be to look at the code that has been provided and figure out how you can use the existing methods, without modifying them, to support the code you need to write.

The two things you need to add to the game involve recursion and multi-threading. In the Monster class, you will see that it implements Runnable. The Runnable interface has a single method, void run(), that you must override. Classes that implement Runnable are intended to be executed in their own thread\* (process); the overridden run method is what happens in the new separate thread of execution.

*\*There are two ways to start a new thread running: extending the Thread class and overriding its run method and implementing the Runnable interface. As there is no reason to sub-class Thread in this program, Monster implements Runnable. See the powerpoint for more info. See the end of World's constructor to see how independent Monster threads are created.*

If you scroll toward the bottom of Monster.java, you will see a place marked <YOUR CODE GOES HERE>. That is the only thing you should change in this file - the addition of code that makes the monster sleep (pause), and the code that makes the monster move in a random direction.

The monsterMove method in the World method should be called by a monster's run method. The monsterMove method checks for all the conditions that might be illegal, so your run method does not need to do this. If you look at this method in the World class, you will see that if the proposed location is not valid or not passable, then nothing happens. If there is currently another monster at the proposed location, then nothing happens (monsters don't attack each other). If the Avatar is at the proposed location, then the monster gets to attack the Avatar and do the appropriate damage. In this case, the monster stays at its current location (Avatar and monsters never overlap). Otherwise, the monster makes its move to the new location, incurring any damage associated with the new location (i.e. if the new location is lava). **Note:** since only the World object knows the outcome of the monster's call to monsterMove, the World object updates the calling Monster object by calling setLocation and/or incurDamage.

In the World.java file, you need to write the code that recursively lights the appropriate tiles (within the player's "torch radius"). You will see a place marked <YOUR CODE GOES HERE> in this class also. Think about this carefully.

You will need to write the code for a recursive method that gets called by the (public) light method:

private int light(int x, int y, int currentX, int currentY, double r)

All the Tile objects begin unlit. The recursion starts at the Avatar's current position (note that locations in the project are <x, y> pairs, rather than <row, col> pairs as in row-major order matrix processing). The light method will call itself recursively for the positions to the north, south, east, and west (do not recurse on the diagonals). The light method retains the initial (x, y) starting position so it can calculate how far the (currentX, currentY) position is from it. You must of course be careful to limit the recursion with appropriate base cases:

* Base case 1: Current position is off the map
* Base case 2: Current position has already been visited (or the Tile is marked as lit).
* Base case 3: Current position is opaque
  + Opaque cells should still be lit, but they should not propagate the search; this is what causes certain parts of the map to appear black despite being within the radius of the torch.
* Base case 4: Current position is outside the torch radius. A position is considered "outside" if the Euclidean distance between the (x, y) index of the Avatar and the (x2, y2) index of the tile is greater than or equal to the torch radius.

The avatarMove method is called when the handleKey method tries to move the Avatar. Similar to the monster, if the proposed location is not valid or passable, the Avatar stays put. If there is a monster at the location, the Avatar attacks it and the Avatar stays put. Otherwise, the Avatar moves to the new location incurring any damage associated with the new location (i.e. if the new location is lava).

You can test your game by running Ultima's main method. Keyboard controls are WASD.

**(Advanced) Game enhancements**

As is, Ultima is a very nice game engine. But think of all the ways it could be improved! Examples:

* Add multiple levels
  + You could add a door tile, maybe that requires a key obtained from the last monster, that moves to the next level
* Add loot
  + Weapons, armor, gems that can purchase items in a shop
* Add a boss monster that appears at the end of each level (when all other monsters are defeated)
* Add spells / projectile weapons (that can shoot outwards towards monsters, rather than requiring proximity to attack)
* Upgrade the graphics
  + The StdDraw class draws images to screen based on their actual size (on disk), size the images appropriately (the images included in the project file are really small). You will also need to update the Tile class which maintains the tile size constant.
    - StdDraw can scale images natively but can lead to rounding issues.
  + Add HP (hit point) bars to the player / monsters.
  + [Here](https://www.dropbox.com/sh/efczml3qu6qy5u4/AAAAQe5zYLiwuQxD6zEYAIhBa?dl=0) are some better tiles (images). You'll need to make / find some assets on your own.

**(Advanced++) Add monster AI**

Add some AI / pathfinding capabilities to the monsters. For example, an aggressive monster may always move towards the player along the shortest path or may attempt to move in front of the player to cut them off. A less aggressive monster may only have a chance to move towards the player or may only target the player only if within a certain range.

**(Advanced++) Multiplayer**

Add support for more than one (at a time) player character.

*Project adapted from* ***Ultima 0.1***

*https://katie.mtech.edu/classes/csci136/Assignments/07\_Threads/*