

CSCI 136 Assignment – Ultima 0.1

CSCI 136 Assignment – Ultima 0.1

March 5th, 2023

Due Date: Tuesday, 3/21/2023 11:59PM

Points: 30

Topic: Threads, Game Development

NOTE: I have set the due date for this assignment to be the Tuesday after Spring Break. This will give you the opportunity to seek help from our TAs after break if needed. Do your best to not depend on this. Hopefully you can get it done or close to done before break ...

In this assignment, you will extend the prior version of Ultima. You will implement Python threads. You will learn how to create code that protects data/operations shared between different threads.

If you were not able to get the last lab to work correctly, code is provided (Lab6Solution.zip) that you can use as a basis for this assignment.

After following all instructions below, please submit your Python scripts (`Avatar.py`, `Tile.py`, `World.py`, and `Monster.py`) in Moodle. Upload these as individual files – do not zip them. Be sure the script files have header sections that include the file name, your name, credits, a description of the file, the class and other useful information. Make sure your classes and all methods are thoroughly commented.

- `Avatar.py`
- `Tile.py`
- `World.py`
- `Monster.py`

CSCI 136 Assignment – Ultimate 0.1

Grading:

This assignment is worth 30 points. You will be graded according to the following criteria:

Grade Item	Points
Program Compiles and Runs	6
Comments on All Classes and Methods	3
Tile Changes	3
Avatar Changes	3
Monster Class	6
World Changes	4
Used Threads for Each Monster	4
No Concurrency Issues on World Instance	4
Total	30

..

Getting Started

You can use your previous Ultima assignment as starting point for this assignment. If you were not able to get the last lab to work correctly, code is provided ([Lab6Solution.zip](#)) that you can use as a basis for this week's assignment.

Files (ultima0.1.zip)

The file `ultima0.1.zip` contains the images, Python library code, and new configuration files for running the game. `Ultima.py` has been changed for the new and improved game, and you should use this as the main game loop - you should not make any changes to `Ultima.py`. In the improved game, there are monsters you must go around and do battle with. You are provided with a stub version of `Monster.py` that you will complete with your own code. You attack monsters by running into them. Monsters attack you in the same way. You can also walk on lava, but it costs you one hit point.

Tile.py

As in the previous lab, a `Tile` object represents an individual position in the Ultima world. You will use the same tiles as in the previous version. You already have code to handle most of this. You need to add a method to the `Tile` class: `getDamage()`.

Here is the specification you should implement for the `Tile` class:

```
class Tile
-----
# Create a new tile based on a String code
__init__(self, string code)

# Return whether this Tile is lit or not
boolean getLit(self)

# Change the lit status of this Tile
setLit(self, boolean value)

# Draw at index position (x, y)
draw(self, int x, int y)

# Does this type of Tile block light?
boolean isOpaque(self)

# Can the Avatar walk on this Tile?
boolean isPassable(self)

# Returns 1 if the tile is Lava, 0 otherwise
int getDamage(self)
```

Avatar.py

The Avatar class has several additions. The Avatar includes hit points (life, or health). Once your hit points reach 0, the game is over, and you lose. The Avatar has a damage amount, which is the number of hit points of damage the Avatar causes when attacking a monster. The `__init__()` method should be updated to handle these new attributes.

If the Avatar incurs damage (from a monster or from walking on lava), the new hit point value after the damage is displayed in the title line of the StdDraw canvas window. This is done for you automatically in the Ultima game loop.

Your Avatar data type must implement the following specification, including the new methods `getHitPoints()`, `incurDamage()`, and `getDamage()`:

```
class Avatar
-----
# Create a new Avatar at index position (x,y)
# Modified to include additional parameters and functionality
__init__(self, int x, int y, int hp, int damage, double torch)

# Get the current x-position of the Avatar
int getX(self)

# Get the current y-position of the Avatar
int getY(self)

# Update the position of the Avatar to index (x,y)
setLocation(self, int x, int y)

# Get the current torch radius
float getTorchRadius(self)

# Increase torch radius by 0.5
increaseTorch(self)

# Decrease torch radius by 0.5, minimum is 2.0
decreaseTorch(self)

# Draw at the current position
draw(self)

# Get the number of hit points left
getHitPoints(self)

# Reduce the Avatar's hit points by the damage amount
incurDamage(self, int damage)

# Get the amount of damage the Avatar causes monsters
getDamage(self)
```

CSCI 136 Assignment – Ultimate 0.1

The Avatar's line in the game text file will now have 5 numbers, for example:





```
11 2 20 3 100.0
```

This example line specifies that the Avatar starts at position (11, 2), starts with 20 hit points, causes 3 hit points of damage, and has an initial torch radius of 100.

Monster.py

The Monster class represents a monster that roams around the World randomly. A monster knows 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 the World object so they can call methods in World from their `run()` method (namely the `monsterMove()` method). If a monster is damaged it displays its remaining hit points in red text over the monster's image for three cycles of play. If a monster's hit points are 0 or less, the monster has been killed and no longer is shown, and its thread should terminate.

Each monster has its own Python thread that, periodically, attempts to move the monster 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 so, 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.

Name	Filename	Image	Code
Skeleton	skeleton.gif		SK
Orc	orc.gif		OR
Slime	slime.gif		SL
Bat	bat.gif		BA

CSCI 136 Assignment – Ultimate 0.1

```
class Monster
```

```
-----  
  
# Constructor initializes monster characteristics, including a  
# copy of the World object it has been instantiated in, a code  
# for the monster type, attributes for the avatar (initial  
# position, hit points, and damage), and a number representing  
# the number of ms it should sleep between moves.  
__init__(self, World world, String code, int x, int y, int hp, int damage,  
int sleepMs)  
  
# Reduce this monster's hit points by the amount damage  
incurDamage(self, int damage)  
  
# Draw the monster  
draw(self)  
  
# Get the number of remaining hit points of this monster  
int getHitPoints(self)  
  
# Get how much damage this monster causes  
int getDamage(self)  
  
# Get current x-position of this monster  
int getX(self)  
  
# Get current y-position of this monster  
int getY(self)  
  
# Move this monster to a new location  
setLocation(self, int x, int y)  
  
# Worker thread that periodically moves this monster  
run(self)
```

CSCI 136 Assignment – Ultimate 0.1

Monsters are defined at the end of the game text file. You can assume the values in the file are valid starting locations. Monsters won't start on top of each other, on top of a mountain, etc.

Here is an example which specifies the items below:

```
SK  3   3   10  3 1000
OR  6  19   8   2 1000
BA  20  10   4   1  500
SL  25  16   6   2 1500
```

- Skeleton at (3, 3) with 10 hit points and causes damage of 3. The skeleton attempts to move every 1000 ms.
- Orc at (6, 19) with 8 hit points and causes damage of 2. The orc attempts to move every 1000 ms.
- Bat at (20, 10) with 4 hit points and causes damage of 1. The bat attempts to move every 500 ms.
- Slime at (25, 16) with 6 hit points and causes damage of 2. The slime attempts to move every 1500 ms.

World.py

The World class requires some modifications. The constructor must now parse a file with more information about the Avatar as well as information about the monsters. The World object must create and keep track of the Monster objects. Each monster has a thread, so you will need to create one thread per monster. You will also need to implement two methods that handle attempting to move the Avatar or a Monster. Finally, you will implement two methods that determine the end of the game: `avatarAlive()` and `getMonsters()`. The game ends when the avatar no longer has any hit points left, or all monsters have been eliminated.

CSCI 136 Assignment – Ultimate 0.1

```
class World
-----
# Load the tiles and Avatar based on the given filename
__init__(self, String filename)

# Handle a keypress from the main game program
handleKey(self, char ch)

# Draw all the tiles and the Avatar
draw(self)

# Set the tiles that are lit based on an initial position (x,y)
# and a torch radius of r. Returns the number of tiles that were
# lit up by the algorithm.
int light(self, int x, int y, float r)

# Recursive lighting method called by light(). (x, y) is still
# the position of the avatar, and (currX, currY) is the
# position being considered for lighting.
int lightDFS(self, int x, int y, int currX, int currY, float r)

# Turn all the lit values of the tiles to a given value. Used
# to reset lighting each time the avatar moves or the torch
# strength changes.
def setLit(self, value)

# Is the Avatar still alive?
boolean avatarAlive(self)

# Attempt to move given monster to (x, y)
monsterMove(self, int x, int y, Monster monster)

# Attempt to move Avatar to (x, y)
avatarMove(self, int x, int y)

# Return number of alive monsters
int getNumMonsters(self)
```


CSCI 136 Assignment – Ultimate 0.1

The `monsterMove()` should be called by a monster's `run()` method. 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 attacks the Avatar and does the specified 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). Since only the World object knows the outcome of the monster's call to `monsterMove()`, the World object must update the calling Monster object by calling `setLocation()` and/or `incurDamage()`.

The `avatarMove()` method should be called when the `handleKey()` method tries to move the Avatar. Like 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).

Helpful Hints:

Do I need to follow the given specifications? Yes. You must implement all methods as described. You may add additional methods if they are helpful to your code.

Why does `monsterMove()` get passed a monster object? The `monsterMove()` method in the World class gets called by the monster's thread. For the method to do things like damage the monster for walking on lava or updating its location, it needs a reference to the object. While the monster `run()` method could do this, it could cause concurrency problems.

How do you pass a monster object to `monsterMove()` ? The `run()` method can use the `self` keyword which is a reference to the object running the method.

How do you change the font for your hit point display? `StdDraw.setFontSize(12)`

How do you display hit points for only three moves? You might want to implement a "timer" attribute in your Monster class to keep track. When a monster incurs damage, reset the "timer" to 0. When a monster is drawn, only draw hit points if the timer is less than 3, and if so, increment the timer.

How do you test if you have handled concurrency correctly? Try running the `10x10_full.txt` world. This world has monsters at all grid positions aside from the Avatar's position. The program should not crash and the monsters should not end up on top of each other.

If a monster calls `time.sleep(1)` or similar between moves, will the monster move exactly every second? No, it might take a little longer than a second. `time.sleep()` makes sure your thread isn't scheduled for execution for at least the specified period of time. Thereafter the

CSCI 136 Assignment – Ultimate 0.1

thread must wait to be scheduled on the CPU which could take more time. For the purposes of our game, this is close enough.

Submit Your Work

After following all instructions above, please submit your Python scripts (`Avatar.py`, `Tile.py`, `World.py`, and `Monster.py`) in Moodle. Upload these as individual files – do not zip them. Be sure the script files have header sections that include the file name, your name, credits, a description of the file, the class and other useful information. Make sure your classes and all methods are thoroughly commented.

- `Avatar.py`
- `Tile.py`
- `World.py`
- `Monster.py`

EXTENSIONS

~~You will not submit the answers to these extensions as part of your assignment submission. You should work through and understand them to learn more and practice. You may be held responsible for the content within, so you really should work through them. With that said, this are outside the scope of what will be submitted for this lab. Do not change your original submittal to address these extensions.~~

There are no extensions for this lab. Do your best to get it right. If you want to experiment, do it separate from what you submit.