**Day 1**

**I. Vẽ Bản Đồ**

* Basic Setting
* Chiều rộng của màn hình (maxTileRows)
* Chiều cao của màn hình (maxTileCols)
* Thiết kế class Tile:
* WallTile ~ tile[0] ~ < >
* LandTile ~ tile[1] ~ [ ]
* WaterTile ~ tile[2] ~ ( )
* FireTile ~ tile[3] ~ { }
* Đọc bản đồ từ file text

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3

1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 4 4 4 4 1 1 1 1 1 1 1 1 4 4 4 4 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

……………………………………..

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

* Xây dựng class Map có tính năng

- Hiển thị bản đồ với các ký tự <>, [], (), {}

- Hiển thị dấu vết của player (đã xong)

- Quản lý và hiển thị dấu vết của các monsters (**chưa hoàn thiện xong**)

**II. Thiết kế class Character và Player để**

- Player được hiển thị cùng với bản đồ.

**III. Thiết kế class Lion, Wolf, …**

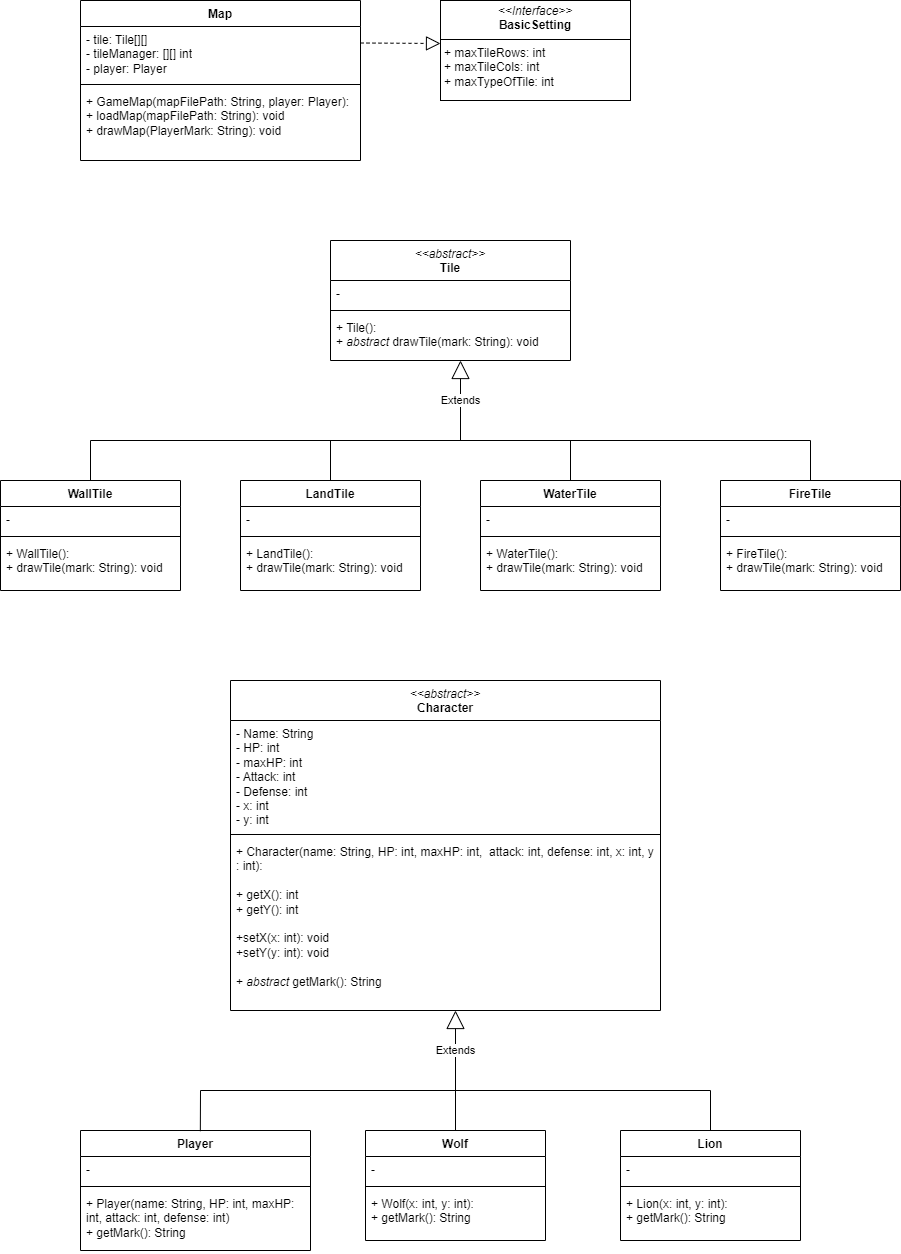


Figure 1.UML for Day1

**Day 2**

**I.** **Hiển thị các monsters trên bản đồ**

1. Thiết kế class GameMain ở đó:

- Player sẽ được khởi tạo.

- Danh sách các quái vật được khởi tạo cụ thể vào được quản lý (lưu trữ) bởi mảng một chiều (sử dụng List).

- Class GameMain được liên kết với class Map

* Quái vật nên được quản lý bởi class Map hay class GameMain???

Answer: chúng ta sẽ quản lý các đối tượng không thể di chuyển (Tile và Item) trong game thông qua class Map và quản lý các đối tượng có khả năng di chuyển (player và monsters) thông qua class GameMain.

1. Chỉnh sửa lại class Map để hiển thị thêm các quái vật:

- Chỉnh sửa lại hàm drawMap để nó hiện thị tất cả các ô Tile ngoại trừ các ô Tile mà vị trị của các ô này trùng với vị trí của một trong các quái vật. Đối với mỗi ô Tile mà vị ví của nó trùng với vị trị của *một quái vật nào đó,* chúng ta cần phải xác định “*một quái vật nào đó”*  chính xác là một con quái vật cụ thể nào. Hay nói cách khác, **cho trước một vị trí (x, y) chúng ta cần xác định xem quát vật nào trong array có vị trị này (hoặc tất cả đều không có)**. Để làm được điều đó, chúng ta xây dựng thêm 2 hàm:

+ boolean containMonsterAt(int x, int y): Hàm này hỗ trợ cho việc kiểm tra liệu có bất kỳ quái vật nào nằm trong vị trí (x, y)

+ Character correspondingMonsterAt(int x, int y): Hàm này trả về quát vật có vị trí (x, y). Chẳng hạn, nếu các quái vật được lưu trữ bởi mảng một chiều như sau

monster[0] = Wolf(3,3)

monster[1] = Lion(5,5)

monster[2] = Wolf(7,7)

monster[3] = Wolf(9,9)

thì correspondingMonsterAt(7, 7) sẽ trả về monster[2]

Cuối cùng, hàm drawMap sau khi được update sẽ như sau:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **public** void drawMap()  {  **for**(int i = 0; i < maxTileRows; i++)  {  **for**(int j = 0; j < maxTileCols; j++)  {  **if**(i == gm.player.getY() && j == gm.player.getX() ) *//Draw player first*  {  tile[tileManager[i][j]].drawTile(gm.player.getMark());  }  **else** if(containMonsterAt(i, j)) *//Draw monsters*  {  tile[tileManager[i][j]].drawTile(correspondingMonsterAt(j, i).getMark());  }  **else** *//Draw tiles*  {  tile[tileManager[i][j]].drawTile(" ");  }  }  System.out.println("");  }  } | |

1. **Hiển thị các Item trên bản đồ**
2. Xây dựng class Item, Weapon, Axe, Sword

* Nếu vị trí các Item được quản lý bởi 2D-array thì Item có cần thuộc tính x ,y ?? (Hay nói cách khác, nếu ta sử dụng 2D-array để quản lý vị trí các Item thì thuộc tính x, y của Item là không cần thiết và ngược lại.)

1. Cách quản lý các Item

- Các Item được lưu trữ bởi List

- Xây dựng hàm setItem để tạo ra các Item ở các vị trí mong muôn trong bản đồ. Lưu ý hàm này sẽ hoạt động khác nhau ở mỗi bản đồ khác nhau 🡺 Class Map nên trở thành *abstract* và hàm setItem nên dành cho việc override.

3. Cuối cùng ta chỉnh sửa lại class hàm drawMap để các Item được hiện thị

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | **public** **void** drawMap()  {  **for**(**int** i = 0; i < maxTileRows; i++) *//i ~ y-coordinate in xy plane*  {  **for**(**int** j = 0; j < maxTileCols; j++) *//j ~ x-coordinate in xy plane*  {  **if**(i == gm.player.getY() && j == gm.player.getX() ) *//Draw player first*  {  tile[tileManager[i][j]].drawTile(gm.player.getMark());  }  **else** if(containMonsterAt(j, i)) *//Draw monsters*  {  tile[tileManager[i][j]].drawTile(correspondingMonsterAt(j, i).getMark());  }  **else** if(containItemAt(j, i)) *//Draw items*  {  tile[tileManager[i][j]].drawTile(correspondingItemAt(j, i).getMark());  }  **else** *//Draw tiles*  {  tile[tileManager[i][j]].drawTile(" ");  }  }  System.out.println("");  }  } | |

A screenshot of a computer

Description automatically generated

Figure 2. UML for Day2

**Day 3**

1. **Xây dựng chức năng di chuyển cơ bản của nhân vật:**

- moveUp, moveDown, moveLeft, moveRight : các hàm này sẽ giúp nhân vật có thể di chuyển tất cả các vị trí trên bản đồ, các hàm này cần bảo đảm rằng nhân vật không di chuyển ra khỏi ngoài khung của game.

- Xây dựng chức năng di chuyển random cho các monsters.

1. **Kiểm tra sự đụng độ (collision) của các nhân vật trước khi di chuyển:**

- Kiểm tra player và monsters có di chuyển vào các ô Solid (Wall tile) (đã xong)

- Kiểm tra player có di chuyển vào ô có Item. (chưa xong)

- Kiểm tra player có đụng độ (trong range) với bất kỳ monster nào trên bản đồ. (chưa xong)

- Kiểm tra các monsters có đụng độ (trong range) với chính nhau. (chưa xong)

A screenshot of a computer

Description automatically generated

Figure 3. UML for Day3

**CÁC GIAI ĐOẠN LÀM DỰ ÁN**

1. **Tìm Cấu Trúc Dữ Liệu để lưu trữ các đối tượng và Hiển Thị Các Đối Tượng**

- Vị trị của các ô Tile được quản lý bởi mảng 2 chiều.

- Các monsters được quản lý bởi List.

- Các items được quản lý bởi List.

1. **Thiết kế tính năng Di Chuyển cho nhân vật (bao gồm player và monsters)**

- player và monsters không được di chuyển vào các ô Solid (Wall)

- monsters có thể di chuyển random hoặc tìm đến player.

1. **Thiết kế các Tương Tác cho các đối tượng trong game**

- Xây dựng sự tương tác cho 4 đối tượng chính: player, monster, map, item

1. **Sử dụng UI**