CIT 594 HW6

Zhiyuan Li
Yi Shang
Di Wu

Technique Requirement

Data Structures

We use at least 3 kinds of data structures, namely:

1. Array and 2 dimensional array
2. ArrayList
3. HashMap
4. Priority Queue
5. Stack
6. Graph and BFS algorithm
7. Complex data types

Java Graphics

This is a GUI-based game implementation, so we use Java graphics in our project, namely Java Swing library.

Design Requirement

Design Pattern

1. We have been sticking to MVC model to implement our project.
2. We use the Factory Method Pattern to facilitate the implementation of each window controller.
3. Also, when initializing the game board and placing monsters, we are using the Factory Method Pattern to facilitate the implementation.

Interface

```
/************************
 ********   Site   ********
 ************************/
/**
 * This class is used to store site info
 */
public class Site {

        /**
         * constructor
         */
        public Site(int x, int y) { }

        /**
         * gets x position
         */
        public int getX() { }

        /**
         * gets y position
         */
        public int getY() { }

        /**
         * gets the Manhattan distance between this site and the other
         */
        public int manhattanTo(Site that) { }

}
```

```
/**************************
 ********   Rogue   ********
 **************************/
/**
 * This class is used to store rogue info
 */
public class Rogue {

	/**
	 * constructor
	 */
	public Rogue(Game game) { }

	/**
	 * rogue power up
	 */
	public void powerup() { }

	/**
	 * rogue take damage
	 */
	public void takeDamage(int damage) { }

	/**
	 * the number of sword the rogue currently have
	 */
	public int getNumberSword() { }

	/**
	 * the current rogue hp
	 */
	public int getHp() { }
```

```
    /**
     * if the rogue has any sword
     */
    public boolean hasSword() { }

    /**
     * adds a sword to rogue
     */
    public void addSword() { }

    /**
     * removes a sword from rogue
     */
    public void removeSword() { }

    /**
     * check iif rogue is dead
     */
    public boolean isDead() { }

}
```

```
/*****************************
 ********   Monster   ********
 *****************************/
/**
 * This class represents an abstract monster
 */
public abstract class Monster {

        /**
         * constructor
         */
         public Monster(Game game, String name) { }

         /**
          * gets the name of the monster
          */
        public String getName() { }

         /**
          * gets the game
          */
        public Game getGame() { }

        /**
         * gets the damage
         */
        public int getDamage() { }

        /**
         * gets the dungeon
         */
        public Dungeon getDungeon() { }
```

```
        /**
         * gets the size
         */
        public int getSize() { }

          /**
           * takes a legal move for the monster
           */
           public abstract Site move();

}
```

```
/**************************
 ********   Game   ********
 **************************/
/**
 * Game class to control whole game, offer method for Game controller
 */
public class Game extends Observable {

        /**
         * Constructor for game
         */
        public Game() { }

        /**
         * Used for set level map
         */
        public void setLevelMap(String filename) { }

        /**
         * gets the rogue of this game
         */
        public Rogue getRogue() { }

        /**
         * gets the monster site info of this game
         */
        public HashMap<Monster, Site> getMonsterSiteMap() { }

        /**
         * gets the dungeon
         */
        public Dungeon getDungeon() { }
```

```
/**
 * gets the position of monster
 */
public Site getMonsterSite(String name) { }

/**
 * sets the position of monster
 */
public void setMonsterSite(HashMap<Monster, Site> monsterSite) { }

/**
 * gets the position of rogue
 */
public Site getRogueSite() { }

/**
 * sets the position of rogue
 */
public void setRogueSite(Site rogueSite) { }

/**
 * gets rid of the power up location if it exists
 */
public boolean removePowerUpSiteMap(Site s) { }

/**
 * gets the powerUpSiteMap
 */
public ArrayList<Site> getPowerUpSiteMap() { }

/**
 * gets the tunnelSite
 */
```

```
public Site getTunnelSite() { }

/**
 * check if current rogue site is tunnel site
 */
public boolean isTunnelSite() { }

/**
 * check is current rogue site is monster site
 */
public boolean isMonsterSite() { }

/**
 * remove the monster
 */
public void removeMonster(Monster m) { }

/**
 * return which monster caught the rogue
 */
public Monster caughtBy() { }

/**
 * sets the sword site
 */
public void setSwordSite() { }

/**
 * check if current rogue site is sword site
 */
public boolean isSwordSite() { }

/**
```

```
         * gets the swordSite
         */
        public Site getSwordSite() { }

}
```

```
/*******************************
 ********   Dungeon   ********
 *******************************/
/**
 * This class represents a dungeon, which is composed of rooms, corridors
and walls
 */
public class Dungeon {

        /**
         * constructor
         */
        public Dungeon(char[][] board) { }

        /**
         * returns dimension of the dungeon
         */
        public int size() { }

        /**
         * returns character representation of this board
         */
        public char[][] getBoard() { }

        /**
         * tells if site is a corridor site
         */
        public boolean isCorridor(Site site) { }

        /**
         * tells if site is a room site
         */
        public boolean isRoom(Site site) { }
```

```
    /**
     * tells if this site is a wall
     */
    public boolean isWall(Site site) { }

    /**
     * tells if s1 to s2 is a legal move
     */
    public boolean isLegalMove(Site s1, Site s2) { }

}
```