Project

Dragon Maze

CS 5060 Intensive Programming, Fall 2012

237 points

Due: 11:59 pm December 6, 2012

Assignment description

This is your final assignment. It is a small project in which you will create a game called *Dragon Maze*. When the game starts, the player is inside a maze with a dragon, and the objective is to escape from the maze without getting killed by the dragon.

For this project you will have more flexibility in choosing how to do things both in terms of how you design your classes and how you organize your code. However, you should follow basic design principle and coding style rules. Grading will be based on at least the following criteria:

- Completeness: the program has all the features outlined in the assignment description.
- Correctness: the program not only has the required features, but the also works as expected.
- Code quality: code should be well organized and easy to understand, and it must include helpful comments. Remember to follow the style rules and design your class structure and methods carefully.
- Efficiency: code should use appropriate data structures and algorithms, and it should be fast enough. Since the game worlds are usually small most algorithms are fast enough, but an algorithm that would be too slow for larger worlds could receive partial credit only.
- Output: output should be clear and easy to understand, with appropriate labels and formatting.
- Errors: the program should be able to handle error conditions gracefully.

Even though the game is small, with a lot of things that could be added to make it into a complete game, your submission will be evaluated as if it were a real project. You will receive a detailed comments intended to be a final evaluation for you to learn from. Think of your submission as a code sample for a job application. Hence, if you cannot finish everything, it is better if you have part of the functionality with good code than all of the functionality with bad code. Think that you would want to impress your evaluator, and also your classmates. Some of the best projects will be shown in class on the last day!

Your submission must satisfy the following general requirements:

- Use at least two data structures (you may use data structures that we have not implemented).
- Have at least one recursive method.
- Use classes and inheritance

- Include one personal feature/extension (you can choose among several options).
- Include a README.txt file with details about your work.

The project may seen a little bit long, but if you are organized you should be able to complete it. Make sure you have things clear before you start coding. There is only a couple of big things you have to do, but the other things are simple.

Starting the game (50 points)

When the game starts, there should be a menu with at least the following options:

- Load a maze.
- Generate a random maze.
- Help with instructions on how to play the game.
- About option that displays name of the game, your name and your A number.
- Exit.

Load maze: The player specifies the name of the file to load. After completing the maze, the game goes back to the menu.

The files should be in a `maps` folder, and the user should only have to type the name of the file.

Random maze: The player chooses a difficulty level (easy, medium, or hard) and a maze is generated. After completing the maze, the game goes back to the menu.

Refer to the following link for ideas on how to generate the maze:
`http://en.wikipedia.org/wiki/Maze_generation_algorithm`.

Grading:

- Menu, help and about options: 10 points.
- Loading maze: 20 points.
- Maze generator: 20 points.

World representation and classes (85 points)

The world is saved in a file with the following format. The first line has three numbers: a number of rows r ($1 \leq r \leq 80$), a number of columns c ($1 \leq c \leq 80$), and a time limit t ($t \geq 0$, where 0 means no time limit). The next r lines describe the initial state of the world using the following symbols:

- `'.'` for an empty cell,

- `'X'` for a wall,
- `'D'` for a dragon,
- `'H'` for the hero (player),
- `'E'` for the exit.

Example:

```
5 11 50
XXXXXXXXXXX
X..X......X
X.DX...X..X
X....H.X..E
XXXXXXXXXXX
```

Note: For auto-generated mazes, there is no file, but the maze is displayed in the same way on the screen.

Your should have the following classes:

- `DragonMaze`: Main program.
- `World`: Contains global information about the game, including the `Grid`, time, etc.
- `Grid`: Contains a description of the world.
- `GridCell`: A cell of the `Grid`. Each cell has a location (x, y), and it can have one `Actor`.
- `Actor`: Any object that can be in a `GridCell`.
  - `Wall`: It is not possible to walk over walls.
  - `Exit`: The player wins when the exit is reached.
  - `Dragon`: Dragons move on their own.
  - `Hero`: The hero is controlled by the player.

Grading:

- DragonMaze (10 pts),
- World (10 pts), Grid (10 pts), GridCell (10 pts),
- Actor (10 pts), Wall (5 pts), Exit(5 pts), Dragon (10 pts), Hero (10 pts).
- Put your classes in a package named: `cs5060.project.dragonmaze`. Use subpackages to organize your classes. (5 pts)

Playing the game (10 points)

Your program should output the initial state of the game and the state of the game after each time step as follows (leave an empty line after each time step):

```
Time limit: 50
Current time: 5
[status message]
XXXXXXXXXX
X..X.....X
X.DX...X..X
X....H.X..E
XXXXXXXXXX
```

A time step is counted each time the player attempts a move, even if the player's robot cannot move because there is a wall on the way. If the player wins the game, output the state of the world followed by a line containing the word "`SUCCESS`". For example:

```
Time limit: 50
Current time: 9
The hero managed to escape from the dragon!
XXXXXXXXXX
X..X....D.X
X..X...X..X
X......X..H
XXXXXXXXXX
SUCCESS
```

If the time expires or the hero dies, output the state of the world followed by a line containing the word "`FAILURE`". For example:

```
No time limit
Current time: 3
The hero was captured by the dragon...
XXXXXXXXXX
X..X....D.X
X..X...X..X
X...D..X..H
XXXXXXXXXX
FAILURE
```

Note: You can come up with your own status messages.

Player actions (17 points)

When playing the game, the player can input the moves by typing:

- `'l'` to move left,
- `'r'` to move right,
- `'u'` to move up,
- `'d'` to move down,
- `'w'` to skip a turn,
- `'q!'` to quit the maze.

Example:

```
Time limit: 50
Current time: 0
The hero is trapped inside a maze with a terrible dragon.
Will (s)he be able to escape...?
XXXXXXXXXXX
X..X......X
X.DX...X..X
X....H.X..E
XXXXXXXXXXX

u
Time limit: 50
Current time: 8
You can hear the steps of the hero moving up...
and the dragon moving down...
XXXXXXXXXXX
X..X......X
X..X.H.X..X
X.D....X..E
XXXXXXXXXXX
```

Grading:

- Move and wait: 15 points.

- Quit: 2 points.

Dragon actions (20 points)

When playing the game, the dragon can choose to do the following actions:

- Move closer to the hero: 15 points
- With a small probability, destroy a wall (it takes three consecutive turns to destroy a wall).

Grading:

- Move: 15 points.
- Destroy wall: 5 points.

Note: Moving closer to the hero means that the number of steps required to reach the hero is reduced.

Extra features (30 points)

Choose one of the following game features:

- The hero must get a key before opening the exit door (use `'k'` to represent a key in the map).
- Use a bomb to destroy walls (use `'b'` to represent a bomb in the map). Bombs cannot kill the dragon.
- After the time limit, the game does not end, but you add another dragon.

Also, choose one of the following items:

- Freeze orb: after collecting a freeze orb, the hero can freeze the dragon if the dragon is at most 3 steps away, and the dragon cannot move for the next 3 turns. A freeze orb can only be used once. (use `'f'` to represent a freeze orb in the map).
- Energy drink: after collecting an energy drink, the hero can move twice in one time step, for 10 time steps (use `'e'` to represent an energy drink in the map).
- Invisibility cloak: after collecting an invisibility cloak, the dragon will not be able to see the hero, for 10 time steps. (use `'i'` to represent an invisibility cloak in the map).

Grading:

- Game option: 15 points.
- Item: 15 points.

Maps (10 points)

Create two interesting maps (5 points each). The maps must be as follow:

- size must be at least 20×20,
- maps cannot be very similar,
- it should be necessary to use a special item to exit the map.

Submission (15 points)

Submit a `zip` file with all your files. Make sure you include the package folders.

Include a `README.txt` file with details, special instructions, and comments about your program.

Include your name and A number at the top of each file. Name the `zip` file `prj_firstName_lastName.zip`. For example, if your name is John Smith, name the file `prj_John_Smith.zip`.

Grading:

- Zip file: 5 points.
- README file: 10 points.