GRADE

Functionality: There is almost nothing you could do to cause glitches or unintended behavior. User input is limited to typing which is vetted by switch statements that default to nothing. One thing that could cause the program to crash is if you spend a lot of time getting to a high level, and then find a light tile which would make the program run very slow/crash because it has to draw so much tile graphics. We suggest giving us full points here.

Design: Some problems were handled cleverly/well, but sometimes we prioritized speed over elegance. The TableList is kind of unnecessary since it's just an ArrayList, but there is some functionality to it we wanted to have. Some things are unnecessarily public, but that doesn't hurt anything. There is nothing too much that we find gross or unsettling, but there is probably some fat on the steak that could be trimmed off, so to speak.

Creativity: DUNGEON CRAWL draws inspiration from the dungeon crawler genre, but is not a replication or expansion of an existing game. The method of generating dungeons is unique (as far as we know), and we don't think there is an existing game that has all the features that DUNGEON CRAWL has to offer. Also, we think we deserve credit for making fun and comedic graphics, making the game even spookier, and winning the 2009 game of the year award.

Sophistication:
We believe we worked at least six times harder on this than for any lab--so by that metric, we satisfy the requirements for full points here.
In addition, the underlying algorithms and systems for generating the maze, navigating through through the main menu, creating the blindness, implementing the graphics, and organizing the leaderboard are all reasonably sophisticated. We think we deserve full points in this category.

Broadness:
1. Java libraries that we haven't seen in class
    We used lots of input/output libraries as well as fonts to make the game look pretty.

2. Subclassing
    Player, Chaser, and Subtile are all subclasses that utilize variables and methods from their parent class. While not completely necessary, it avoids copy and pasting the same code.

3. Interfaces (`implements` an interface you created)
    NONE

4. User-defined data structures

The map could be seen as a data structure (double linked array?), but the queue would definitely count as user defined, and is needed for the chaser to correctly chase the player.

5. Built in data structures

ArrayLists are used in the TableList class, and matrices are used in multiple places.

6. File input/output

The leaderboard text file is read and written by the program. Image files are used to draw stuff.

7. Randomization

Tons of randomly generated things are used to make the game different and interesting.

8. Generics

Used in the queue data structure, I could have just defined queue to only hold characters, but I felt like generics would be better semantically.

Since we have 7/8, we should get full credit for this area.

Code Quality:

The code is annotated--not obsessively, but enough that we believe anyone in our class or at a higher level would be able to figure out what's happening without our explanations or the program specification document. It ought to be clear what each method does and what each variable means. Organization and cleanliness is also not, admittedly, obsessive--but overall we don't believe there are any significant offenses. While we do not expect all points with respect to this category, we think we're deserving of a decent score.