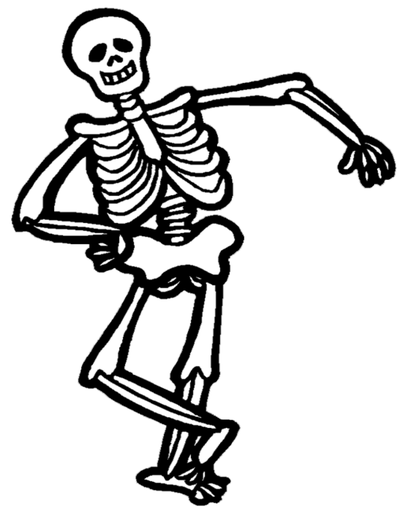
COSC 365 Programming Assignment 1

Escape the Graveyard



For this programming assignment, you will use C# to write a text-adventure game in object-oriented style. In this game, the player will try to find a key to escape a graveyard and evade skeletons hiding in the ground, collecting coins along the way. The player will type commands to examine locations and move around the level.

Your program will take the path to a file as its only command line argument. That file will contain text commands that describe a level. Your program will read these commands to create the necessary objects to represent the level. As your program reads commands from the user (via `stdin`) to play the game, it will produce output in response to the commands. At the end of the game, some additional information will be printed out.

You will be given a source file with some starter code called `etg.cs`, which can be compiled by running:

```
UNIX$ mcs etg.cs # produces etg.exe
```

Your program can be run interactively:

```
UNIX$ mono etg.exe test/1/level
```

You can also "simulate" user input by redirecting a command file to `stdin`:

```
UNIX$ mono etg.exe test/1/level < test/1/input1
```

Your job is to complete the program using object-oriented features (specifics are outlined below).

A reference executable is included, which can be run with the provided levels and inputs to produce the correct output. Additionally, an included grading script can be used to compare your output to that of the reference executable.

Requirements and Constraints

- You submit only a single C# file called `etg.cs`, which contains your whole program. Don't split the code into multiple files. Don't `tar` or `zip` it.
- Your program's output matches the reference executable *exactly* for the provided inputs.
- The player wins the game if they reach the exit location (defined in the level file) AND have a key.
- The player loses the game in three ways:
  - Get attacked: they reach a location with a hiding skeleton.
  - Die of exhaustion: they take more than `2*w*h` actions, where `w` is the width of the level and `h` is the height of the level.
  - End of input: no more input is provided, but ending conditions have not been met.
- Statistics are printed out when the game ends. Example:

```
Game ended after 0 turn(s).
  LOCATION: 0, 0
  COINS:    0
  KEY:      False
  DEAD:     False
```

- Your program has the following classes:
  - `ETG`: the program's class. Will contain your `Main()` method.
  - `Game`: the game's main driver class. Will contain your `Player` and `Level` objects, among other things. It will also handle input.
  - `Player`: the class containing information about the player.
  - `Level`: contains a grid of `Locations`.
  - `Location`: contains zero or more `Entity` objects. Handles interaction with each entity when the player inspects or occupies the location.
  - `Entity`: an `abstract` base class for game objects. Defines at least these two methods:
    - `look()`: describes the entity (if it can be seen) when the player looks at its containing location.
    - `interact()`: defines what happens when the player goes to a location containing the entity. For example, a skeleton may kill the player or a key may be taken.

- Use inheritance to create a class derived from `Location` that represents the exit location in the level. This class should override the behavior of `Location` so that the player does not see or interact with any entities at the exit. Instead, it should print messages about the gate and key according to the reference executable.
- Use inheritance and polymorphism to create classes for keys, loot (treasure chests with coins), and skeletons. These classes should inherit from `Entity` and implement the missing functionalities: `look()` and `interact()`.
- Keys and loot can be seen and taken, while skeletons can't be seen and kill the player on `interact()`.
- Proper encapsulation techniques should be used. For example, the `Player` class should hold the player's location and be responsible for updating it.
- Once a key is taken, it is removed from the location.
- If loot is taken from a treasure chest, it becomes empty, but remains at the location.
- The player may not move or look past the bounds of the level.
- The player's coordinates are printed before each action.
- You may use any classes under C#'s `System` namespace, but may NOT use any outside libraries or packages.

Grading

- **Program Correctness: 50%**

Does your program run correctly and produce the right output for the given inputs? This part of your grade can be computed by using the provided grading script.

- **Paradigm Adherence: 50%**

Does your program solve the problem using the object-oriented fundamentals discussed in class and required from this write up?

NOTE: Your program will be graded for correctness by running it with the provided grading script on the Hydra lab machines. Ensure that you test on those machines before submitting so that you may be confident in that portion of your grade.

Tips

- It is highly recommended that you use the reference executable to play the game for a few of the test levels to understand the desired output.
- Use the grading script to determine where your output differs from that of the reference executable in order to narrow down missing functionality or output formatting.