Dungeon Crawler

Contents

[Declaration 1](#_Toc511120000)

[Tutorial 2](#_Toc511120001)

[Critical Appraisal 2](#_Toc511120002)

[Classes 3](#_Toc511120003)

[Testing 3](#_Toc511120004)

# Declaration

MIT License

Copyright (c) 2018 Tilman Wirawat Raendchen

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

# Tutorial

Playing this game could not be any easier. I went out of my way to implement a way for anyone to pick up the game and immediately start playing. Understanding the game is another feat, however.

I wanted to make the game follow the specification as closely as possible, which means that the player has to walk through a forest and come across enemies and rocks, both of which drop items that can be collected. The player can then choose to go into the sewer, where there are also enemies to be found. There are also doors, which can be unlocked with either enough keys or a wand.

The game itself was constructed in a very modular fashion, meaning that all parts can be taken apart and brought back together for maximum modularity. This also means that I have not constructed “orchestrated” monster fights, but rather a single monster fight that has some variety.

In order to complete the game, the player needs to collect enough keys in order to be able to open the doors in the Sewer. Doors can also be opened by obtaining and equipping the ultimate weapon: the ‘Wand’.

There is a small grinding factor as well. You cannot just spam the Enter key and be done with it. There is a degree of difficulty that comes with the game. If you want to circumvent this, take a look at the constructor method for the Game class.

# Critical Appraisal

There were moments during development when I was at a loss because of a bug that appeared out of nowhere. But then I googled the error for 20 minutes and I would eventually be able to fix it.

I tried to make my game as modular as possible simply for good practice. During my final phase of commenting the code, I noticed some classes that I could have modularised much better. An example of such an instance is my Inventory and Gear dilemma. Both of these classes share the same parent type, and I think it would have been better to create a base class that shared that parent type, such as an ItemSpace.

However, I do think I did a well enough job with the modularity of the overall program. After all, this is what allowed me to create unlimited unit tests for my testing needs. To create the BaseEnemy class that inherits from the Character class, which then allowed me to create multiple enemies, for example, was a good call I believe.

There is one aspect that I recognise as not having fulfilled that was required by the specification, which is that my game doesn’t feature doors that are randomly open or closed. I got so much into it that I simply forgot about this feature. All of my doors are now inherently closed on creation.

Overall, I’m very satisfied with the outcome of my game. I might even work a little more on it when I have free time and enhance it. Perhaps it will become the next big game? 😊

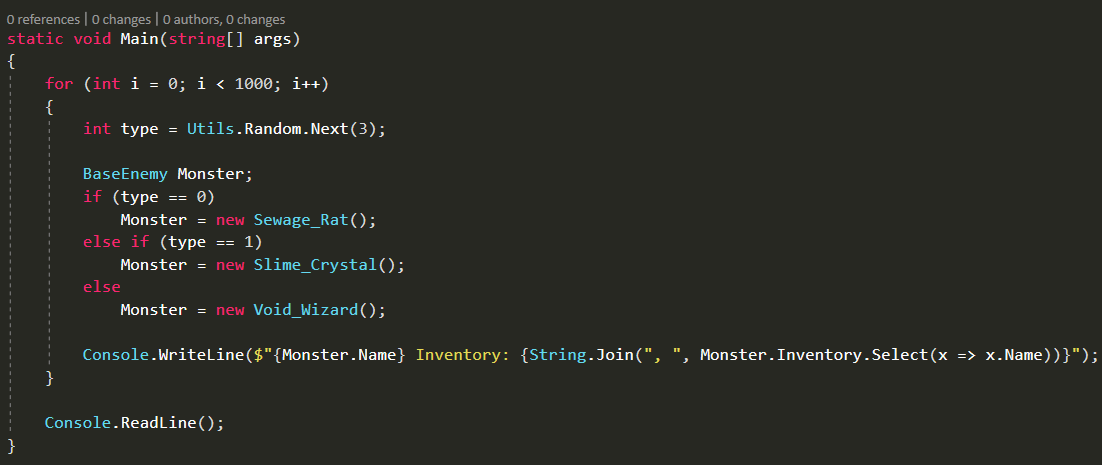
# C:\Users\TilmanRaendchen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\UML_Diagram.pngClasses

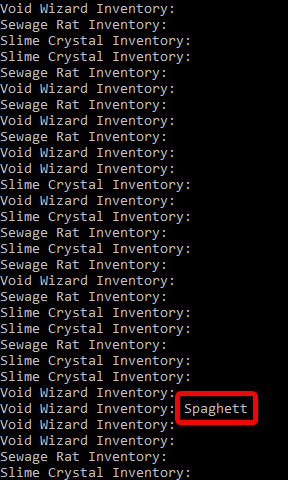
This is a UML diagram I spent some hours on designing that accurately represents all classes and their inheritance. This image can be enlarged either in Word or by saving it to file and viewing it in another Image Viewing Application.

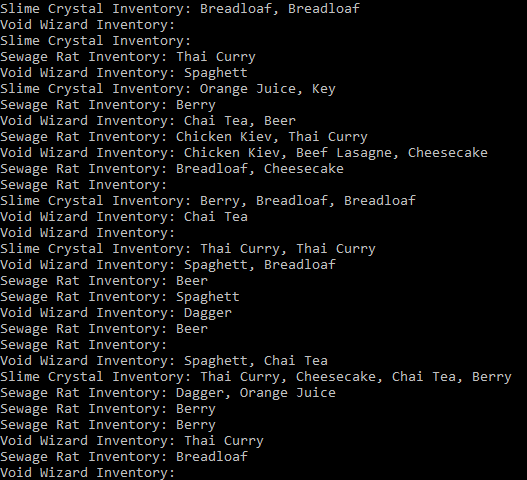
# Testing

I think the way I test my projects is quite universal in the gameplay programming paradigm. I usually **prototype** functions and features and make use of **agile methodologies** by testing it throughout. I use a mix of both **white-box** and **black-box** testing; **grey-box** testing essentially. By this I mean that I will test both the functionality of the feature and the usability, as well as the design of it. The whole point of agile methodologies is that the client is involved in the feedback phases, but since I am more or less the client and the developer, I can race through these stages in no time.

One example of this is what happened at some point during development. The basic explanation is that I was play-testing the game as a whole, but got annoyed at the fact that monsters didn’t drop anything. I took a look at the code but couldn’t figure out what was wrong just from looking at it. So I created a unit test:



This would show me the inventories of any Monster I spawned, so I spawned a thousand Monsters just to make sure. And surely enough:

More or less every 100th Monster had a single item. There was definitely something wrong. The only thing I could do now is go to the source and step through the program. This allowed me to see where I accidently divided by a hundred, making the drop chance 0.01% for nearly every item. Fixing that allowed me to get the desired outcome: