PyMUTA (Python Modular Ugly Text Adventure)

A text based adventure game in Python 3.

# Introduction

This is the documentation written for my game MUTA. I have designed the game with one of my largest goals for it to be extendable and easy for others to read and adapt. This document will explain the different documents present in the files, which will help a user understand what each files main purpose is and how it can be manipulated or extended with ease. I should note that this began as a coding project with a book called Make Your Own Python Text Adventure (Johnson, 2018), I would definitely recommend it to someone starting Python, but not necessarily someone who doesn’t understand coding practices.

Contents

[Introduction 1](#_Toc513709028)

[The Varying Files 1](#_Toc513709029)

[game.py 1](#_Toc513709030)

[commands.py 1](#_Toc513709031)

# The Varying Files

Within the main game folder, there are a bunch of important files that anyone can extend or change to their desire, which will be mentioned here. A couple which won’t need mentioning however, is the Launcher shortcut used to start the game, and the \_\_init\_\_.py file which simply initialises the folder as something python is allowed to look at, to find the other files.

## game.py

This is the main game script, which contains the play loop needed for the game to run. The code starts with importing the valid files it needs (such as the commands.py file) and then shows a brief welcome message with a menu selection. On the user input, the script will choose to try and import the save.py file (if there is one), or simply start the game new. After this the play() is defined, if you wish to add any of your own commands, this is where you can place the arguments on the user input to do it. The game.py lastly features a if \_\_name\_\_ == '\_\_main\_\_': segment to ensure the script is running as the main module, and not being imported, this can be removed if you wish to import game.py for whatever reason.

## commands.py

|  |
| --- |
| **Importation** |
| **Get Player Input** |
| **Accepted Inputs** |
| **Different Commands** |

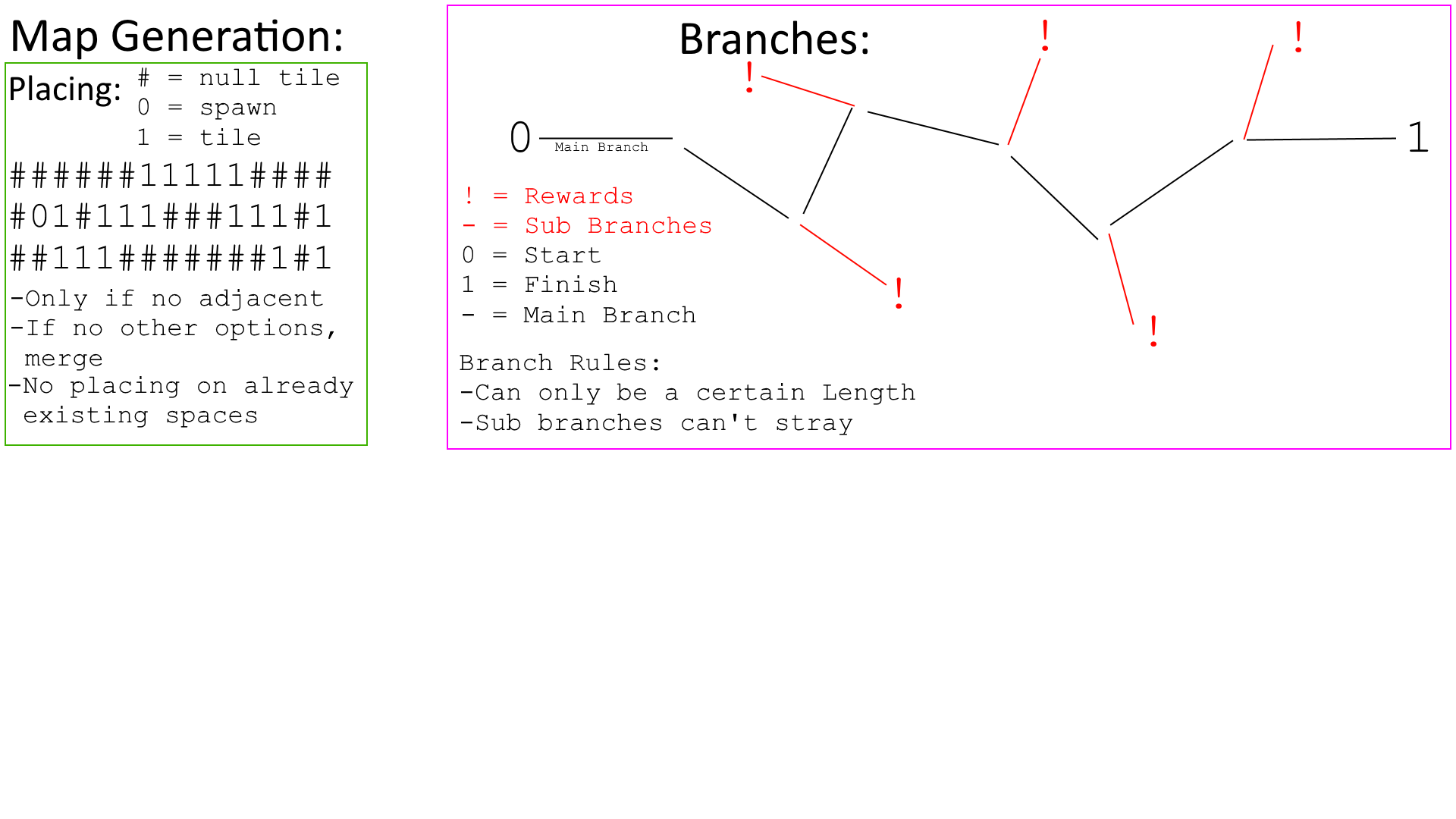
This module looks a little frightening at first, but with a bit of check work you can see it’s actually structured in a reasonable way. Firstly is the importation of the items database, so that the inventory and relative commands work, we then accept the players input, which is sent back to the play loop, and then if it is an accepted result, the script launches the command the user inputted. The accepted commands are the third part of this document, I have put each command separate to others, and commented them a lot to ensure that a reader can understand what each one does and how it does it. More explanation on the different commands and the way they work is further in the document.

## items.py

The items.py module is pretty simplistic in reading its’ structure, the hardest part is remembering to update the inventory list with any items that are added. This module is an items database used to keep items and their values as class objects, meaning they can be called and checked for their variables at any time. The structure for example objects is clear in page comments at the top of the module. After this is the list of items to add to the inventory list. It is important to remember that items.py is only ever called once, and that is at the start of the commands.py as a list of objects. The list inventory is what should be altered.

## mapmanager.py

This module is for creating the map. At the time this is being written, the map generator is barebones, but there are some development concepts below.



After the map has been generated, the map codes are written in page comments, this is so the user / developer can see how the map decides what is on each tile. These can be easily extended on the eventmap function. The checkmap function is used to read where the player is, in the pre-defined size of the map, then it checks this space, if it is a ‘#’, the play can’t move there and the function returns False.

The eventmap is the main function that takes