PyRYO Documentation

Release 1.0a

Ege Emir Ozkan

November 24, 2016

Contents

Contents:

CHAPTER 1

About Me

My name is Ege Emir Özkan, I am a Python developer that has worked on a few projects, this is my first module.

CHAPTER 2

Introduction

I find it almost unthinkable that there isn't a single easy to use module made specifaclly for Python. I wanted to fill this gap in the best way I can, the solution I found is: PyRYO. It isn't *a* module, it is a *collection* of modules:

- rpgModular (NPCs and player characters)
- Charter (for maps)
- Platonian (Questing module) (To do)
- Raven (Image and Musics) (To do)
- Nexus of Stories (A simple client that can be used for any game made with PyRYO) (WIP)

2.1 Many Worlds, One Life

PyRYO worlds are connected to each other, a character can be used in *any* number of worlds, this is done via Van Helsing System and Nexus of Stories client.

Ability to use the same character however has a caviat, a character can become overpowered with all thoose experiences. Which is why, if a character dies in **a** world, he/she dies in **all** the worlds. It is a strict perma-death system.

CHAPTER 3

Van Helsing System

PyRYO comes with a universal client for all PyRYO scripts, This is known as the *Van Helsing System* to use your games with VHS however you need to align with some *requirements*

3.1 File Stracture

- Characters.db
- client.pyw
- Character Inventories
  - Your Character.inventory
- Worlds
  - <Name of Your Game>
    - info.minfo
    - Data
      - <ChapterName>
      - 1-Arrival.memoir

3.1.1 Characters.db

Characters.db is an sqlite3 file that holds informations about characters and npcs, so that a player can use his/her character not only in your game but also in other people's games. Characters.db can be created with the setup script or mannualy however we will talk about it later.

3.1.2 client.pyw

client.pyw is the main client of the PyRYO, known as the universal client.

3.1.3 Character Inventories Folder

Character Inventories is the folder in which inventories of player's characters is held in a file (<character-name>.inventory) this means every inventory item can be carried across multiple gamewords.

3.1.4 Worlds Folder

Worlds folder hold the games developed by any number of developers using PyRYO framework.

3.1.5 info.minfo

info.minfo is the file that holds the description of your game, it is shown as a world detail in the client.

3.1.6 Data Folder

Data Folder is where you keep your chapters (aka episodes) within other folders and any number of items including your music, images and auido.

3.1.7 *.memoir files

*.memoir is a special type of text file that keeps the texts that will be presented to the player via PyRYO framework.

3.2 Note to independent developers

To all those beautiful people who decided to use my module to create an independent game, I love you all and if you won't use my client you really don't have to use this file structure either since you will ship your game with your own client. *However* It is my advise to you: It may be better if you use a similer structure as certain folders (inventory) and files (*.db) files are so intertwine with the framework that they are automatically generated and framework may not work properly without them.

CHAPTER 4

Charting Your World

Wheter you are creating a sci-fi, a fantasy or any type of game really, most fundemental point in design is a world for your events to happen. In PyRYO *Worlds* this is done via a special module: *charter*

You first need to 'import' charter: `import charter` from there you can do anything.

Let's start by creating a room

4.1 Creating a Room

Room creation is rather straightforward, a room is an instance of the clas `Room` and can be created as:

```
example_room = Room(room_id="room name", connections = [])
```

This creates a room with *no* connections, keep in mind that, `connectinos= None` is **not** a correct usage since `None` is not a list.

4.2 Adding or deleting a connection

In PyRYO, connections are rooms that are connected to another room. To add or remove a connection for instance:

```
#To connect example_room_2 to example_room:
example_room.edit_room(roomName="example_room_2", switch = "add")
#To remove this connection:
example_room.edit_room(roomName="example_room_2", switch = "del")
```

Keep in mind that roomName is the room_id of the desired room and *not* the variable name you've asigned to that room.

4.3 Get Connections

You can use `example_room.get_connections()` to get a list of rooms connected to `example_room`

4.4 Lock State

Each room comes with a variable named `lockstate` which defines if a room is locked (unaccessible) or unlocked (accessible), lockstate can be changed with:

```
# To unlock the door:
example_room.set_lock_state(state="unlocked")
#or
example_room.lockstate = "unlocked"
#To lock the door:
example_room.set_lock_state(state="locked")
#or
example_room.lockstate = "locked"
```

Keep in mind although both versions are equally right, using `set_lock_state(lockstate)` command with an invalid lockstate will send an error and won't change it, wherehas asigning a lock state directly will change the lockstate regardless of the state asigned, which can be used to asign custom lockstates to a Room without changing the source code.

CHAPTER 5

Characters and NPCs

5.1 Characters

Characters are controlled by `Character` and `CharacterExist` classes in PyRYO, former is used in initial creation wherehas `CharacterExist` should be used to 'load' an existing character. A character consists of its:

- Name
- Skills
  - Fighting
  - Scholarship
  - Stealth
- Health
- Class
  - Weaponsmaster
  - Explorer
  - Riflesmaster

5.1.1 Creating a character

To create a character in PyRYO, one can use:

```
character= rpgModular.Character(name="CharacterName", fighting=25, scholarship=25
                                stealth=25, health=500, weapons="Explorer")
```

If you don't provide the `weapons` variable, character will be created as a Weaponsmaster All classes start with a default weapon:

Class	Default Weapon
Weaponsmaster	Conscript's Saber
Riflesmaster	Conscript's Rifle
Explorer	Explorer's Machette

Then, a file known as <charname>.inventory will be created in Character Inventories folder to keep this character's inventory. Wherhas his health and stats will be written to a database

5.1.2 Loading a character

Loading an existing character is easy, it is done via `character = rpgModular.CharacterExist("charname")`, this lets you to interract with the character. This gives access to folowing methods:

character.showSkills() Returns the skills of the character

character.showStats() Returns health, level and dps of the character

character.showInventory() Returns character inventory

character.addInventory(item) Adds item to character inventory, `item` is `str`

character.takeDamage(damage) Substracts `damage` from current `health` value, `damage` must be `int`

character.die() Kills the character

5.2 NPCs

NPCs are non-player entities that is controlled by `NPC` class they have the following values:

- Name
- Stats
  - Health
  - DPSYield
- Player based values
  - LevelOfThePlayer
  - agressionFlag
  - flag

`LevelOfPlayer` and `DPSYield` go into an algorthim to create the DPS value of the NPC, `agressionFlag` takes the values "Enemy", "Neatural" and "Friendly" wherhas flag takes "Active" or "Inactive". NPCs are defined with:

```
npc = NPC("npcName", health=400, DPSYield=25, LevelOfPlayer=3, agressionFlag="Friendly", flag="Active
```

`NPC` class caries the methods:

npc.takeDamage(damage) Substracts `damage` from current `health` value, `damage` must be `int`

npc.die() Kills the NPC

CHAPTER 6

Indices and tables

- genindex
- modindex
- search