Big Typernatural Project Documentation

Release 0.1.0

EwanCoder

June 18, 2014

Contents

Big TyperNatural Project, BTP is a **game** written in **Python**, powered by **PyGame** and based on **keyboard typing** so this game doesn't even have mouse support. And this is not a trouble at all - it's a feature.

Note: If you just want to start playing, you can go to *Getting Started* page.

These docs are powered by **Sphinx** and intended to help both **users** to start playing and any **developers** (including me) to understand actual code and maintain it easily with comprehension of each and single line. Thus, these docs have both "user" side and "developer" side, which are actually pretty entwined together.

CHAPTER 1

Contents:

1.1 About

This page describes everything about the project: the **idea** itself, the **history** of making and the **abstract** things. If you wish some **technical** project info, you may be interested in *Development Lab* as well as in *User Guide* if you wish some **how to play and what's the game** info. While *User Guide* page is intended to help users get started and describes all about the game, this page however describes all about how this game has been created, what concepts and ideas it follows and other **general information** on the project.

Contents

1.1.1 Ideas and concepts

BTP is **typing text**-based **rpg** game, written in **python** using **pygame**. Really great game feature is that alongside with gaining experience, new levels, clothes and having a good time you will actually be mastering **real life skills** in fast keyboard typing because of battle system, walking system and all other which are **typo-like** (touchtyping guys will be able to handle some bosses at start).

This game is basically a **keyboard trainer**, but fancy keyboard trainer. You don't need anymore these stupid programs like "Keyboard SOLO" or any touchtyping teaching programs, now you can learn how to type touchtypingly through a **game** with fun and joy.

Typing

The game is made on 'typing' system - you need to type on your keyboard to get results.

- You need to type 'left' to go left, you need to type 'talk to Katie' to talk to Katie. All possible options shown onto the screen at the place you are currently located.
- Battle system is typing system. You need to type words faster that it do mobs so you can kill them. It sounds strange, but it's perfect in action.

Text-based

Text-based games are usually quest-games which shows description about current place you located at, upon background image describing current place. You read about it, your imagination blends text and image and you're choosing any option to do from an actions list. The next scene shown is based onto your actions.

RPG

Role-playing game is a game where player takes a 'role' into a game, playing as character with lots of skills, intentory, some items, hitpoints, experience, level and other stats. It's usually very addictive type of games where you can level up constantly and gain more profit.

Python and Pygame

This is the carcass on which this game is builded: **python** is a programming language, when **pygame** is a cool library for easily making games (it's a wrapper upon SDL by the way). You can read about it more at *Development Lab* page.

1.1.2 Game plot

Distant future. All kinds of creatures (daemons, elfs, angels) were actually existing all this time but in the other worlds, on other planes. As far as noone could reach another in eternity. Time passed, civilisations rised and faded. But then the big Disaster happend: all the worlds blended together by the hand of old powerful creature, so the great grief war begun between all kinds of creatures including humans and our world - the Earth. The Earth was destroyed, as well as all other worlds. It was "Common world" now. There was 200 years of bloody quarrels and battles for survival, until peace was made. You had been born 20 years after that, after the peace was made in new cruel world, but it's still dangerous out there. It is not the Earth nor one of the other planets now. All our technology, trains, cellphones had long gone as well as knowledge of that, other civilisations faded too. It's just ruines and villages and swords and shields now. And some magic, of course, for another worlds have had one. And the sharp fangs of deadly foes waiting in dark forests. And you, fighting for survival.

1.1.3 Way of Arts

I am very much all-rounded person, I like everything to do. I like bodybuilding, I like programming, I like cooking (and cookies), building electronics, recording my own music and lots of other stuff. So when the question came to music and paints, I've decided to make them all by myself (of course, you can help me if you want). Yes, it's hard, and long, and I don't paint well (at all), but I don't have many options here, and when I tried to draw something - (it came out pretty ugly) I liked the process itself. So these are my ways of arts in making the game:

- **Coding** - this is an art in some kind, really. I love making perfect and beautiful code
- **Music production** - I know how to do a song
- **Painting** - that's my lame side, I get it. But it's never late to learn something new! :)

1.1.4 Software

I prefer using free opensource software, so I needed to find something to fulfill my requirements. And I have found everything I need.

- **Coding**. For coding I use my **Arch Linux** plus **ViM editor** with **Jedi plugin** for python autocompletion. You can read more on software at the *Development Lab* section.

- **Music**. In the past, I've had some experience with Image-Line FL Studio, I downloaded really cool VSTs and made metal songs. But in real life they are all cost around 500 bucks, and now I'm an opensource man. So I've decided to use **Linux MultiMedia Studio (LMMS)** alongside with it's default instruments which is capable of making epic symphonic and orchestral music like I need.
- **Drawing**. As for painting, I've found awesome program called **Krita** (it's opensource and crossplatform) which some do call "Photoshop killer". It has really awesome capabilities of drawing graphics, and I liked it. Now I'm going to buy a graphics tablet and going to draw all the graphics needed for my game.

1.1.5 History

Finally, I'm going to tell you briefly what and how have inspired me to do such a thing - write text based rpg game in nowadays world, and what stages I have been through while developing it. I've been developing it since 2008. It was my pro-skill to create a game as a way of **learning** new programming language. It's funny, it's interesting. And it's significantly **useful**! Cause, you know, the best way to learn language is to **practice it**. I set my goal - I do that and I learn new language. So, I were developing games firstly in Basic, Pascal, Delphi, then in lots of other languages such as C, C++, Java, then I started creating web-browser games using php, mysql and javascript (html and css are stylesheet and markup languages, which I do not count as programming languages). So there was lots of languages, I don't even remember all list of them.

I were developing this games and learnt different things. From how to read files up to how to create inventory and skills and items... This was really great practice. Why **was**? Because now I am **actually** writing The Game. The real one. Because I'm in love with **python** and I don't think that I'll find anything more suitable for my current goal.

In the past it was Big Supernatural Project - big supernatural roleplaying text-based game based on CWTV Supernatural show. One version of the game was even capable of playing with two characters at once - Dean and Sam (two brother, main heroes of Supernatural TV Show).

But when I had thought to make it real (last time it was with C++ and QT) - I understood that there're lots of licences in real world, copyrights and other crap. So, even if my game will be opensource and free (which I am direct to), if it will become awesome and famous (which is not likely to happen) some companies might be offended, maybe even litigate with me and other crap. So that was when I decided to make a game based on Supernatural world, but without any characters from TV Show (actually I was intended to make some cross-reference with them). And I thought that it would be great to have the action going in other age - in 16th, for example.

At last, when I fell in love with python and decided to make a real game like real real game, I comprehended that this was not a good idea to make a supernatural-based game. I need to write my own story - that's the way of art. I'm currently watching, reading and enjoying (in all prospects) the great saga of George R.R. Martin "A Song of Ice and Fire" (widely known for "Game of Thrones" caption - first book's name and TV show's name). He have inspired me that great storyline must consist thousands of characters entwined each with other (and dying in packs). And I am intended to write such a story, but within the game. Well, who knows, maybe I'll write a book based upon this plot. Always wanted to write my own book.

The last thing I had needed to do is to find **epic music** themes for my game, free **sounds** for game effects and fantasy **images** for my storyline. At first I had managed to find some images and music, but it was too hard to find and alongside I had realised that they are all non-free like Supernatural TV Show, for all in our world has its patents and licences :) And I've solved my problem by making my own music and images (at least, I'm going to try).

Note: Anyone capable of drawing and interested in, please contact me for I need help :) Of course this will benefit you only with your name in the titles, for I don't have much money. Any musicians and programmers are welcomed as well.

1.2 User Guide

Contents

1.2.1 Getting Started

Welcome, brave sir User. I see, you wanna play. Here are quick and simple instructions to start playing:

How to get it

First, you need to get the game and packages it relies upon. You need 3 things:

1. Python package. Download whether 2 or 3 version, it's not crucial, yet I am using 3rt and most newest, bleeding edge version.
2. PyGame package. Download lastest available version for your OS.
3. The game itself:

- Stable version
- Development version

Note: If you are on *nix you can install both **python** and **pygame** through your package manager:

```
apt-get     install  python pygame
aptitude    install  python pygame
pacman      -S       python pygame
```

Just make sure you're installing version of pygame compatible with the version of python.

How to run it

All you need it to execute `game.py` game file in the root game folder. You can do it via **python**. Run python in the directory with the game and print `game.py`. Then the game will start. Also you can run it just as executable:

```
#Add eXecutable flag to game.py file if it's not executable yet (should be, though)
chmod +x game.py

#Run the game
./game.py
```

1.3 Development Lab

Contents

Todo

- Write on this page (dev.rst) something about pygame (I can read more about pygame here)
- I can read here more on software that I use for coding - arch linux + vim + jedi plugin (I've not described all the plugins and other stuff) [from about.rst, line 58]

1.3.1 Branching

This is main github repo of the game. You can clone it for your sake. As you can see, it has multiple branches.

- master - main branch for **stable releases**. There will go versions like *0.1*, *0.2*, *1.0* - official stable releases.
- dev - **development** branch. All development goes here, but separate big features being developed on separate branches created just for these features. Here goes versions *0.0.7*, *0.0.7.4*, *0.1.1* and etc.
- docs - separate branch for **docs**. It is made for good and clean commits history and splitting documentation & code development, but because this documentation uses `sphinx autodoc extension`, it's essential to merge **dev** branch to **docs** branch each time before creating docs for new features.

When the big feature being developed, it's useful to create separate branch just for it. And if it's big enough and consists from separate parts or features, new separate branches could be created from this branch. For example - req branch was created for solving issue with virtualenv not building my req.txt, pygame and other stuff. Now it's perfectly working with readthedocs.

1.3.2 Modules

BTP main module - game.py

Contents

About

This is main game executable file which contains main logics of the game and manipulates other modules and classes (basically, `screens` module). At first, we set `game.CAPTION` and `game.SIZE` constants, which is responsible for main window title and size respectively.

```
CAPTION = 'Big Typernatural Project'
SIZE = (1000, 700)  #: Should be at least the size of images in Images folder
```

Note: Size of the window must be smaller or at least the same as any image used as background, otherwise the image won't cover all screen and there will be black areas.

Then we create new pygame window using those constants.

```
if __name__ == '__main__':
    pg.display.set_caption(CAPTION)
    screen = pg.display.set_mode(SIZE)
    Game().loop(screen)
    pg.quit()
```

Last two lines are basically creation of main game class `game.Game` and a proper pygame quit. The `game.Game` class is looping through whole lots of game logics **while True**, so it's okay that we're quiting our game right after `game.Game` creation.

Note: The `__name__ == '__main__'` thing is basically checking whether we're running our game standalone way (form console, as executable, etc.) or we're importing it. Even though I'm not intended to import it and you're should not too, it's proper way to protect your project from undesired execution.

Imports

`screens`	For wrapping screens around (see *Interfaces wrapper - screens.py*)
`os`	Determine whether file exists or not
`pickle`	For saving/loading Pers() object (savegame)
`random`	For executing battle only if the chance is right
`pygame`	As a core of the whole game it is imported in each module

Classes

Game This is the main logics class which rules the game **while true** through `screens` module.

class `game.`**`Game`**

`loop`(*surface*)
Main function loop of the game

We have only one method here and nothing else - `game.Game.loop()`. So, basically, we don't need a class - we can easily make it a function or even write all the code within `__name__ == '__main__'` condition, but I like OOP concepts and I like extensibility of classes and object. So for now it will stand a class, but when the stable *1.0* version is out - we'll see.

Todo

Make something about attr -> param linking support

Our `game.Game.loop()` method have only one attribute - `game.Game.loop.surface` which is the screen area for drawing all the stuff.

At first, we set some variables:

```
pers = Pers()
started = False # Prevent multiple game loads + game quit to menu trigger
```

- `game.Game.loop.pers` - our `game.Pers` object to work with. All the data about your character is stored within it. Your hitpoints, your current location and etc.
- `game.Game.loop.started` - boolean which tells us whether game is already loaded or is it should yet be loaded/created.

Then goes main cycle which loops **while True**.

```
while True:
    # Create menu
    while pers.name == '':
        login = screens.Menu().main(surface)
        # Create loginscreen
        if login:
            pers.name = screens.Login().main(surface)

    # Load game of save new game, then started=True
    if not started:
        if os.path.isfile('Saves/' + pers.name):
            pers.load()
        else:
            pers.save()
            screens.Introduction().main(surface, pers.name)
        started = True
    (pers.place, mobs) = screens.World().main(surface, pers.place)
    if random.randrange(0, 100) < mobs['Chance']:
        pers = screens.Battle().main(surface, mobs, pers, Mob())
    # Save each loop
    pers.save()
```

Here we create menu using class `screens.Menu` and it's method `screens.Menu.main()`. When 'New game / load game' option is selected, menu returns something (not important what) into the local `login` variable which, if not empty, will load `screens.Login.main()` method from `screens.Login`.

Well, the login screen will return a name when it is entered to login screen and Enter is pressed. The name will be stored in `game.Pers.name` variable and then whole cycle will stop working. Because `game.Pers.name` won't be empty anymore.

The next step is to save game and start intro or to load game if user entered the name which already exists in Saves folder. Here we're using `os` module to check whether savegame exists or not.

If it exists, we use `game.Pers.load()` method which loads whole object data by means of `pickle`. If it isn't, firstly we're saving whole object data to the file and then we're executing `screens.World.main()` method form `screens.World` class, which is intended to tell user some pre-history of game world and give hime base knowledge of what he'll be waiting within the game.

After that, the variable `game.Game.loop.started` sets to True and this part of code never repeats too.

Next step is finally the world loading. Class `screens.World` is showing world until any action was performed. When an action was performed, `screens.World` returns next place to `game.Pers.place` and mobs record for new place to local variable `mobs`.

There's our `random` module comes in handy: we're using it to calculate chances that we're in trouble and monsters are attacking.

Then we're saving a game (after successful battle) and all begins again

Note: There will be lots more logics added to calculate, it's early development stage of the game. In the future, you'll be able to 'cleanse' locations so that mobs you've killed won't attack you again or even will respawn after some time. Now, however, if you'll kill any monsters (hah, battle is not yet ready) and you'll try to load the game - they can attack you again.

Mob There's not much done yet about this class. It's here just for an example of what will be done soon.

class game.**Mob**(*maxhp*)

Creates new monster who is to fight you

maxhp = 20

Standard mob hitpoints

Source code:

```
maxhp = 20  #: Standard mob hitpoints
hp = maxhp

def __init__(self, maxhp):
    self.maxhp = maxhp
```

Pers Handles all Player data.

class game.**Pers**

load()

Load self Pers() object from file

name = ‘’

Player name stored internally for easy self save/load

place = ‘OldManHouse’

Standard first place of a game (changeable through the game)

save()

Saves self Pers() object to file

Source code:

```
name = ''                 #: Player name stored internally for easy self save/load
place = 'OldManHouse'     #: Standard first place of a game (changeable through the game)
maxhp = 20
hp = maxhp

def save(self):
    """Saves self Pers() object to file"""
    with open('Saves/' + self.name, 'wb') as f:
        pickle.dump(self, f)
        print('Game saved as ' + self.name)

def load(self):
    """Load self Pers() object from file"""
    with open('Saves/' + self.name, 'rb') as f:
        self = pickle.load(f)
        print('Game loaded as ' + self.name)
```

Quest This class will contain lots of tasks, jounral, other things to handle all the quests system and other logic. Yet it is far not done.

class game.**Quest**

Source code:

```
print('Class for quests and tasks and notes')
```

Interfaces wrapper - screens.py

Contents

About

1.4 Features and TODO

Contents

Todo

- Write on this page (dev.rst) something about pygame (I can read more about pygame here)
- I can read here more on software that I use for coding - arch linux + vim + jedi plugin (I've not described all the plugins and other stuff) [from about.rst, line 58]

(The *original entry* is located in /var/build/user_builds/btp/checkouts/latest/Documentation/dev.rst, line 7.)

Todo

Make something about attr -> param linking support

(The *original entry* is located in /var/build/user_builds/btp/checkouts/latest/Documentation/game.rst, line 61.)

CHAPTER 2

Your way around

If you are just user who wanna play, use *Getting Started* guide. If you want to read about the game something more, you can read *About* section as well as whole *User Guide*.

For developers, however, there's *Development Lab* page. Also you can check out *Features and TODO* page to see what have been done so far and what is intended to be done.

Python Module Index