Tech Basics 2
- Assessment

By: Carlotta von der Linde

Matrikel-Nr.: 3033292

Due Date: 22.03.2019

Instructor: Helena Lingor

Module: Tech Basics 2

Semester: Wintersemester 2018/2019

Table of contents

Introduction of the idea

In the following, I will describe and discuss my project for the tech basics 2 class , which is a short text-based adventure game with a simple graphical user interface. The included code originates out of my programming.

When I started working on this game in the last semester, my idea was to create a text-based adventure game with similarities to ‚Fortnite - Battle Royale'. In this online game, up to 100 players get thrown into an area (which gets smaller, as the game processes) to fight against each other until only one is left. Even though the main aspect in ‚Fortnite' is to fight the other players, there is still some game of strategy involved, as each player has an inventory and is able to collect and use items smartly. The vision I have for the adventure game I am creating is, to be a game with a (more) complex story behind it and a different distribution of the proportions of fighting and logical thinking. The game is ought to be an adventure game, with an overarching horror story: The User affiliates a group of people, who want to save their families from humans, infected with a mysterious virus, that turns them into something similar to zombies. At the time where the story takes place, those humans are living locked up in a ‚haunted' town, but the wall protecting the surroundings begins to get dilapidated. The player then can decide, whether he wants to join the people on their mission or not. If he joins, he clicks himself with varying buttons through the story and at some points his decisions can obtain different outcomes.

Progress since the prototype

Last semester, I thought that the story would be the first thing that I would extend, if I would continue working on this project. I found the story ended way too abruptly and wanted to implement some kind of map, to make navigating through the game more fun and clear. Additionally, I wanted to build another data structure for the game state (an inventory, with objects for fighting, or solving puzzles) to broaden the story and evoke more suspense. At different places in the map, I imagined different objects to discover, characters to meet and ask for help and especially more options for the user to encroach.

When I started this semester to engage with this game again, especially with the implementation of a GUI, I came to the conclusion that I would not be able to realise all of my ideas and visions by the time I would have to be finished.

As our task was to implement a graphical user interface, I put my main focus on the presentation of the game and did not change the story content of the game very much. Contrary to what I had expected at the beginning, I had to change the complete structure of the game except for the part where the text is printed. Since I had planned last time to structure my next game better and in tree structure, I tried to realise that in this game. Moreover, to improve the style and to make it easier for others to read, I tried to write the code as advised in the ‚Style Guide for Python Code' and ‚Einführung in Python 3'.

The game now consists of three different files - ‚story.json' , ‚story.py' and ‚main.py':

2.1 Story.json

In the file ‚story.json' the content and process flow of the story is saved. It contains the whole story the player is told and experiences. This file simply consists of the individual ‚story points' that make up the story/experience. A ‚story point' can be a scene, an event or anything similar. For practicability, every ‚story point' has a unique name, with which it can be called. Each ‚story point' consists of different components:

```
{
  "beginning": {
    "message": "Hello! What's your name, adventurer?",
    "expect": {
      "type": "input",
      "options": ["Confirm"],
      "var_name": "username"
    },
    "response": {
      "text": [
         "Hi $username ...!"
      ],
      "options": [". . ."]
    },
    "next": ["greeting"]
  },
```

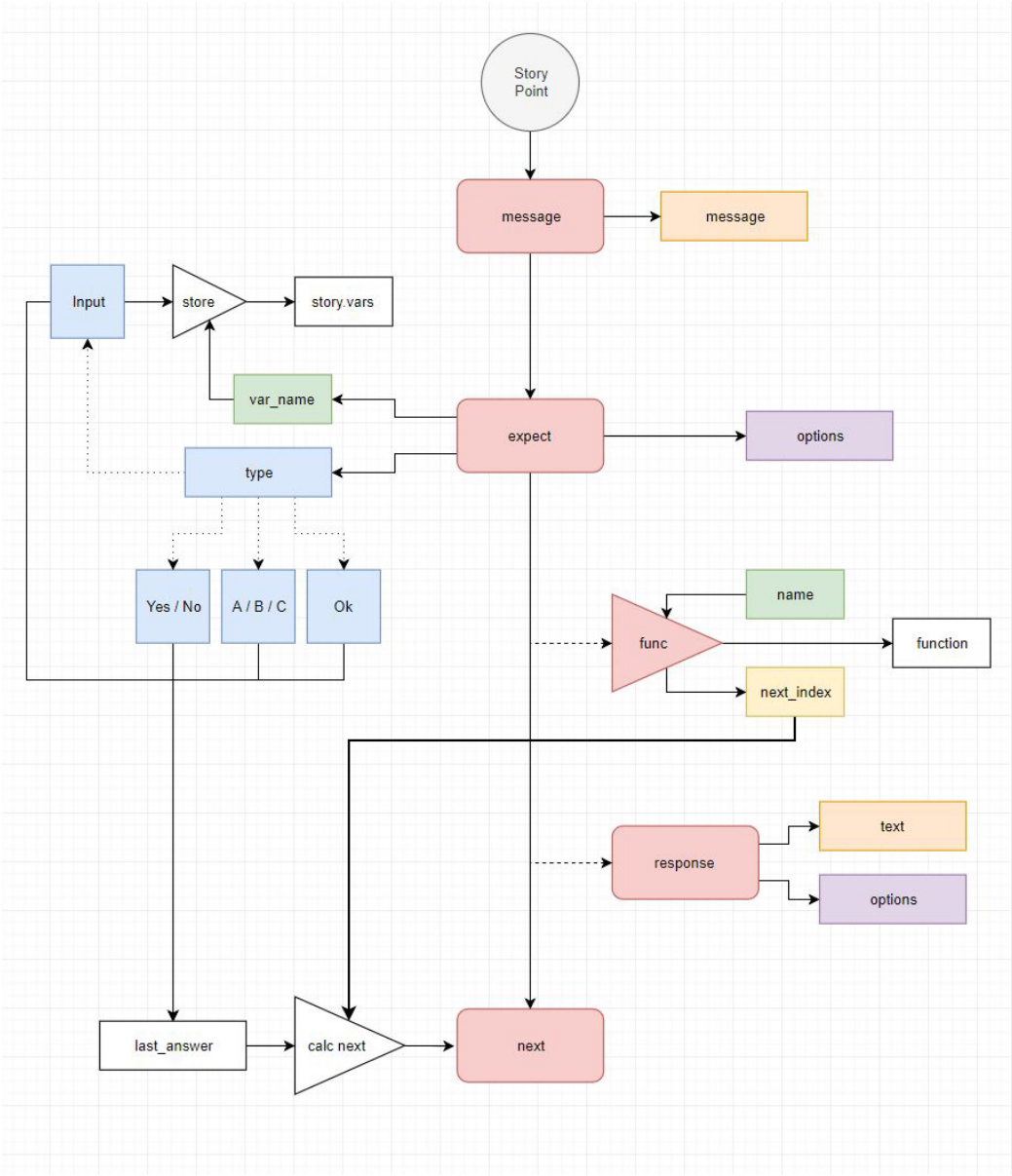
Example of a ‚story point'

The following table lists the functionalities of the different components:

Name	Function
„story_point_name“	Unique name of the story point.
„message“	Contains what will be written in the message box of the application. This will be shown to the User when reaching this story point. Can be a question/statement/...
„expect“ **„type“** **„options“** **„var_name“**	To store the expected answer information. ———————————————————— Type of answer can be „ok" / „yesno“ / „abc" / „input“. ————————————————————- Available options the user has, they will be shown inside of the buttons. ————————————————————- „var_name“ is optional. For "type"=input only: Name of the variable to store the input in.
[optional] **„func“** **„name“** **„next_index“**	The function needs to be defined in the calling class of Story(). ———————————————————— Set name of the function to call. ———————————————————— Index is used to determine which response and next story point will be used according to the function outcome. This value needs to be set by the method set_next_index().
[optional] **„response“** **„text“** **„options“**	Available responses to the given answer. ———————————————————— Response text depending on the given answer. ————————————————————- Available button content, each for one response text.
„next“	following story points according to the given answer.

The JavaScript Object Notation, short JSON, is a compact data format in a textual form that is easy to read. It is used for the datex and retention of structured data between two applications.

The following flowchart outlines the functionality of the story.json file:



2.2 Story.py

The file story.py is the description of the class „story“. This class is necessary to retrieve the current state of the story. Moreover the „running-through“ trough the story can be controlled and navigated here. This can be achieved with the Getters ans Setters concept. I used those to access or request certain variables of the story class. Each function is somewhat self-descriptive and will return or take the data in the right format.

Furthermore there are some 'globals' described in the __init__ function of the story class. They will be accessed by the getter and setter functions.
Finally there is a function to replace the placeholders '$variable' inside of the story content with the contents of the corresponding variables. These variables are stored in an array inside of the story class.

2.3 Main.py

In this file the main() method is constructed, as well as the application to initialise and describe the user interface. Moreover, the class ‚story' is created and used.

What was important for me creating the application, was that the window can be pulled bigger or smaller by the user in hindsight to a comfortable usage. This is given by the __init__ function right in the beginning of the file. With defining self.story = Story(story.json), we can have recourse to the story.json file further on.

In addition, with set_debug_mode the program can show information about the state of the story for better debugging. While programming the application, I became extremely confused and lost the overview over all the different paths and their outcomes, so the messages in the console were one of the most helpful things I integrated. With the method ‚create_widgets' for instance, I created two frames (frame 1 = text frame, frame 2 = button frame, text input frame) and the buttons that should occur in frame 2 (How wide are they? / Wich function will they call?). In main.py the layouts of the widgets are defined as well. I decided to hide all the created buttons and then show the ones that are necessary for each question. In ‚show_message' it is checked which buttons need to be shown at wich time - according to what question is asked - and when the input window should appear. One of the most important methods is pressed_btn, because it determines what happens after a button is pressed. With an if-clause the program checks checks the last answer that was given and calculates which respond should be printed out next. Otherwise it checks for the user input, the function call and if the response object exists. Afterwards, the functions necessary for the story happening are defined.

3. Process in the development:

1. At first, I created the layout for the user interface (main.py)
2. Then, I created the first structure for story.json, which I adjusted and extended many times
3. Afterwards I created the class story in story.py
   A. The reading of the content of the story from story.json
   B. First getters and setters
4. I connected the functions of the class story with the user interface
5. Many iterations of the steps 1 to 4 while implementing additional functions or after the occurrence of errors

3.1 Inventory:

Since I wanted to implement the feature of an inventory but was lacking time, i will outline a way to realise that.

1. Create a function to be called when some object can be found.
2. The function will simply create a new variable called something like "found_sword" and will set the value to true
3. With an other function the value of the variable can be queried to influence the story.

4. Challenges

When I started working with the project again, I needed some time to understand my own coding again. This is why I implemented many comments in the code I wrote, so if I would wanted to add something after sometime, It would not be so hard to understand. Moreover this turned out to be very useful in main.py, as the functions with different outcomes lead to different ‚story points‘.

I noticed very quickly that I wanted to do too much in a too short time. I would have loved to include background pictures and sounds, to make everything a bit more of a fun experience, but was not able to implement it.

Furthermore, it challenged me to keep the layout the way I wanted it, when the window could be pulled as big/small as wanted.

The biggest challenge I encountered but was able to cope with, was that every condition would be fulfilled for a correct process flow of the story. This was very hard because of the many different possibilities of a story point.

Sources:

- Rossum, Guido van; Style Guide for Python Code: https://www.python.org/dev/peps/pep-0008/#string-quotes [last access: 12.03.2019].

- Klein, Bernd; Einführung in Python 3: Für Ein- und Umsteiger; Nov.2017.