Assignment 5: Infinite Story

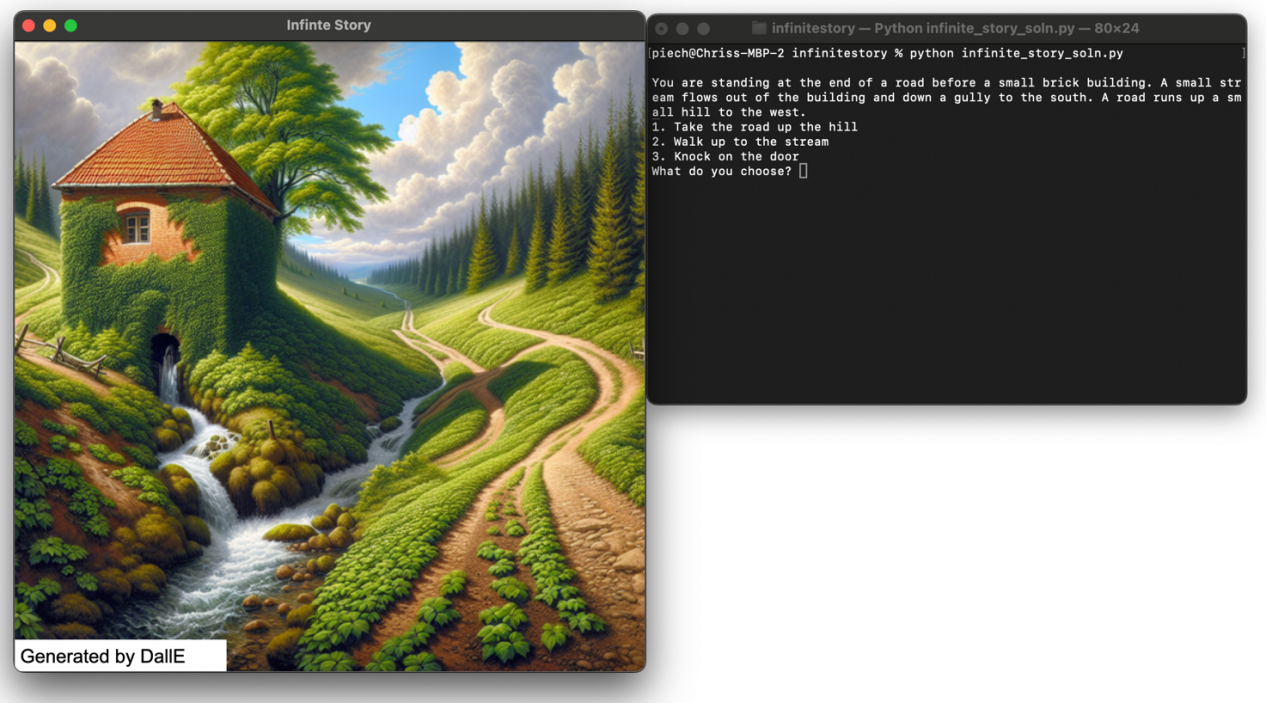
Assignment designed by Chris Piech, inspired by Eric Roberts. Handout written with Anjali Sreenivas, Yasmine Alonso, Katie Liu. Ethics by Javokhir Arifov and Dan Webber. Test scripts by Iddah Mlauzi and Tina Zheng. Advised by Mehran Sahami and Ngoc Nguyen, and more!

Overview

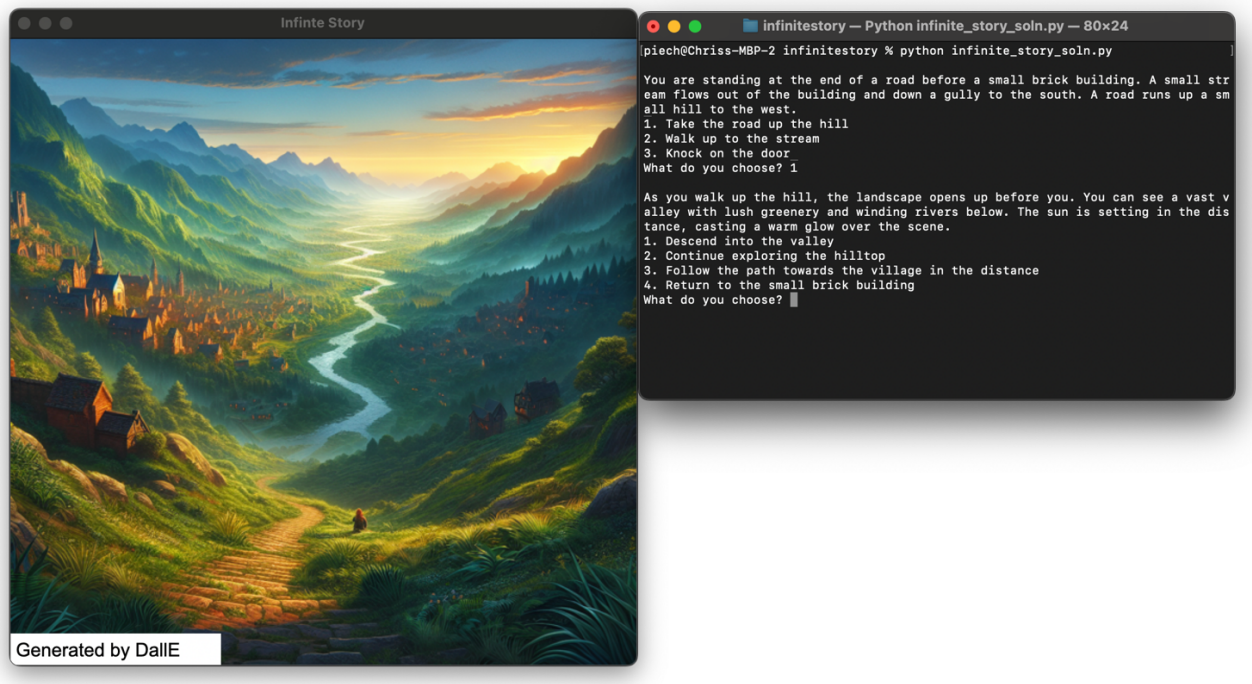
You are exploring a world. Almost every scene is normal, but if the hero explores enough, they will slowly start to uncover the mysterious parts…

In this assignment, you will write a choose-your-own-adventure adventure game that harnesses the power of generative AI. You will get to apply the knowledge you have learned about dictionaries and combine that with making requests to ChatGPT to ultimately help guide your user through a mystical adventure. In the end, you will think about the ethical implications underlying the use of generative AI in storytelling applications.

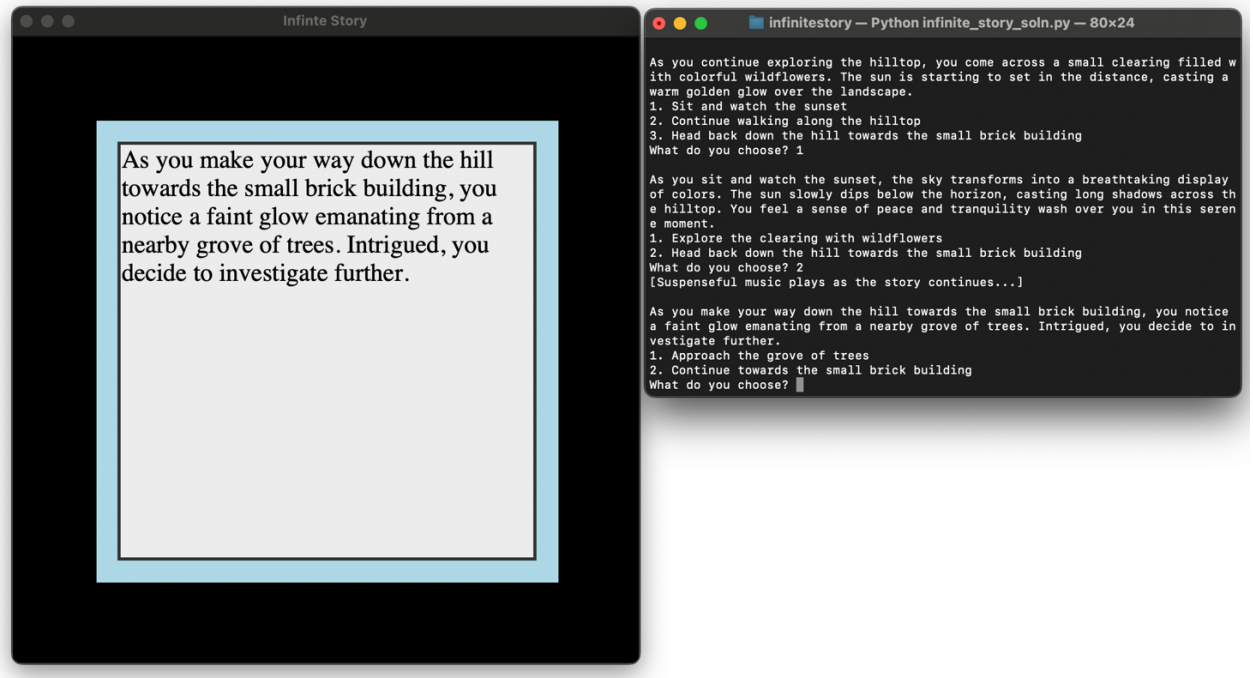
When the user starts the program it will look like this:



The user can then choose what action to take next. Here they chose 1, to take the road up the hill:



Up to this point, it seems like a standard storytelling app. The magic happens when the user gets to a scene that hasn't been written yet. Instead of crashing, or preventing the user from continuing, the app will make a request to ChatGPT to generate the next scene. The story continues!

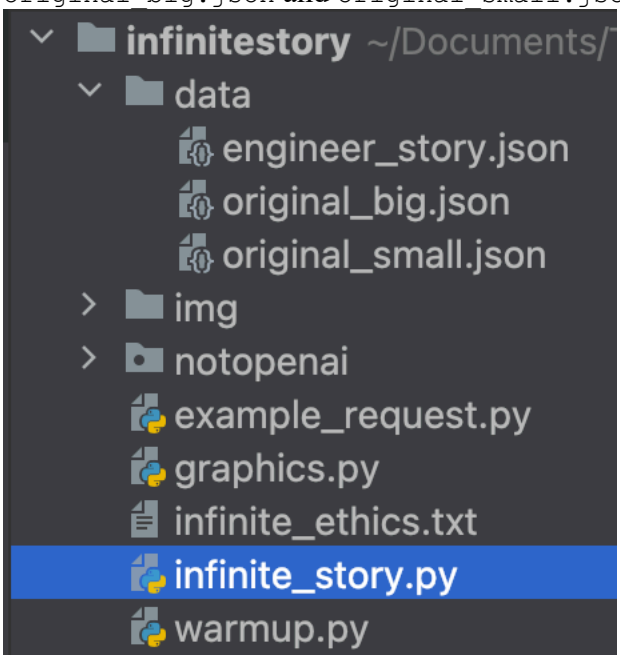


One of the goals of this assignment is that you get practice decomposing your program. In the milestones we will give you suggestions of helpful functions. However the control flow decisions are ultimately in your good hands.

Milestone 1: Loading the Story

A key limitation for any story of this nature is that you will eventually hit a dead end. Program `warmup.py` to find the "dead ends" in a story. Dead ends are scene keys that are referenced in the story, but are not defined.

In order to complete this warmup we need to first understand how story data is stored. The data for a story is saved as a nested dictionary. We've created two story dictionaries for you, `original_big.json` and `original_small.json`. These files are stored in a folder called `data`:



First, load in a story dictionary. Since each story is in json format you can load it with the following line of code which should be at the start of your `main` function:

```
story_data = json.load(open('data/original_small.json'))
```

You will routinely access this dictionary throughout your program. You should only load the file once.

Before proceeding, take some time to understand how the data in story_data is organized. Here is an excerpt of the story_data dictionary:

```
{
  "plot": "You are exploring a world. Almost every scene is normal, but if the hero explores enough of the normal parts (eg more than 10 scenes), they will slowly start to uncover the mysterious parts. Most of the tone is simply setting a beautiful and uplifting landscape filled with wonder.",
```

```
  "scenes": {
    "start": {
      "text": "You are standing at the end of a road before a small brick building. A small
stream flows out of the building and down a gully to the south. A road runs up a small hill to
the west.",
      "scene_summary": "Start of the story. Standing in front of a brick building.",
      "choices": [
        {
          "text": "Take the road up the hill",
          "scene_key": "overlooking_valley"
        },
        {
          "text": "Walk up to the stream",
          "scene_key": "next_to_gully"
        },
        {
          "text": "Knock on the door",
          "scene_key": "knocking_on_small_brick_building"
        }
      ]
    },
    "knocking_on_small_brick_building": {
      "text": "You knock on the door, but no one answers. You hear a faint sound of a dog
barking from inside.",
      "scene_summary": "Nobody answers.",
      "choices": [
        {
          "text": "Go back to the start",
          "scene_key": "start"
        }
      ]
    },
    ... # more scenes, hidden for space
}
```

Wow! A truly nested dictionary. Don't be intimidated by it. Instead, understand it step by step.

Story: Every story is a dictionary with two keys: a key "plot" that is associated with a string description of the overall story line, and a key "scenes" that is associated with a dictionary containing all the data of the scenes in the story (see Scenes below).

Scenes: A dictionary with all the scene data. In the scenes dictionary *scene keys* are associated with all the data for that given scene. Each scene has a "text" description, a "scene_summary" and a list of choices that the user can take from the scene (see Choices below). Here is scene_data associated with the scene key "knocking_on_small_brick_building". It only has one choice:

```
{
  "text": "You knock on the door, but no one answers. You hear a faint sound of a dog barking from inside.",
  "scene_summary": "Nobody answers.",
  "choices": [
    {
      "text": "Go back to the start",
      "scene_key": "start"
    }
  ]
}
```

Choice: A choice has both the "text" of the choice and the "scene_key" that the hero will go to if they take this choice.

Now that you have the story_data loaded, it is time to program `warmup.py` to find "dead ends":

- Loop over all the scenes in the story.
- For each scene, loop over all the choices for that scene.
- Each choice has a value associated with the key "scene_key".
- If the scene_key is not a key in story_data["scenes"], then it is a dead end. Print out scene_key.

For example if you ran `warmup.py` with the story constant set to "original_small.json" file it should print out the following keys:
next_to_gully
descend_into_valley
watching_sunset
continue_exploring_hilltop
return_to_small_brick_building

We print out `descent_into_valley` because it is referenced by a choice, but it is not in the scenes dictionary (it is a dead end). We do not print out `knocking_on_small_brick_building` as it is a key in the scenes dictionary. Recall that for this milestone you should write your code in `warmup.py`. Test your code on both original_small and original_big. The order of dead ends does not matter.

You can also try these **optional** challenges. Can you print out these values using story_data:

- The "plot" of the story?
- The "text" description of the scene with key "start"?
- The "scene_key" you would go to if you took the first choice from the "start" scene (the choice that has text "Take the road up the hill")?

Milestone 2: Printing a Scene to the Console

Installing packages:
For the rest of the assignment you will be programming in **infinitestory.py**. Before you can run your code, you will need to install the `requests` package. You can do this by running
python3 -m pip install requests
in your terminal. You may need to replace `python3` with `python` or `py` depending on your system.

Write a function to print a single scene to the console. It should only take one parameter, the dictionary associated with that scene. Write your code in `infinite_story.py`. In this new program you should load the story_data json in the same way as in Milestone 1. You can test your code on either the big or small story data. We recommend that at this point you start working with `original_big`.

Your function should print out the "text" for the scene as well as the "text" for each choice associated with the "choices" key in the scene data. Choices should be numbered for the user, starting at 1. For example, if you called the function with the data for the "start" scene, it should print out the following:

You are standing at the end of a road before a small brick building. A small stream flows out of the building and down a gully to the south. A road runs up a small hill to the west.
1. Take the road up the hill
2. Walk up to the stream
3. Knock on the door

*Aside: We are eventually going to render a scene by both printing out a description of the scene to the console **and** render an image associated with the scene. Rendering the image is in a future Milestone.*

Recall that a scene dictionary looks like this (this one is for the start scene):

```
{
  "text": "You are standing at the end of a road before a small brick building. A small stream flows out of the building and down a gully to the south. A road runs up a small hill to the west.",
  "scene_summary": "Start of the story. Standing in front of a brick building.",
  "choices": [
    {
      "text": "Take the road up the hill",
      "scene_key": "overlooking_valley"
```

```
    },
    {
      "text": "Walk up to the stream",
      "scene_key": "next_to_gully"
    },
    {
      "text": "Knock on the door",
      "scene_key": "knocking_on_small_brick_building"
    }
  ]
}
```

Observe that every scene has a description ("text") as well as a list of next scene options that the user can choose from ("choices"). Notice that choices is a list and that each element in the list is a dictionary of its own. The "text" key in each inner dictionary is the text description for the choice.

Your function should work for any scene, not just the start. Note, that you do not need to use the "scene_summary" key, that will come in handy later.

Milestone 3: Get a Valid Choice

Write a function to get a valid choice from the user. This function should also only take in one parameter, the data dictionary of a scene. It should return the choice the user made.

After you print a scene to the console, including the choices, you will need to get a valid choice from the user. In this milestone you will write the function that gets that information from the user.

Ask the user what they choose, with input prompt `"What do you choose? "`. Their response needs to be one of the integers that corresponds to a printed choice for the scene. Until the choice is valid, print out `"Please enter a valid choice: "` and then re-prompt the user to enter a valid choice. When you get a valid choice, return the choice made.

Here is an example, where we call the get valid choice function for the "start" scene. Note that the scene has already been printed using the function from the previous milestone. User input is in blue italics. The user enters two invalid choices, before they eventually enter a valid choice, `1`, which corresponds to the text "Take the road up the hill"

You are standing at the end of a road before a small brick building. A small stream flows out of the building and down a gully to the south. A road runs up a small hill to the west.
1. Take the road up the hill
2. Walk up to the stream
3. Knock on the door

What do you choose? *cat*
Please enter a valid choice: *5*
Please enter a valid choice: *1*

Pro Tip: Python lists are 0-indexed, but our choices are indexed starting at 1! How might you take care of this?

Milestone 4: Scene Flow

Now that you have functionality to print out a scene, and get a valid choice for a scene, it is time to stitch our story app together! In main(), your job is to infinitely run scenes by transitioning from one scene to the next based on what the user inputs.

Pro Tip: You should have a variable which keeps track of the current scene key. The first scene always has the scene key "start".

In an infinite loop you should:

1. get the scene data for the current scene
2. print the current scene
3. get a valid choice from the user
4. update the current scene key

Suggested: To make it easier for the user to read, print out an extra (blank) line to the console before you print out a scene.

Woohoo! Look at your infinite story coming together! If you play for long enough, you will eventually reach a scene_key which is not in your story. Your program will likely crash with an error that says `KeyError` and complains that a scene key is not in your scenes dictionary. So what do we do if we don't have a scene description?! We will resolve this issue in the next milestone! 🙂

Milestone 5: Handling Infinity

When we get to a scene key where we don't have scene data (i.e. the key does not exist in our story_data dictionary, a dead end), we are not going to crash. Instead, we are going to call upon ChatGPT for help in generating a new scene, including appropriate choices! Then, we will continue our program with the newly created scene. For this milestone, define a function that creates a new scene when the scene key doesn't exist in your story_data dictionary. Specifically you should:

1. Print "[Suspenseful music plays as the story continues...]"
2. Construct a prompt to ask Chat GPT to generate the next scene
3. Add the scene into your story_data, so that if the story brings the hero back to the same scene key, you display the same information.

The prompt should be formatted like this:
`"Return the next scene of a story for key [scene key]. An example scene should be formatted in json like this: [example scene]. The main plot line of the story is [plot]."`
scene_key is the key of the scene to be generated. The plot should be the value associated with "plot" in story_data. The example scene should be the start scene's data turned into a string. You can cast a dictionary to a string, just like you can cast an integer to a string, using a command like `str(start_scene_data)`. It is important to give ChatGPT an example json so that it knows what format you are expecting scenes to be in.

So how do we communicate with ChatGPT to generate our next scene when we do not already have the data in our dictionary? To do this, we will first need to understand a new key concept: APIs.

APIs (Application Programming Interfaces) allow software applications to communicate and interact with each other. They define the methods and data formats that applications can use to request and exchange information, allowing computer scientists to integrate various services, such as ChatGPT, seamlessly into their own projects without a lot of added complexity. Think of them as functions on a remote computer that you can call over the internet. They are a super powerful tool!

See `example_request.py` for a full example of making an API "call" to OpenAI.

```
chat_completion = CLIENT.chat.completions.create(
  messages=[{
    "role": "user",
    "content": """What is the capital of all countries in east africa?
    Reply in json where keys are countries""",
  }],
  model="gpt-3.5-turbo", # the GPT model to use
  response_format={"type": "json_object"} # we want our response in json format,
)
```

This is an example of an API "call", where we are sending a prompt to OpenAI's chat API (client) to generate a response based on the prompt "What is the capital of all countries in east africa? Reply in json where keys are countries". It also specifies that the input message is from the user ("role": "user"), indicates the GPT model to be used ("gpt-3.5-turbo"), and specifies that the response should be formatted as json (response_format={"type": "json_object"}).

In order for the call to work we need to first initialise the API client. Notice the line in your program:

`CLIENT = NotOpenAI(api_key="yourapikey")`
You are going to need to replace "yourapikey" with your own API Key. To get your API key see the NotOpenAI handout.

The `chat_completion` that you get back from the API call is a rather complicated object. To get just the json response you can use the following two lines:

```
# get the content of the response
response_str = chat_completion.choices[0].message.content
# turn the string into a dictionary using loads
new_scene_data = json.loads(response_str)
```

If ChatGPT has done its job, `new_scene_data` will now be a new dictionary for a scene, complete with text, summary and choices. You can now add this new scene to your story_data dictionary.

Aside: Why is the API called NotOpenAI? Its just a joke. OpenAI is the company that created GPT-3.5. To keep things free for you we are routing all of your requests through our paid account. We call it NotOpenAI because we are not OpenAI 🙂*. The API is identical to the OpenAI API, and if you want to switch over to your own paid OpenAI key, you can! See NotOpenAI handout for details.*

Milestone 6: Visualize Scene Images

Perhaps you arere travelling in a lush valley along a winding river, passing by quaint and charming brick cottages. Or maybe, once dusk has come upon us, you continue following this glowing, luminous stream to encounter a mysterious building with light pouring out of it. These visuals bring magic to the entire assignment – it is up to you to display them now to enhance the user's experience! For all of the scenes in original_big and original_small, we have asked DallE to generate a corresponding image. They are all saved in the img folder.

Important: we may not have images for every possible scene that needs to be displayed, especially when you get into newly created scenes. If we don't have an image for the current scene, simply just display a black rectangle over the canvas (or feel free to spice it up and display something else of your choice as an extension!). You do **not** need to be generating images yourself!

Note: you might need to run `py3 infinite_story.py` (Windows) or `python3 infinite_story.py` (Mac) in your PyCharm terminal in order for the images to appear.

It is important to understand where these images reside, and how we can access them. Notice that in your infinite story assignment folder, there is a sub-folder called img. Within this img folder, we will have all of the images (stored as .jpg images) that can be used to craft our story! Each image is named "[scene_key].jpg" where scene_key is the key in the "scenes" of the original_big story_data dictionary. For example "img/start.jpg" has this
image:

Which corresponds with the start scene in the original_big story_data dictionary.

A path to an image is a string that tells you how you can get from a starting directory (in this case, the current folder for your infinite story assignment) to your end destination (the desired image). We separate folders (directories) with slashes (`'/'`) to indicate what directories must be “clicked through” to get to our destination. That is why `"img/start.jpg"` is the path to this image above.

As mentioned before: we have two potential cases, either we have an image for our scene (in which case we will display it) or we don't have an image for the scene (in which case we will display a dark screen, or something of your choice). It is up to you to do this! You will be using your handy dandy canvas library functions to display your images. 🙂

Some helpful tips for this part of the assignment:

- Create a canvas in your main function using this line:
  `canvas = Canvas(CANVAS_WIDTH, CANVAS_HEIGHT, "Infinite Story")`
- You can use your knowledge of concatenating strings to construct a path to your desired image! It will look something like this, where [your scene key] is replaced by the key of the current scene:
  `path = "img/[your scene key].jpg"`
- To check if we are able to find an image, we can use
  `os.path.exists(path)`
  This will evaluate to `True` if the inputted path is a valid one (meaning that the image exists and we can use it), or to `False` if the inputted path is an invalid one (meaning the image doesn't exist, and we should display our black screen).
- To display an image using canvas, we can use
  `canvas.create_image_with_size(x, y, img_width, img_height, path)` where x and y are the coordinates of the upper left corner of your image.

An optional suggestion: as a small, honest extension, write text on top of every image that says "Generated by DallE". Otherwise users might not know that these were AI generated.

CONGRATULATIONS!! Look at your program, what magic! You just implemented your first nested dictionaries, and your first generative AI project! We are so proud of you. ❤️

Milestone 7: Reflection

Awesome! Now, we have given creative control over storytelling to our code and the AI model. However, is it really as simple as that? Generative AI is an exciting new tool, but it comes with a lot of complicated questions. Is the AI doing a "good" job? Is the AI copying stories from human content creators? What does it mean for our society if we can't tell the difference between AI content and human content. Finally: what values and biases does this generative AI use? In this last milestone we are going to think about that last question.

Consider all of the choices you make when telling a story. Narratives, settings, and even names were all things that you, as a storyteller, would have decided. When we prompt our AI model, what choices does it make for us? Whose stories is it really telling?

Change the story to `engineer_story.json`. This story only has a single scenes, so it should start generating as soon as the user makes a choice.

Run the story a few times and take note of the names that it generates for your co-workers. You can explore statistics behind any names generated using this website:

https://forebears.io/forenames. Feel free to create your own stories or to change the prompt. After, answer these questions in the `infinite_ethics.txt` file:

Q1: You should notice that these names generated by OpenAI for this story are quite popular in a lot of countries. What type of countries/regions keep showing up for these names? Do you feel like they are distributed evenly across the world population? As a hint, compare the names generated in your story to the list of authors of this handout! Justify your answer. Write at least two sentences
Q2: How comfortable do you feel letting the AI model decide the names used in the story? Are you willing to trust the model with other storytelling decisions? There is no wrong position, but please justify your answer. Write at least two sentences.
Q3: Given how ChatGPT handled names in this context, what sorts of issues could you imagine coming up if instead of asking ChatGPT to generate a story, you were using ChatGPT to evaluate candidates for a job?

Submission

Here are the files to submit for this assignment:

- warmup.py
- infinite_story.py
- infinite_ethics.txt

As well as any files you need for your extensions.

Extensions

Better scenes with `history`

Sometimes the scenes generated by OpenAPI can be non-sensical, because the prompt doesn't give the language model access to the *history* of things that have happened. In this extension we are going to append to the prompt a line like "Here is what has happened so far: [history]" where [history] is a list of the scenes the hero has visited and the choices they have made. Develop a way to keep track of the history.

Important: use the "scene_summary" part of each scene when adding to your history of visited scenes. This will keep history from getting too large.

Make your own creative story

One way to go above and beyond is to create your own story. You can write your own plot, your own scenes and even create your own images. Interested in an alternative history where the university is run by the Olhone? Make it! Want to write a story about a detective in the 1920s? Go for it! This is a great way to show off your creativity and storytelling skills, while practicing authoring json.

Open-ended actions

What if the user wants to do something that is not in the list of choices? For example, what if the user wants to "jump in the stream" when the choices are "Take the road up the hill", "Walk up to the stream", and "Knock on the door"? How would you handle this situation? One option is to always give the user the option for an open ended action, then use ChatGPT to generate a new scene based on the user's input. This is a challenging extension, but it is a great way to make your story more interactive.

Important: to make the story still feel believable, you should also introduce a new dynamic where ChatGPT can decide if you action actually happens. For example if the user enteres "fly to the moon" ChatGPT could decide that this is impossible. You could either make that have no impact on the scene the user is in, the action just fails. Alternatively ChatGPT could generate a scene for the failed action state! Get creative.

Deeper ethics analysis

For this extension, you will need to think about the ethical implications of using generative AI in storytelling applications. In the last milestone, you explored the names generated by OpenAI for the story. The ethical issues with large language models and generative AI are deep, and a lot of them are currently being debated! Can you use your ability to call OpenAI from python in order to uncover some interesting ethical insight?

Tracking objects

For this extension, you will need to think about how to keep track of objects in your story. For example, if the user picks up a key in one scene, you will need to remember that they have the key in the next scene. Could that key then be necessary to take a particular action? This is a challenging extension, but it is a great way to make your story more interactive.

Displaying All Scenes

In the assignment specification, when you encounter a scene with no image, you simply draw a black square. What if you instead rendered the text of the scene description. You might find it helpful to use `create_text_area`, a canvas function we haven't used yet. This function takes in a string, which could be multiline and renders it to the canvas, wrapping the text.

```
# create_text_area function
canvas.create_text_area(x, y, end_x, end_y, text, font_size, font)


# example call
canvas.create_text_area(0, 0,
  CANVAS_WIDTH,
  CANVAS_HEIGHT,
```

```
  text="this is a test",
  font_size=24,
  font="Times")
```

Saving Extension

For this extension, you will need to think about how to save the state of your story. You can save either just the new scenes created by open ai, or you could also save the story progress! You should use json.dump to save variables to file. Here is an example:

```
json.dump(my_variable, open('saved_data.json', 'w'))
```

Use Your Imagination 🌱

The joy of an extension is sometimes to think of something that nobody else is doing. We say go for it!