How to Write a Text Adventure in Python Appendix A: Saving A Game – Let's Talk Data

By Phillip Johnson

Merry Christmas! This has been one of the most requested features for me to add to the tutorial, so here goes. While it modifies the free tutorial code, the concepts here can also apply to the book.

Saving and loading data from code to a binary or text format is referred to as "serialization" and "deserialization". When different computer systems need to share data, this is often done with XML or JSON. If that's not a concern, Python can use the builtin pickle library.

Saving the game

First, we'll need to create the action for the player. Let's start by adding the action to the player. First add `import pickle` to the top of the file, then add this method:

```
66
67
68
69
70
```

```
def save_and_exit(self):

    pickle.dump(self, open( "saved_player.p", "wb" ))

    pickle.dump(world._world, open( "saved_world.p", "wb" ))

    print("Game saved!")

    exit()
```

The `pickle.dump` method converts an object to a binary format for Python. The first parameter is the object to save and the second parameter specifies where to save the object. We need to create a save for both the world itself (tiles) and the player.

Next, go to the actions file and create an action for this method:

```
61
62
63
```

```
class SaveAndExit(Action):

    def __init__(self):

        super().__init__(method=Player.save_and_exit, name="Save and
Exit", hotkey='x')
```

Finally, we have to make sure this shows up as an option for the player, so add `moves.append(actions.SaveAndExit())` to the end of the list of actions in `tiles.py`:

```
39
40
41
42
43
44
45
```

```
def available_actions(self):
    moves = self.adjacent_moves()
    moves.append(actions.ViewInventory())
    moves.append(actions.SaveAndExit())
    return moves
```

If you run the game now, you can save the game and you should notice two files appear in the directory of your game code.

Loading the game

To load the game, we basically have to do the same process in reverse. So we'll check to see if the save files exist, and if they do, transform the data into the game objects.

Since we need to handle new games and saved games, I renamed the `play` method to `game_loop` and created a new `play` method:

```
11
12
13
14
15
16
17
18
19
20
21
```

```
def play(saved_world=None, saved_player=None):
    if saved_world and saved_player:
        world._world = saved_world
        player = saved_player
    else:
        world.load_tiles()
        player = Player()
    game_loop(player)
def game_loop(player):
```

The new `play` method has optional parameters for the saved objects. If they are present, we manually set the world and player to those parameters. Otherwise, we default to creating a new world and player.

Next add these imports:

```
from pathlib import Path
import pickle
```

Now we'll create a method that checks to see if the files are present, and if so load them and pass them into our new `play` method. Here we'll use `pickle.load` which reads in a file and unpacks it into an object.

```
def check_for_save():
    if Path("saved_player.p").is_file() and
Path("saved_world.p").is_file():
        saved_world = pickle.load(open("saved_world.p", "rb"))
        saved_player = pickle.load(open("saved_player.p", "rb"))
        save_exists = True
    else:
        save_exists = False
```

I wanted to give the player the option of loading the saved game or starting a new one, so I added this code too:

```
46
47
48
49
50
51
52
53
54
55
56
57
58
59
```

```
if save_exists:

    valid_input = False

    while not valid_input:

        load = input("Saved game found! Do you want to load the game?
Y/N ")

        if load in ['Y','y']:

            play(saved_world, saved_player)

            valid_input = True

        elif load in ['N','n']:

            play()

            valid_input = True

        else:

            print("Invalid choice.")

else:

    play()
```

Notice how we use the two variants of `play`, sometimes with parameters and sometimes without.

Last, we have to change the entry point of our code:

```
61
62
```

```
if __name__ == "__main__":

    check_for_save()
```

Now when you play the game, you should see it pick up your save files and load the game at the spot when you saved it.

As always, the full code is on GitHub!

Click here for Part 1 of the abridged tutorial.