Week 2 documentation

Luc Wehrens i6331384 & Sebastian Schmülling i6304128

To start, we coded two separate python scripts to parse the generated .txt files. We decided on uploading the generated .txt files to our GitHub we set up for this project, so that we would both be working with the same, consistent files.

The python scripts parse the generated .txt files into dictionaries, which we do by figuring out the patterns by which the different entities and events are separated. The code then looks for these patterns, and when it finds one, it knows to start a new instance.
Finally, these dictionaries are then inserted into an array, one by one. This way we end up with an array of dictionaries for both generated entities and events, which we can easily loop through.

In our third python script, we do just that. This code inserts our parsed data into our actual database, which we do with the help of the sql.connector python library. This library allows for python to connect to the SQL server, and execute queries.
We parse the generated data using our previously mentioned code, and then loop through the dictionaries.

First up are the generated events, as these also hold entities.
We first take the title of the generated event, e.g. “player_with_team”, and separate this string based on the underscores. This way, we know what table we should insert the entities in, in this case, “player” and “team”. Then we take the entities, and add them to the array that holds all entities, but not before adding the ‘category’ at the first index. This way, all entities will have the same format. Finally, we add the components which make up the event, to the ‘action’ table in our database. Done.

Finally, the generated entities.
First, we check which category the entity is. Not all categories match up perfectly to our database, so some hard-coding was in order (e.g. both generated ‘vendors’ and ‘npc’ should go into our ‘npc’ table). Then, for each category, we checked which attributes of the generated data matched up to our ERD. We took that data and added it to our database, ignoring the data which did not match.
If a generated entity holds data for a different table as well, e.g. generated “Character” entities also hold “Class” data, we take that data, and insert it into the appropriate table (in this case, “Class”). Then we get the id of that entity, and insert it back into appropriate column of the original entity, i.e. the “class_id” column in the “characters” table, in this case.

To summarize, we:
1. parse all generated events and entities;
2. loop through the generated events, and
    a) format the corresponding entities and add them to the generated entities;
    b) insert the events into the ‘action’ table
3. loop through all generated entities and add them to the database, taking care to check for foreign data in the entity.

The only real problem we encountered was with the ‘character’ table in our database. No matter what, even with foreign key constraints disabled, we could not insert data into this table, and VSC would give an error. We eventually narrowed down the issue to it being called ‘character’, and we found out that somehow, it collided with the MySQL syntax. Changing the table to be called ‘character**s**’ instead, worked, and we could now insert data again.