Rahul Rajagopalan
6.170

Adventure Game Design Notes

The design problem presented in this assignment was to develop an application that would be able to express adventure games that contained all the requested features (including multiple rooms, the concept of an objective, and items that can be used to make progress) and was also decoupled from the actual game scenario. One of the requirements was that the game configuration should use a SQLite database to avoid passing around a lot of data in cookies and to give an opportunity to practice SQL.

To resolve these issues, I stored almost all of the game configuration in the database. Session state only contains the ID for the current player, which is used to retrieve the remaining relevant data. As the game progresses, the database is updated. This approach avoids passing large cookies around, and also does not duplicate data between the database and Python objects. To keep players separate, a fresh set of rows is created for each game. A possible optimization might exist by extracting the immutable components of the game configuration into their own tables; this would reduce duplication of data, but it might also slow down the application because it would need to do more joins.

A full description of the database schema can be found in the object models and the schema design notes.

The available moves for the player are not stored in the database; they can be inferred from the data that exists. gameengine.py contains the code that does this. This required a bit more coding, but reduced the workload for the game designer as they do not have to explicitly specify actions.

Game configurations are stored in gameconfigurations.py. The functions in this file accept a GameBuilder object, and send messages to it to specify the game. This approach has several advantages:

- Game scenarios are completely decoupled from the database; as the GameBuilder handles the details. In fact, the configuration functions do not even have to know that a database exists at all.
- The configuration is written in a declarative style through careful naming of methods on GameBuilder. It is designed to be human-readable.
- It's possible to use a different type of storage by making a different object that implements the same interface as GameBuilder, and passing it to a configuration function. The configuration would not need to change.
- Game configuration can be swapped out simply by calling a different configuration function. I only needed to change one line of code in __init__.py when I switched from the simple game I had written for testing to the "Save the Internet" game that is presented to end-users.
- It might even be possible in the future to expose the configuration "language" to users to allow them to design their own games.

The primary drawback of this design is that it does require creating a large amount of database rows with every game. This is addressable though. If performance is an issue, the database could be denormalized to reduce the number of queries. Also, if it is decided that storing the data in session state is acceptable, a new GameBuilder implementation could be defined to do that, with minimal change to the rest of the code.