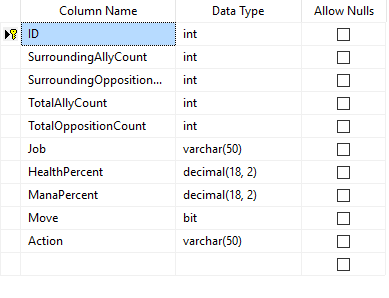
Database & API Structure

The decision to implement a database and API was very late in the game which is why it did not receive the same amount of attention as the rest of the project. It is also not one of the areas in which the project was going to place much focus. Instead it was a quick implementation to complete an additional task.

The database was created using SQL server 2014 and the data structure is very simple.

If the game was to expand and introduce a more in depth experience the database would hold additional data with expanding tables. The nature of the implemented API calls only required the data to be in a single format so the database shares these traits in a single table. If the database was to expand the data would be broken down into smaller and more specialize tables.

# Database Design



As states above the database is very simple, it is only used for one purpose and that is the storing and retrieving of one data transfer object. The elements stored into the database have been picked out as they are the key components within the AI formulas.

The database is stored on Microsoft Azure as it links very well with SQL server 2014. Using SQL server I was able to log into the azure database and query it without ever needing to go back to the azure portal.

# Server & API Calls

The API is set up on a very simple node.js server. The server doesn’t do anything fancy it just responds to two route calls required for the game.

First of which is a GET request to /api/Turns.

For this call to be successful the request must hold every database element from SurroundingAllyCount to the ManaPercent (See above). These variables are then passed into the SQL query as the WHERE attributes. The database returns all the entries from the database matching the corresponding parameters. This call is only ever made by the AI as the human controlled characters do not require this information. The information retrieved is then used to help formula the AI’s interactions within the game. As this element of the project was seen as an addition to the initial scope it was not given enough time to perfect. Given the game development had more time this is one of the areas that would see a lot of rework and refinement.

The second API call is the put request to /api/Turns

The game will take most of the information needed for the API at the beginning of that characters turn, as well as setting up the request call. When the player has completed an action and movement the API request will be sent, with the additional Move and Action attributes. It was done this way because the information used in formulating a complete turn is based on the variables or situation at the beginning of that characters turn.

The server and API are stored on a digital ocean droplet which has been linked to a Distelli build pipeline. This enables the API and Server to be updated as soon as the new code is committed to the Git Hub repository. Although this was not necessary it was fantastic practice for future work and is similar to the work flow a professional web application would follow.

To aid the debugging process of the API and Database a piece of software called Postman was used. This is a simple chrome extension or downloadable application that allows the user to experiment with the URL requests without having to run them in game. Without this the entire API development would have been a lot slower and more error prone.