**Database Design**

Table 1: PlayerTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | posX | posY | posZ | energy |
| Kris | 2.54 | 0.49 | 3.12 | 69 |
| Ryan | 4.32 | 0.49 | 1.14 | 32 |
| Vince | 1.76 | 0.49 | 1.48 | 53 |
| Charlie | 2.74 | 0.49 | 2.88 | 47 |
| Joe | 2.42 | 0.49 | 3.51 | 10 |

The PlayerTable stores the data for the Player character. It stores its’ position on the X, Y and Z axis. When the client loads the data from the server, it will take all of these variables and use them for the position of the character and its’ energy. This way, the game state of the player would be saved and loaded from the database. The id is used for identification of the user, as the game shouldn’t allow several players with the same name, it’s an easy way to identify the player inside the database. There is no need to save any extra data inside this database table like velocity, for instance, because it would make the player load a game and instantly be running in a direction. Having only the necessary columns would also make the database more performance efficient as well as easier to manipulate or use.

Table 2: EnemyTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | posX | posY | posZ | energy |
| Kris | 0.54 | 0.49 | 0.12 | 31 |
| Ryan | 1.12 | 0.49 | 0.33 | 54 |
| Vince | 4.61 | 0.49 | 3.32 | 97 |
| Charlie | 3.88 | 0.49 | 4.12 | 87 |
| Joe | 4.52 | 0.49 | 0.37 | 8 |

The EnemyTable works in the same way as the PlayerTable. The reason for that is simply because the enemy inside the game has very similar functionality to the player of the game. It’s also just a single entity therefore it wouldn’t add more complexity to the database by having any extra identification IDs.

Table 3: MushroomsTable

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id | mushID | posX | posY | posZ | currentAge | deathAge |
| Kris | 1 | 2.22 | 0.5 | 2.22 | 31.12 | 67 |
| Kris | 2 | 3.33 | 0.5 | 3.33 | 10.11 | 99 |
| Kris | 3 | 4.44 | 0.5 | 4.44 | 22.22 | 44 |
| Kris | 4 | 1.11 | 0.5 | 1.11 | 64.64 | 75 |
| Kris | 5 | 1.99 | 0.5 | 1.99 | 6.66 | 31 |
| Kris | 6 | 2.88 | 0.5 | 2.88 | 4.82 | 80 |

The MushroomsTable stores the data for the mushrooms correlated to a player’s game state. The id is used to identify which mushrooms are needed for the game state we’re about to load. The game would look for all entities inside the table with the id of the character name and use the data to spawn them. The next value is mushID. This value is used in order to know which of the mushrooms the data is for. The posX, posY and posZ variables are used for the position of the mushroom. They’re being stored in order to return the mushrooms to their original position on the game scene. The currentAge is the age of the mushroom. Since the mushrooms age during gameplay, this value is used in order to return them to their exact age from the point of saving. The deathAge is used, because when spawned, the mushrooms get assigned a random variable that would indicate their length of life. When loading back to the saved game state, this value is needed in order to return the mushrooms to their exact point of their lifetime before saving.