**Side Battle RPG Database**

Document by Raf

Last Updated: August 23, 2018

Table of Contents

[Getting Started 2](#_Toc522796556)

[The SQLite Database 3](#_Toc522796557)

[Database Diagram 3](#_Toc522796558)

[Database Project Conventions 3](#_Toc522796559)

[Attachable Tables 4](#_Toc522796560)

[Regular Tables 5](#_Toc522796561)

[Database: Many-to-Many Relationship Tables 6](#_Toc522796562)

[TypesLists: The Special Table 7](#_Toc522796563)

[Other Notes 7](#_Toc522796564)

[The Visual Studio Solution’s Database 8](#_Toc522796565)

[File Structure 8](#_Toc522796566)

[XAML View and C# Controller 8](#_Toc522796567)

[All About the Table Templates Folder 8](#_Toc522796568)

[Header 8](#_Toc522796569)

[Map Builder 8](#_Toc522796570)

[Header 8](#_Toc522796571)

[Header 8](#_Toc522796572)

[Header 8](#_Toc522796573)

[Header 8](#_Toc522796574)

[Enemy Battle Simulator 9](#_Toc522796575)

[Header 9](#_Toc522796576)

[Header 9](#_Toc522796577)

[Header 9](#_Toc522796578)

[Header 9](#_Toc522796579)

[General Comments/Tips 9](#_Toc522796580)

[Header 9](#_Toc522796581)

[Header 9](#_Toc522796582)

[Header 9](#_Toc522796583)

Getting Started

Things to Note:

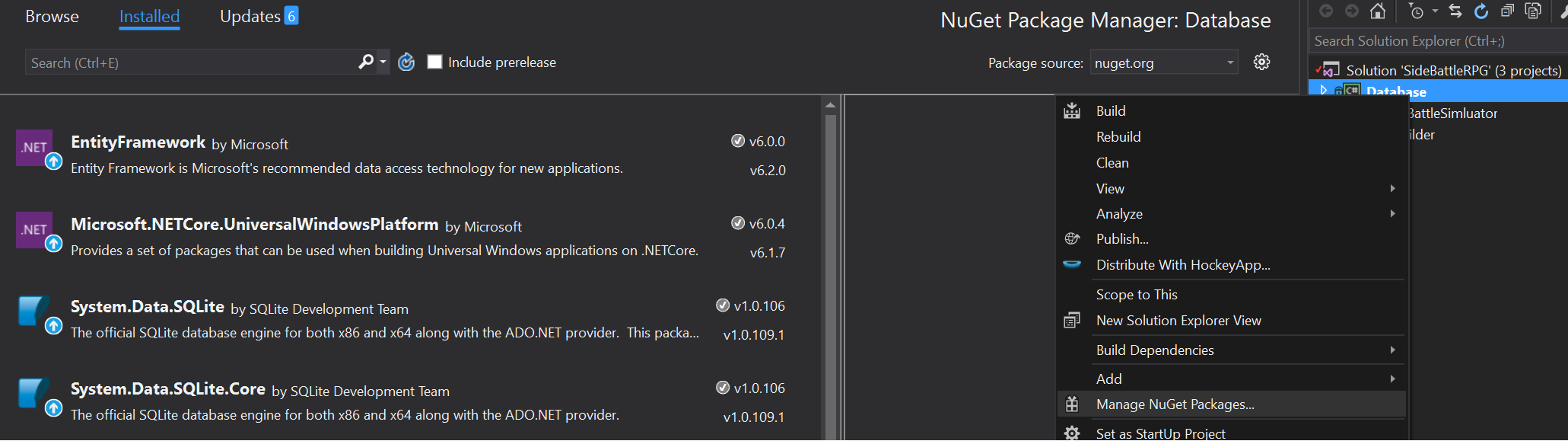
* The README file in the base directory of the repo has the project overview.
* The program is operated on Windows 10
* The program is in C#, XAML (markup language), and uses SQLite: You can learn these along the way
* The database layout is based off the Object-Oriented Programming Style
* The files in the Visual Studio Solution is a deviation of the Model-View-Control framework
* Make sure you have at least 10 GB in your storage (Much more than required, but just in case)
* You do not need to use the command line at all

Install the following:

* Visual Studio 2015: <https://visualstudio.microsoft.com/vs/older-downloads/>
* SQLite Studio (Windows Installer): <https://sqlitestudio.pl/index.rvt?act=download>
  + Alternatively, any program that supports handling data, in SQLite, will do

Install only if Visual Studio forces you:

* UWP support for Visual Studio 2015
* System.Data.SQLite: This should already be installed from clone the Git repo. It not, then right click on the project and click “Manage Nuget Packages” (On the bottom right)

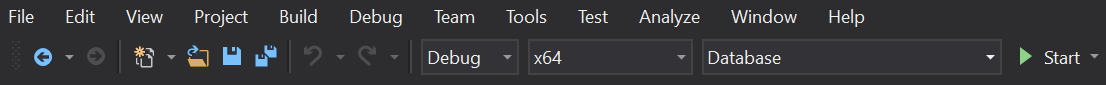


At the top left search bar, type in “sqlite”, select “System.Data.SQLite”, and install the latest version. Iterate for each project under solution “SideBattleRPG”.

After installation – Loading the project into your workspace:

* Visual Studio solution: Clone the repository, and sync the files from Github <https://github.com/github/VisualStudio/blob/master/docs/contributing/publishing-an-existing-project-to-github.md>
* SQLite Database: Please refer to the README file in the DataDump folder for the instructions

You can switch between working on projects by selecting the combo box, on the bottom right of the image below, with the name “Database”



In the Visual Studio Solution: if you run the “Database” project (Press F5 or CTRL+F5) and see a title called “Database” with a NavBar on the middle of the screen, then you got it all setup. Congrats!

# **The SQLite Database**

## Database Diagram

ER Diagrams

## Database Project Conventions

The naming convention is strict:

* All tables must be named in singular format (i.e. “CREATE TABLE Player" instead of “… Players”)
* All tables must have a primary key by table name plus “ID" (i.e “Player\_ID INTEGER PRIMARY KEY”)
* Any key referencing BaseObject must have “ON DELETE CASCADE ON UPDATE CASCADE”
* No sub-sub-classes: Use attachable tables instead
* Many-to-many relationship tables must have “\_To\_” in between the relating classes (i.e. Player\_To\_Skill)
* The left name before the “\_To\_” in many-to-many relationship tables (e.g. The “Player” in “Player\_To\_Skill”), should be the host class. This means that Player\_To\_Skill is being modified while the user is updating the “Player” page, and not the “Skill” page, in the application. This rule is not enforced, but highly recommended for design consistency.
* TypesLists is the only table that breaks conventions: it is the only special case in the database

Failing to follow these conventions may disrupt the database application’s logic.

## Attachable Tables

These include Super-classes and DDO’s (Dumb Data Objects).

**BaseObject:** The most important table in the whole database. Provides the most common attributes for almost all the tables that will be the classes of the game. It consists of:

* The ID of a table item
* Name
* Description
* Main Image
* Creation Date
* Updated Date

**Stats:** DDO that stores numbers indicating Attack, Defense, etc. Mapped onto several tables

**PassiveEffects:** Generalized as a superclass for states and passive skills

* ElementRates: Modifies the affected player/enemy’s elemental resistancies/weaknesses, when they have the effect intact
* HPRegen and SPRegen: The set amount of HP/SP at the end of every turn
* SPConsumeRate: The modified amount of SP consumed when the player/enemy is affected
* TurnEnd1 and TurnEnd2: Number of turns the effect lasts (e.g. The enemy is sleeping for 2 to 4 turns)
* TurnEndSequence: Determines when the effect activates (e.g. After action, after turn, etc.)
* GetHitRemove: The probability of a state being removed, after getting hit by a player/enemy
* ComboDifficulty: How difficult it will be to execute combo skill
* Counter and Reflect: The probability the target deflects physical and magical attacks, respectively
* ExtraTurns: The number of extra turns
* Physical and Magical Damage Rate: Modified amount of damage given and taken, in %
* DisabledToolType1 and DisabledToolType2: Tool types that canoot be used
* StatModifiers: The changed stats while the player/enemy is affected (Acts as buffs and debuffs)

**Tool:** Generalized as a superclass for the item, skill, and weapon classes

* Type, formula (as in ‘Damage Formula’), and HPSPModType: refer to how the tool is classified
* HPAmount and SPAmount: set values to how much of the tool’s target HP/SP is gained/lost
* HPPercent and SPPercent: same as above but instead is based off the target’s Max HP/SP
* HPRecoil: amount of HP, in %, the user loses after using the tool
* Scope: determines the tool’s potential targets
* ConsecutiveActs: number of times the user applies the tool on the target, in the same turn
* RandomActs: number of times the user’s tool hits targets, by random
* Element: tools element, based on the TypeLists table
* Power: The magnitude of damage/recovery the tool gives to the target
* Accuracy: The probability of the tool hitting the target
* CriticalRate: The probability of the tool inflicting a critical hit
* Priority: Users applying with higher priority tools will always move before other battlers
* ClassExclusive1 and ClassExclusive2: If either is not set to “None”, then the tool can only be applied by users in that BattlerClass

## Regular Tables

The main tables the database works with. All these tables rely on the BaseObject table.

**Achievement:** Accomplishments by the player (Mostly manipulated by the Event class)

**Animations:** Ignore for now – The tool’s animation sequence

**BattlerClass:** A set of base stats, movesets, and wieldable weapons for the player/enemy. It is named BattlerClass, instead of Class, because of ‘Class’ is a keyword on C#

* UpgradedClasses: A battlerclass can upgrade into two different advanced BattlerClasses – Reserved for base classes
* UseableWeaponTypes: Weapon types (from TypesLists table) the BattlerClass can use
* ScaledStats: The base stats

**Command:** Ignore for now – A single command for the tool’s animation

**EGSingleEnemy:** If dealing with the ‘EnemyBattleSimulator’ project, the ‘EnemyBattleSumilator’ section below. Ignore otherwise.

**Enemy:** Opponents and obstacles against the player

* ElementRates: The enemy’s elemental resistancies and weaknesses
* BossType: Determines if an enemy is a regular enemy, a mini-boss, a regular boss, or final boss
* Exp: Amount of experience points obtained by defeating the enemy
* Gold: Money obtained by defeating the enemy
* BattlerClassID: The enemy’s class
* ScaledStats: If Enemy has a BattlerClass, this determines the deviation from the BattlerClass’s base stats (from -3 to +3). If BattlerClass is null, then this will be the enemy’s custom base stats from (0 to 8.5)

**EnemyGroup:** If dealing with the ‘EnemyBattleSimulator’ project, see the ‘EnemyBattleSumilator’ section below. Ignore otherwise

**Environment:** The general background of the area the player is travelling in. MapBuilder needs this

**Event:** Ignore for now, unless it involves the ‘MapBuilder’ or EnemyGroup project. Also, for skills with special cases

**Item:** A consumable tool that helps players inside and outside of battle

* DefaultPrice: If shops don’t have custom item prices, this will be the the amount money they are sold for
* Consumable: Determines whether the item disappears from the inventory, after use
* PermStatMods: If the item is used, the target’s stats will permanatly change
* TurnsInto: What a used consumable item turns into (i.e. Eating “Giant cheese” turns into “2/3 Cheese”)

**Map:** Important class for the ‘MapBuilder’ project; more can be added onto this table, upon making Maps

**PassiveSkill:** A permanent or equippable passive effect on the player/enemy

* HPMin/HPMax/SPMin/SPMax: The % values determining when the skill is in effect, based on HP/SP
* AnyState: Skill will be in effect if the user has a state
* NoState: Skill will be in effect if the user does not have a state
* StatesActive1 and StateActive2: Only activated when either of the states are present in the user
* StatesInactive1 and StateInactive2: Only activated when both states are not present in the user
* ExtraEXPGain: The extra amount of EXP gained
* ExtraGoldGain: The extra amount of gold gained

**Platform:** Important for the ‘MapBuilder’ project

* FloorDamage: The amount of HP, for every N seconds, the player loses, when they are on that platform
* BounceVelocity: How high the player jumps above ground (Almost always set to 0, for no bouncing)
* LandingDamp: Slipperiness of the platform: (Set to 1 for the regaulr amount of friction)

**Player:** Playable characters – The ones who make up your party

* ElementRates: The player’s elemental resistancies and weaknesses
* The four other non-key attributes determine the player’s friendliness and teamwork ability
* NaturalStats: Uniquely deviated stats from the Player’s BattlerClass

**Projectile:** Ignore for now – Might be integrated with tools, somehow

**Skill:** A tool embedded and used by a player/enemy

* SPConsume: Number of SP the skill consumes (0 to 100)
* NumberOfUsers: Any value greater than 1, means the skill requires teamwork, button combinations, etc.
* Charge: The amount of turns the user remains idle, before using the skill
* Warmup: The amount of turns the player/enemy needs to wait, at the beginning of every battle, before being able to use that skill
* Cooldown: The amount of turns the player/enemy needs to wait, after using the skill, before being able to re-use the skill again
* Steal: Determines whether the skill can steal items from the target (Rarely ever used)

**State:** A passive effect obtained by being exposed to a tool

* MaxStack: The maximum number of times the state can be planted onto the target
* Stun: Prevents the target from moving
* KO: Determines that the target, with the state, is unconscious with 0 HP
* Petrify: Same as KO, except the HP is the same
* StepsToRemove: The number of steps taken in the overworld to remove the state

**Weapon:** A wieldable tool for players and enemies

* Type: The weapon type based on the TypesLists
* Range: The amount of distance the weapon can reach – some value from 1 to 9, inclusively
* DefaultPrice: If shops don’t have custom item prices, this will be the the amount money they are sold for
* DefaultQuantity: Number of times the weapon can be used, before breaking (Set to 0: unlimited usability)

## Many-to-Many Relationship Tables

Tables that are bridges to connect foreign keys to other tables.

**Player\_To\_BattlerClass:** Players have BattlerClasses and a BattlerClass is owned by many Players

**Player\_To\_Player:** Determines relation levels between two players

**Player\_To\_Skill:** The level required for the player to reach the skill

**Player\_To\_State:** Determines how effective the state is, against the player

**Skill\_To\_Enemy:** The enemies summoned, when the skill is used (Rarely used)

**Skill\_To\_Player:** The players summoned, when the skill is used (Rarely used)

**Tool\_To\_State\_Give:** The probability of the target getting secondary effects after using the tool

**Tool\_To\_State\_Receive:** The probability of the user getting secondary effects after using the tool

**NOT IMPLEMENTED YET**

**PassiveEffect\_To\_State:** If the player/enemy is inflicted with a passive effect, the state rate determines how much more (or less) they will be vulnerable to that state.

**… Some others I probably forgot about**

## TypesLists: The Special Table

Stores elements with only a single name. Each of the types specified below only indicate a single text attribute. By design perspective, the TypesLists table groups all the tables that would only have a “Name” attribute, instead of having many DDO tables with different names and the exact same columns.

**Elements:** An attribute affecting damage rates between tools and players/enemies. In this project’s context, it would be natural sources such as water, fire, air, etc.

**Weapon Types:** Wieldable weapons used by players and enemies

**Tool Types:** Categorizes the general effects of the tool

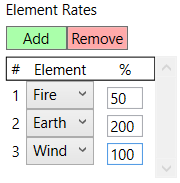
**Tool Formulas:** A mapped value for the tool, indicating which formula is used (i.e. damage formulas for skills)

## Other Notes

Suggestions to improving the database design is open and encouraged.

One issue with the general database design is how the many-to-many relationships are laid out. Most of these types of tables are very similar, if not identical, to each other. The similarities are due to maintaining the foreign key constraints.

Another issue is how the tables attributes map into an element from TypeLists. For example, the Player class has an attribute called “ElementRates”. ElementRates is a text attribute storing List IDs for a specific Type-List, followed by another number. The following table data below would set ElementRates = ‘1\_50\_3\_200\_4\_100’, instead of using foreign keys, and store it into the database upon committing the transaction.



The numbers 1, 3, and 4 refer to the selects IDs of the elements in the TypesLists table. 50, 100, and 200 refer to the rates on the table. Any solution for this issue, that does not introduce redundancies on the database, like constant string values, should replace this old system.

# **Visual Studio Solution Project: Database**

Mimics the Model-View-Control framework. The files are very closely linked to the SQLite database. Each .xaml file (XAML View) contains a .xaml.cs (C# Controller) file inside of it.

## File Structure

Pages and User Controls (A.K.A. page templates act like the following)

**BaseControls:** The main navigation and operations handler for the whole database. The only needed base controls is the navgivation bar, the table list, and the buttons footer. All of these have already been implemented.

**Classes (Page):** The page under the navigation bar’s selected options. These are the main controls for viewing and changing the content of a regular table.

**ClassesUnstructured (Page):** Pages that do not follow the Classes format. This section is currently reserved for TypesLists. Ignore it otherwise.

**ClassTemplates (Template):** The main controls for viewing and changing the content of an attachable table.

**TableTemplates (Template):** Even though it functions properly, this section is a complete mess. See the next section “All About the Table Templates Folder” (Two sections below this one) if you want to work with it. Basically, these manage TypeLists mapping and many-to-many database relationships.

**Utilities:** (Mostly) static classes that help with the database. See the comments on their respective files. **ObjectOperations** and **SQLDB** are the two most important utilities. It is highly advised to read their comments, before starting on the Database Project.

## All About the Table Templates Folder

Text

## Application Control Flow

Upon starting “Database”, the code goes as follows:

## Quick Setup

To create a new class, template. Highlight the

There are several pages and templates that have already been implemented. Use those as examples to support more classes.

# **Visual Studio Solution Project: Map Builder**

## High Level Overview

There is nothing implemented here yet. The map builder is a recently added project and is supposed to have data stored to build maps on the game. You can start anywhere in this section, mechanics wise. Two high level design are provided (The “design ideas” below) for this section. If you are not able to think of a decent third design idea, then work with one of the two designs below. Add any ideas you have in mind, or something else that you might find convenient. Asides from what is mentioned on the rest of this section, anything goes.

## Database Related Information

If you’re currently not comfortable dealing with the database, then skip this section.

The

## Design Idea 1: Hand-drawn Maps

This design is preferable for the scope of the project. Maps are more fluid and less customizeable/forced. On the downside, it is a slower and more complex system to manage. If you’re looking for challenge and get more experience as a programmer, this would be the one to pick.

## Design Idea 2: Tile-based Maps

This design is less preferable, but faster and easier to work with.

# **Visual Studio Solution Project: Enemy Battle Simulator**

## High Level Overview

Text

## Database Related Information

Text

## Header

Text

## Header

Text

# **General References**

## UI Plan for Menus/Inventory

Text

## UI Plan for Battling

Text

## Final Comments

Text