RPG Database UI System

Document by Raf

Last Updated: August 25, 2018

Table of Contents

Getting Started

Before reading this document, make sure you have already read the **README** file on the root directory of the SideBattleRPG repository.

Install the following programs:

- 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:

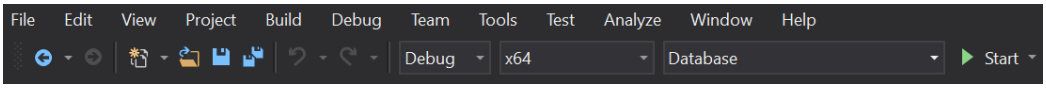
- Support for UWP apps for Visual Studio 2015
- System.Data.SQLite API: This should already be installed from clone the Git repo. If not, then:
    - Right click on the project (under solution “SideBattleRPG”), you want to install the API into, and click “Manage Nuget Packages”
    - On the top left 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 projects into your local workspace:

- Visual Studio solution: Open Visual Studio and Fork/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 following steps below
    - Open SQLite Studio (or another SQLite program): Create a new database and connect to it.
    - Copy and paste everything from "TablesAug2018.sql" onto your SQLite editor.
    - Highlight everything (CTRL+A) then run the code (F5)
    - If running gives an error (i.e. your database isn't populated with tables), then try to fix it somehow or refer to the next step. If running does work, then skip the next step.
    - If fixing it doesn't work, contact Raf for a copy of the db file. You will not be needing the new database you just created. You can either delete it or use it as a test/backup database.
    - If you're having security issues with writing on the database: place the db file in "C:\Users\<YourUserName>".
    - Go to SideBattleRPG/Database/AccessDB.cs (In your cloned/forked repository)
    - On line 8, change the substring "C:\Users\User\" to the directory your database resides in, then hit CTRL+S to save.

Author’s Note: I don't know how to properly do version controlling in sqlite. When updating the datbase, I suggest exporting your current version of the database, then sending it to the DataDump folder, for now.

You can switch between working on projects by selecting the combo box, that says “Database”, at the top of the program’s window.



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

This is the database for the entire project. It is a tweaked version of an ER diagram

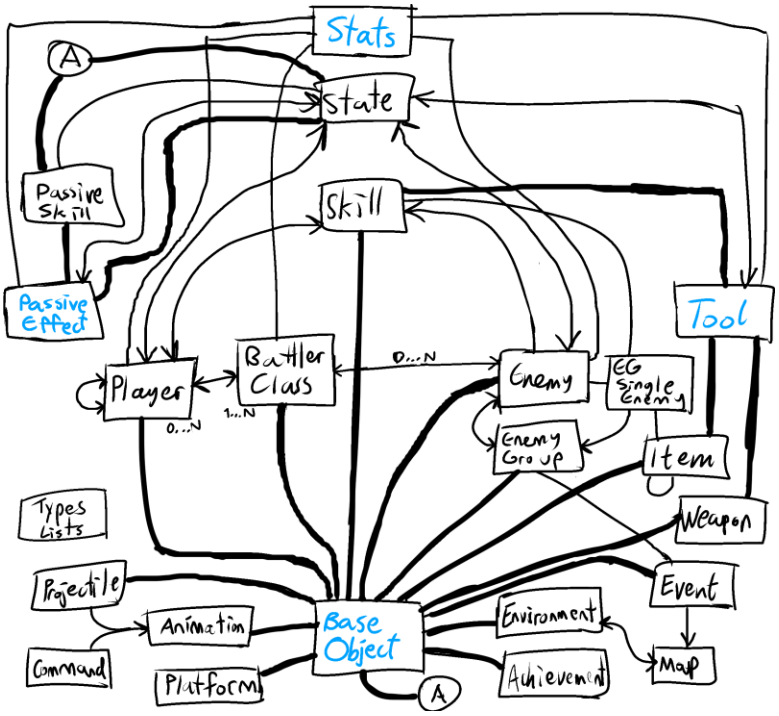


Diagram Notes:

- Table --- Table: one-to-one relation
- Table ← Table: One-to-many relation
- Table ←→ Table: Many-to-many relation (Filtered out many-to-many relations)
- The ER diagram table attributes and diamonds have been filtered out (too cluttered otherwise)
- Superclasses are treated as tables that can be attachable (Not using ISA connventions to satisfy app logic)
- Table names highlighted in blue are attachable tables
- The tables with blue names do NOT have foreign keys from tables with non-blue names, despite one-to-one dependent constraints from other tables
- Diagram may not be complete and will be updated

Database Project Conventions

The naming convention is strict. Failing to follow these conventions may disrupt the database application's logic.

- 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 (See section below)
- Many-to-many relationship tables must have "_To_" in between the relating classes (i.e. Player_To_Skill)
- The left name before "_To_" in a many-to-many relationship table must be the host class. This means that the table "Player_To_Skill" is being modified while the user, in the application, is updating the "Player" page, and not the "Skill" page. This rule is not enforced, but highly recommended for consistency.
- TypesLists is the only table that breaks conventions: it is the only special case in the database
- For the sake of this document: the tables will be categorized into 3 different types (Next 3 sections below)

Attachable Tables

These include DDO's (Dumb Data Objects) as well as tables that would act as Superclasses or abstract classes in an object-oriented environment.

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 table that stores numbers indicating Attack, Defense, etc. Mapped onto several tables:

- HP = Max Health (In range [1, 2000])
- Atk = Attack (In range [1, 350])
- Def = Defense (In range [1, 350])
- Map = Magic Power (In range [1, 350])
- Mar = Magic Resistance (In range [1, 350])
- Spd = Speed (In range [1, 350])
- Tec = Technique (In range [1, 350])
- Luk = Luck (In range [1, 350])
- Acc = Accuracy (In range [-100, 100])
- Eva = Evasion Rate (In range [-100, 100])
- Crt = Critical Hit Rate (In range [-100, 100])
- Cev = Critical Hit Evasion Rate (In range [-100, 100])

PassiveEffects: Generalized table for the State and PassiveSkill tables

- 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 tables

- 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/NoState: Skill will be in, or not in, effect if the user has 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: An effect obtained by being exposed to the tool's secondary effects (e.g. Posioned, Frozen, Asleep, etc.)

- 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 that might need to be added, eventually

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 tables, acting as DDOs, 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

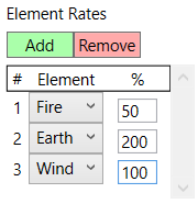
Tool Formulas: A mapped value for the tool, indicating which formula is used (i.e. damage formulas for skills)

Other Notes

Suggestions to improving the database design are 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, taken from the database UI application, would set ElementRates = ‘1_50_3_200_4_100’, instead of using foreign keys, and store it into the database upon committing the transaction.

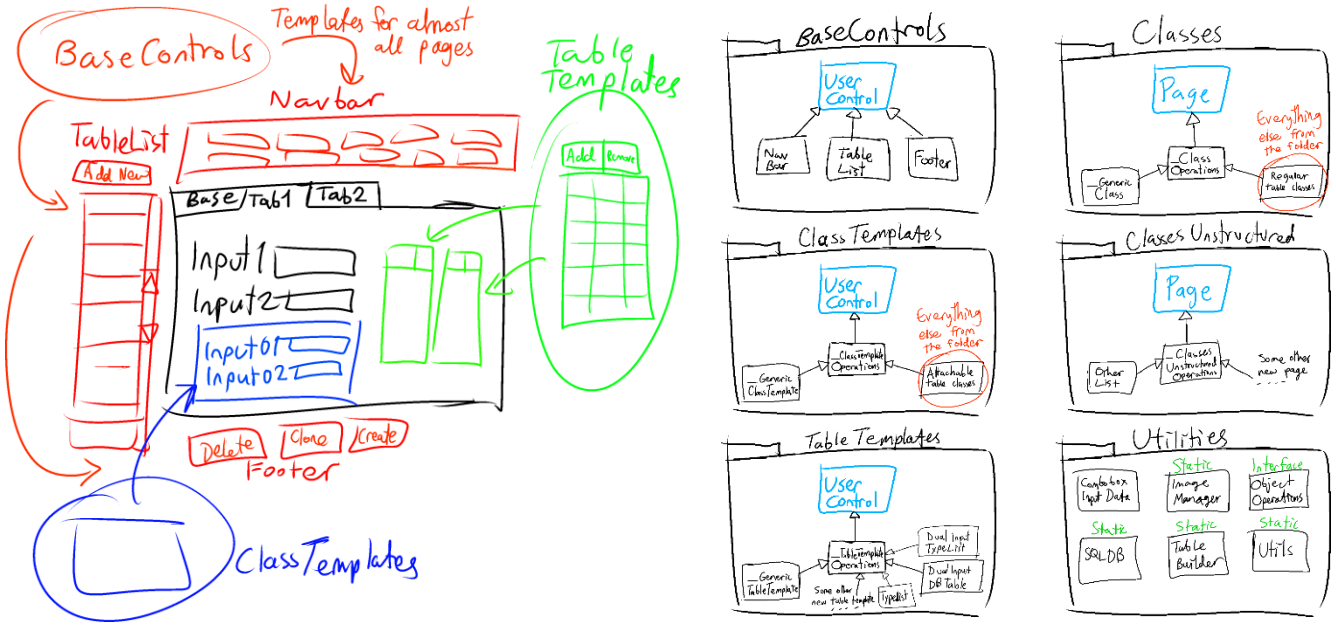


The numbers 1, 3, and 4 refer to the selects IDs of the elements in the TypesLists table. 50, 200, and 100 refer to the rates on the table. Any solution for this issue, that does not introduce redundancies on the database, like constant string values, should replace this old system.

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. Take time to explore the files. Some variables from the C# file have been declared in the XAML file.

File Structure



Referring to the diagram on the right: The database UI project is divided into six folders with the following content inside them (excluding UserControl and Page). The arrows indicate generalization (i.e. NavBar is a subclass for UserControl). Please note that UserControl and Page are from the WPF API on Visual Studio itself. Do not try to look for them in the folders.

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: the page the user lands on. 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

The folder is structured as follows:

_TableTemplateOperations: The main and general control for providing the database operations. DO NOT modify this file unless you know what you're doing.

_GenericTableTemplate: Works in a similar manner to the Classes and ClassTemplates folder. This only exists to provide a shortcut for creating a new table template. This is not needed very much.

DualInputDBTable: Reads Many-to-many tables, such as Player_To_Skill, Skill_To_State, etc. This table displays a UI table with two rows: Row 1 is a combo box of content read from the database, and row 2 is a text input field, which is not always required.

DualInputTypeList: On a UI level, it serves as an identical function to DualInputDBTable. The difference between this and DualInputDBTable is that reads only stored data from the TypesLists table (See "Other Notes"; the last section under "SQLite Database")

TypeList: A table with a single textbox input. This table template is hardcoded only to serve as an interface to the TypesLists table.

Some notes about handling table templates on a page:

- ClassTemplates can use table templates, but not the other way around
- NEVER use the InitializeNew() method for table templates. Use the Setup(…) method under SetupTableData(…), which is a method in the pages (Classes folder) and templates (ClassTemplates folder)
- The following diagram is an example of how to initialize a table template (In Player.xaml.cs):

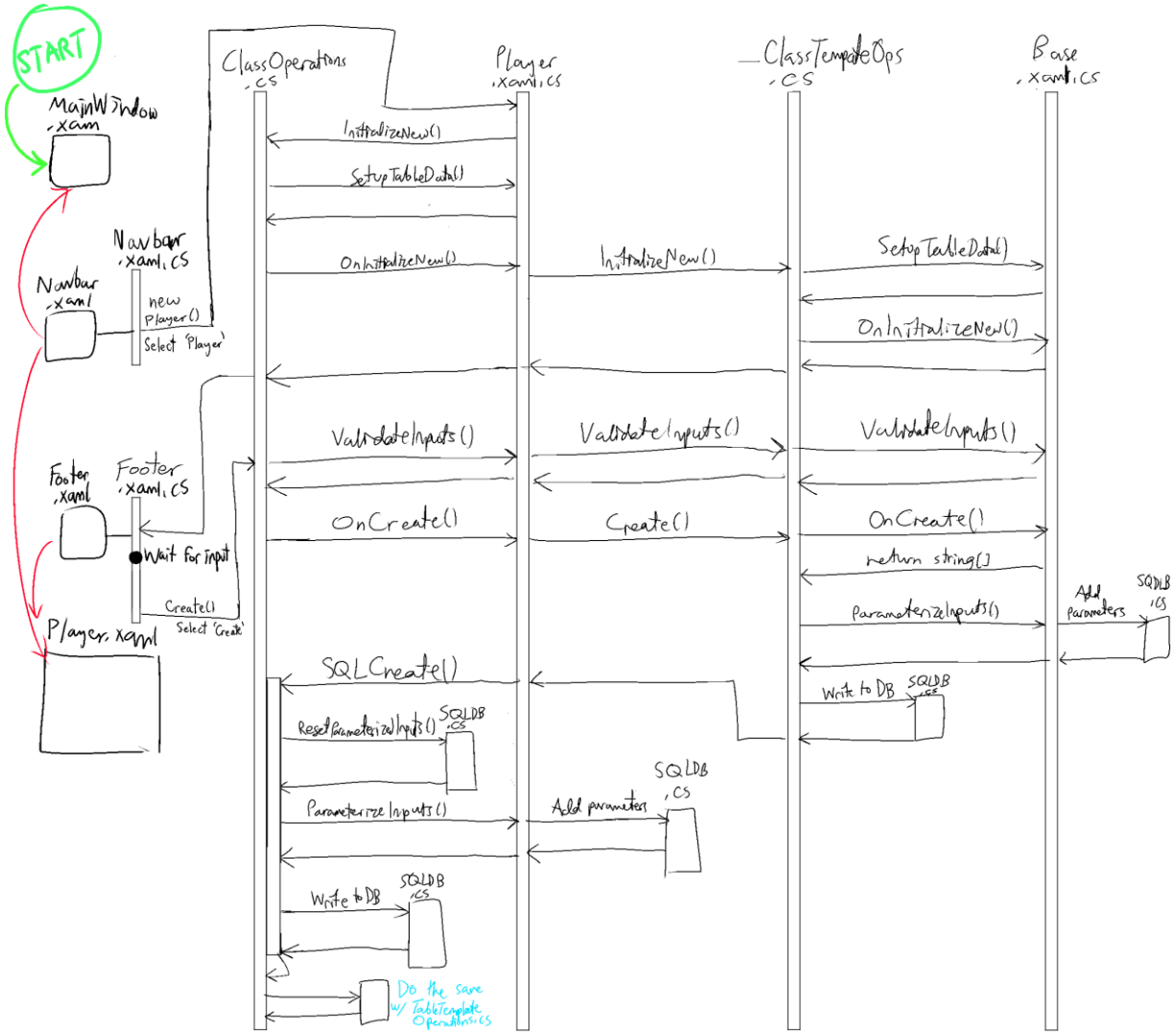
```
17      protected override void SetupTableData()
18      {
19          ClassChoices.Setup("Player", "BattlerClass", "Possible Classes", new List<string> { "Class" });
20          SkillChoices.Setup("Player", "Skill", "Skill Set", new List<string> { "Skill", "Level" });
21          StateRates.Setup("Player", "State", "State Rates", new List<string> { "State", "%" });
22          ElementRates.Setup("Player", "TypesLists", "Elements", "Element Rates", new List<string> { "Element", "%" });
23          Relations.Setup("Player", "Player", "Compatibilities", new List<string> { "Player", "Comp." });
24          SkillChoices.AttributeName = "LevelRequired";
25          StateRates.AttributeName = "Vulnerability";
26          ElementRates.AttributeName = "ElementRates";
27          Relations.AttributeName = "CompanionshipTo";
28      }
29
30      protected override void OnInitializeNew()
31      {
32          Base.InitializeNew();
33          NatStats.InitializeNew();
34          NatStats.HostTableAttributeName = "NaturalStats";
35          CompanionshipInput.Text = "100";
36          SavePartnerRateInput.Text = "100";
37          CounterattackRateInput.Text = "100";
38          AssistDamageRateInput.Text = "100";
39      }
```

  - If the attribute is a table template, then the template goes under SetupTableData()
  - None of the table template attributes are under OnInitializeNew()
  - The final argument for Setup(…) is a list of columns that will be on the table
  - Specifically, for both the DualInputDBTable and DualInputTypesList table templates:
    - Number of columns must be exectly one or two
    - AttributeName is set if the table requires the text input column
    - AttributeName is only set if the table has two columns
    - AttributeName's value is the name of the extra attrbute in a many-to-many datbase table (e.g. The fourth attribute in the Player_To_Skill table is called LevelRequired)
    - ElementRates needs 5 arguments due to being a DualInputTypeList template: the table does not have a Player_To_TypesLists table, but rather goes into TypesLists, checking List_Type = "Elements"

Explaining this section any further will make the content more confusing, so experiement around with the table builder. See what the variable names can do, and they are set up that way. If you have a better way to design the table templates, a solution that elegantly handles many-to-many tables and TypesLists data, then use that instead.

Application Control Flow

The following diagram is a sample of how the application's code runs when the user initializes input then creates the player. Similar rules apply to the "read then update" action.



Quick Setup

It is highly recommended that you do not directly add new files, from scratch, to the Classes, ClassTemplates, and TableTemplates. Inconsistencies and more bugs will come from it, otherwise.

Below are templates/shortcuts to creating the files that will interface with the database tables.

Creating a new View + Controller for Classes: Creates a new page class to directly interact with a specific regular table from the database.

- Go to BaseControls > NavBar.xaml > NavBar.xaml.cs
    - There are SetupFromClick() functions that have been commented out. Go to the table you will be planning to interface with (e.g. Enemy, Platform, State, etc.), and uncomment the line
- Go to Classes: Single left click "_GenericClass.xaml"
- Press CTRL+C then CTRL+V: Two files called "_GenericClass – Copy" (.xaml and .xaml.cs) will be made
- Right click on "_GenericClass – Copy" and rename it to the name of the table you will be working with
- Go to the .xaml.cs version of the newly renamed file and replace both of instances of _GenericClass to match the file's title

```
7    public partial class _GenericClass : _ClassOperations
8    {
9        public _GenericClass()
```

- Go into .xaml version of the newly renamed file and change "Database.Classes._GenericClass" (line 2) to "Database.Classes.<TableName>" where TableName is the name of table you are working with
- At Visual Studio's navigation bar: Go to Build > Rebuild Solution (May take more than several seconds)
- If rebuilding didn't work, then try Build > Build Solution. If that doesn't work, then give the compiler some time; you might need to restart the application if it still does not notice it
- If all else fails, then try all the steps again to see what went wrong (XAML compiler is far from perfect)

Creating a new View + Controllers for ClassTemplates: Creates a template for the pages. These directly interact with a specific attachable table from the database.

- Go to ClassesTemplates: Single left click "_GenericClassTemplate.xaml"

```
8    public partial class _GenericClassTemplate : _ClassTemplateOperations
9    {
10       public _GenericClassTemplate()
11       {
12           InitializeComponent();
13           ClassTemplateTable = "_GenericClassTemplate";   // DO NOT FORGET TO SET THIS
14       }
```

- Press CTRL+C then CTRL+V: Two files "_GenericClassTemplate – Copy" (.xaml and .xaml.cs) will be made
- Right click on "_GenericClassTemplate – Copy" and rename it to the name of the attachable table you are working with
- Go into the .xaml.cs version of the newly renamed file and replace all three instances of _GenericClass (lines 20, 22, and 25) to match the file's title
- Go into .xaml version of the newly renamed file and change "Database.ClasseTemplates._GenericClass" (line 2) to "Database.ClassTemplates.<TableName>" where TableName is the name of attachable table you are working with
- Follow the last three steps of "Creating a new View + Controller for Classes" (right above)

Creating a new View + Controller for TableTemplates: Creates a table template for the paes. These directly interact with specific columns for many-to-many tables from the database. The procedure is very similar to how content from the classes "Classes" folder is made. The only difference is the name that needs to be replaced. The name of the file also does not have to match any table in the database.

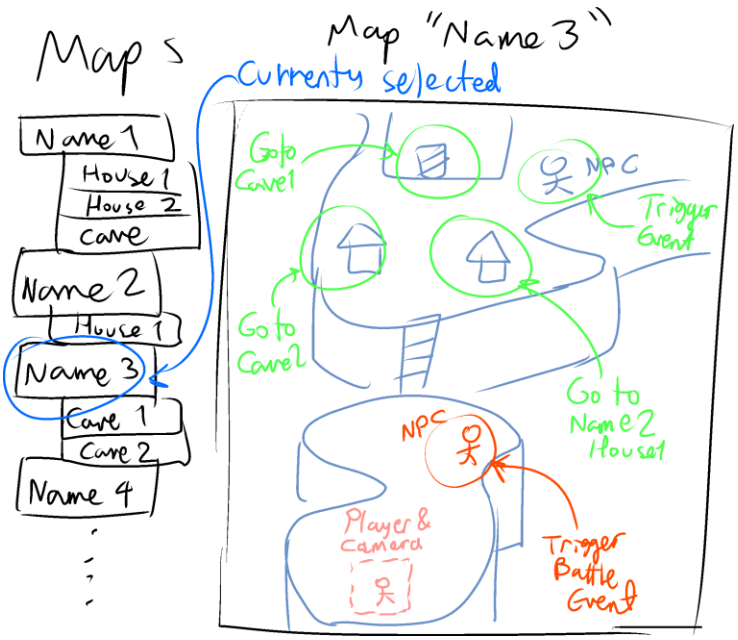
There are several pages and templates that have already been implemented. Use those as examples to support interfacing with more database tables.

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 for the game. You can start anywhere in this section, mechanics wise. One general idea (diagram below) and two high level designs 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. Add any other 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.

The diagram below is a general overview of what the map system interface could look like. The maps can be organized into a directory-like structure. Clicking the buttons, on the left of the diagram, will view the specified map. If you have a design choice that you think is better than this, then implement that instead.



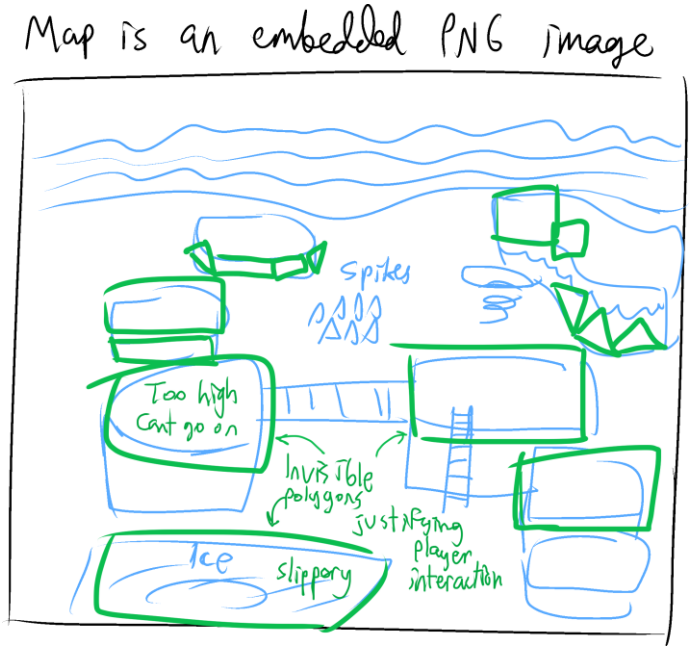
Managing the Data

The important database tables are Map, Event (NPC interactions, cutscenes, moving to other maps, etc.) and Environment. These three objects are all retrieved from the database. However, the tables for these three objects are currently empty. While making data and variables for the application: if you are not planning to manage the three tables, by either populating the table or creating new attributes, then just use C# classes in instead. Make sure your disposable classes have store the same variables, and more, from the its respective table attributes – i.e. The Map table has a Width and Height attribute; your disposable Map class will need a Width and Height attribute as well.

You may need to directly test the map by travelling on it with a playable character, for this application. Collision logic needs to be considered. The player will also be falling, being behind objects, going over and under bridges.

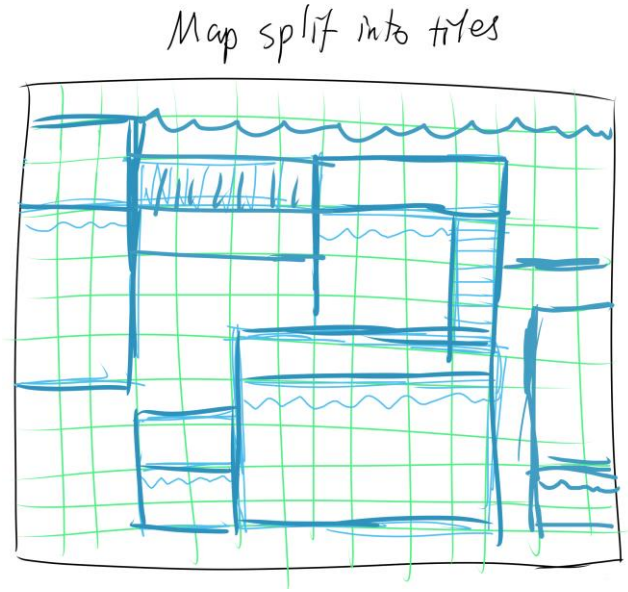
Design Idea 1: Hand-drawn Maps

This design is preferable for the scope of the project. Maps are more fluid, customizeable, and less forced. On the downside, it is a slower and more complex system to manage. If you are looking for challenge to get more experience as a programmer, this would be the one to pick. You do not have to be an artist to test the maps for this design. Just go to Google Images and find a map.



Design Idea 2: Tile-based Maps

This design is less preferable due to the stiffness of the visuals. It is faster and easier to work with. Stiff diagonal tiles may not be worth implementing, for this project. All tiles are squares.



Visual Studio Solution Project: Enemy Battle Simulator

High Level Overview

Like the map builder, this project is also empty. This is supposed to act as a turn-based battle tester for the game. Battle animations, audio, and music are not expected to be implemented. The main component of this project is to set up enemy troops that the player's party will encounter throughout the game. (e.g. three slimes and two dragons, twos slimes and one chimera, etc.)

This application provides two tabs to the user: "Team" and "Enemies".

Please note: "EG" means Enemy Group and "TG" means Team Group

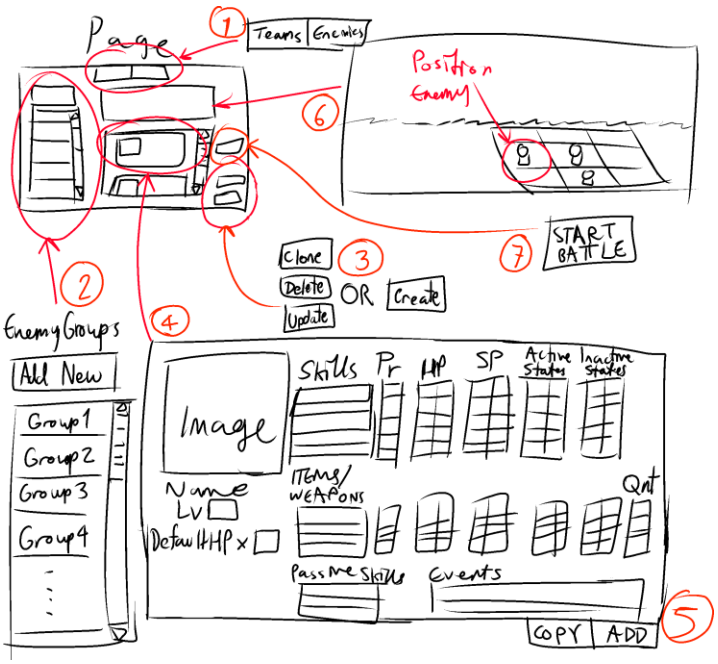
Managing the Data

Almost all tables related to battling are needed. The most important tables are: Enemy, EnemyGroups, and EGSingleEnemy. The high-level details on these tables are specified on the "Enemies" page below. None of the database tables have been populated with actual content yet.

Use disposable C# classes that mimic the attributes for their respective tables. For example: for Player class, refer to the attributes of the Player table, and it's related attachable/many-to-many class tables. For the C# classes, you will need more than just tables attributes. The extra class variables will be a designed based on your decisions.

The Enemies Page

The "Enemies" page will be accessed through the "Enemies" button in a small navigation bar. The page format will be specified on the diagram below:



1. The navigation bar between two options: toggle between the “Teams” and “Enemies” pages. Discard any changes to teams/enemy groups upon traversing through the pages.

2. The list EnemyGroup objects from the database. You can either modify the tables on the database or store the EnemyGroup objects through a File/IO mechanism. If you are choosing the latter, then it is only a temporary method until you decide to use the database.

3. Operations on the selected, or new, EnemyGroup object. As a reference: you can use the UI layout, code format, and operations of the “Database” project on the Visual Studio Solution.

4. The sample layout and attributes of an EGSingleEnemy object to be stored as a table in the database. Again, if you are using File/IO to store information, it will only be a temporary method until you decide to use the database.

   The EGSingleEnemy will have a level (Enemy information, including stats, will be stored on the separate “Enemy” table/C# class) and list of skills/items/weapons, A.K.A. tools they will use. the specific EGSingleEnemy can also have passive skills, but no other equipment. For every tool, the enemy will have conditions or when they will use the tool, to enhance their AI. The AI design is up to you: try to make the AI as smart as you can. Regular enemies will not be intelligent, but boss enemies will have need to use the AI implementations.

5. Each EnemyGroup will have a list of EGSingleEnemy objects (assumed max is 9). The “Add” and “Copy” operations will add an extra EGSingleEnemy to the list. The changes will not be saved until the “Update”, “Create”, or “Clone” button is clicked.

6. Indicates the position of the enemies of a single EnemyGroup, on a 3x3 board, during battle. The left side of the board will have enemies that are positioned to be closer to the the party (i.e. snipers and mages would be on the middle/right side of the board, strategically). These will be stored in the EnemyGroup table/object.

7. Starts the battle with the current input settings. Recommended to prompt the user to save the EnemyGroup to the database or File/IO.

Please note that the EnemyGroup and EGSingleEnemy tables are outdated and may now be incomplete. Feel free to modify them.

The Teams Page

The “Teams” page can be accessed through the “Teams” button in a small navigation bar. The page format will be the exact same thing with two exceptions. There is no “Start Battle” button and the box for specifying players, number 4 on the diagram above, is different.

Each party member will be retrieved from the Player table in the database, just like how EGSingleEnemy will need to read the Enemy table. The database is currently empty, so make test data initialized from C# classes instead. For this application, ignore the list of skills stored in the Player_To_Skill table of the database.

Author's Note: Design inconsistencies will be present on the next paragraph, sorry.

Anything on Teams Page will NOT store/retrieve anything into/from the database. Unlike EnemyGroups and EGSingleEnemies, the teams page is entirely designed for testing purposes only present on this application. A "TeamGroup" and "TGSinglePartyMember" table is not needed in the database, because the party is dictated by the events in the game's Story Mode. Only use File/IO and dispoable C# classes to save and read data from "TeamGroups" and "TGSinglePartyMember".

During Battle Testing

A battle is executed when the user of the application selects "Start Battle" in the "Enemies" tab. A new window will pop up, disabling the host window, or the user will be directed to a new page (Your choice). You could also use the console if you are generally not into designing a user interface. The user of the application can end the battle anytime, by closing the window or hitting a button.

As mentioned at the beginning of the section: Mimic the database by using C# classes. You may need to add more variables on your C# classes. It is highly advised to read the PDF **Battle Flow Chart** in the Documentation section of this repository.

The following list are formulas that are applied during battle: Please see the "Stats" table attributes in page 3.

Physical Standard Damage Formula = (1.5 * pATTACK – 1.25 * pDEFENSE) * MULTIPLIER

- pATTACK = (UsersAtk + UsersWeaponAtkBoost) * UsersAtkBuff
- pDEFENSE = (TargetsDef + PartnerDefBoost) * TargetDefBuff

Magical Standard Damage Formula = Magical Standard: (1.5 * mATTACK – 1.25 * mDEFENSE) * MULTIPLIER

- mATTACK = (UsersMap + UsersWeaponMapBoost) * UsersMapBuff
- mDEFENSE = TargetsMar * TargetMarBuff

MULTIPLIER = EnvironmentBoost * ElementalDamage * CriticalHitDamage * UserSkillPower

- EnvironmentBoost = Based on the environment, certain elemental attacks will be stronger – Values are usually 0.25, 1, and (Ignore this field for now: just set it to 1)
- ElementalDamage = The target's element rate against the tool's element – e.g. An ice attack is used against a target whose ElementRate on ice = 150%; the target will receive 1.5 times more damage
- CriticalHitDamage = If the user lands a critical hit on the target then set it to 3, set to 1 otherwise
- UserSkillPower = A constant for a skill/item's strength (0.5 = weak, 1 = average, 3 = strong, etc.)

Probability of hitting of the target = 95 * (UserTec / TargetSpd) * (UserHitRate / TargetEvasionRate)

Probability for a Critical Hit = 2 * ((UserTec ^ 1.1) / TargetSpd) * (UserCrit / TargetCritEva)

The following stats from user and target are evaluated through values in the database. You'll have to make arbitrary numbers for now to intialize your own classes, stats, players, and enemies. Experiment on different numbers.

There is another factor that has been removed from the formulas. When party members are better friends with each other, their stats slightly increase. This will not be considered for now.

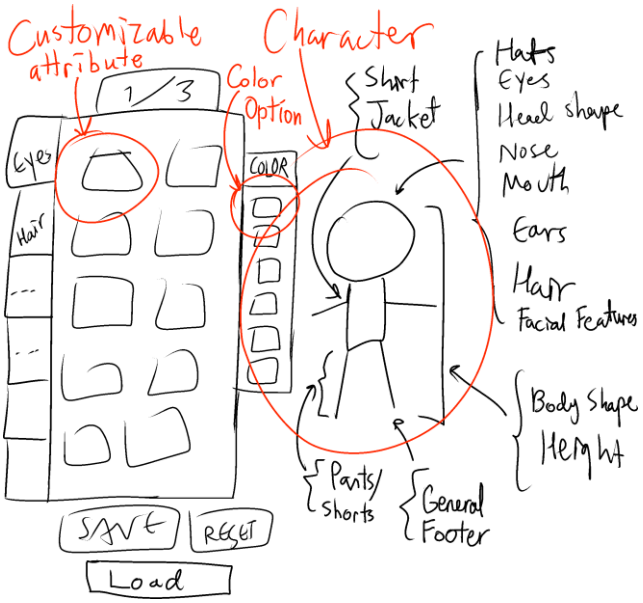
Visual Studio Solution Project: Character Creator

High Level Overview

The RPG will have many sprites and NPCs, so a character creator will be needed. Quadrupedals do not have to be supported in this application. Their sprites may have to be added manually.

You will be playing as your Avatar (the first and main party member). To make the Avatar, a more restricted version this character creator will be used. The restriction only allows customization on the biological facial features, plus the body shape and height. The Avatar will not have customizeable clothes due to how their class armor dictates what they will be wearing. To make the Avatar look less like the NPCs, some of the customizeable attributes will be different. For example, only the Avatar character can have spiky protagonist hair whereas only the NPCs can have another type of hair.

The diagram below indicates a rough idea of what the application might look like.



The list on the diagram does not includes all features. Other accessories may be needed.

Managing the Data

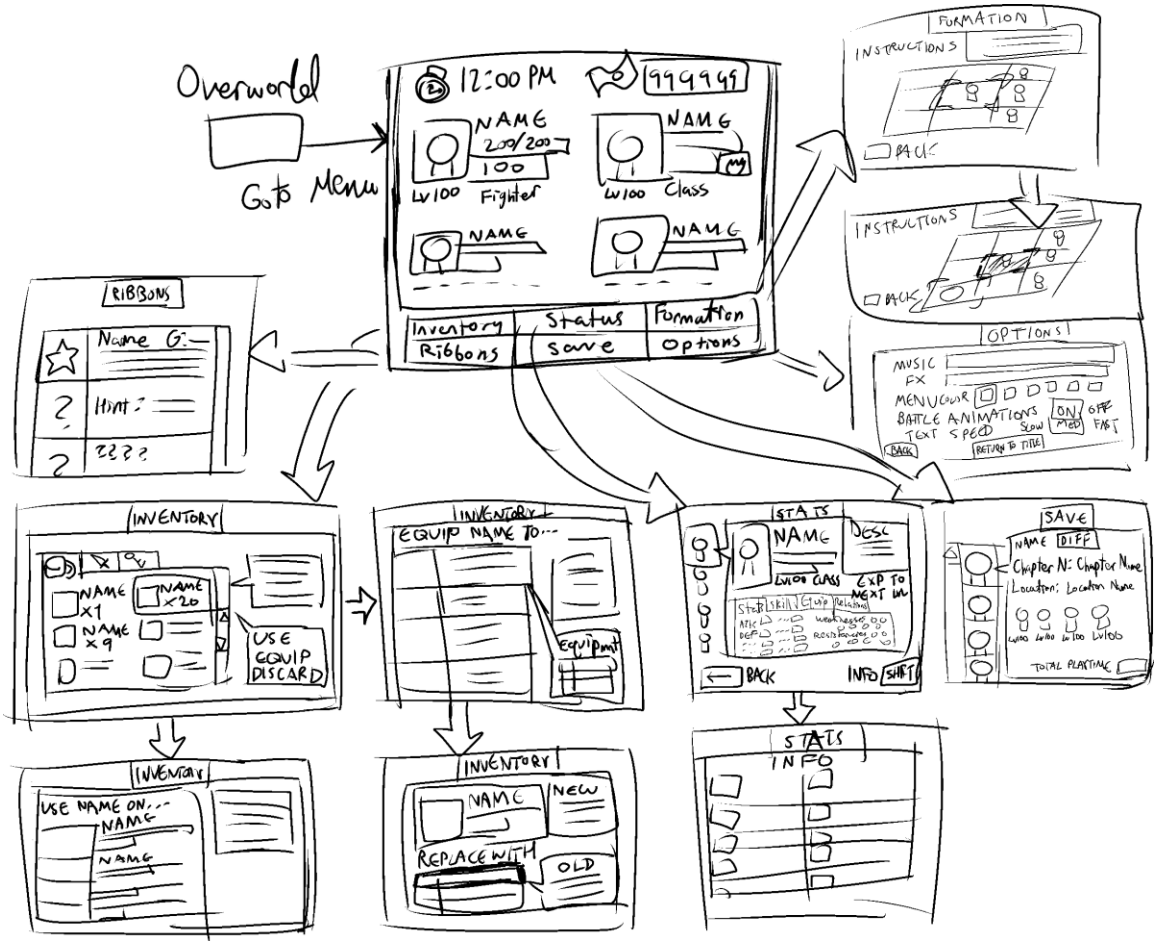
Here are some possible functionailities and requirements for this application:

- Reset should set the default customizable attributes of the character
- The way to read and write the image list of customizable attributes will be up to you (different types of hats, etc.), whether it be many image blobs in the database, many text files, or some other better way
- Managing the data (read/write) for the character and color options will be decided by you as well
- A User Interface is highly recommended for this application
- Make sure this application can load the same character into spritesheets – Multiple angles of the same customizable attribute will need to be made
- Do not gender-lock (stuff like “Select Male or Female”, etc.) the character creator: you will be doing more unnecessary work than needed

None of these reqiurements, except for the gender-locking part, are absolute. These functionailities/requirements only suggest what could be done for the character creator.

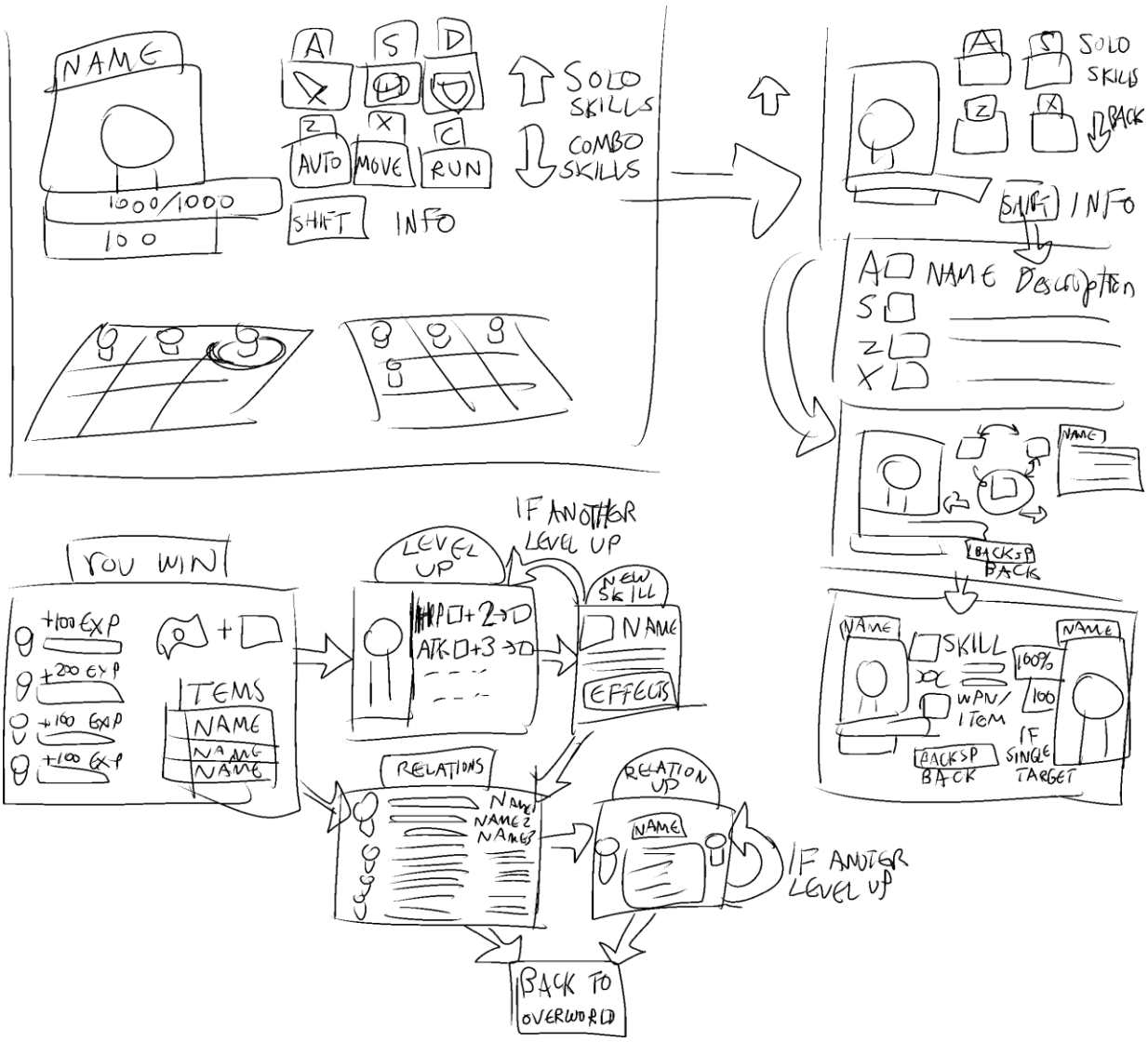
General User Interface Plans

UI Plan for Menus/Inventory



UI Plan for Battling

See **Battle Flow Chart** in the Documentation section of the repository for more information



The Entire Project's Current State

What is Already Completed

The only sections that have been worked on are the portions that are the closest to handling the database.

- The Raw Database (In SQLite program):
  - All planned tables are set up: General format and design is confirmed for most part
- The Database Interface (Visual Studio):
  - The Player, Achievement, OtherLists, and Skill pages are fully functional, and can be clicked, from the navigation bar
  - Almost all class templates have been implemented
  - The planned table templates have been implemented
  - All needed utilities, so far, have been implemented
- Anything else that comes from the previous commit messages
- Documentation is up to date

What Needs to be Done

Everything else: a few items may be listed, but there is lots of work to do in each of them. This list is not extensive.

- The Raw Database (In SQLite program):
  - More potential many-to-many relation tables need to be added
  - Cleanup junk, such as unneeded constraints and attributes (low priority)
- The Database Interface (Visual Studio):
  - Creating pages that can interface with the remaining regular tables (Under Classses Folder)
  - The class template to interface with the PassiveEffect table: Interfacing with the State and PassiveSkill tables depend on this
- These projects have nothing in them, and will need to be developed
  - Mapbuilder
  - CharacterCreator
  - EnemyBattleSimulator
- Undecided Conceptual Stuff (Any suggestions are welcome)
  - Conjuring up a way to store sprites into the database without execessive manual labour for drawing and animating said sprites
  - How to properly store tool animations and team combos
  - How to properly design projectiles to handle tool animations and combos
  - Handling Events: All types of NPC/Environment interaction and managing special types of tools
- Better version control and improved design for the SQLite database (low priority)
- Potential improvements on the TableTemplates folder (low priority)

Final Comments (From Raf)

For conveience, make sure that all projects (or at least the one you're working on), and most files, have the following import/using statements:

- System.Date.SQLite
- Database.Utilities.SQLDB
- Database.AccessDB

Again, what is said about the CharacterCreator, BattleEnemySimulator, and MapBuilder in this document is not definite. The design implementation sections about those files are just suggestions on how I to approach them. You are free to make those files your own convenient way.

This may interfere with compability, so I'll mention this: Interactive Database, and MapBuilder use a different set of Visual Studio libraries and APIs, compared to BattleEnemy Simulator and CharacterCreator. The Interactive Database and MapBuilder use a Windows Foundation Presentation format (WPF), whereas the other two use Windows Forms Projects (WinForms). WinForms is simpler than WPF but provides less features and may be incompatible with a couple of the files in Database.Utilities. If you're having trouble with managing the limitations of WinForms, then let me know, and I'll tell you how to switch the project to WPF.

When you're working on the project, please do NOT push to the master branch.

This project is not meant to be competitive in any way. I will take almost full responsibility of any accidents against the project. I'm not a fan of micromanaging, but I might have to, for the interactive and rawdatabase sections. If you think anyone else might be interested in helping with the development of this project, then feel free to invite them over.

If you have any questions, or complaints on my handwriting, then contact me through my Discord: Mr_Per50na#2496