# Aetalag Tabletop RPG Asset Manager
# for UTM CSCI 352

Chase Perritt and Danny Lillard

**Abstract**

**"Aetalag Tabletop RPG Asset Manager" is a modular asset management system for tabletop roleplaying games. The targets for "Aetalag Tabletop RPG Asset Manager" are those who wish to "make it easy" when working with TTRPG assets: dungeon masters and players alike. The project is currently in the development phase, about halfway through now.**

## 1. Introduction

"Aetalag Tabletop RPG Asset Manager" is a modular asset management system for tabletop roleplaying games built to assist the user in TTRPG character building and upkeep via gathering information from online databases and organizing the information in an accessible and dynamic manner. Our target audience is, of course, TTRPG players. We expect the target audience to benefit from "Aetalag Tabletop RPG Asset Manager" due to its ability to better keep track of TTRPG elements in the heat of a game: initiative scores, health points, etc.

### 1.1. Background

Tabletop RPGs, or role-playing games, are an increasingly common hobby in modern society that involves creating and acting out characters in fictional scenarios. One game is typically called a "session," and multiple sessions are called a "campaign." One such example of a tabletop role-playing game is the widely popular Dungeons and Dragons (5th Edition). D&D [5e] sessions tend to take place in high fantasy, magical settings, however, the rules governing them are quite real and concrete. Some terms that would be useful to know are as follows:

Sheet: A page in a character's portfolio, typically used to track their stats, equipment, and/or spells.

Ability Scores: 6 integer values while determine what the character is skilled in, and not. The scores are: Strength, Dexterity, Constitution, Intelligence, Wisdom, and Charisma. These values are intergeral to D&D, and effects all parts of the game. Based on ability scores are modifiers, modifers are calculated by: (ability score - 10) / 2

Dice: D&D is ruled by seven dice, inorder of sides: 4, 6, 8, 10, 12, and 20 sided dice. The player is expected to use these dice in real life. Within the program, the syntax to represent dice is: d-sideNum.

Hit Points: Hit points determine how much damage a player can take. It is calculated by xRy + CONmod + misc. Where xRy is a dice roll, with x being the roll of the dice and y being the max of that die.

Initiative: This is an integer that is used to determine when an entity is allowed to do actions in the list of all other entities. For example, a ghost with initiative 3 goes after a person with initiative 4.

Proficiency Bonus: Proficiency Bonus is reliant upon a character's total level and determines a level of experience with a skill, weapon, method of spellcasting, or other field. Calcuated by: (x / 4) + 1 rounded up, where x is level.

Armor Class represents how difficult it is to hit a given character and is reliant upon the character's equipped armor, their Dexterity modifier, and any secondary or miscellaneous modifiers. Calculated by: DEXmod + Secondarymod + misc.

Saving Throws: Saving Throws represent a character's attempt to resist a spell, trap, poison, disease, or other threat. Calculated by: Xmod + misc. + proficiency (if applicable).

### 1.2. Challenges

A particular challenge in mapping out both the ruleset of D&D 5e is that, though we have access to source reference documents and particular websites that compile all of the information from several source books and pdfs, there is simply too much information out there. This is a gargantuan undertaking, and a simple solution would simply be to add less content than we hope: to cut out the equipment databases and alternate features in favor of the minimum required amount.

## 2. Scope

Completion of the project entails the existence of a D&D 5e character maintenance tool that rivals traditional pen and paper sheets in terms of quality of life assurance and ease of access to information. While this software may not render the pen and paper style obsolete, it should prove valuable as a beginner's introductory tool to tabletop roleplaying or as a reference for rules and monster stats for Dungeon Masters. As for stretch goals we would like to give the user to save their sheet at a .pdf or .txt, and have an automatic initiative roller.

| Use Case ID | Use Case Name | Primary Actor | Complexity | Priority |
|---|---|---|---|---|
| 1 | Closing the Program | User | Low | 1 |
| 2 | Adding a New Entity to “HP Tracker” | User | Low | 1 |
| 3 | Deleting an entity from “HP Tracker” | User | Low | 1 |
| 4 | Editing an entity in “HP Tracker” | User | Low | 1 |
| 5 | Adding a new entity to “Initiative Tracker” | User | Low | 1 |
| 6 | Deleting an entity from “Initiative Tracker” | User | Low | 1 |
| 7 | Editing an entity in “Initiative Tracker” | User | Low | 1 |
| 8 | Sorting entities in ”Initiative Tracker” | User | Low | 2 |
| 9 | Creating a Blank Character Sheet | User | Medium | 2 |
| 10 | Creating a Sheet With “Make it Easy” | User | Low | 2 |
| 11 | Editing Sheet Values | User | Low | 2 |
| 12 | Changing Drop Down Choices | User | Low | 2 |
| 13 | Saving a Sheet | User | Low | 3 |
| 14 | Saving a Sheet As | User | Low | 3 |
| 15 | Opening an Old Sheet | User | Low | 3 |

TABLE 1. USE CASE TABLE

## 2.1. Requirements

Most requirements for “Aetalag Tabletop RPG Asset Manager” are derived directly from the standard pen and paper style of managing a character. Each requirement listed below deserves to be here because it makes the player’s life easier, whether that requirement does something better than pen or paper, or adds new functionality.

### 2.1.1. Functional.

- Users are able to create new characters - By clicking on the “New Sheet” button on the “Aetalag Tabletop RPG Asset Manager” window the user is able to create a new character from scratch
- Users are able to open old sheets - By clicking the “Open Sheet” button on the “Aetalag Tabletop RPG Asset Manager” window the user is able to open a saved sheet
- User is able to use the “Make it Easy” functionality to create a new character - By selecting the “Make it Easy” button on the “Aetalag Tabletop RPG Asset Manager” the user is able to quickly make a character
- Users are able to keep track of the HP of entities - Using the “HP Tracker” subsection of the “Aetalag Tabletop RPG Asset Manager” users should be able to add new entities, edit their values and name, and remove entities from the subsection
- Users are able to keep track of the initiative scores of entities - Using the “Initiative Tracker” subsection of the “Aetalag Tabletop RPG Asset Manager” users are able to track the initiative of entities. They should be able to add entities, sort the entities, name the entities, and delete them as well
- Users are able to request more information on the HP, AC, Speed, Saves, and Initiative blocks from the “Character Sheet” window - By clicking on the dog ears present on these information block users are able to request more information about the blocks and edit information inside
- Users need to have a save file for each character - analogous to a sheet of paper this will keep track of all his or her last saved values for said character
- Users need to be able to change the values of a character - through text boxes, he or she will be able to alter the character’s values, after unlocking said boxes

### 2.1.2. Non-Functional.

- Speed of loading and saving - previous saves and the act of saving must load quickly for the user, in less than two seconds

## 2.2. Use Cases

| | |
|---|---|
| Use Case Number: | 1 |
| Use Case Name: | Closing the Program |
| Description: | The use wants to close out of the program. |

- User selects the “x” button on “Aetalag Tabletop RPG Asset Manager” window.
- The program closes.

| | |
|---|---|
| Use Case Number: | 2 |
| Use Case Name: | Adding a new entity to “HP Tracker” |

Description: User wants to create a new entity to the “HP Tracker” section on the “Aetalag Tabletop RPG Asset Manager” window. Allowing them to quickly view HP.

- User Selects the “+” button present to the top right of the “HP Tracker” section of the “Aetalag Tabletop RPG Asset Manager” window.
- A new HP bar appears with the fillable fields: “Enter Name” and “0/0” as current and max health, which each can be edited.
- User fills out these fields.

Termination Outcome: User has a new filled HP tracker.

Use Case Number: 3

Use Case Name: Deleting an entity from “HP Tracker”

Description: User wants to delete an entity from the “HP Tracker” section on the “Aetalag Tabletop RPG Asset Manager” window.

- User selects the “X” button present in the block with that “HP Tracker” object.
- That item is removed from the list.

Termination Outcome: User has one less item in the “HP Tracker”.

Use Case Number: 4

Use Case Name: Editing an entity in “HP Tracker”

Description: User wants to edit an entity from the “HP Tracker” section on the “Aetalag Tabletop RPG Asset Manager” window.

- User selects the the item they want to select, currentHP, MaxHP, or Name, present in the block with that “HP Tracker” object.
- The user types in the new value.
- The value is changed.

Termination Outcome: User has a new value.

Use Case Number: 5

Use Case Name: Adding a new entity to “Initiative Tracker”

Description: User wants to create a new entity to the “Initiative Tracker” section on the “Aetalag Tabletop RPG Asset Manager” window. Allowing them to track that entities initiative

- User selects the “+” button present to the top right of the “Initiative Tracker” section of the “Aetalag Tabletop RPG Asset Manager” window.
- A new initiative tracker appears with the fillable fields: “Enter Name” and “0” as the initiative, which each can be edited.
- User fills out these fields.

Termination Outcome: User has a new filled initiative tracker.

Use Case Number: 6

Use Case Name: Deleting an entity from “Initiative Tracker”

Description: User wants to delete an entity from the “Initiative Tracker” section on the “Aetalag Tabletop RPG Asset Manager” window.

- User Selects the “X” button present in the block with that “Initiative Tracker” object.
- That item is removed from the list.

Termination Outcome: User has one less item in the “Initiative Tracker”.

Use Case Number: 7

Use Case Name: Editing an entity in “Initiative Tracker”

Description: User wants to edit an entity from the “Initiative Tracker” section on the “Aetalag Tabletop RPG Asset Manager” window.

- User Selects the the item they want to select, Initiative or Name, present in the block with that “Initiative Tracker” object.
- The user types in the new value.
- The value is changed.

Termination Outcome: User has a new value.

Use Case Number: 8

Use Case Name: Sorting entities in ”Initiative Tracker”

Description: User wants to sort entities in the “Initiative Tracker” section on the “Aetalag Tabletop RPG Asset Manager” window.

- User fills in the Initiative of each item with a value.
- The user clicks the “Sort” button present on the “Initiative Tracker”
- The values are sorted, low to high.

| | |
|---|---|
| Termination Outcome: | User has a sorted list. |
| Use Case Number: | 9 |
| Use Case Name: | Creating a Blank Character Sheet |
| Description: | The user wants to create a new character from scratch (without using the “Make It Easy” functionality). |
| | • User Selects “New Sheet” button on the “Aetalag Tabletop RPG Asset Manager” window. |
| | • A blank sheet is opened in a new window. |
| | • The user will then be able to fill out any blank fields in the sheet. |
| Termination Outcome: | User has a new filled sheet. |
| Alternative: | Sheet Manager window is already open |
| | • User Selects “New” on the “Aetalag Tabletop RPG Asset Manager” window. |
| | • A blank sheet is opened in a new window. |
| | • The user will then be able to fill out any blank fields in the sheet. |
| Use Case Number: | 10 |
| Use Case Name: | Creating a Sheet With “Make it Easy” |
| Description: | The user wants to create a new sheet, the easy way. |
| | • User Selects “Make it Easy” button on the “Aetalag Tabletop RPG Asset Manager” window. |
| | • The “Make it Easy” window is opened. |
| | • The user selects the drop down menu items, and fills in the values that they want. |
| | • The user names a save file in the windows save file as dialog. |
| Termination Outcome: | A sheet is created & saved with the users selected values. |
| Use Case Number: | 11 |
| Use Case Name: | Editing Sheet Values |
| Description: | The user wants to edit values in their sheet. This use case is a general catch for all the different things that can be edited, there are far too many to have a case for each. The rule is that if the data is encased in a text box it can be edited. |
| | • User Selects a TextBox in their “Sheet” |
| | • The user types in a new value. |
| Termination Outcome: | The value is changed. |
| Use Case Number: | 12 |
| Use Case Name: | Changing Drop Down Choices |
| Description: | The user wants to change a value in a drop down menu. |
| | • User Selects the drop down. |
| | • The user selects a value in the drop down. |
| Termination Outcome: | The value is changed. |
| Use Case Number: | 13 |
| Use Case Name: | Saving a Sheet |
| Description: | The user wants to save their sheet. |
| | • User Selects “Save” button on “Sheet” window. |
| | • The sheet values are complied and saved. |
| Termination Outcome: | The .aetalag file is updated. |
| Alternative: | The sheet is not saved at all. |
| | • The “Save As” protolog runs. |
| Use Case Number: | 14 |
| Use Case Name: | Saving a Sheet As |
| Description: | The user wants to save a sheet under a new name. |
| | • User Selects “Save As” button on “Sheet” window. |
| | • A windows file explorer window is opened. |
| | • User saves their file as a .aetalag file. |
| | • The sheet values are complied and saved. |
| Termination Outcome: | The .aetalag file is created. |
| Use Case Number: | 15 |
| Use Case Name: | Opening an Old Sheet |
| Description: | The user wants to open a sheet they have been already working on. |
| | • User Selects “Open Sheet” button on the “Aetalag Tabletop RPG Asset Manager” window. |
| | • A windows file explorer screen is opened. |
| | • The user navigates to the .aetalag file they wish to open. |
| Termination Outcome: | The sheet is loaded. |

## 2.3. Interface Mockups

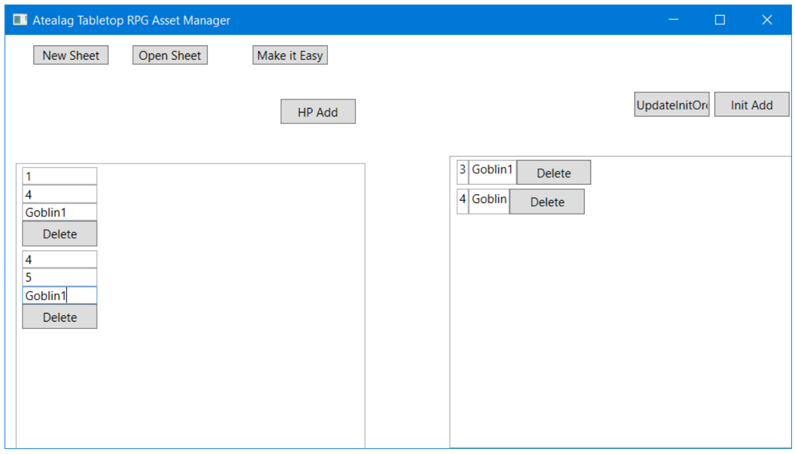


Figure 1. GUI for the Main Window.

Here we have the "Main Window" the first window the user is greeted with. To the left, we have a HP tracker, where the user can add, delete and edit entities. To the right we have a initative tracker, where the user can add, delete sort, and edit the initiative of entities. Above these objects we have three buttons. "New Sheet" which opens a blank Character Sheet, "Open Sheet" which opens a saved sheet, and "Make It Easy" which opens the Make It Easy window.

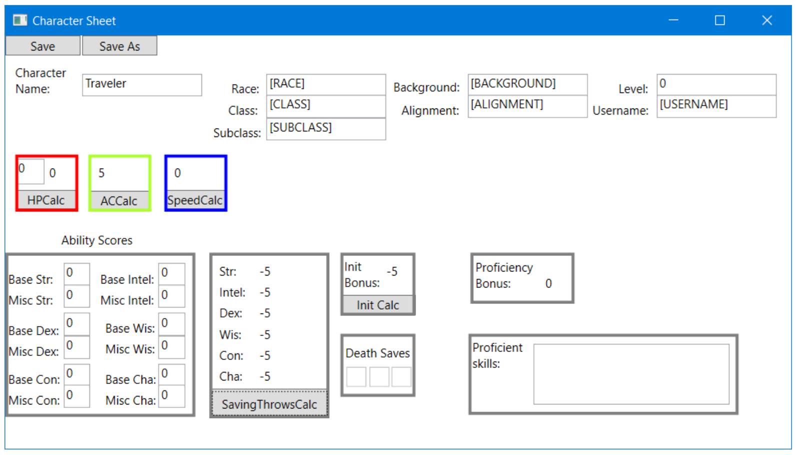
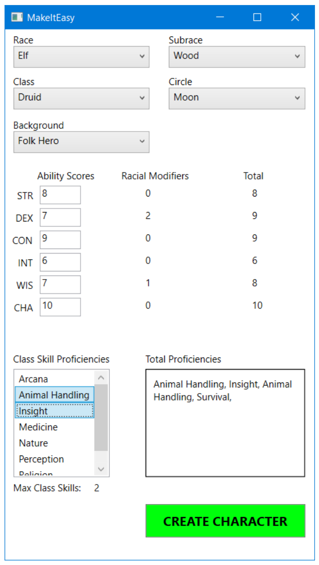


Figure 2. GUI for a Character Sheet.

This GUI is where the user interacts with his or her character. The Save and Save As buttons work as expected. Following down the GUI we have 8 text boxes, all labeled, which are simple text fields for the user to fill in. With Make It Easy most of these fields are already filled in. Next we have the HP, AC, and Speed boxes. When each of these objects buttons are clicked a new window will appear showing the calculation for their values. These windows can be found in the "Calcs" GUI figure. Next we have an ability score display, with simple text boxes that are coupled to Publisher objects. Following is the Saving Throws, and Initiative displays, which have windows and subscribers attatched to them as well. Finishing out we have death saves, which is only present on the XAML, proficiency bonus, a simple box coupled to the level number, and Proficient Skills, a simple textbox.

Make It Easy features dynamic Combo boxes that changed based on the other boxes values. All of the values from these five boxes are fed into the Character Sheet Window. Following we have Ability Scores, fed in from the user, racial bonuses, fed in from databases, and total score, calculated from the bonuses and the user input. Next we have proficiencies, created by the user's background and their own choices. The Create Character button will prompt the user to save the sheet, then



MakeItEasy

Race: Elf
Subrace: Wood
Class: Druid
Circle: Moon
Background: Folk Hero

| | Ability Scores | Racial Modifiers | Total |
|---|---|---|---|
| STR | 8 | 0 | 8 |
| DEX | 7 | 2 | 9 |
| CON | 9 | 0 | 9 |
| INT | 6 | 0 | 6 |
| WIS | 7 | 1 | 8 |
| CHA | 10 | 0 | 10 |

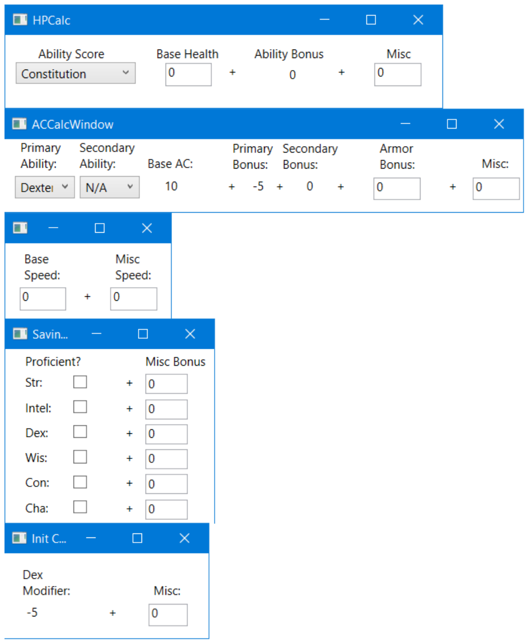
Class Skill Proficiencies: Arcana, Animal Handling, Insight, Medicine, Nature, Perception, Religion

Total Proficiencies: Animal Handling, Insight, Animal Handling, Survival,

Max Class Skills: 2

Figure 3. GUI for a Make It Easy character.

loads a sheet with their respective values.



HPCalc

Ability Score: Constitution | Base Health: 0 + Ability Bonus: 0 + Misc: 0

ACCalcWindow

Primary Ability: Dexte | Secondary Ability: N/A | Base AC: 10 + Primary Bonus: -5 + Secondary Bonus: 0 + Armor Bonus: 0 + Misc: 0

Base Speed: 0 + Misc Speed: 0

Savin...

| Proficient? | | | Misc Bonus |
|---|---|---|---|
| Str: | ☐ | + | 0 |
| Intel: | ☐ | + | 0 |
| Dex: | ☐ | + | 0 |
| Wis: | ☐ | + | 0 |
| Con: | ☐ | + | 0 |
| Cha: | ☐ | + | 0 |

Init C...

Dex Modifier: -5 + Misc: 0

Figure 4. These are the windows that are present from the calculation button on the HP, AC, and Speed boxes, the saving throws display and the initiative calculation.

Here are all the calculation windows from the Character sheet. With HP, AC, Speed, Saving Throws, and initative all accounted for. All values are dependent on ability score modifiers and such are coupled to Subscribers. The windows with a drop down menu allow the user to change their ability score coupling, and will update the new value. All values with a box around them can be edited by the user.

## 3. Project Timeline

1) Requirements:
   We planned to have all of our requirements established by the submission and review of the first project proposal draft on the 10th of February, 2020. This goal was achieved and in line with hand in.
2) Design:
   We planned for our major design decisions to be fleshed out by March 12th. This goal was achieved, the design was later changed during development due to faulty design.
3) Implementation:
   We hoped to have finished implementing the primary, promised functionality of "Aetalag Tabletop RPG Asset Manager" by April 23rd. This goal was not completely achieved, much of the original, ambitious functionality was not present, with the bare essentials being present.
4) Verification:
   We hoped to have fixed all bugs and leaks by April 23rd. Currently no bugs are known in the program.
5) Maintenance:
   After the end of April, any additional functionality and systems could be added at any point in the future.

Our time budgeting was less then ideal, a combination of life events other school work and a few crises made it harder than average to budget time. But this is usually the hardest part of any project be that software or otherwise.

## 4. Project Structure

When the program is opened the user is greeted by the Asset Manager window. This window is used to: track HP, track iniative, open sheets, and create new sheets, using "New Sheet" or "Make It Easy."
Make it Easy is a key feature of our program. It allows users to easily and quickly create a basic character. The window of this feature follows from the Asset Manager window.
Following Make it Easy, open sheet and new sheet we have the Sheet window. This window displays all information needed for a basic character.
In some of the values present on the Sheet window there are window that can be opened which will display the calculation for that value.

### 4.1. UML Outlines

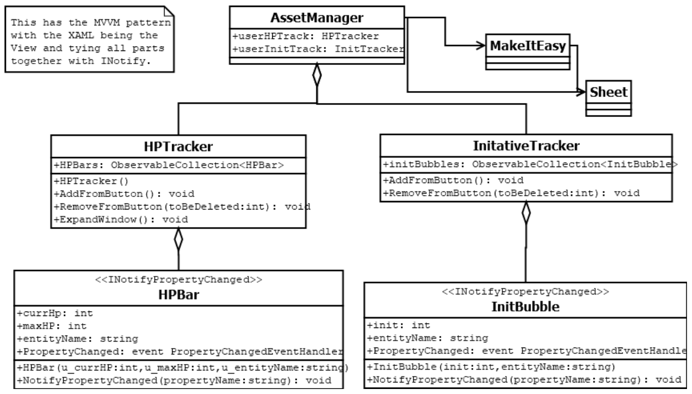

This has the MVVM pattern with the XAML being the View and tying all parts together with INotify.
AssetManager
+userHPTrack: HPTracker
+userInitTrack: InitTracker
MakeItEasy
Sheet
HPTracker
+HPBars: ObservableCollection<HPBar>
+HPTracker()
+AddFromButton(): void
+RemoveFromButton(toBeDeleted:int): void
+ExpandWindow(): void
InitativeTracker
+initBubbles: ObservableCollection<InitBubble>
+AddFromButton(): void
+RemoveFromButton(toBeDeleted:int): void
<<INotifyPropertyChanged>>
HPBar
+currHp: int
+maxHP: int
+entityName: string
+PropertyChanged: event PropertyChangedEventHandler
+HPBar(u_currHP:int,u_maxHP:int,u_entityName:string)
+NotifyPropertyChanged(propertyName:string): void
<<INotifyPropertyChanged>>
InitBubble
+init: int
+entityName: string
+PropertyChanged: event PropertyChangedEventHandle
+InitBubble(init:int,entityName:string)
+NotifyPropertyChanged(propertyName:string): void

Figure 5. UML for the Asset Manager window.

This UML is for the first screen the user sees, the Asset Manager Window. This screen needs to have a dynamic list of objects, where the list, and the object inside the list, can be edited. Following the order of the UML the classes; on the bottom, the HPBar and InitBubble are the objects inside the list. They are dynamically edited using the <<INotifyPropertyChanged>>class present in Visual Studios. These two objects act as the "Model" for the MVVM pattern. Next, the HPTracker and InitiativeTracker hold the list of objects. They are the "ViewModel" for the MVVM pattern, defining logic for working with the list. Not present in the class is the XAML mark up that allows the user to see, and edit the Model and work with the View Model being the "View" for the MVVM pattern. Following, we have the AssetManager class, which is a container class to hold the parts of this class. The arrows to the "Sheet" and "MakeItEasy" classes are to show that the a sheet or Make It Easy can be accessed from this window.

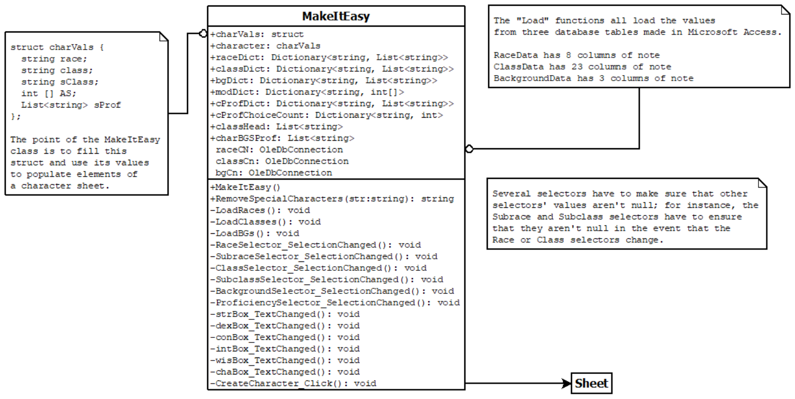

struct charVals {
string race;
string class;
string sClass;
int [] AS;
List<string> sProf
};
The point of the MakeItEasy class is to fill this struct and use its values to populate elements of a character sheet.
MakeItEasy
+charVals: struct
+character: charVals
+raceDict: Dictionary<string, List<string>>
+classDict: Dictionary<string, List<string>>
+bgDict: Dictionary<string, List<string>>
+modDict: Dictionary<string, int[]>
+cProfDict: Dictionary<string, List<string>>
+cProfChoiceCount: Dictionary<string, int>
+classHead: List<string>
+charBGSProf: List<string>
raceCN: OleDbConnection
classCn: OleDbConnection
bgCn: OleDbConnection
+MakeItEasy()
+RemoveSpecialCharacters(str:string): string
-LoadRaces(): void
-LoadClasses(): void
-LoadBGs(): void
-RaceSelector_SelectionChanged(): void
-SubraceSelector_SelectionChanged(): void
-ClassSelector_SelectionChanged(): void
-SubclassSelector_SelectionChanged(): void
-BackgroundSelector_SelectionChanged(): void
-ProficiencySelector_SelectionChanged(): void
-strBox_TextChanged(): void
-dexBox_TextChanged(): void
-conBox_TextChanged(): void
-intBox_TextChanged(): void
-wisBox_TextChanged(): void
-chaBox_TextChanged(): void
-CreateCharacter_Click(): void
The "Load" functions all load the values from three database tables made in Microsoft Access.
RaceData has 8 columns of note
ClassData has 23 columns of note
BackgroundData has 3 columns of note
Several selectors have to make sure that other selectors' values aren't null; for instance, the Subrace and Subclass selectors have to ensure that they aren't null in the event that the Race or Class selectors change.
Sheet

Figure 6. UML for the Make It Easy window.

The Make It Easy class is a massive class that compiles databases and loads values for a character. These values are then fed into the “Character” charVals variable. That character is then used to create a save file that opens a sheet.

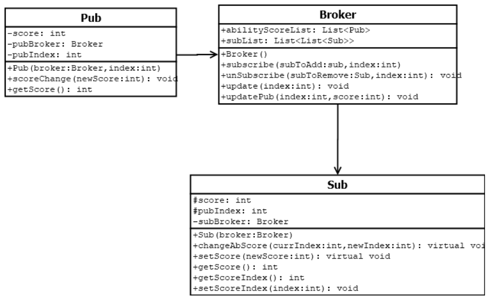

Pub
-score: int
-pubBroker: Broker
-pubIndex: int
+Pub(broker:Broker,index:int)
+scoreChange(newScore:int): void
+getScore(): int
Broker
+abilityScoreList: List<Pub>
+subList: List<List<Sub>>
+Broker()
+subscribe(subToAdd:sub,index:int)
+unSubscribe(subToRemove:Sub,index:int): void
+update(index:int): void
+updatePub(index:int,score:int): void
Sub
#score: int
#pubIndex: int
-subBroker: Broker
+Sub(broker:Broker)
+changeAbScore(currIndex:int,newIndex:int): virtual void
+setScore(newScore:int): virtual void
+getScore(): int
+getScoreIndex(): int
+setScoreIndex(index:int): void

Figure 7. UML for the Publisher Subscriber design pattern.

This is the Publisher-Subscriber pattern we are using. The “Pub” objects are attached to ability scores. While the “Sub” objects are attached to any object that is dependent on an ability score. The “Broker” object works between the Pub and Sub, dynamically updating Subs and also subscribing/unsubscribing Subs to their respective Pubs.

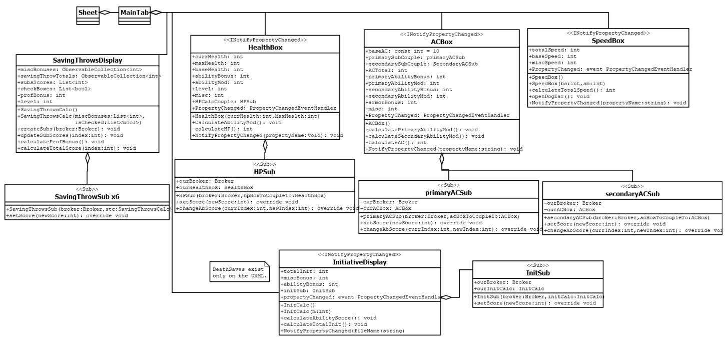

Sheet
MainTab
SavingThrowsDisplay
+miscBonuses: ObservableCollection<int>
+savingThrowTotals: ObservableCollection<int>
+subsScores: List<int>
+checkBoxes: List<bool>
-profBonus: int
+level: int
+SavingThrowsCalc()
+SavingThrowsCalc(miscBonuses:List<int>, isChecked:List<bool>)
+createSubs(broker:Broker): void
+updateSubScores(index:int): void
+calculateProfBonus(): void
+calculateTotalScore(index:int): void
<<Sub>>
SavingThrowSub x6
+SavingThrowsSub(broker:Broker,stc:SavingThrowsCalc)
+setScore(newScore:int): override void
<<INotifyPropertyChanged>>
HealthBox
+currHealth: int
+maxHealth: int
+baseHealth: int
+abilityBonus: int
+abilityMod: int
+level: int
+misc: int
+HPCalcCouple: HPSub
+PropertyChanged: PropertyChangedEventHandler
+HealthBox(currHealth:int,MaxHealth:int)
+CalculateAbilityMod(): void
-calculateHP(): int
+NotifyPropertyChanged(propertyName:void): void
<<Sub>>
HPSub
+ourBroker: Broker
+ourHealthBox: HealthBox
+HPSub(broker:Broker,hpBoxToCoupleTo:HealthBox)
+setScore(newScore:int): override void
+changeAbScore(currIndex:int,newIndex:int): override void
<<INotifyPropertyChanged>>
ACBox
+baseAC: const int = 10
+primarySubCouple: primaryACSub
+secondarySubCouple: SecondaryACSub
+ACTotal: int
+primaryAbilityBonus: int
+primaryAbilityMod: int
+secondaryAbilityBonus: int
+secondaryAbilityMod: int
+armorBonus: int
+misc: int
+PropertyChanged: PropertyChangedEventHandler
+ACBox()
+calculatePrimaryAbilityMod(): void
+calculateSecondaryAbilityMod(): void
-calculateAC(): int
+NotifyPropertyChanged(propertyName:string): void
<<INotifyPropertyChanged>>
SpeedBox
+totalSpeed: int
+baseSpeed: int
+miscSpeed: int
+PropertyChanged: event PropertyChangedEventHandler
+SpeedBox()
+SpeedBox(bs:int,sm:int)
+calculateTotalSpeed(): int
+openDogEar(): void
+NotifyPropertyChanged(propertyName:string): void
<<Sub>>
primaryACSub
-ourBroker: Broker
-ourACBox: ACBox
+primaryACSub(broker:Broker,acBoxToCoupleTo:ACBox)
+setScore(newScore:int): override void
+changeAbScore(currIndex:int,newIndex:int): override void
<<Sub>>
secondaryACSub
-ourBroker: Broker
-ourACBox: ACBox
+secondaryACSub(broker:Broker,acBoxToCoupleTo:ACBox)
+setScore(newScore:int): override void
+changeAbScore(currIndex:int,newIndex:int): override void
DeathSaves exist only on the UXML.
<<INotifyPropertyChanged>>
InitiativeDisplay
+totalInit: int
+miscBonus: int
+abilityBonus: int
+initSub: InitSub
+propertyChanged: event PropertyChangedEventHandler
+InitCalc()
+InitCalc(m:int)
+calculateAbilityScore(): void
+calculateTotalInit(): void
+NotifyPropertyChanged(fileName:string)
<<Sub>>
InitSub
+ourBroker: Broker
+ourInitCalc: InitCalc
+InitSub(broker:Broker,initCalc:InitCalc)
+setScore(newScore:int): override void

Figure 8. UML for the Main Tab window.

Here we have the Main Tab uml, this is the window which the player interacts with after the the Asset Manager and Make It Easy. Beginning at the top we have a “Sheet” class. This class holds logic for creating new tabs and opening files, it also holds the “MainTab.” Next we have MainTab, a container class for all the objects on the Main Tab. Following from the Main Tab object we have the displays for the data. These objects display the data for users, and provide logic for calculating the score of their displays. Next we have <<Sub>>objects, these couple their display objects to a publisher.

### 4.2. Design Patterns Used

Firstly, the MVVM design pattern is used in order to dynamically update the HP and Initiative Tracker on the “Atealag Tabletop RPG Asset Manager” window. This design pattern is a perfect solution to our problem of needing to dynamically update a list of elements. Secondly, we use a publisher-subscriber design pattern to dynamically update stats that work off of ability scores. Making it such that any change to an ability score will automatically update all stats dependent on it. And also allows said stats to quickly decouple from an ability score and couple to another.

The MVVM design pattern is present on the “AssetManager” UML diagram. Where the HPBar and InitBubble are the Views, the HP and Initative Tracker are the View Models and the GUI provided by XAML markup is the view. The Pub-Sub Pattern is present on the “Pub-Sub” UML diagram and on the “Main Tab” where any class marked <<Sub>>is a child of the Sub class and is thus is a subscribed to a publisher.

## 5. Results

As of handin four we have designed the project, both in UML and in GUI. Now all is needed is to begin development. As of handin five we have begun development. Right now we are about halfway through. Here at the end of the project we were able to deliver a minimum requirments build of the project. Fufilling all of our most basic desired functionality. Retrospectively this project was much harder then origonally expected, a combination of becoming familiar with new tools, the COVID-19 crisis and being new to developing large software products resulted in a less then optimal development pattern. But that is okay! This is a learning experience. We have both grew due to this.

### 5.1. Future Work

We can add, tabs for Equipment, Skills, spells, abilities, and notes, and also have databases for all that information. A different save/load system would be beneficial as well. As of 4/25/2020 there are no plans for continued development of Aetalag TTRPG Manager. As finals class and more is learned between the two of us I am sure that this project will be finished in time.