[D&D-inator]

# Team Members

* Andrew Emlund
* Spencer Southam
* Gabe Hill
* Alice McClain

# Project Description

DnDinator is a DM utility for Fifth Edition D&D. The application handles the creation and management of characters. Has a built in Dice roller, Loot stat generator, and Combat damage roller. Launch options lead to a version of the app for a DM and a version for the Players.

# Part 1: Functional Requirments

*Below is an example set of functional requirements. Your team must fully define what your system will do. Language and terminology must be appropriate for laypeople, as this document is for the client.*

## Glossary

* DM – Dungeon Master, also reffered to as Game Master/Manager
* Player – Individual participating in the game, has less application functionality than the DM

## Priority

Each requirement has a priority level of [1], [2], or [3].

1. Must-have functionality critical to the problem solution.
2. Highly desirable feature that should be included.
3. Optional requirements that will be completed if time allows.

[1] Launch argument for DM vs Player

[1] Character sheet storage

[2] Character sheet creation

[1] Spell Repo

[1] Item Repo

[1] ?Effect Repo?

[1] Dice Roller with options for different type of dice, Including how many of that dice to roll

[2] Combat dice roller that takes in the effects of items held by the attacker

[1] Loot stat generator

[1] Loot type Repo

[1] Item Repo

## Book

1. A librarian can add a book to the system – see UC5. [1]
   1. The system stores the following required information about a book:
      1. Publication Type [1]
      2. Title [1]
      3. Author [1]
      4. Publisher [1]
      5. Publication Date [1]
      6. ISBN [1]
      7. Replacement Price [2]
   2. The system stores the following optional information about a book:
      1. Keywords [1]
2. A librarian can checkout books for a patron – see UC10 [1].
   1. Etc., etc., etc. (LOTS MORE REQUIREMENTS HERE)

# Part 2: Objective Coverage

* Data persistence via writing files to disk and reading them back
* JavaFX for the application's GUI
* Proper encapsulation and separation of components into their respective MVC categories
* HashMaps will be used alongside enums to properly access and store class and race data
* Proper teamwork, communication, and load distribution will be used to ensure each group member pulls their own weight
* UML will be used to properly structure the application, and provide reference to that structure
* User input will be gathered and checked
* Output will be formatted in an intelligent way
* Overrides will be used for all applicable methods
* Proper optimizations, such as StringBuilder, shall be used where applicable

# Part 3: Basic Design

# Part 4: Project Plan

* Phases Base Level
* Phase one: Continued planning and distributing of jobs for the program
* Phase dos: pretesting and editing
* Phase san: polishing and finalizing
* Phase vier: demonstrating and final execution!