Edward Kennedy
Dr. Pulimood
CSC 415
4 November 2016

__**Use case descriptions**__

Add creature to initiative

Actor: Game Master (GM)

Scenario

1. The GM selects “Add creature to combat”
2. The GM chooses from a list of creatures to add, and selects a quantity
3. The GM chooses to either manually input the creatures’ initiative or click the “roll” button to automatically generate initiative scores
4. The system adds the creatures into the turn order depending on their initiative

Exceptions

1. The GM wants to add a new creature rather than choose an existing one - GM selects “Add new creature” from the bottom of the creature list; see the use case for **Add creature type to database**

Add or remove status effect from creatures

Actor: Game Master (GM)

Scenario

1. The GM selects “Add status effect”
2. The GM enters the effect name, description, stat modifications, duration, and whether or not saves are allowed (and, if so, the type of save)
3. The GM marks off every creature affected by the status
4. The GM clicks “Apply effect”
5. All selected creatures gain the status effect with the given parameters

Exceptions

1. Not all input fields are filled - the system will display an error message and prompt re-entry if the name and duration are not submitted; all other fields are optional

Add creature type to database

Actor: Game Master (GM)

Scenario

1. The GM selects “Add new creature”
2. The system presents an input dialog, prompting for all creature information
3. The GM enters the information and clicks “Save creature”
4. The creature is added to the database

Exceptions

1. A creature with this name already exists - the system will save the creature but append a hyphen and a letter to the name to differentiate between creature variants.
2. Not all input fields are filled - the system will display an error message and prompt re-entry if the name is not submitted; all other fields are optional.

Edit creature database

Actor: Game Master (GM)

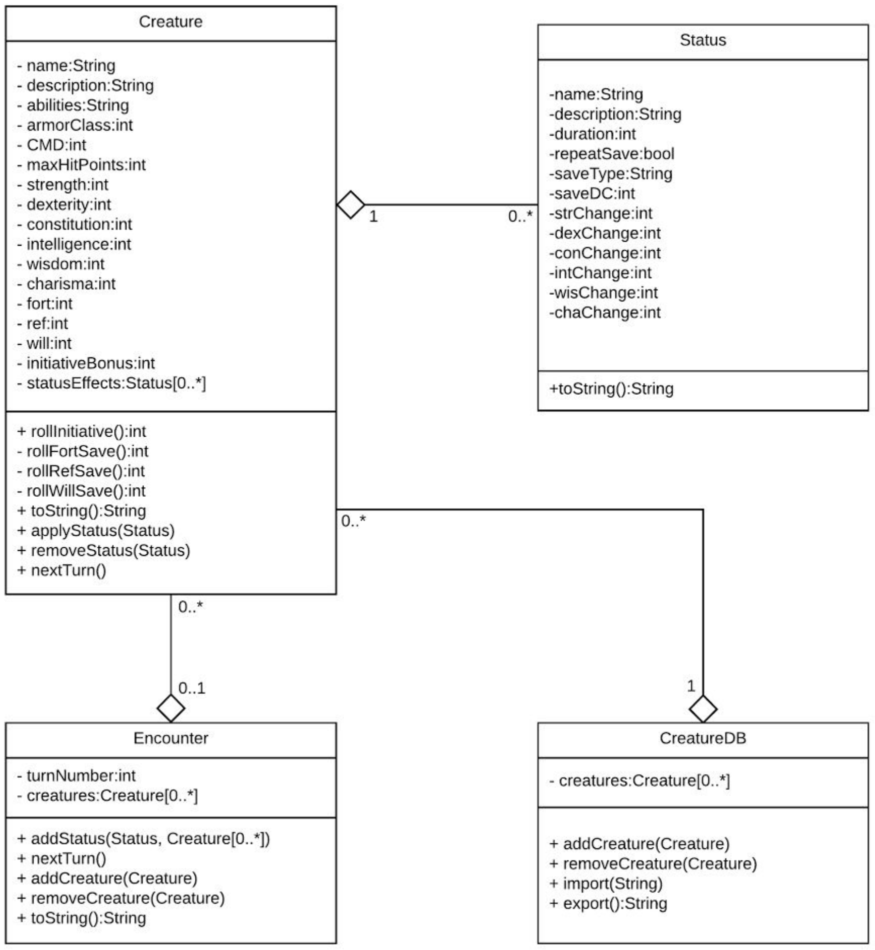
Scenario

1. The GM selects “View and edit creatures”
2. The system displays a table showing basic information on all creatures saved in the database
3. The GM clicks on an individual creature
4. The system shows the full information of the creature as editable fields
5. The GM modifies any desired fields and clicks “Save creature”

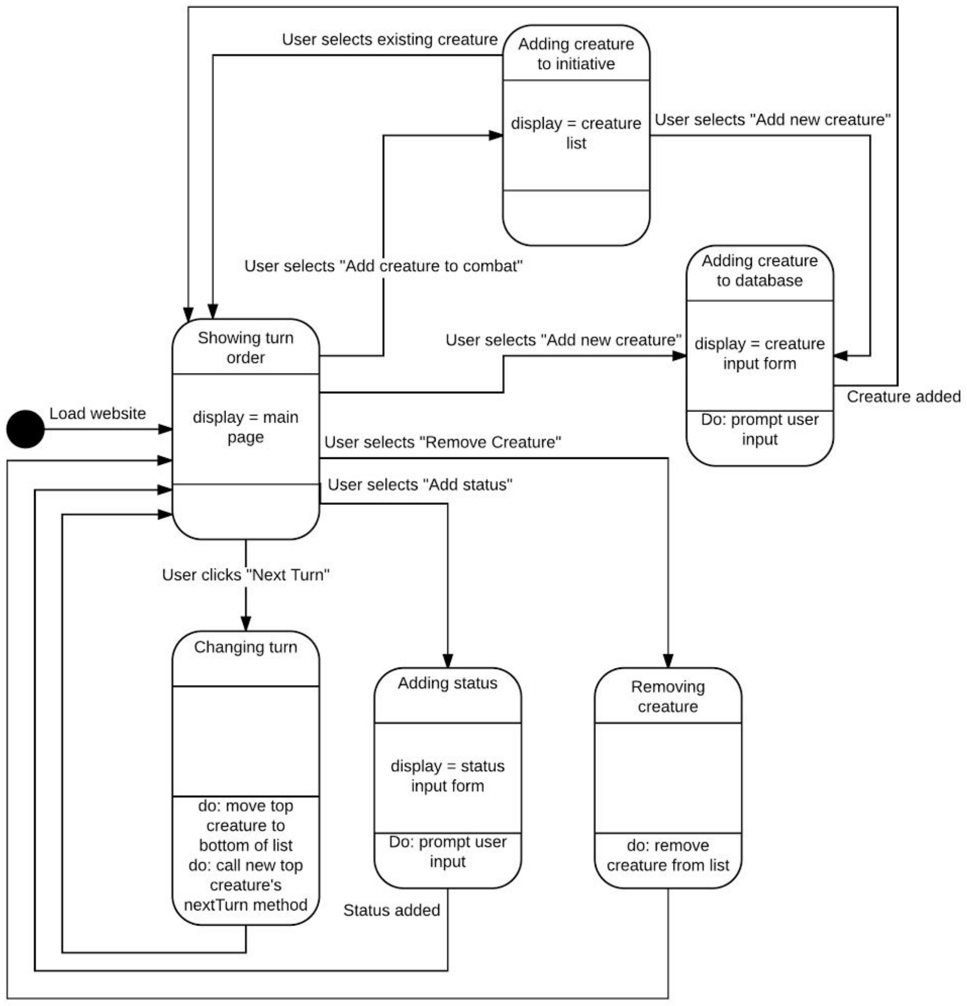
Exceptions

1. The creature name is removed - as the name is a necessary field for the creature, the system will display an error message and prompt re-entry

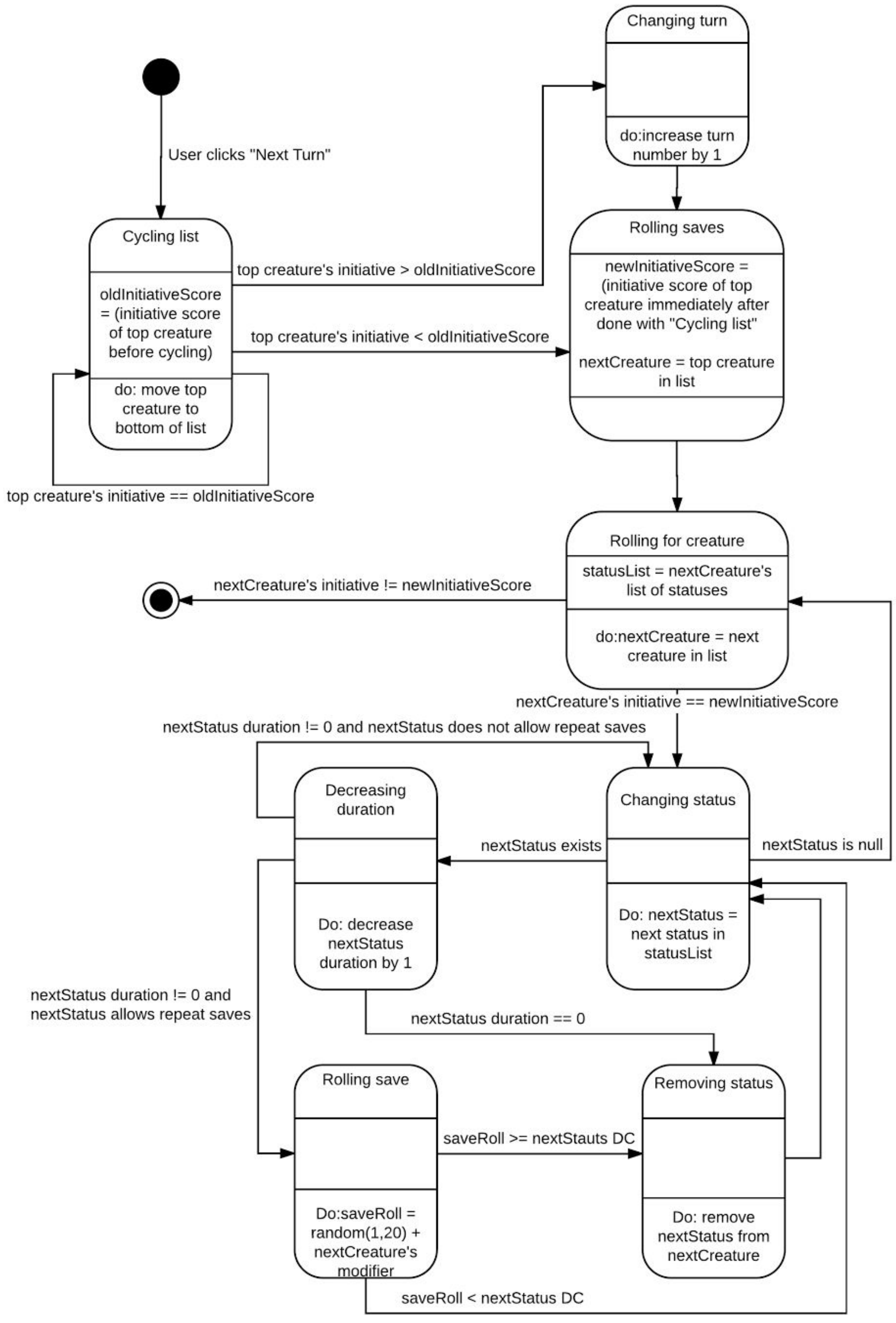
Detailed design class diagram



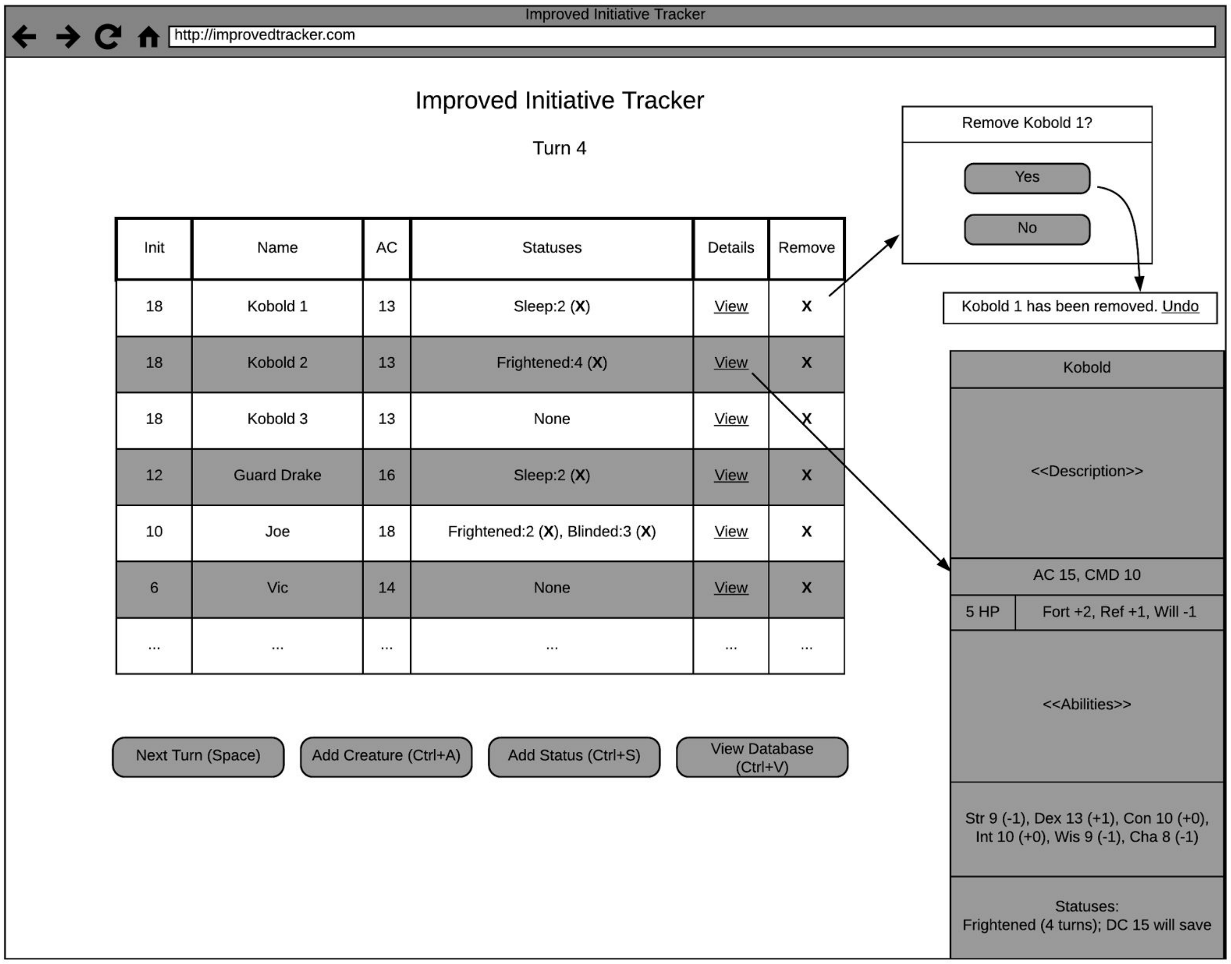
System Statechart


User selects existing creature
Adding creature to initiative
display = creature list
User selects "Add new creature"
User selects "Add creature to combat"
Adding creature to database
display = creature input form
Do: prompt user input
Creature added
Showing turn order
display = main page
User selects "Add new creature"
Load website
User selects "Remove Creature"
User selects "Add status"
User clicks "Next Turn"
Changing turn
do: move top creature to bottom of list
do: call new top creature's nextTurn method
Adding status
display = status input form
Do: prompt user input
Status added
Removing creature
do: remove creature from list

Detailed Statechart: turn change algorithm


User clicks "Next Turn"
Cycling list
oldInitiativeScore = (initiative score of top creature before cycling)
do: move top creature to bottom of list
top creature's initiative == oldInitiativeScore
top creature's initiative > oldInitiativeScore
top creature's initiative < oldInitiativeScore
Changing turn
do:increase turn number by 1
Rolling saves
newInitiativeScore = (initiative score of top creature immediately after done with "Cycling list"
nextCreature = top creature in list
Rolling for creature
statusList = nextCreature's list of statuses
do:nextCreature = next creature in list
nextCreature's initiative != newInitiativeScore
nextCreature's initiative == newInitiativeScore
nextStatus duration != 0 and nextStatus does not allow repeat saves
Decreasing duration
Do: decrease nextStatus duration by 1
Changing status
Do: nextStatus = next status in statusList
nextStatus exists
nextStatus is null
nextStatus duration != 0 and nextStatus allows repeat saves
nextStatus duration == 0
Rolling save
Do:saveRoll = random(1,20) + nextCreature's modifier
saveRoll >= nextStauts DC
Removing status
Do: remove nextStatus from nextCreature
saveRoll < nextStatus DC

UI



Improved Initiative Tracker

http://improvedtracker.com

Improved Initiative Tracker

Turn 4

Init	Name	AC	Statuses	Details	Remove
18	Kobold 1	13	Sleep:2 (**X**)	View	**X**
18	Kobold 2	13	Frightened:4 (**X**)	View	**X**
18	Kobold 3	13	None	View	**X**
12	Guard Drake	16	Sleep:2 (**X**)	View	**X**
10	Joe	18	Frightened:2 (**X**), Blinded:3 (**X**)	View	**X**
6	Vic	14	None	View	**X**
...	...	...	...	...	...

Next Turn (Space) · Add Creature (Ctrl+A) · Add Status (Ctrl+S) · View Database (Ctrl+V)

Remove Kobold 1?

Yes

No

Kobold 1 has been removed. Undo

Kobold

<<Description>>

AC 15, CMD 10

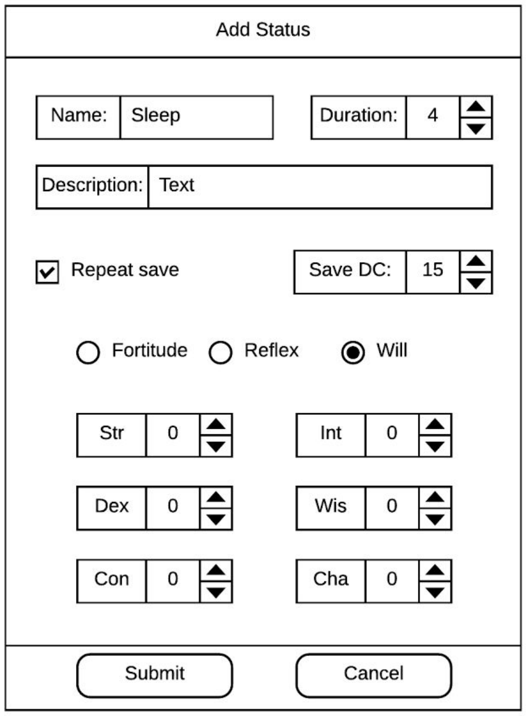
5 HP | Fort +2, Ref +1, Will -1

<<Abilities>>

Str 9 (-1), Dex 13 (+1), Con 10 (+0), Int 10 (+0), Wis 9 (-1), Cha 8 (-1)

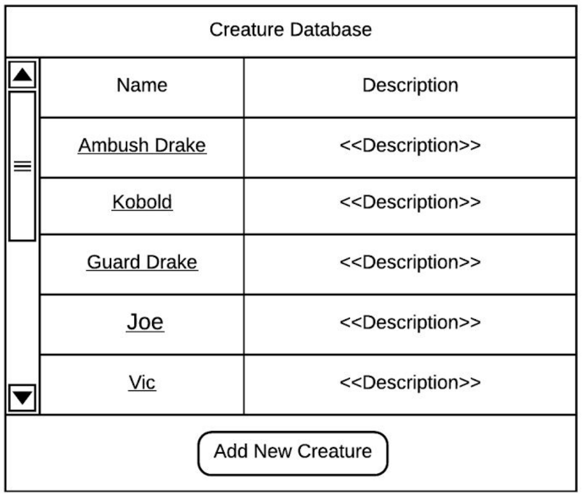
Statuses:
Frightened (4 turns); DC 15 will save

UI (Submenus)

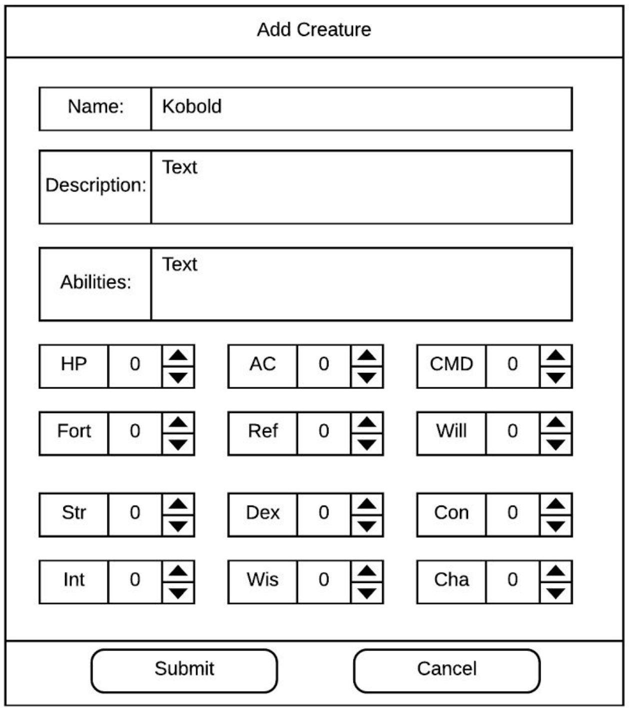

Add Status

Name: Sleep
Duration: 4
Description: Text
[✓] Repeat save
Save DC: 15
Fortitude Reflex [✓ Will]

Str	0	Int	0
Dex	0	Wis	0
Con	0	Cha	0


Creature Database

Name	Description
Ambush Drake	<<Description>>
Kobold	<<Description>>
Guard Drake	<<Description>>
Joe	<<Description>>
Vic	<<Description>>


Add Creature

Name: Kobold
Description: Text
Abilities: Text

HP	0	AC	0	CMD	0
Fort	0	Ref	0	Will	0
Str	0	Dex	0	Con	0
Int	0	Wis	0	Cha	0

Eight Golden Rules

1. *Strive for Consistency*

The UI is developed in a way that should be familiar to players of Pathfinder; while many different sources have information on creatures or status effects in different orders, there are general groups that are kept together. This UI groups the content in a way that makes sense to Pathfinder players: AC/HP/CMD grouped together, Fort/Ref/Will together, and Str/Dex/Con/Int/Wis/Cha together. In addition, the UI is internally consistent; different sections always appear in the same order (e.g. Fort then Ref then Will, or Str then Dex then Con). This makes it so that users can easily learn and remember where to look for specific pieces of information.

2. *Enable Frequent Users to Use Shortcuts*

   The system offers shortcuts for key functionality to help speed up use of the site (for example, space to change turns). This will allow users of the system to grow comfortable with a few simple ways to streamline use of the site.

3. *Offer Informative Feedback*

   All actions will have a noticeable effect on the interface. For example, all buttons will cause separate dialogs to either open or close. In addition, when certain actions are completed (such as adding a new creature to the database or to combat) the change is reflected in those screens. Also presented will be a box displaying the last action performed (along with an undo button) for a short amount of time. This will not only let the user know that their input was recognized by the system but also give them the ability to reverse their actions.

4. *Design Dialogs to Yield Closure*

   All dialogs will show short messages of completion as they close (discussed above), signifying that the task has been completed successfully. Different windows are also geared towards carrying out one specific task, so that when a dialog closes it signifies the end of a task.

5. *Offer Simple Error Handling*

   Most fields will restrict the type of input to ensure validity (such as the fields that require numbers). Other fields will be checked when the user attempts to click submit. If any errors are found (such as a required field being left blank) the user will be notified of the offending field and given an explanation of how to correct the error.

6. *Permit Easy Reversal of Actions*

   Users can easily remove status effects or creatures that were added by mistake through the remove buttons. In addition, the page prompts for confirmation on actions such as removing creatures to ensure that the user wants to go through with the action. Past that users will be briefly shown messages detailing the last completed action and including an undo button, so that users can reverse any undesired actions.

7. *Support Internal Locus of Control*

   All aspects of the system are user defined and user driven; even though some aspects such as saving throws are done automatically, they wait to trigger until the user manually chooses to change turns. All actions will provide user feedback (as discussed above) to demonstrate that the user is the one driving the system.

8. *Reduce Short-Term Memory Load*

   Users are always shown the various shortcuts available so that the information is easily referenced. In addition, the main page strives to hold all information that is necessary at a glance, such as a creature’s name, initiative score, armor class, and current statuses; the goal of the system is to store that information for a user so that they don’t have to constantly remember it.

Test Case Design

The testing procedure will start with unit testing. As classes are built, unit tests will be developed in parallel. Tests will be made to ensure that each class can be properly constructed, and that all major methods work (in general: setting and accessing various fields, constructing objects). Unit testing will be simple to implement for the Status class, as a Status does not need to interact with other classes for its functionality to be tested. Similarly, for the Creature class, most methods work exclusively within the Creature class's primitive data types. However, for the methods that interact with a Status, placeholder objects will need to be created in order to test Creature as a standalone unit. Similarly, testing of the CreatureDB and Encounter classes will be focused more on their integration with other units.

Integration testing will involve ensuring that the classes interact correctly amongst one another. For this project, bottom up integration testing will be used. The lowest level of this will be making sure that a Creature correctly stores and modifies a Status through its applyStatus, removeStatus, and nextTurn methods. From there, it will involve making sure that CreatureDB correctly stores and preserves its Creature objects. This includes making sure that creatures can be added, properly accessed, and removed from the database without affecting other objects. Then, the Encounter class will be tested. The Encounter is responsible for correctly maintaining a list of Creatures, and beyond that ensuring that statuses are correctly managed by the objects in its Creature list. The basic integration testing will involve adding, accessing, and removing Creatures; beyond that, it will involve adding Statuses to various creatures, and going through turns to ensure that the statuses are updated.

Finally, system testing will look at the project as a whole. Preliminary testing will be for ensuring that all core functionality exists and is outwardly visible. Then, the system will be used for several mock combats to test the design and flow of the interface. System testing also involves checking the error handling of the system. So, tests will also involve attempting to break the system (with improper input) to see whether or not the error handling works.

For implementing these testing practices, Ruby's Test::Unit and ActionDispatch::IntegrationTest will be used. An alternative would be RSpec. Rake can be used to create tasks to execute these tests so that they can be easily run after major updates. Once the project is live, PIWIK will be helpful in determining the usability of the site. It will help track what pages need to be made easier to use and which need to be made more accessible, based on the page analytics.

Test cases:

Functionality Tested	**Inputs**	**Expected Output**	**Actual Output**
Add creature to initiative	Full creature information	Main screen shows new creature	
Add creature to initiative	No creature chosen	No change to the main screen	
Add status effect to creatures	Full status effect information; some creatures selected	Selected creatures show the new status	

Add status effect to creatures	No name provided for status effect	Nothing changes; system prompts for name entry explaining its necessity	
Add status effect to creatures	No creatures selected	Nothing changes in the system	
Advance the turn tracker	The encounter includes several creatures	The list is cycled so that creatures with the next highest initiative scores are shown; the next creatures’ statuses are updated	
Advance the turn tracker	As above, but the creatures with the lowest initiative are currently at the top	As above; in addition, the turn counter increases	
Advance the turn tracker	No creatures are in the combat	Nothing changes	
Add creature type to database	Full creature information is given	The creature database includes the new creature with all of its information as provided	
Add creature type to database	No name given	The system does not add the creature, instead prompting the user to enter the name	
Add creature type to database	Name repeats an existing creature in the database	The creature is added to the database with a hyphen and letter appended to the name to differentiate between the similarly named creatures	
Edit creature database	Full information given	The database now reflects the changed	

		version of the creature, but no Creature objects in the Encounter are changed	
Edit creature database	No name given	The system prompts the user to enter the name	