Mage Helper

Max Norfolk

Introduction:

Mage Helper is an app that acts as a character sheet for dungeons and dragons 5th edition (5e). It provides an interactive experience for the user to modify their character. It is very useful for some things that are difficult to keep track of such as the weight (and player's carrying capacity.) It also easily allows looking up of spells. There are some apps on the Google Play store that do similar things to this one.

The first technical feature is the use of fragments. Fragments are used in the Main Activity to allow the user to quickly and easily view information about their character. This is combined with the Bottom Navigation bar to allow the user to quickly switch which fragment is currently being displayed.

There are many views that are extended to provide some new and interesting behavior. These are described in greater detail in the Implementation section.

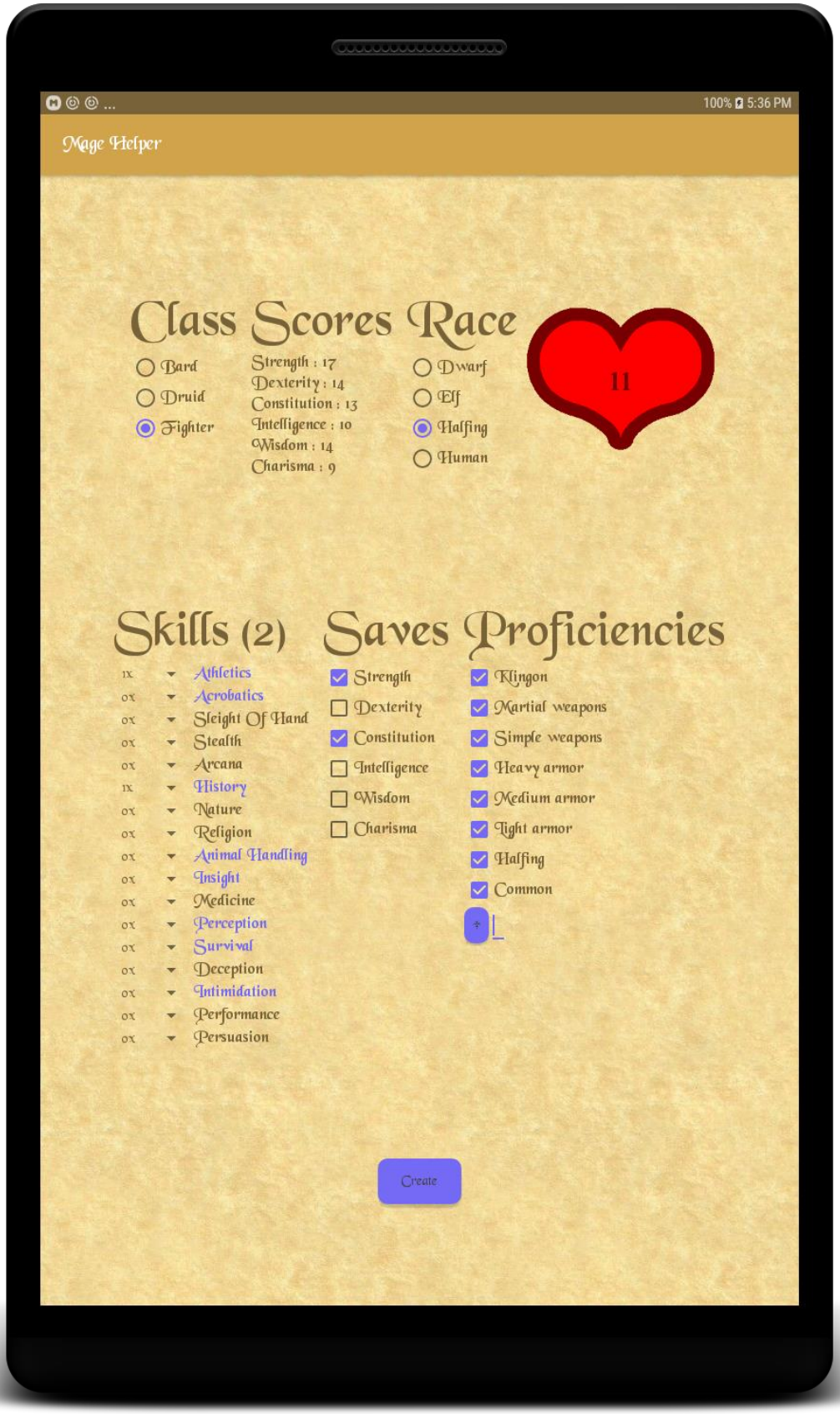
We discussed in class a little bit how to have different layouts depending on the orientation of the device. I have implemented for all fragments and activities (except the loading screen, as it did not need one) an alternate layout if the orientation is portrait or landscape. This is done through additional XML files, or getting the orientation at runtime. For example, the Dice Fragment the number of rows, the number of dice in each row and the order of the dice is all determined at runtime.

User Experience:

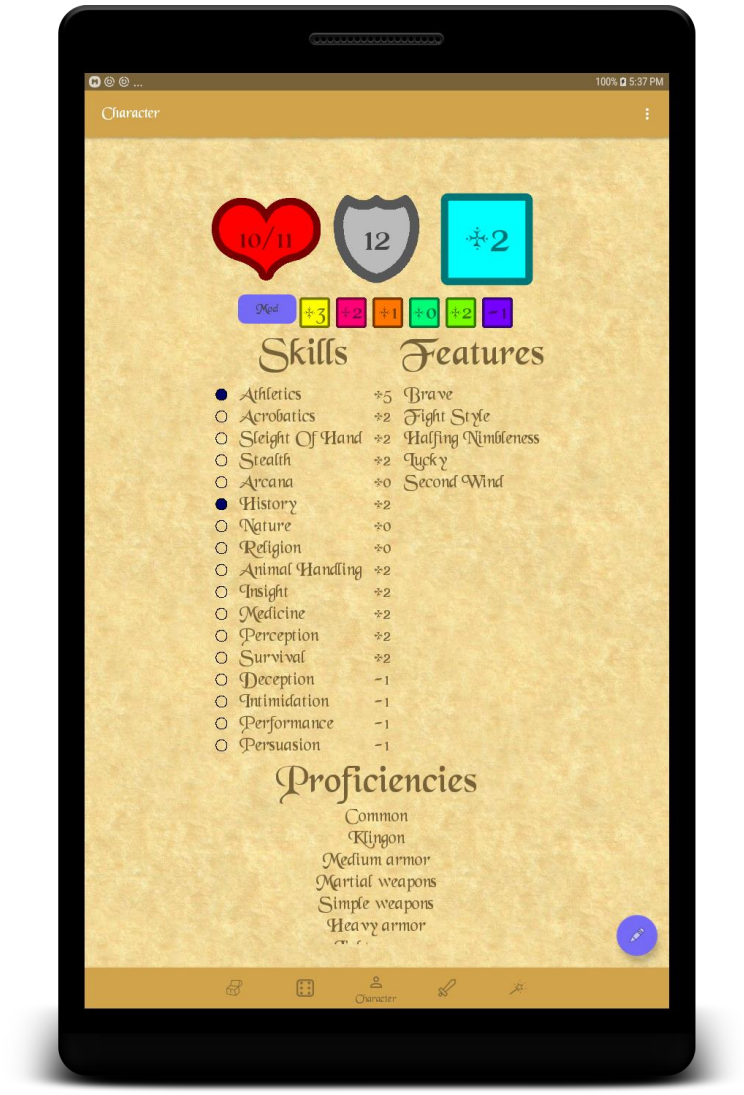
The user is greeted with a loading screen, and after a few seconds, they are transitioned to a new activity which allows them to create a character

The user then enters in the information they want to for their character. Here is an example of a character.

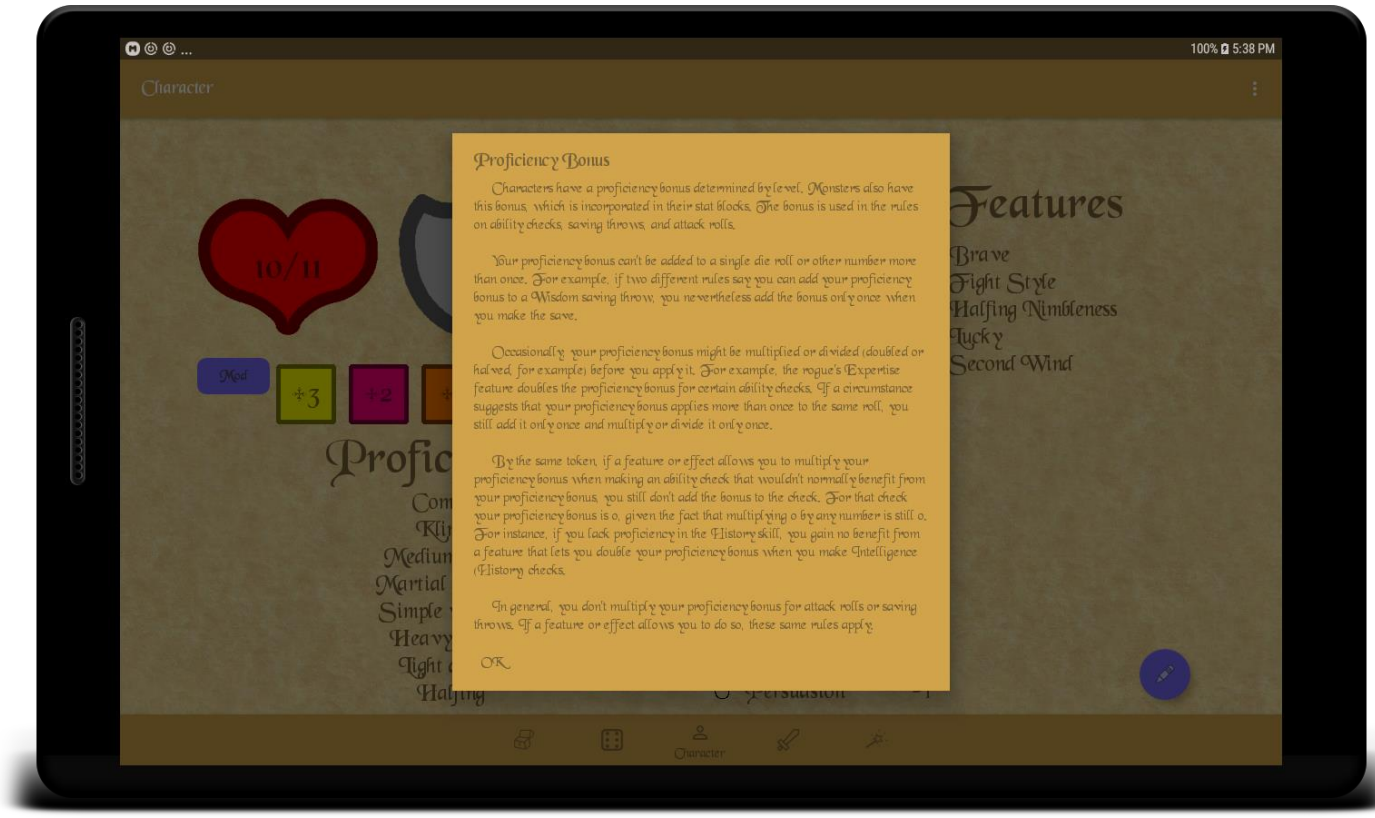
This character is a Halfling fighter. Notice how their scores are set, as well as proficiencies.


100% 5:36 PM
Mage Helper
Class
Bard
Druid
Fighter
Scores
Strength : 17
Dexterity : 14
Constitution : 13
Intelligence : 10
Wisdom : 14
Charisma : 9
Race
Dwarf
Elf
Halfing
Human
11
Skills (2)
1x Athletics
0x Acrobatics
0x Sleight Of Hand
0x Stealth
0x Arcana
1x History
0x Nature
0x Religion
0x Animal Handling
0x Insight
0x Medicine
0x Perception
0x Survival
0x Deception
0x Intimidation
0x Performance
0x Persuasion
Saves Proficiencies
Strength
Dexterity
Constitution
Intelligence
Wisdom
Charisma
Klingon
Martial weapons
Simple weapons
Heavy armor
Medium armor
Light armor
Halfing
Common
Create

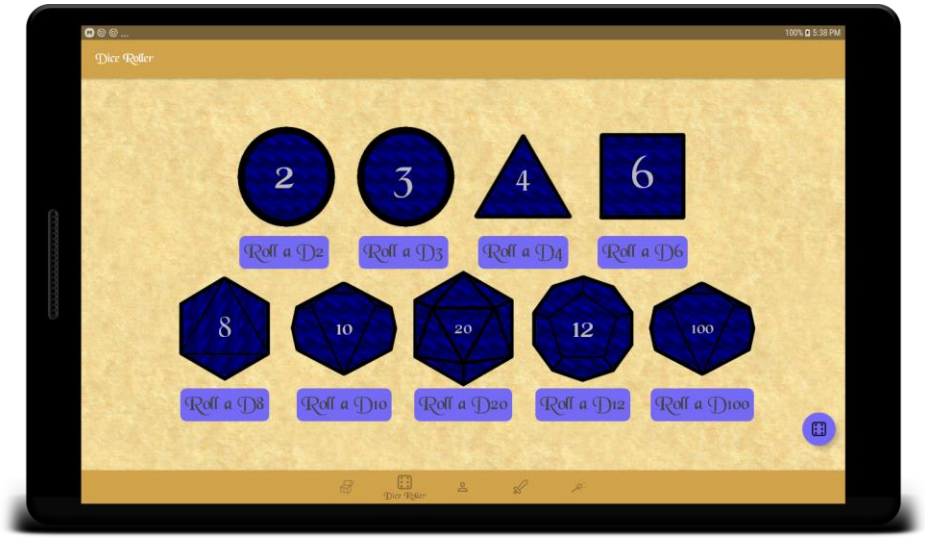
After creating a character, they transition to the main activity, here they can view all their stats features



Many of the text views are clickable, which opens up the rules about the topic. For example, proficiency bonus:

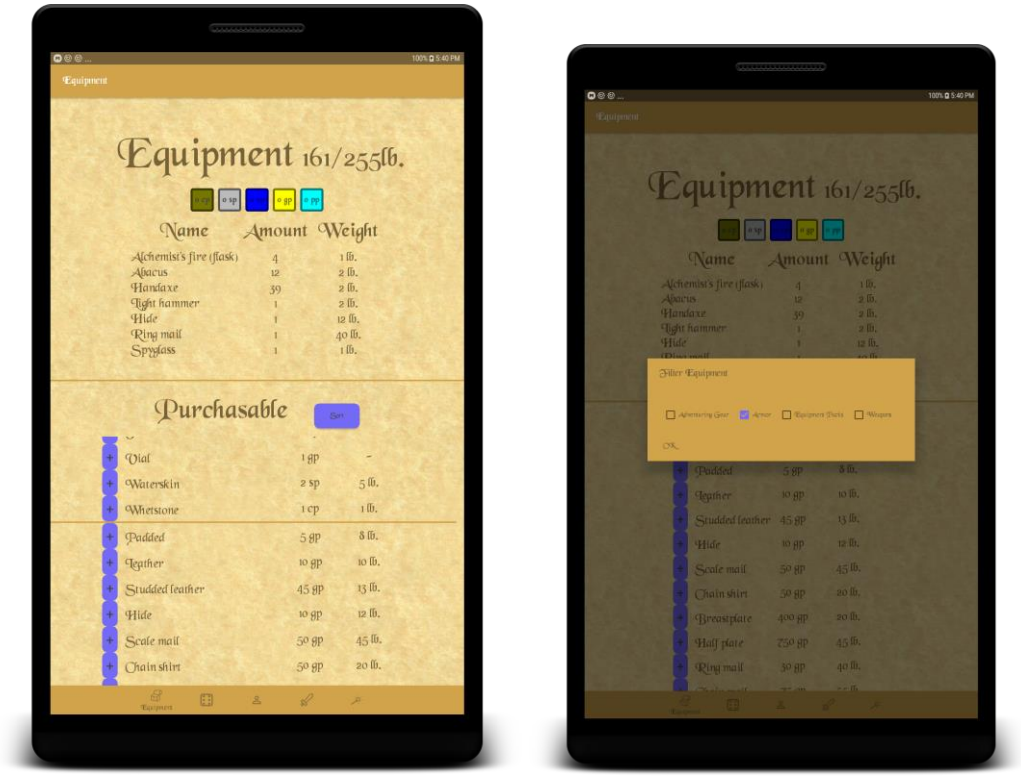


The user can also roll dice by clicking the button under a dice.

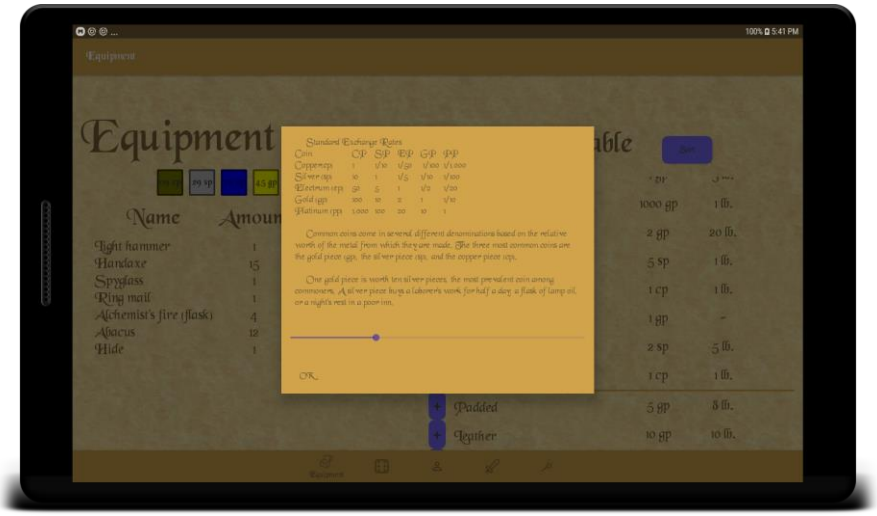


They can also click the floating action button to roll an n-sided dice n number of times.

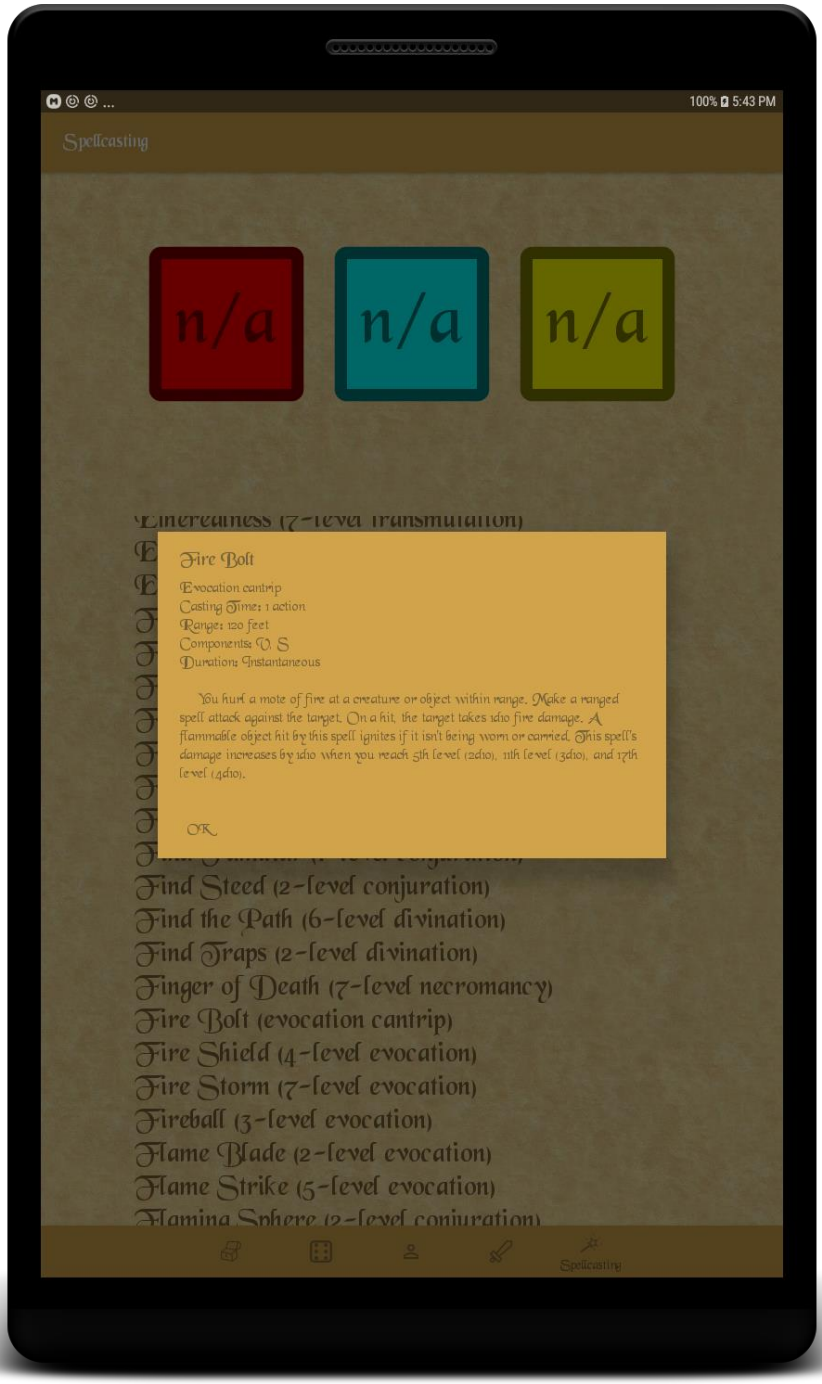
The user can modify their equipment by switching to the equipment fragment. Here they can add items to their equipment and modify their coinage (how much gold, silver, copper... they have)



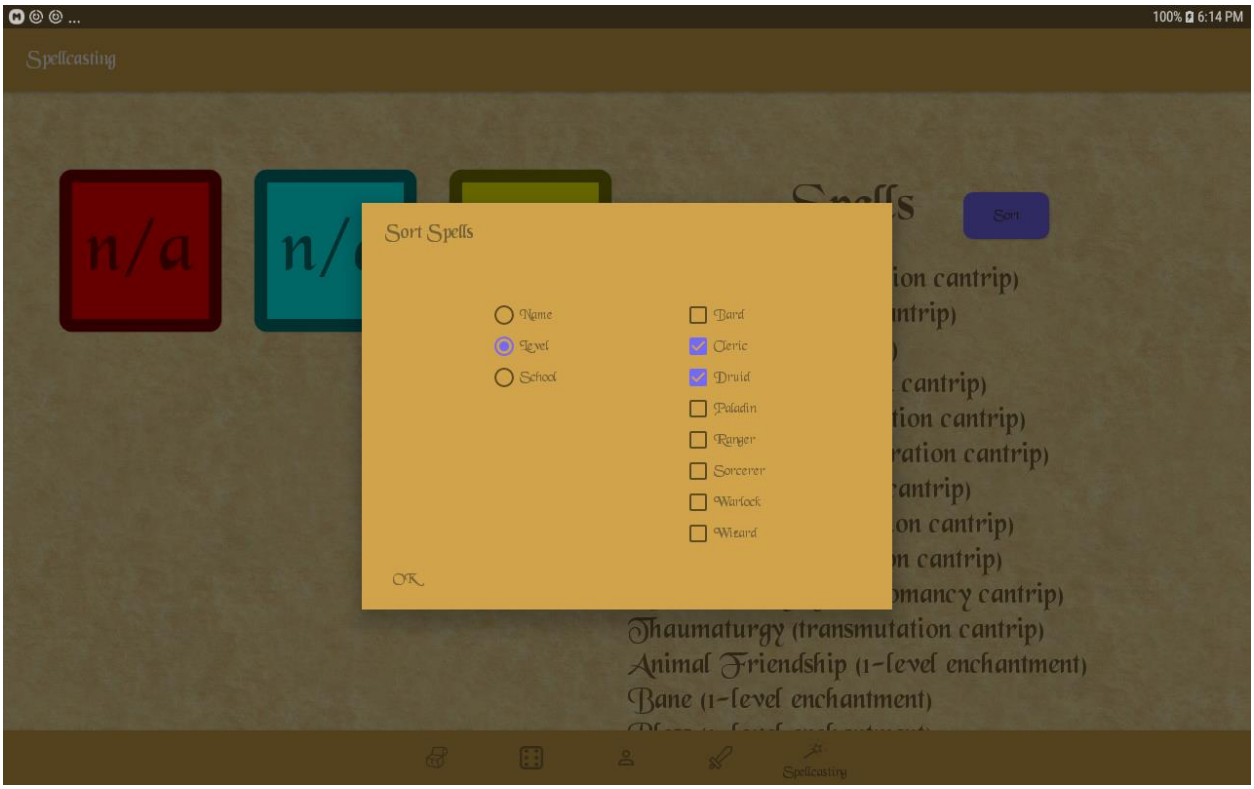
They can filter which items appear in the Purchasable equipment, making it easy to find particular items. The users can modify all of their coinage by using a Maxiumless seekbar. This seekbar is a seekbar that when the user holds it to the end of the seekbar the maximum increases over time. The seekbars for coinage when they are increasing their maximum, the amount it increases by increases over time to easily add large quantities of gold silver or any other coin.



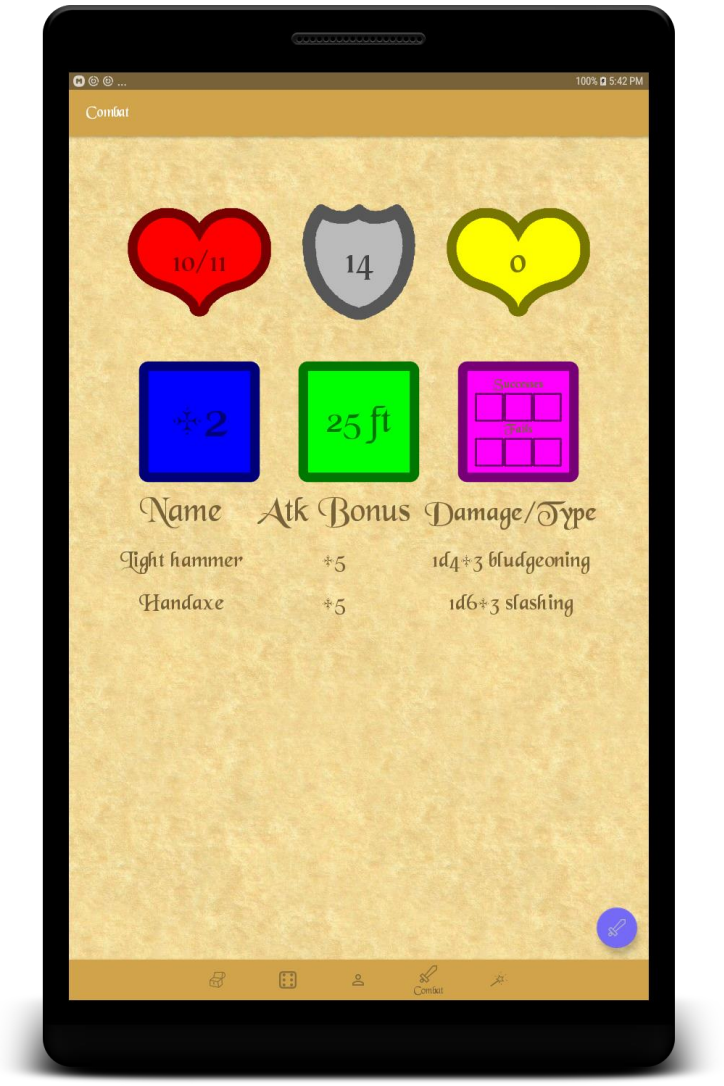
The app also holds has all the spells in SRD version 5.1. (The srd is the open source spells, classes, races ect. For Dungeons and Dragons.) Our fighter cannot cast spells, so notice how for the Spell Save DC, Spell attack bonus and spell casting modifier are all filled with n/a. Later on I will change the character to a bard, and we can look how it updates. All the spell text views are clickable and will open a stat similar to one you would find in an official DND book.



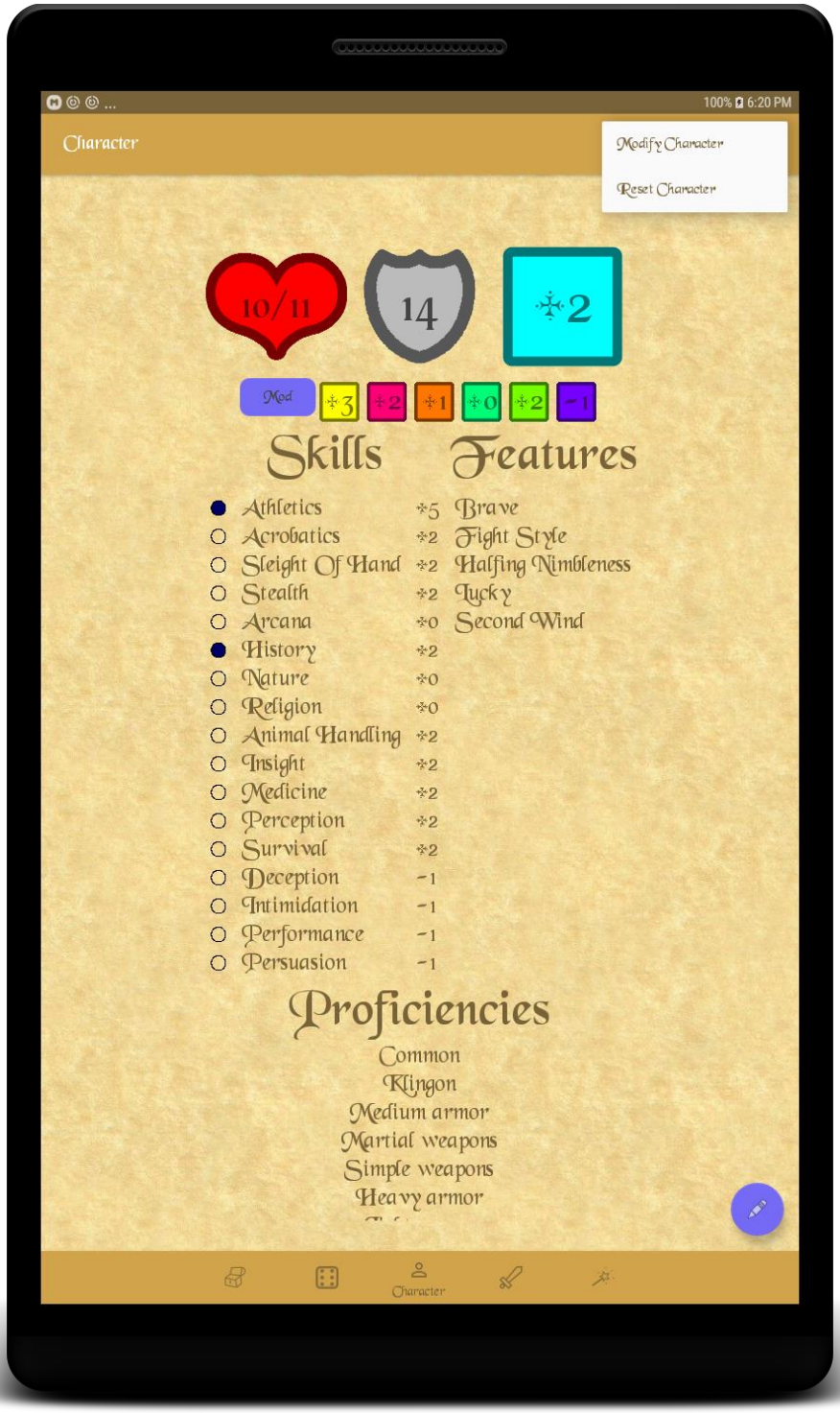
Spells like equipment can be filter to only show spells for a particular class. Their sorting can also be modified, by level, name or school of magic. School of magic is not typically useful, however if you play a Eldritch Knight (not implemented) knowing the school of magic is very useful. For level, cantrips are considered 0^{th} level spells



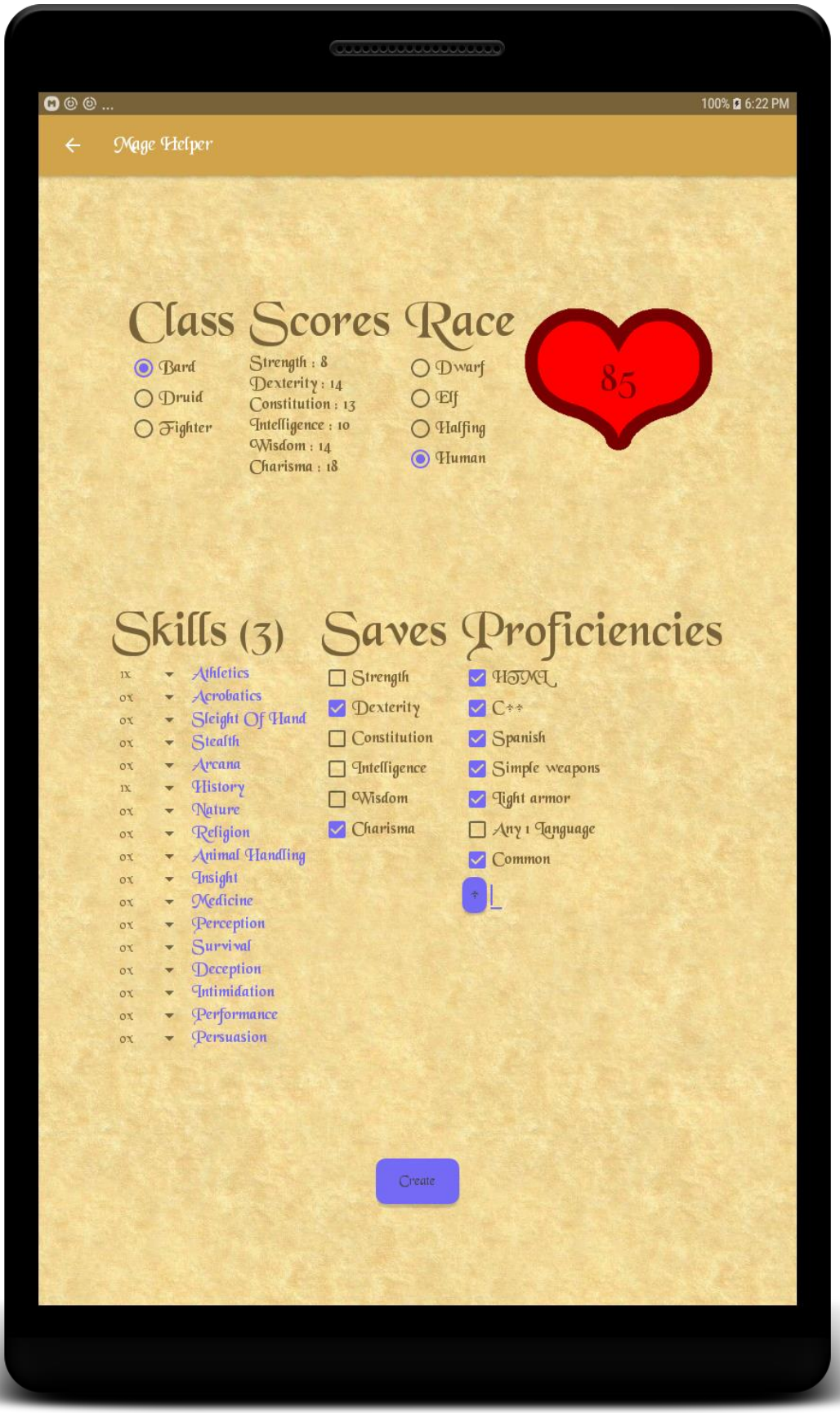
The player also has their combat fragment to view all things useful during combat. This contains their AC, hitpoints, temporary hitpoints, initiative, speed, death saving throws and all the different weapons they can use for attacking. Clicking the floating action button in the bottom will open up information about the the weapon properties. You can also click the Damage/Type textview to open up information about all the different damage types. The weapons shown are directly from the equipment, and are added to the player through the equipment.



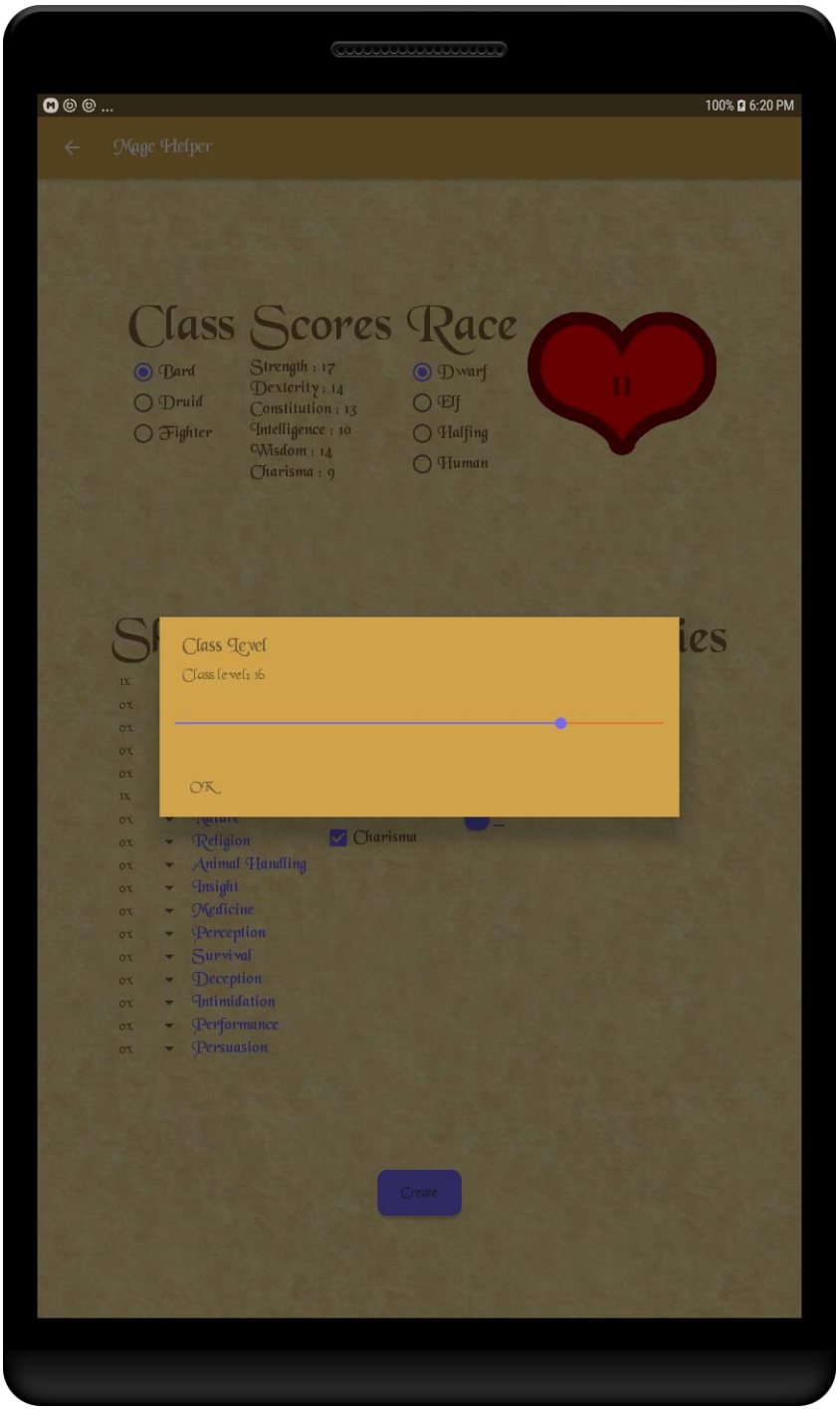
If you get bored of your character, or need to level up or modify a stat, you can do so from the character fragment. You can also reset your character. Both of these send you to the second activity you saw.



I created a human bard, just so we can see some nice things that appear when you are a spell caster.

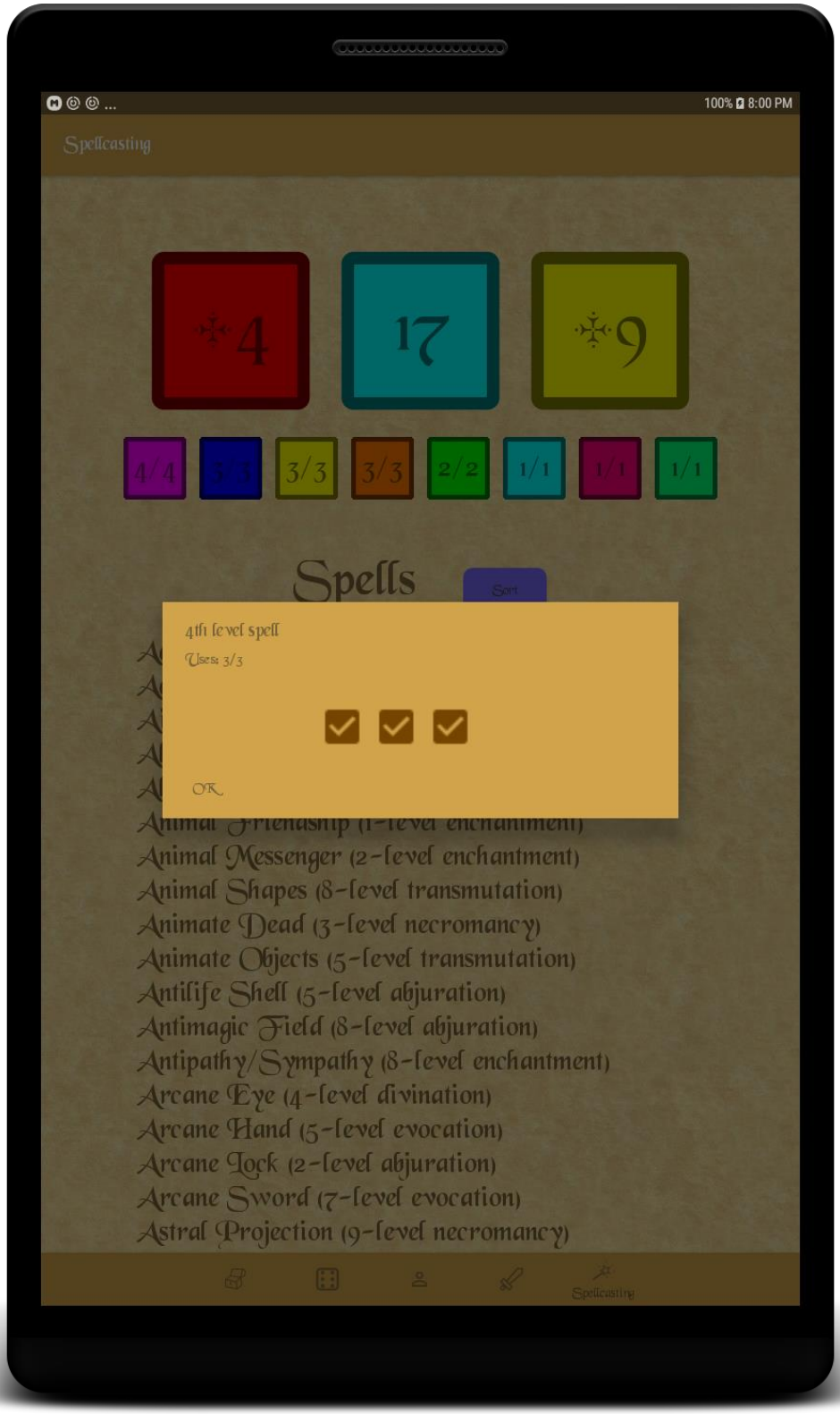

100% 6:22 PM
Mage Helper
Class
Bard
Druid
Fighter
Scores
Strength : 8
Dexterity : 14
Constitution : 13
Intelligence : 10
Wisdom : 14
Charisma : 18
Race
Dwarf
Elf
Halfing
Human
85
Skills (3)
1x Athletics
0x Acrobatics
0x Sleight Of Hand
0x Stealth
0x Arcana
1x History
0x Nature
0x Religion
0x Animal Handling
0x Insight
0x Medicine
0x Perception
0x Survival
0x Deception
0x Intimidation
0x Performance
0x Persuasion
Saves Proficiencies
Strength
Dexterity
Constitution
Intelligence
Wisdom
Charisma
HTML
C++
Spanish
Simple weapons
Light armor
Any 1 Language
Common
Create

Something about humans, you can see, is that they have proficiency in Any 1 language, in addition. Just to show how custom proficiencies can be added, I've added C++ and HTML as languages our bard is proficient in. We also can set the level by clicking on the class text view.

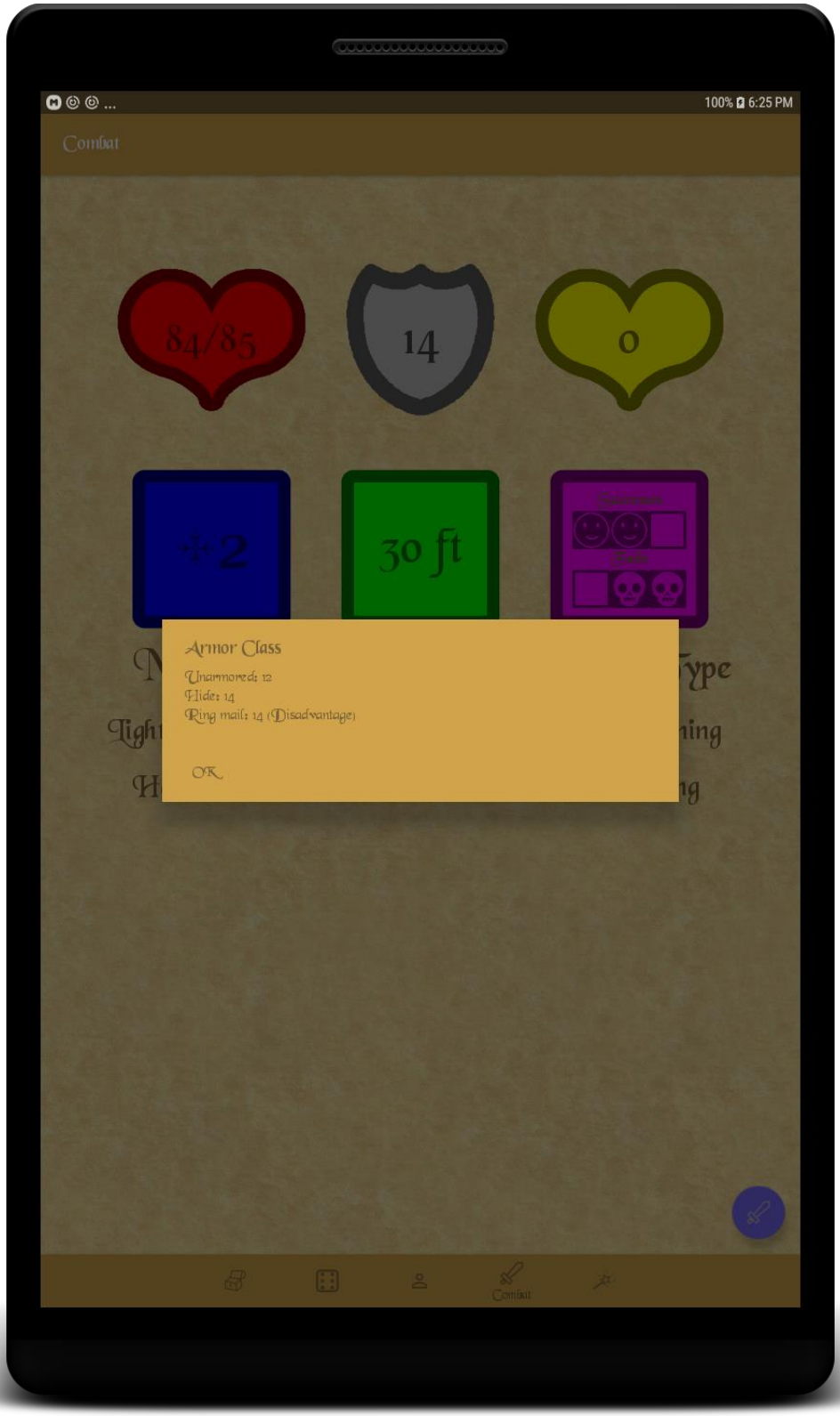


Now if we go to the spellcasting activity you can see that we no longer have n/a, and we have slots showing instead. Also look at the fact that we have 8th level spells

Just a quick example of what the spell slots look like when you click on them. The checkboxes modify the text number of uses remaining.



We skipped over armor class, but it shows the largest armor class as the text well as all of the other potential armors when you click on the text.



You can also click on a weapon and it will bring up all useful information about it

Any expendable ability shows the number of uses with checkboxes (or a seek bar if it is 5 or more uses) similar to Spell slots. For example your number of Bardic Inspiration die is equal to your charisma modifier (min 1). We can see this works as intended.

You can test this app, by creating a character and making sure everything displays correctly. You can click on spells make sure they have the correct text. The majority of things should save after closing the app (onPause in reality). There are a few exceptions like Success & failures for saving throws as these aren't needed, but health and weapons should save. The number of expended uses for everything is not saved.

Implementation:

The app composes of 3 activities and 5 fragments. All of the fragments and activities have a landscape and portrait view. Some are defined programmatically (for example the Dice Fragment) while others have defined XML files for the alternate orientation.

Something that is quite interesting is the boxes that are everywhere. The boxes drawable is actually a white square with a gray outline. The tint for the drawable is set, so we have a plethora of colors without needing a new file for every different color box. This idea is also used by the heart for the red and yellow heart. The shield symbol does this, however since it has no variations, it does not need to.

The MyDialog class is a class that simplifies the AlertDialog class to allow the programmer not to have to simplify many things, including issues with the dialog being opened multiple times. It allows the programmer to set the Title, Message and view for the alert dialog. It also features 1 listener, which is OnClose. This listener is called whenever the view is closed with an integer parameter which is sued to represent whether the dialog was closed with the OK button or by cancelling and clicking outside the dialog.

The most common views you will see are ClickableTextView. This is a custom class implemented by me. It is a very basic class that extends the TextView class. It makes use of the MyDialog class, and displays one on click.

The ClickableTextView class is the superclass to the ExpendableClickableTextView class which sets the view of the MyDialog to either a seekbar or checkboxes (depending on the max of the expendable object).

The MaximumlessSeekBar class is a subclass to the SeekBar class. This is a simple short class that modifies the original seekbar's behavior so when the user reaches the end of the seekbar the limit increases. This class has a few methods to modify the speed before each increase and the amount to increase it by. This class also allows the user to add their own OnSeekBarChangeListener, without killing the MaxiumlessSeekBar's behavior.

Something that was more difficult than it may seem is the use of custom drawables for the checkboxes. This required a lot of work to correctly resize the drawables, and because of this the Death Saving Throws box is one of my favorite things in the entire app. The background box is clickable and the checkboxes also are useable.

Another difficult thing was increasing the progress bar as resources were loaded. This was difficult because attempting to increase the bar as other threads were running, the app would wait until it was finished before redrawing the bar. To overcome this, I had a thread which ran after a slight delay. The last thing this thread did was call the next thread with another delay, and so on until you reached the end. This slight delay allowed the app to redraw everything. These threads are in the LoadResources class.

To save time the spell and equipment textviews are created at the start of the activity and stored. This is very useful, to not have to create these and style them everytime we want to filter or rotate the device. Doing it this method requires a bit more code to make sure they aren't recreated unless they need to, and to remove their parent whenever we filter them, however this is well worth it since we do not have to create hundreds of text views which will be garbage collected the second the user modifies one filter.

The backend part of the app is vastly complex, so I will only go into detail a few interesting things. You will notice that a few of these classes were not used in the app. Just about all of these are functional, but I didn't have time to make use of them.

The Parser interface is implemented by a few classes. The purpose of this is to read .5e files, which I used to store data. I wanted to be as efficient as possible while storing and reading files, so I thought it was a good idea to make my own file type to store and read files. To make adding object types easy to add, I use reflection to access the static method generate(String) from any class that implements the Generateable interface. Reflection is also used to create generic arrays at runtime for the ParserArray<T> class.

The EnumMap<K,V> and the EnumSet<T> classes I rewrote to allow n-null entries. This is useful if the user had the option to choose multiple options. For example, say you wanted a race which had the Common and 2 languages of your choice, You cannot easily store this in the default EnumSet<T> class, however in my rewritten version, you can. Both of these classes heavily use reflection to create generic arrays and to get the constants for the class ect.

The spell class is another area of interest. This class was written to allow users to enter their own spells. Sadly, I did not have time to implement this into the app, however the code is still there. How this code could be used is described in the Misc section as it is not yet implemented. The spell class is used to hold the normal spells in the app.

The Player class was originally intended to be used, however to simplify things for the app, it was converted to the Player2 class. The Player2 class loses lots of options offered by the Player class such as multi-classing, and resistances. The Player2 class simplifies things so it may be used by the app. You could use the Player class for everything the Player2 class does, however storing it is more difficult. The Player2 class has the toString() method and the fromString() method which is used to read and write to the SharedPreferences to store the user's character.

Bugs:

The only major limitations are the long loading time. There is potentially a way to load the resources however I have not had time to explore it yet. The other limitation with this app is the lack of potential customization. The majority of the classes and races in dungeons and dragons 5e are missing. The option for sub-classes and sub-races are also missing. Some of the races and classes are missing features (for example bard's Magical secrets). When the keyboard is opened up the screen shrinks, and some views look incorrect. Feats and multi-classing are also missing. A few of the spells text is formatted slightly strange depending on if they have charts or a stat. If you give yourself too high coinage, the text will stretch outside of the box.

Misc:

The spell class could be used to store user's custom spells. This is not yet implemented. The spell class has the Spell.generate(String) method which will return a spell object from a string of text in the format of a spell from an official DND book. This spell object could call the .toParseable() method which would return a formatted string which could be saved in SharedPreferences and then be read when the app starts up. Again, this is not in the app at the moment, but the code is there, and could easily be added.