Easy-To-Use AI Deck Generation

V0.1 Beta

USER GUIDE

Developed by

Fishagon Studios

fishagon@yahoo.com | Fishagon.com

Welcome

Deck Generator is a very straight forward procedural generation of card decks based on designer assigned ratings. It makes sure that no two AI will have the same deck of cards, but also always falls within a very small margin of deck rating.

The most common use of this asset will be to use the rating as a way to scale difficulty. You can make it so each AI in the game would have a difficulty rating and the Deck Generator can take that number and make a fitting difficulty of deck for the player to play against.

I'm glad you've decided to take the chance with using Deck Generator for your game and I hope that it serves you well. If you have any problems, suggestion, concerns or even just feedback please feel free to connect us at fishagon@yahoo.com so that we may support you as much as we can.

Deck Generator supports Unity 5.3.3+.

Thank you for using Deck Generator!

Setting Up Deck Generator in Your Project

There are only two things you need to have in your game for Deck Generator to work and that's DeckGenerator.cs and any C# class with the name "Card".

In order for DeckGenerator to work it works with a List<Card> that act as decks. Therefore, in your game you will need to have a base class for the type "Card". And within Card you need to have the following two ints: "id" and "rating".

Here is an example of the very least your Card class must contain for Deck Generator:

```
using UnityEngine;
using System.Collections;
using System.Xml;
using System.Xml.Serialization;

public class Card
{
    [XmlAttribute("ID")]
    public int id;

    [XmlElement("Rating")]
    public int rating;
}
```

In this example we use XML, but you can use any other format you would like as long as you're able to compile it into a List.

Once you have your Card class set up you'll want to create a system of creating a database. In the demo scene you'll find that we did this by creating an XML database of cards.

```
<?xml version="1.0" encoding="UTF-8"?>

<ItemCollection>
    <Cards>
        <Card ID="000">
            <Name>Index</Name>
            <Desc>Description of the Index</Desc>
            <Type>Weapon, Shield, Armor, Consumable, Spell, Summon, Companion</Type>
            <Rarity>Potato</Rarity>
            <Rating>500</Rating>
        </Card>
        <Card ID="001">
            <Name>Level 1 Sword</Name>
            <Desc>A basic short sword.</Desc>
            <Type>Weapon</Type>
            <Rarity>Common</Rarity>
            <Rating>010</Rating>
        </Card>
        <Card ID="002">
            <Name>Level 2 Sword</Name>
            <Desc>A basic, yet sturdy short sword.</Desc>
            <Type>Weapon</Type>
            <Rarity>Common</Rarity>
            <Rating>020</Rating>
        </Card>
        <Card ID="003">
            <Name>Level 3 Sword</Name>
            <Desc>A sturdy short sword.</Desc>
            <Type>Weapon</Type>
            <Rarity>Uncommon</Rarity>
            <Rating>030</Rating>
        </Card>
```

Note how in the example we have a Card with the ID=”000”, this is what we call our Index card and acts as template for all other cards. The Deck Generator will automatically ignore whatever card is in the 000 ID, so make sure not to put any cards at the first index of your database.

It’s important to find a good balance for your Card Ratings, the Deck Generator doesn’t know which of your cards are strong and needs your guidance for that.

In our demo example we use the DatabaseLoader.cs and CardCollection.cs scripts to create our card database. Feel free to use them as the foundation for your system if you’d like.

Generating A Deck

Now that you’ve got yourself a card database it is time to generate some decks. This is a simple one function system where you only need to assign a few variables to guide the generator to fitting the rules of your game.

Let’s breakdown the function or you now so that you’ll know how to read the examples and then use it in your own projects.

List<Card> Generate(List<Card> cardDatabase, int deckSizeMin, int deckSizeMax, int dupeLimit, int desiredRating, int iterationLimitor)

List<Card> cardDatabase = This is where you will insert your card database list that you previously created. This are all the possible cards in your game that the generator can use.

int deckSizeMin = The minimum amount of cards that can be in a deck within your game.

int deckSizemax = The maximum amount of cards that can be in a deck within your game.

int dupeLimit = The maximum amount of duplicates of each unique card from the database allowed in a deck. You can use this to limit the generator to using at most 2 of any one card. If you leave null or enter 0, it will allow any amount of a single card (even if your game doesn’t limit duplicates, I recommend imposing one on the generator for best results).

int desiredRating = This is the total sum rating you want the generated deck to match. This is where you can insert your AI’s rating and the generator will know how hard to make the deck.

int iterationLimitor = This is an optional parameter where you can limit the amount of times the generator loops looking for the best combination of cards to match your desired rating. The higher amount of iterations you allow the more processing it will take, whereas the less iterations you allow the less accurate the generation may be. The default amount is 512 and has shown to be both precise and low costing.

The function also returns as a new List<Card>. So to use this function you would write a code similar to the following:

```
49      void CreateDeck()
50      {
51          List<Card> deck = new List<Card> ();
52          deck = DeckGenerator.Generate (DatabaseLoader.cardDB.cc.cards, 10, 15, 2, aiRating);
53      }
```

In that example we create a new deck of cards called “deck” and generate it using a card database from the example DatabaseLoader.cs script, a minimum of 10 cards, a maximum of 15 cards, limit any duplicates to 2, and we pass in an AI’s difficulty rating for our desired deck rating.

It’s also good to note that if you were to generate this code two times in a row with no changed variables, you would most likely receive two completely different sets of cards. You can see this in action by testing it in the included Demo scene.

Other Functions

We have included an additional function into Deck Generator as just a bit of a helpful function after having just generated a deck.

You can use Shuffle() to mix up the cards in a List<Card> to get a more random draw. This function is based on the Fisher-Yates shuffle algorithm and uses a static instanced System.Random instead of UnityEngine.Random.

This function takes no parameters and below is an example of how simple it is to use:

```
49      void CreateDeck()
50      {
51          List<Card> deck = new List<Card> ();
52          deck = DeckGenerator.Generate (DatabaseLoader.cardDB.cc.cards, 10, 15, 2, aiRating);
53          deck.Shuffle ();
54      }
```

Thanks Again

Once again, we would like to thank you for using Deck Generator for your game. Please feel free to contact us about anything, including anything you would like to see added to the asset or even just to show us how you used it!

Tips for Better Results

If you're having a hard time getting satisfactory results with the generator, please take a quick look over these usage tips.

- The generator is math based and heavily reliant on you using balanced ratings for your card database. Consider a large amount of cards (perhaps the weaker ones) with very low ratings. Perhaps cards that would be in almost any deck, give a 5 or 10. The generator will use those often when getting precise ratings.
- Start low rating AI with a "large" rating number. I recommend a deck rating of 150 for starting AI. Base your card ratings around your starting deck rating.
- The generator performs best when it is limited. If your game has a very few amount of cards and you don't place a duplicate limit, you will not get good results. Even if your card game rules don't limit duplicates, put one into the generator. This can be seen in the demo scene to work quite well with a very limited card database.