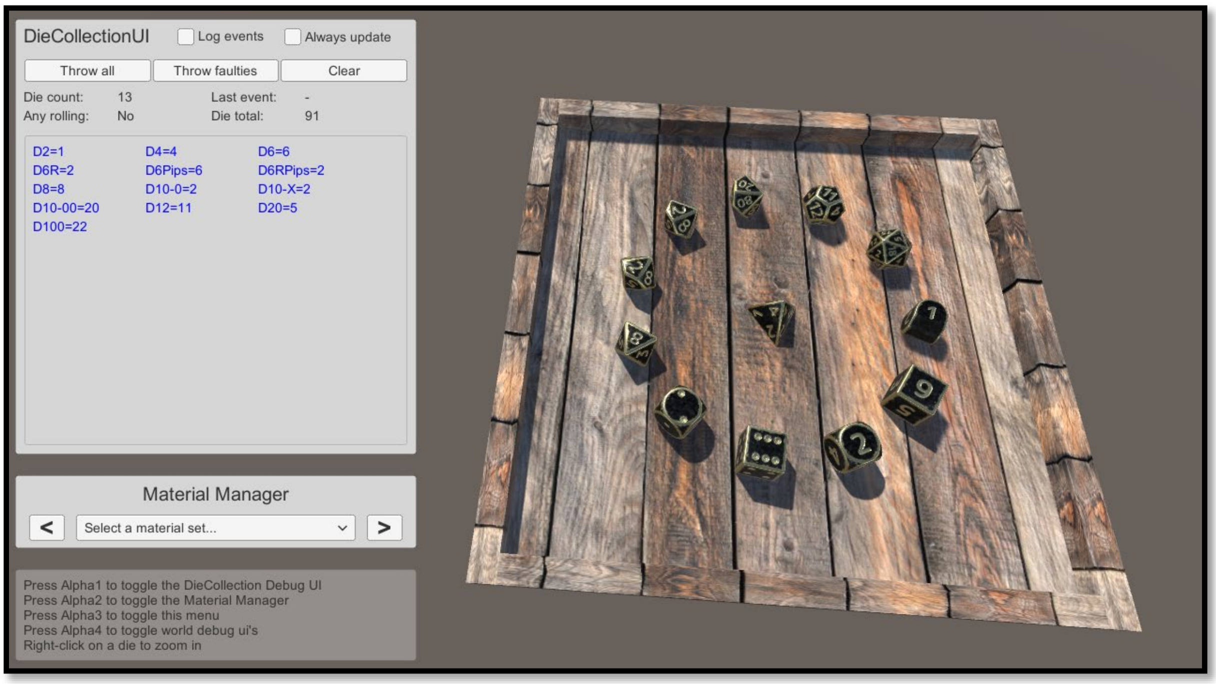
DICE (CREATOR) PACK 1.2 MANUAL



Die count: 13 Last event: -
Any rolling: No Die total: 91

D2=1	D4=4	D6=6
D6R=2	D6Pips=6	D6RPips=2
D8=8	D10-0=2	D10-X=2
D10-00=20	D12=11	D20=5
D100=22		

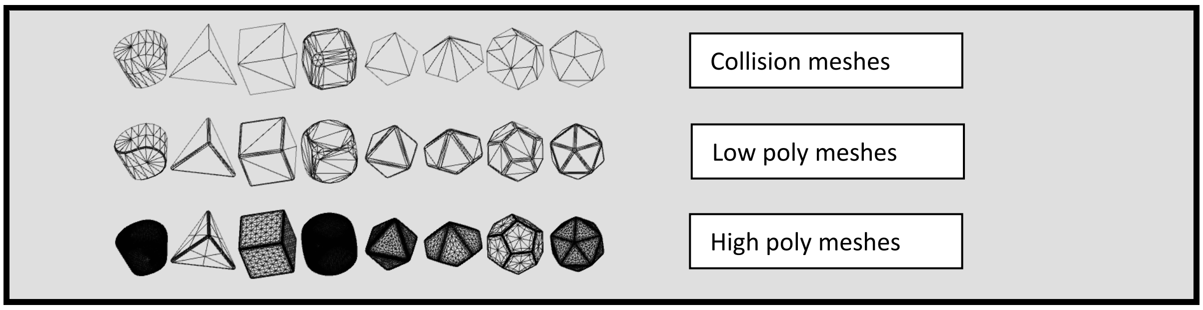
Press Alpha1 to toggle the DieCollection Debug UI
Press Alpha2 to toggle the Material Manager
Press Alpha3 to toggle this menu
Press Alpha4 to toggle world debug ui's
Right-click on a die to zoom in

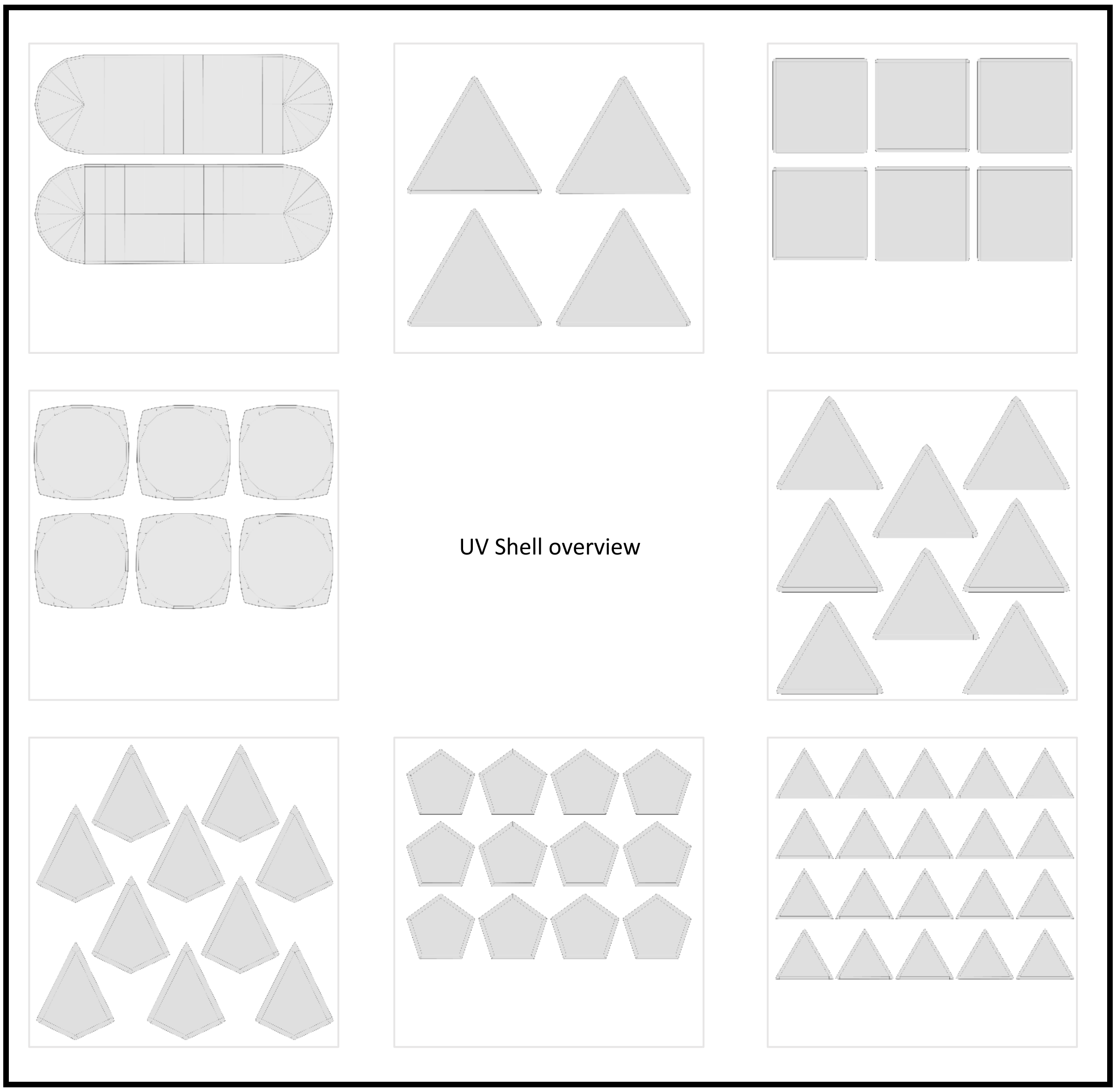
TABLE OF CONTENTS

CONTENTS, FEATURES & TARGET AUDIENCE

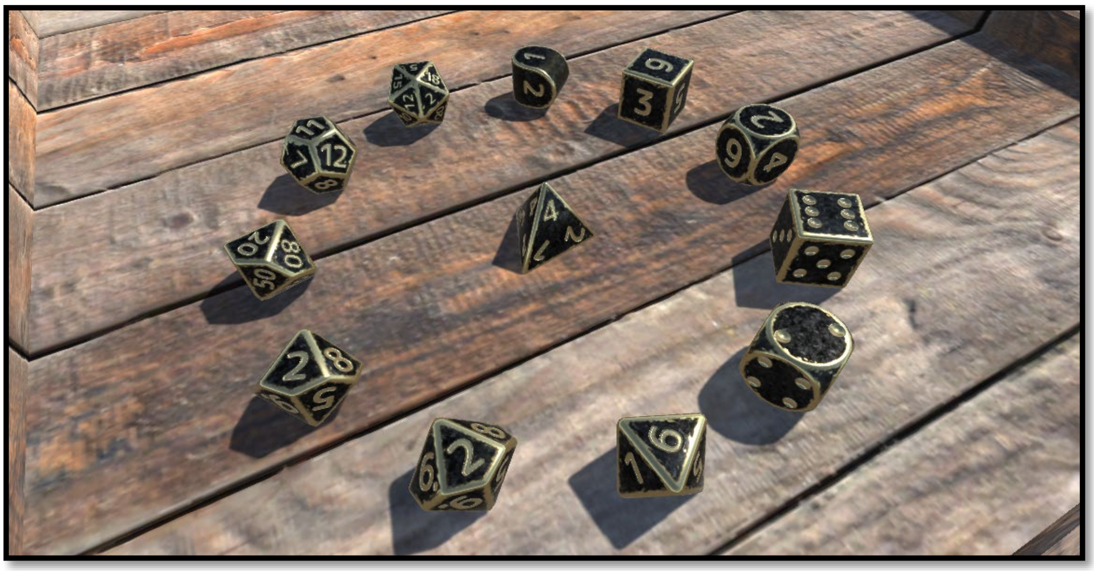
CONTENTS

- Models (.blend/.fbx) for the following RPG dice types:
  D2, D4, D6 (cubic & rounded), D8, D10, D12 & D20 in three formats:
  - ultra low poly ***collision*** meshes
  - low poly including ***well laid out UV's shells***
  - high poly for ***baking*** in e.g. Substance Painter

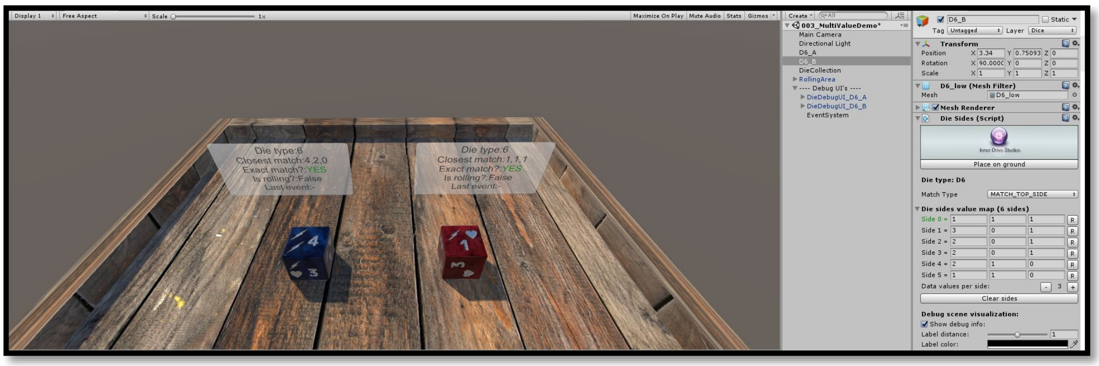




- 12 ready-to-roll *physical* die prefabs: a D2, a D4, a cubic numbered D6, a rounded numbered D6, a cubic pip-based D6, a rounded pip-based D6, a 0-based D10 (0, 1, 2, etc), an X-based D10 (X, 1, 2, etc), a 10-based D10 (10, 20, 30, etc), a D12 and last but not least a D20.



- a multivalued dice example such as you might find in a board game like Descent.



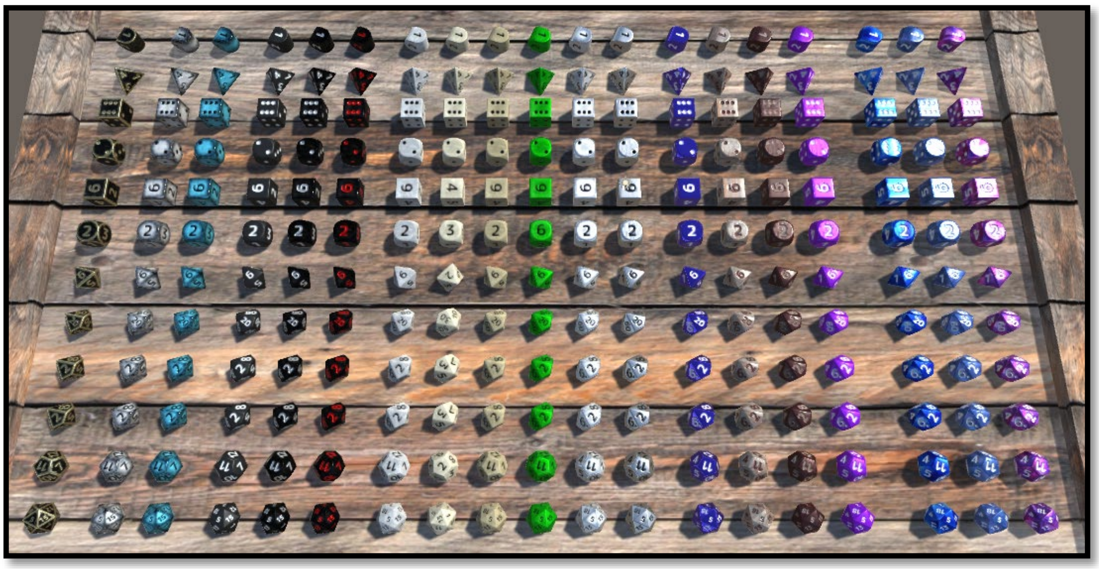
Die type:6
Closest match:4,2,0
Exact match?:YES
Is rolling?:False
Last event:-

Die type:6
Closest match:1,1,1
Exact match?:YES
Is rolling?:False
Last event:-

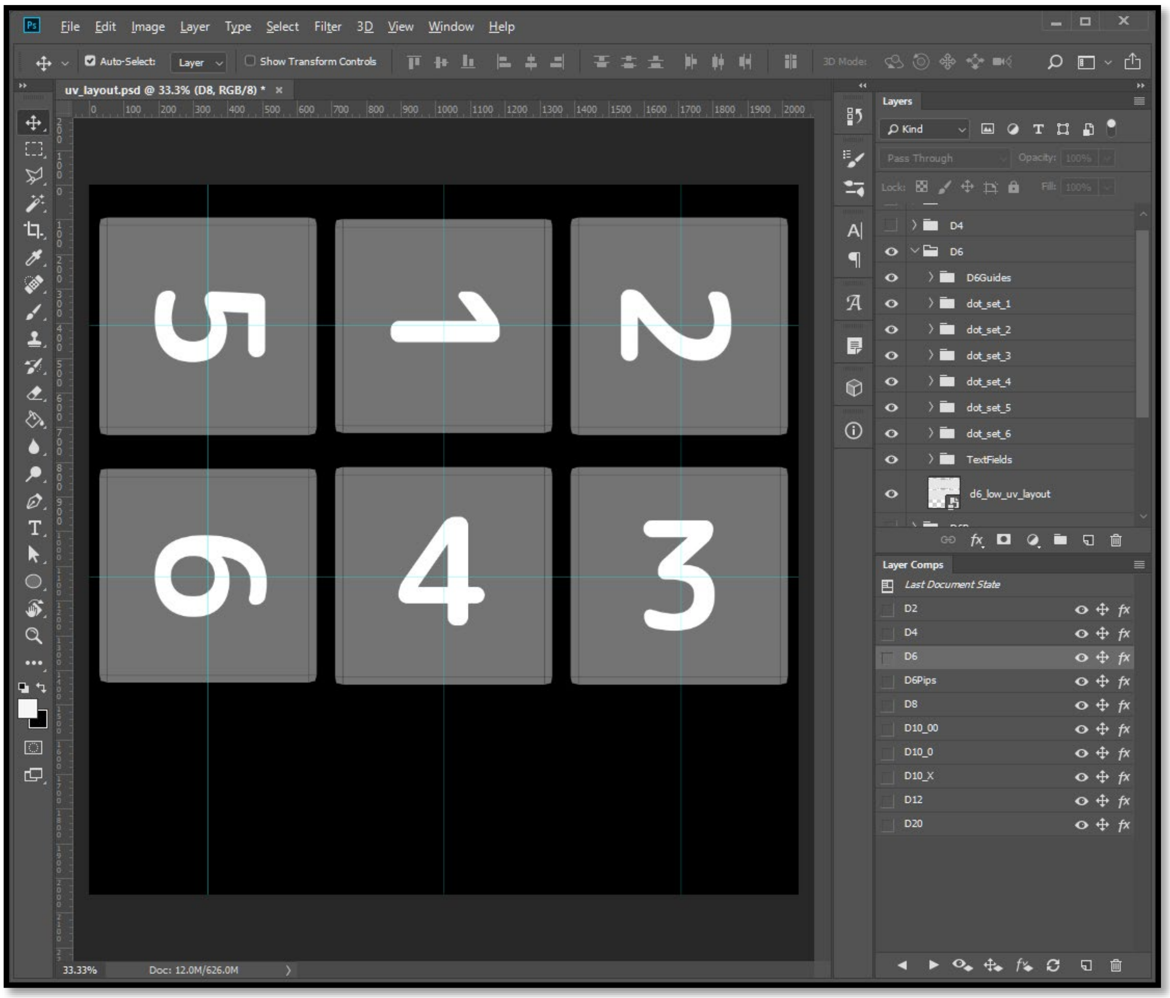
003_MultiValueDemo*
Main Camera
Directional Light
D6_A
D6_B
DieCollection
RollingArea
---- Debug UI's ----
DieDebugUI_D6_A
DieDebugUI_D6_B
EventSystem

D6_B Static
Tag Untagged Layer Dice
Transform
Position X 3.34 Y 0.75093 Z 0
Rotation X 90.0000 Y 0 Z 0
Scale X 1 Y 1 Z 1
D6_low (Mesh Filter)
Mesh D6_low
Mesh Renderer
Die Sides (Script)
Inner Drive Studios
Place on ground
Die type: D6
Match Type MATCH_TOP_SIDE
Die sides value map (6 sides)
Side 0 = 1 1 1 R
Side 1 = 3 0 1 R
Side 2 = 2 0 1 R
Side 3 = 2 0 1 R
Side 4 = 2 1 0 R
Side 5 = 1 1 0 R
Data values per side: - 3 +
Clear sides
Debug scene visualization:
Show debug info:
Label distance: 1
Label color:

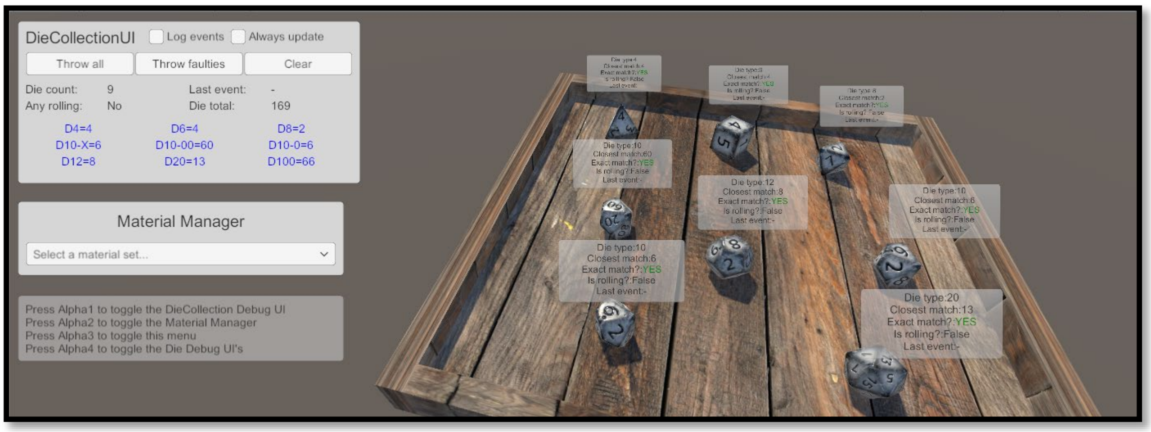
- 19 ready to use texture/materials packs including explanations on how to create your own. Since version 1.2 all materials use the standard convention where opposite sides always add up to the same total (except for the D2 and D4 of course):



- A Photoshop (.psd) template file containing uv layouts for all dice models and text elements at the correct position, scale & rotation allowing you to either create your own dice materials or fabricate black & white bitmap masks to use in Substance Painter or Unity:



- several debug / test UI's to help you construct your dice:


DieCollectionUI ☐ Log events ☐ Always update

Throw all | Throw faulties | Clear

Die count: 9 Last event: -
Any rolling: No Die total: 169

D4=4 D6=4 D8=2
D10-X=6 D10-00=60 D10-0=6
D12=8 D20=13 D100=66

Material Manager

Select a material set...

Press Alpha1 to toggle the DieCollection Debug UI
Press Alpha2 to toggle the Material Manager
Press Alpha3 to toggle this menu
Press Alpha4 to toggle the Die Debug UI's

Die type:10
Closest match:60
Exact match?:YES
Is rolling?:False
Last event:-

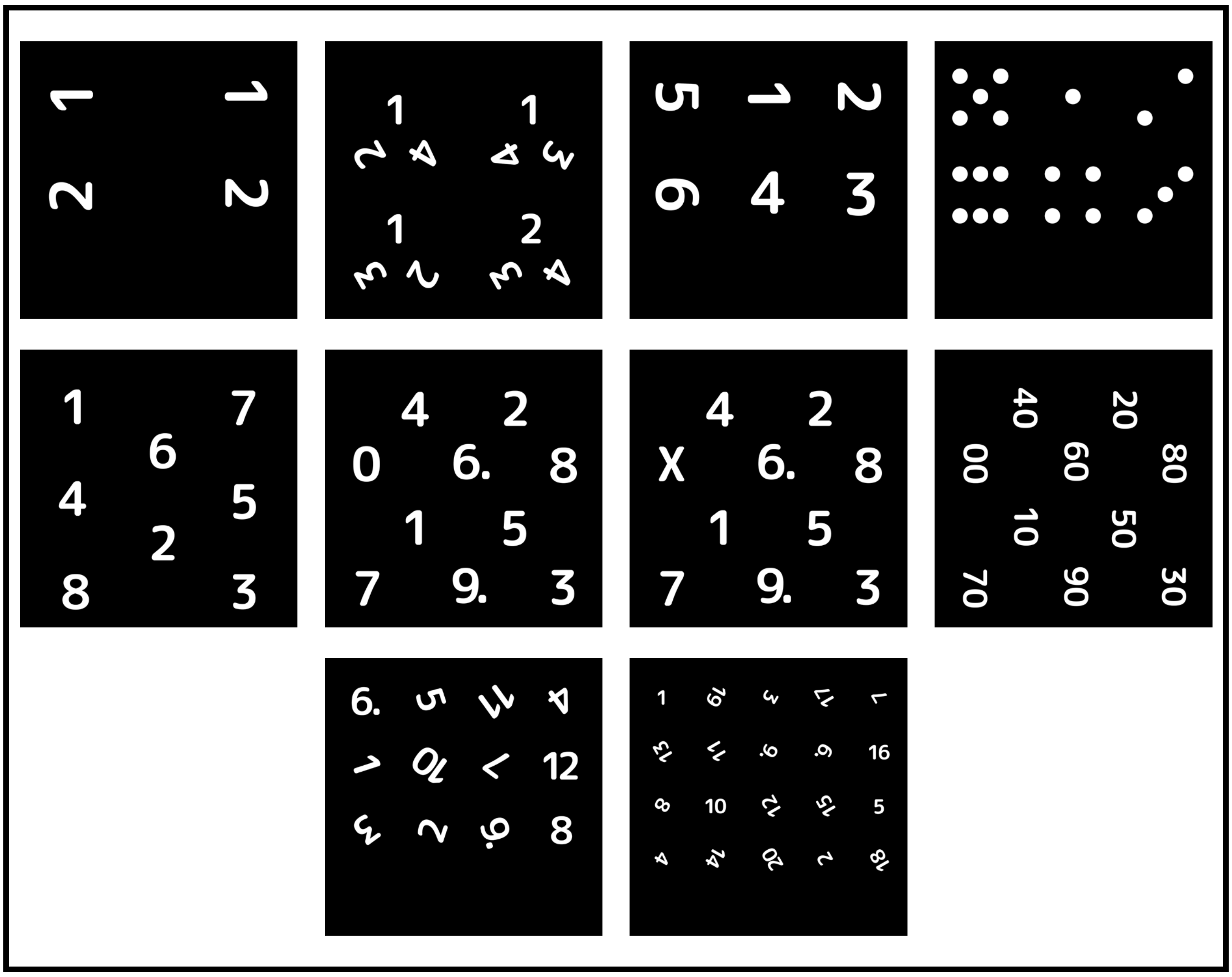
Die type:12
Closest match:8
Exact match?:YES
Is rolling?:False
Last event:-

Die type:10
Closest match:6
Exact match?:YES
Is rolling?:False
Last event:-

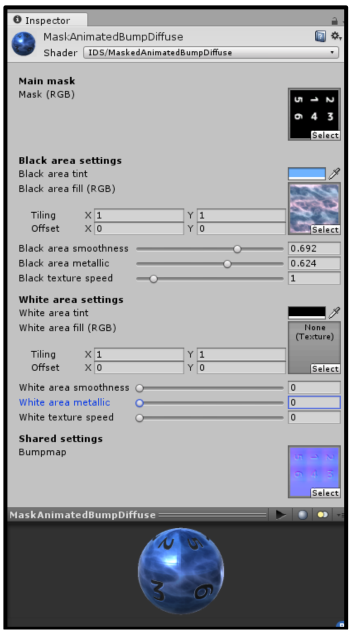
Die type:10
Closest match:6
Exact match?:YES
Is rolling?:False
Last event:-

Die type:20
Closest match:13
Exact match?:YES
Is rolling?:False
Last event:-

- A default set of already exported black & white masks ready to be used in either Substance Painter as bitmap masks or directly in Unity as masks while creating your own dice materials:



- 3 different variations of mask shaders allowing you to quickly create your own dice materials using black and white input masks such as the ones above and different texture variations:



Inspector
MaskAnimatedBumpDiffuse
Shader IDS/MaskedAnimatedBumpDiffuse

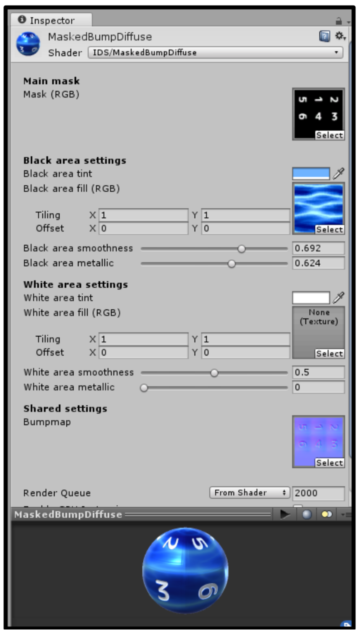
Main mask
Mask (RGB)

Black area settings
Black area tint
Black area fill (RGB)
Tiling X 1 Y 1
Offset X 0 Y 0
Black area smoothness 0.692
Black area metallic 0.624
Black texture speed 1

White area settings
White area tint
White area fill (RGB) None (Texture)
Tiling X 1 Y 1
Offset X 0 Y 0
White area smoothness 0
White area metallic 0
White texture speed 0

Shared settings
Bumpmap

MaskAnimatedBumpDiffuse



Inspector
MaskedBumpDiffuse
Shader IDS/MaskedBumpDiffuse

Main mask
Mask (RGB)

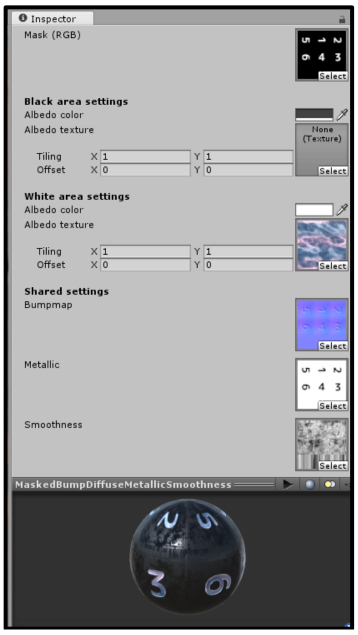
Black area settings
Black area tint
Black area fill (RGB)
Tiling X 1 Y 1
Offset X 0 Y 0
Black area smoothness 0.692
Black area metallic 0.624

White area settings
White area tint
White area fill (RGB) None (Texture)
Tiling X 1 Y 1
Offset X 0 Y 0
White area smoothness 0.5
White area metallic 0

Shared settings
Bumpmap

Render Queue From Shader 2000

MaskedBumpDiffuse



Inspector
Mask (RGB)

Black area settings
Albedo color
Albedo texture None (Texture)
Tiling X 1 Y 1
Offset X 0 Y 0

White area settings
Albedo color
Albedo texture
Tiling X 1 Y 1
Offset X 0 Y 0

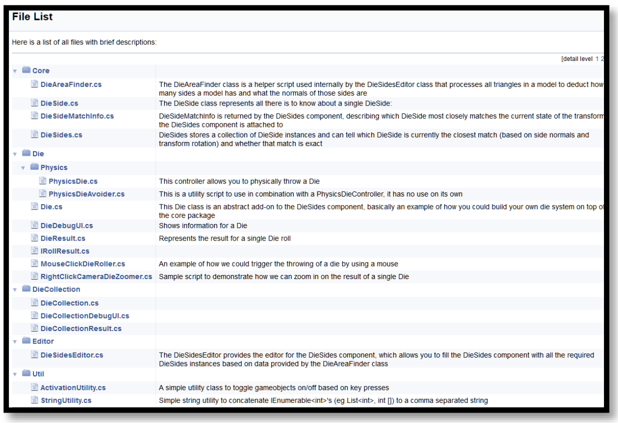
Shared settings
Bumpmap

Metallic

Smoothness

MaskedBumpDiffuseMetallicSmoothness

- fully documented MonoBehaviours and Editor scripts that allow you to create custom dice from scratch or modify anything you'd like to see differently (see features below for more info):

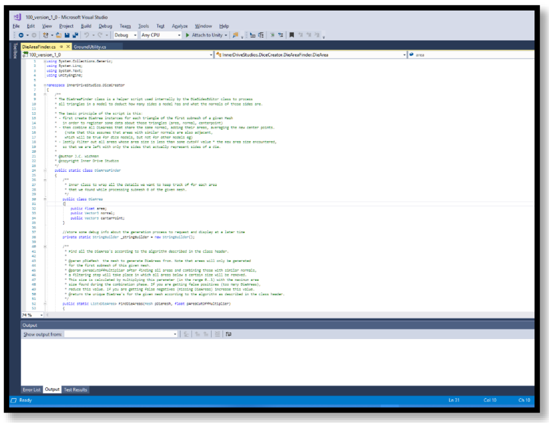


File List

Here is a list of all files with brief descriptions:

[detail level 1 2

▼ Core	
DieAreaFinder.cs	The DieAreaFinder class is a helper script used internally by the DieSidesEditor class that processes all triangles in a model to deduct how many sides a model has and what the normals of those sides are
DieSide.cs	The DieSide class represents all there is to know about a single DieSide:
DieSideMatchInfo.cs	DieSideMatchInfo is returned by the DieSides component, describing which DieSide most closely matches the current state of the transform the DieSides component is attached to
DieSides.cs	DieSides stores a collection of DieSide instances and can tell which DieSide is currently the closest match (based on side normals and transform rotation) and whether that match is exact
▼ Die	
▼ Physics	
PhysicsDie.cs	This controller allows you to physically throw a Die
PhysicsDieAvoider.cs	This is a utility script to use in combination with a PhysicsDieController, it has no use on its own
Die.cs	This Die class is an abstract add-on to the DieSides component, basically an example of how you could build your own die system on top of the core package
DieDebugUI.cs	Shows information for a Die
DieResult.cs	Represents the result for a single Die roll
IRollResult.cs	
MouseClickDieRoller.cs	An example of how we could trigger the throwing of a die by using a mouse
RightClickCameraDieZoomer.cs	Sample script to demonstrate how we can zoom in on the result of a single Die
▼ DieCollection	
DieCollection.cs	
DieCollectionDebugUI.cs	
DieCollectionResult.cs	
▼ Editor	
DieSidesEditor.cs	The DieSidesEditor provides the editor for the DieSides component, which allows you to fill the DieSides component with all the required DieSides instances based on data provided by the DieAreaFinder class
▼ Util	
ActivationUtility.cs	A simple utility class to toggle gameobjects on/off based on key presses
StringUtility.cs	Simple string utility to concatenate IEnumerable<int>'s (eg List<int>, int []) to a comma separated string

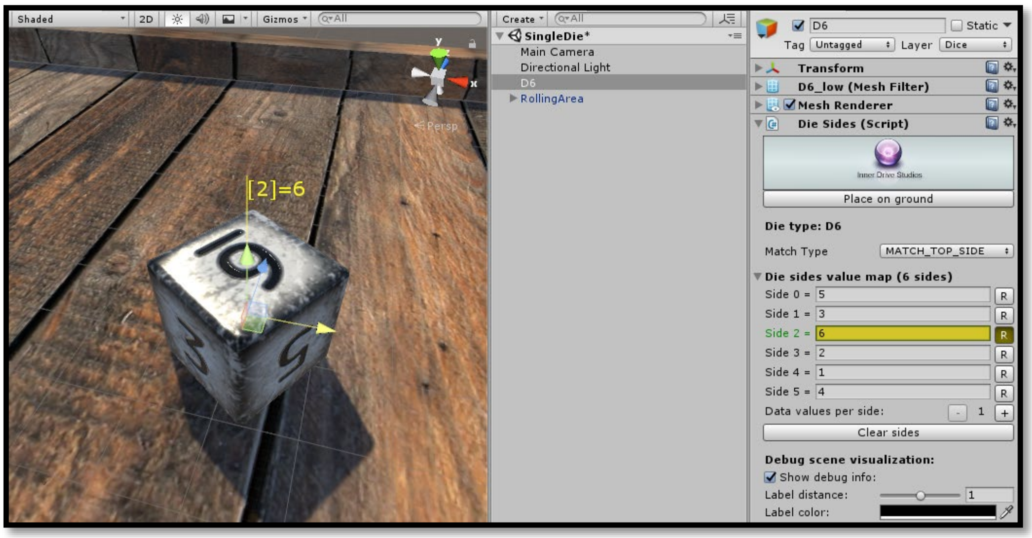

100_version_1_0 - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Attach to Unity
DieAreaFinder.cs
GroundUtility.cs
100_version_1_0
InnerDriveStudios.DiceCreator.DieAreaFinder.DieArea
area
74 %
Output
Show output from:
Error List
Output
Test Results
Ready

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using UnityEngine;

namespace InnerDriveStudios.DiceCreator
{
    /**
    * The DieAreaFinder class is a helper script used internally by the DieSidesEditor class to process
    * all triangles in a model to deduct how many sides a model has and what the normals of those sides are.
    *
    * The basic principle of the script is this:
    * - first create DieArea instances for each triangle of the first submesh of a given Mesh
    *   in order to register some data about those triangles (area, normal, centerpoint)
    * - then combine all DieAreas that share the same normal, adding their areas, averaging the new center points.
    *    (note that this assumes that areas with similar normals are also adjacent,
    *    which will be true for dice models, but not for other models eg)
    * - lastly filter out all areas whose area size is less than some cutoff value * the max area size encountered,
    *   so that we are left with only the sides that actually represent sides of a die.
    *
    * @author J.C. Wichman
    * @copyright Inner Drive Studios
    */
    public static class DieAreaFinder
    {
        /**
         * Inner class to wrap all the details we want to keep track of for each area
         * that we found while processing submesh 0 of the given mesh.
         */
        public class DieArea
        {
            public float area;
            public Vector3 normal;
            public Vector3 centerPoint;
        }

        //store some debug info about the generation process to request and display at a later time
        private static StringBuilder _stringBuilder = new StringBuilder();

        /**
         * Find all the DieArea's according to the algorithm described in the class header.
         *
         * @param pDieMesh  the mesh to generate DieAreas from. Note that areas will only be generated
         * for the first submesh of this given mesh.
         * @param pAreaCutOffMultiplier after finding all areas and combining those with similar normals,
         * a filtering step will take place in which all areas below a certain size will be removed.
         * This size is calculated by multiplying this parameter (in the range 0..1) with the maximum area
         * size found during the combination phase. If you are getting false positives (too many DieAreas),
         * reduce this value. If you are getting false negatives (missing DieAreas) increase this value.
         * @return the unique DieArea's for the given mesh according to the algorithm as described in the class header.
         */
        public static List<DieArea> FindDieAreas(Mesh pDieMesh, float pAreaCutOffMultiplier)
        {
```

- a SideUtility to attach prefabs to all sides of a die as an alternative to the provided material solution
- various scenes clearly demonstrating all the different features, explained in detail further along in this document.

FEATURES

- the ability to reuse existing dice or create your own dice from scratch including a handy editor to fill in data for all die sides quickly and easily. Side detection is based on an algorithm which processes all triangles in a die mesh, filtering out false positives, to end up with a correct set of die sides and normals to aid the 'which-side-is-up-detection' process, foregoing methods that depend on aligning trigger colliders or other awkward solutions:



D6 Static
Tag Untagged Layer Dice

Die type: D6

Match Type MATCH_TOP_SIDE

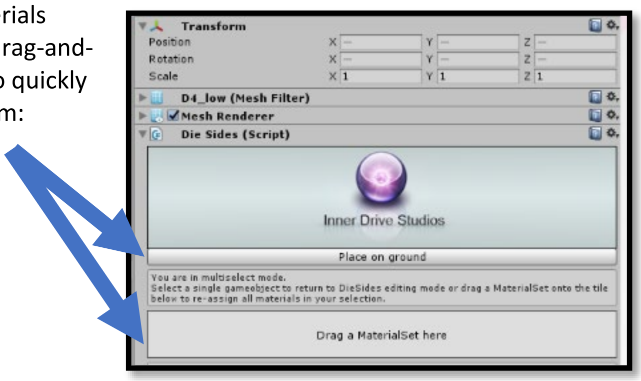
Die sides value map (6 sides)

Side 0 = 5 R
Side 1 = 3 R
Side 2 = 6 R
Side 3 = 2 R
Side 4 = 1 R
Side 5 = 4 R
Data values per side: - 1 +

Debug scene visualization:

☑ Show debug info:
Label distance: 1
Label color:

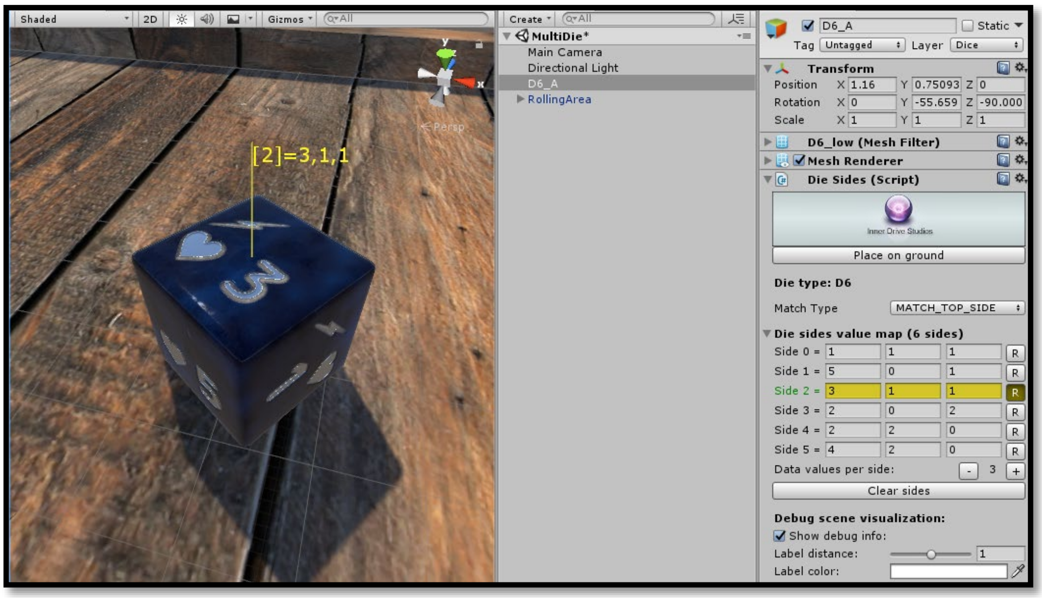
- the ability to quickly change the materials assigned to a set of selected dice by drag-and-drop MaterialSets **&** handy buttons to quickly align all dice with the floor below them:



Position X — Y — Z —
Rotation X — Y — Z —
Scale X 1 Y 1 Z 1

You are in multiselect mode.
Select a single gameobject to return to DieSides editing mode or drag a MaterialSet onto the tile below to re-assign all materials in your selection.

- the ability to have different sets of data for each die side:



D6_A Static
Tag Untagged Layer Dice

Transform
Position X 1.16 Y 0.75093 Z 0
Rotation X 0 Y -55.659 Z -90.000
Scale X 1 Y 1 Z 1

D6_low (Mesh Filter)
Mesh Renderer
Die Sides (Script)
Inner Drive Studios
Place on ground

Die type: D6

Match Type MATCH_TOP_SIDE

Die sides value map (6 sides)

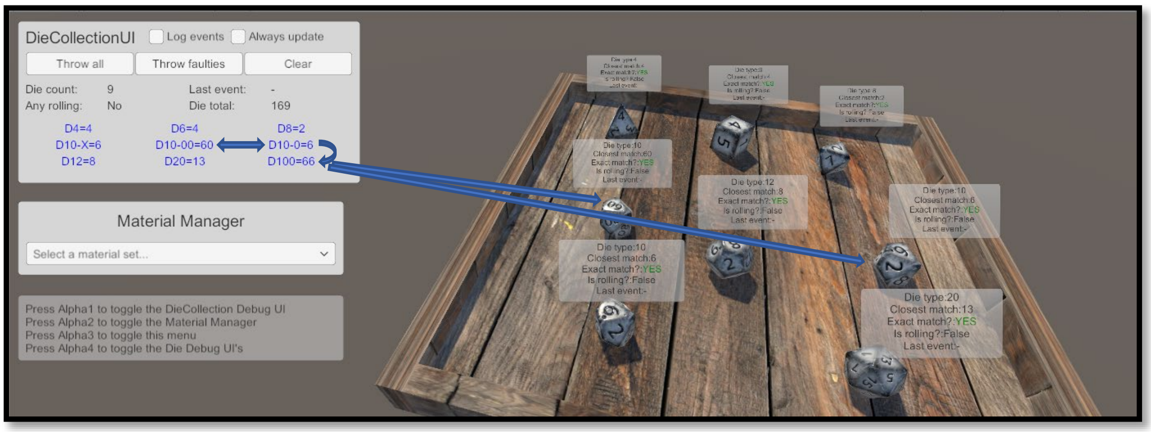
Side 0 =	1	1	1
Side 1 =	5	0	1
Side 2 =	3	1	1
Side 3 =	2	0	2
Side 4 =	2	2	0
Side 5 =	4	2	0

Data values per side: 3

Clear sides

Debug scene visualization:
Show debug info:
Label distance: 1
Label color:

- the ability to create collections of dice and even collections of collections of dice to quickly and easily retrieve totals for all dice in a collection and implement advanced features such as D100's


DieCollectionUI Log events Always update
Throw all Throw faulties Clear

Die count: 9 Last event: -
Any rolling: No Die total: 169

D4=4 D6=4 D8=2
D10-X=6 D10-00=60 D10-0=6
D12=8 D20=13 D100=66

Material Manager
Select a material set...

Press Alpha1 to toggle the DieCollection Debug UI
Press Alpha2 to toggle the Material Manager
Press Alpha3 to toggle this menu
Press Alpha4 to toggle the Die Debug UI's

- event based mechanism (code only, but easy to use) to detect when a roll (single or collection) starts or ends and easily retrieve the result of a roll including whether all the values were exact or not:

Rollable events (eg Die & DieCollection)

```
/**
 * Triggered when isRolling toggles from false to true.
 */
public virtual event Action<ARollable> OnRollBegin = delegate { };

/**
 * Triggered when isRolling toggles from true to false.
 */
public event Action<ARollable> OnRollEnd = delegate { };

/**
 * Triggered when ClearEndResult is called.
 */
public event Action<ARollable> OnEndResultCleared = delegate { };
```

Events thrown by DieCollections only

```
/**
 * Triggered when a child within the collection starts rolling.
 * The child is passed as second parameter to the event handler.
 */
public event Action<DieCollection, ARollable> OnChildRollBegin = delegate { };

/**
 * Triggered when a child within within the collection stops rolling.
 * The child is passed as second parameter to the event handler.
 */
public event Action<DieCollection, ARollable> OnChildRollEnd = delegate { };

/**
 * Triggered when the result for a child within the collection is cleared.
 * The child is passed as second parameter to the event handler.
 */
public event Action<DieCollection, ARollable> OnChildEndResultCleared = delegate { };
```

TARGET AUDIENCE

This package can be used in multiple ways in increasing levels of complexity:

- Use the provided prefabs, materials and starting code to implement dice into your game
- Use the provided prefabs and starting code but create your own materials in Unity using the provided mask shaders & photoshop template, or in Substance Painter (advanced).
- Use the provided meshes and code to create your dice from scratch, using custom values, choosing between single or multiple values per die side
- Use only the provided meshes, code everything yourself using the core utilities and code to tailor everything exactly to your needs, using the project as a learning resource/starting point

In other words, this package provides everything you need to start using the most common (roleplaying) dice, whether you use the provided prefabs/models or your own, whether you use the given materials or create your own, whether you use the provided die side detection system or implement your own, whether you use the provided event based die rolling system or built your own on top of the basic die side detection system.

GOOD TO KNOW

- The values on the various dice are visualized to the user through the use of textures, not by attaching a textfield or anything to each side (although a Side Utility is included since version 1.2). Combined with the low poly die models, collision models and the maps that have been created by baking the details of a high poly mesh onto the low poly models, this ensures high performance without losing any visual fidelity.
- Each die has its own material, if you would like to put all textures in one atlas you will need to use a tool such as Mesh Baker.

SCENE OVERVIEW

This section gives you a quick overview of the provided scenes so that you know which scene to investigate for some more information. More information on specific prefabs & components is provided in the next section. Recreating the 001_GettingStarted scene from scratch is discussed in the 'How do I ...' section at the end.

Scene	**Description**
001_GettingStarted	This scene gives an overview of almost all basic features in one go. It contains prefabs for all different die types, provides an example of a preconstructed area to roll dice in, demonstrates collections, material sets, and the different debug UI's. Run the demo and inspect all the different components for more info. Things to note: • the construction of the 'RollingArea' especially the layer & collider setup • the DieCollection setup especially how the D100 is constructed
002_MultiValueDemo	This scene provides an example of using multi valued dice, eg dice with more than one value per side. Things to note: • how the multiple values have been setup. Note that which column stores which data is not something which is explicitly defined. In other words, whether column 1 contains the range and column 2 the health or vice versa is up to you.
003_MultiMaterialDemo	This scene provides an example of all available materials/materialsets, demonstrating the use of the Unity Standard shader in combination with textures created through Substance Painter, but also the use of the provided MaskShader which allows you to create material variations without going through Substance Painter. Use of this MaskShader is new since version 1.2 and replaces the overlay method of multiple materials from previous versions. See the 'Materials' chapter for more info on the provided materials/materialsets/textures and creating your own materials.
004_DieSideUtilityDemo	This scene contains the results from applying the Side Utility that has been added in version 1.2. Although this is not the recommended way to visualize die faces, in very specific scenario's where the faces should be changeable during gameplay, this might come in handy. This utility is described in the Materials chapter. The scene demonstrates two different approaches, one with Sprites being used as die faces and one with a TextMesh being used as a die face. Note that these are just examples, and not meant as production ready assets perse.

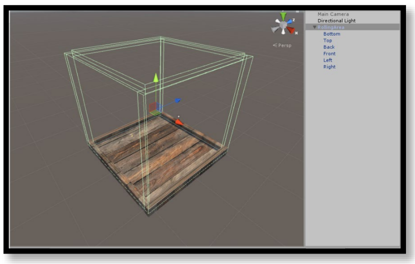
PREFABS & COMPONENTS OVERVIEW

Although all objects in the provided demo scenes can be created from scratch of course, it is often easier to start from existing prefabs. This section gives an overview of how the provided prefabs have been set up and what the provided components do. Please refer to the API docs or the files & folder overview if you need help finding the referenced components in the project.

We'll describe each prefab briefly with an overview of its components, followed by a more in depth description of the prefab's components. Components not directly tied to a prefab will be discussed last.

ROLLINGAREA PREFAB

The RollingArea prefab is a simple space enclosed by 6 colliders to make sure the dice can't escape the rolling area. If you want the dice to respond to mouse clicks, make sure all these colliders *except* the Bottom are on the 'Ignore Raycast' layer. In addition to avoid ending up with dice leaning 'into' a wall, the Back, Front, Left and Right walls have a Physics Die Avoider component attached (see Dice Prefabs and onwards).



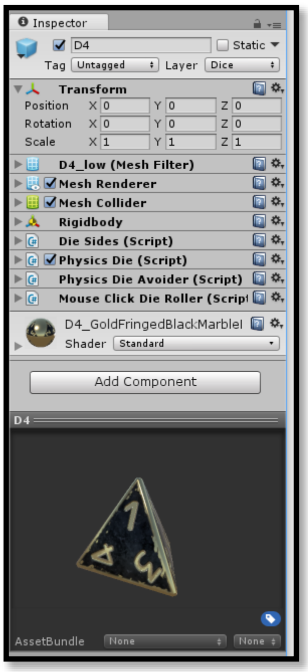
DICE PREFABS (D2, D4, D6, D6PIPS, D6R, D6RPIPS, D8, D10-0, D10-X, D10-00, D12, D20)

Each die prefab implements a die under the influence of physics. This implies at least a number of standard Unity components have to present: Transform, Mesh Filter, Mesh Renderer, Mesh Collider & Rigidbody.

Although the dice (except the rounded D6) will also work without, each MeshCollider references a specific even lower poly *collision mesh* as it improves the functionality and performance of the dice.

In addition a number of die specific components have been added:

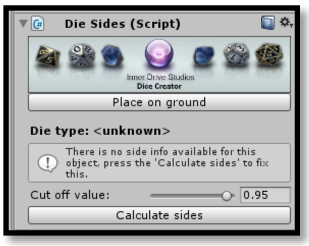
- the DieSides component which stores all the information about the sides of die and the data for each side
- the PhysicsDie component which subclasses the Die class to implement a physics based die based on the data provided by the DieSides component
- the PhysicsDie Avoider component which improves the behaviour of the die by making sure it doesn't end up leaning into a wall or another die leading to a 'non exact' value.
- and last but not least the Mouse Click Die Roller which simply looks for something of type Die and calls .Roll() on mouse up.



```
Inspector
☑ D4                          ☐ Static
Tag Untagged   Layer Dice
Transform
Position   X 0   Y 0   Z 0
Rotation   X 0   Y 0   Z 0
Scale      X 1   Y 1   Z 1
D4_low (Mesh Filter)
☑ Mesh Renderer
☑ Mesh Collider
Rigidbody
Die Sides (Script)
☑ Physics Die (Script)
Physics Die Avoider (Script)
Mouse Click Die Roller (Scrip
D4_GoldFringedBlackMarbleI
Shader Standard
Add Component
D4
AssetBundle None   None
```

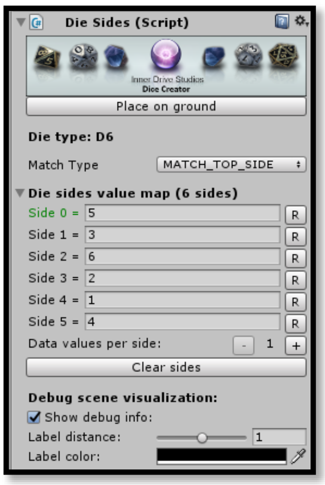
DIESIDES COMPONENT

The DieSides component stores all the information about the sides of a die and the data for each side. When you first add an instance of this component, it will tell you that the Die type is ***<unknown>*** and the only option besides 'Place on ground' is 'Calculate sides'.



```
Die Sides (Script)
Inner Drive Studios
Dice Creator
Place on ground
Die type: <unknown>
There is no side info available for this object, press the 'Calculate sides' to fix this.
Cut off value:  0.95
Calculate sides
```

'***Calculate sides***' searches for a mesh on the same GameObject the DieSides component is attached to, first by looking for MeshCollider and if that can't be found, for a MeshRenderer. Once a mesh is found, all triangles in the mesh are processed to locate a unique set of areas that represent the die sides, printing data about this process as it progresses.


Die Sides (Script)

Die type: D6
Match Type: MATCH_TOP_SIDE
Die sides value map (6 sides)
Side 0 = 5
Side 1 = 3
Side 2 = 6
Side 3 = 2
Side 4 = 1
Side 5 = 4
Data values per side: 1

Debug scene visualization:
✓ Show debug info:
Label distance: 1
Label color:

The '***Cut off value***' determines which areas get pruned during this process; if you end up with not enough sides, this value should be increased, if you end up with too many sides, the value should be decreased.

The '***Match Type***' determines whether side normals aligned with the world up or down vector are considered to indicate a matching side. This should be MATCH_TOP_SIDE for anything but D4's.

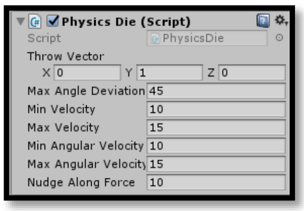
'***Place on ground***' calculates the bounds of the rotated die and raycasts down to find a 'floor' to put the die on. If the die sides have been detected correctly it also rotates the die first to make sure a flat side is aligned with the floor.

The '***Die sides value map***' is created by clicking the '***Calculate sides***' and allows you to enter data for each side. Toggling '***R***' (de)activates the selected side in the viewport, hiding/showing all other normals. If you are implementing a complex die, you can also add more data values per side by clicking the '**+**' button. Of course removing is also possible through the '**-**' button.

The '***Debug scene visualization***' has no effect whatsoever on the functionality of the die, only on the visuals in the scene view during development.

If you select **multiple game objects** with a Die Sides component, the interface changes so you can drag a MaterialSet on multiple instances at once, changing all the materials used in one go (for single die selections, just assign the correct material directly). Also see the Materials chapter.

PHYSICSDIE COMPONENT

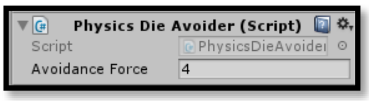

Physics Die (Script)
Script: PhysicsDie
Throw Vector X 0 Y 1 Z 0
Max Angle Deviation 45
Min Velocity 10
Max Velocity 15
Min Angular Velocity 10
Max Angular Velocity 15
Nudge Along Force 10

The PhysicsDie component subclasses the Die class to implement a physics based die, based on the data provided by the DieSides component. The component has two ways to roll the die, by calling Roll() without parameters, causing the inspector parameters to be used, or by passing in specific parameters for the throw vector etc, causing the inspector parameters to be overwritten.

This means that if a die is rolled as part of a collection by triggering Roll() on the collection, the inspector parameters will be used. If you want to *change* this behaviour, you are best of subclassing DieCollection in order to tweak the default behaviour.

Parameter	**Description**
Throw Vector	Direction in which force will be applied to throw the Die.
Max angle deviation	Maximum deviation in degrees from the throw vector specified above.
Min velocity	The die will get a velocity based on the normalized deviated throw vector multiplied with a random number between min and max velocity.
Max velocity	The die will get a velocity based on the normalized deviated throw vector multiplied with a random number between min and max velocity.
Min angular velocity	The die will get a random angular velocity between the min and max values.
Max angular velocity	The die will get a random angular velocity between the min and max values.
Nudge along force	Sometimes a Die stops while it is balancing like a coin on its edge. To prevent that you can enter a non zero nudge along force here. A value of 0 disables this feature. If you are using the low poly collision meshes, you can leave this value at 0.

PHYSICSDIEAVOIDER COMPONENT



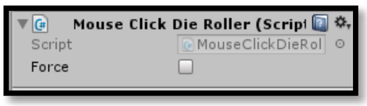
Physics Die Avoider (Script)

Script	PhysicsDieAvoider
Avoidance Force	4

Dice leaning into each other or dice leaning into a wall may cause a Die value to not be exact. To avoid this you can attach a PhysicsDieAvoider to any objects you want the dice to avoid, this includes both dice and environment walls for example (but not the floor or the dice will be "bouncin'-all-day" !).

Upon collision an '***Avoidance Force***' will be added to the objects that actually have a non kinematic RigidBody (so although you can attach this to a wall, the wall itself will not be blown away for example). This causes dice to roll away from each other or from the wall, and reduces the chance of getting these aforementioned non exact values.

MOUSECLICKDIEROLLER COMPONENT



Mouse Click Die Roller (Script

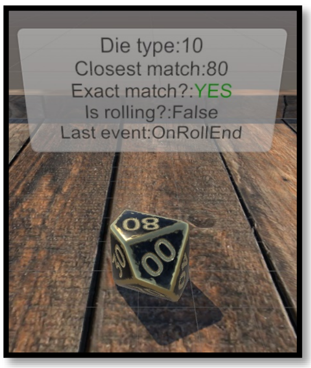
Script	MouseClickDieRol
Force	☐

Assuming the GameObject has a collider attached, this component will simply look for any component of type 'Die', such as 'PhysicsDie' and call its 'Roll()' method.

The '***Force***' parameter specifies whether 'Roll()' will be called even if the die is already rolling.

DIEDEBUGUI PREFAB & COMPONENT

PREFAB



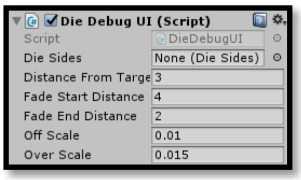
```
Die type:10
Closest match:80
Exact match?:YES
Is rolling?:False
Last event:OnRollEnd
```

The DieDebugUI prefab can be connected to an instance of a DieSides component to display debug information about that specific instance. If the GameObject to which this instance is attached also contains an instance of the Die component (or one of its subclasses such as PhysicsDie), the prefab will also display information about any triggered Die events. As can be seen on the right, the prefab provides information on:

- the number of sides the die has
- which value is currently the closest match
- whether that match is exact
- the rolling status
- the last event triggered

By default the component zooms in on mouse over, and out on mouse out.

COMPONENT



Die Debug UI (Script)

Script	DieDebugUI
Die Sides	None (Die Sides)
Distance From Targe	3
Fade Start Distance	4
Fade End Distance	2
Off Scale	0.01
Over Scale	0.015

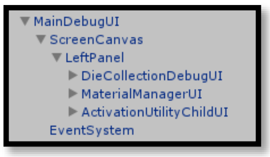
The DieDebugUI Component has 6 parameters:

Parameter	Description
Die Sides	The DieSides instance to display information for
Distance from target	The y distance between the center of the DieSides instance and the debug canvas
Fade Start Distance	If the camera is closer to the instance than this value, start fading out the canvas
Fade End Distance	If the camera is closer to the instance than this value, the canvas is completely faded out
Off scale	The scale of the component when the mouse is not over the component
Over scale	The scale of the component when the mouse is over the component

MAINDEBUGUI

The MainDebugUI is a preconstructed node that contains a correctly setup Canvas and EventSystem to display debug information for a collection of dice. Expanding ScreenCanvas/LeftPanel, we find three different panels:

- DieCollectionDebugUI
- MaterialManagerUI
- ActivationUtilityUI



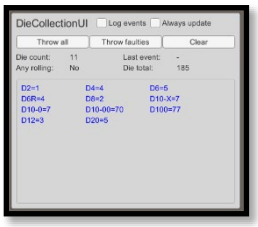
```
▼ MainDebugUI
  ▼ ScreenCanvas
    ▼ LeftPanel
      ▶ DieCollectionDebugUI
      ▶ MaterialManagerUI
      ▶ ActivationUtilityChildUI
    EventSystem
```

All these panels are discussed below. For an overview of the required steps to use the MainDebugUI in an empty scene, please refer to the 'How do I ...' section.

DIECOLLECTIONDEBUGUI PREFAB & COMPONENT

PREFAB

The DieCollectionDebugUI prefab is similar to the DieDebugUI prefab, but instead of displaying information for a single die, it displays information for a collection of dice. Just like the DieDebugUI prefab, it is not meant to be used in production, but rather to provide a starting pointing, demonstrating how to register for events, check the status of dice, get collection totals and so forth.



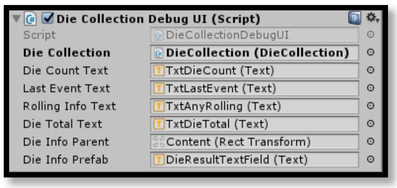
```
Die count:   11        Last event:  -
Any rolling: No        Die total:   185

D2=1        D4=4         D6=5
D6R=4       D8=2         D10-X=7
D10-0=7     D10-00=70    D100=77
D12=3       D20=5
```

Checking '***Log events***' causes event information to be written also to the Debug.Log.

Checking '***Always update***' continuously retrieves the die results instead of only when an event happens.

COMPONENT

The DieCollectionDebugUI component has everything set up already except the '***Die Collection***' reference. Just drag in any collection for which you would like to display the information at runtime.



Die Collection Debug UI (Script)

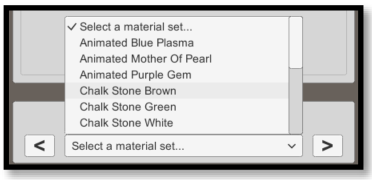
Script	DieCollectionDebugUI
Die Collection	**DieCollection (DieCollection)**
Die Count Text	TxtDieCount (Text)
Last Event Text	TxtLastEvent (Text)
Rolling Info Text	TxtAnyRolling (Text)
Die Total Text	TxtDieTotal (Text)
Die Info Parent	Content (Rect Transform)
Die Info Prefab	DieResultTextField (Text)

MATERIALMANAGERUI PREFAB, COMPONENT & MATERIAL SETS

PREFAB

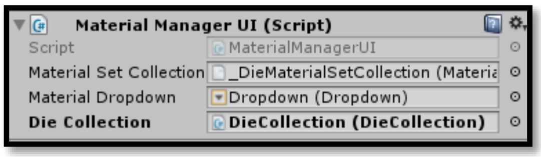
The MaterialManagerUI prefab allows you to change the material applied to a set of dice at runtime, based on the same functionality that is also used by the DieSides editor at editor time.

For more information see the component description below.



COMPONENT

To perform the runtime material switching and avoiding having to load materials from the resources folder at runtime, the Material Manager UI prefab requires a reference to a material set collection, which is an instance of the MaterialSetCollection scriptable object which in turn contains references to instances of the MaterialSet scriptable object. In addition it requires a Die Collection for which you would like to change the material. Using the MaterialSetUtility script it tries to match the listed



Material Manager UI (Script)

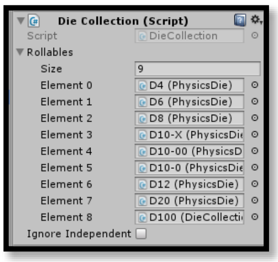
Script	MaterialManagerUI
Material Set Collection	_DieMaterialSetCollection (Materia
Material Dropdown	Dropdown (Dropdown)
Die Collection	**DieCollection (DieCollection)**

GameObjects with materials from the selected set based on naming convention, to which you should adhere closely if you want to add your own sets: everything in the game object name up to the first _ should match everything in the material name up to the first _.

You can create your own MaterialSets and MaterialSetCollection through the Assets > Create > IDS > DiceCreator menu. Another option is to copy a materials folder, rename it and run the 'Rebuild Material Sets' script (also see the Materials chapter). Please have a look at the available sets & collections in the Materials folder for more information.

OTHER COMPONENTS

DIECOLLECTION COMPONENT



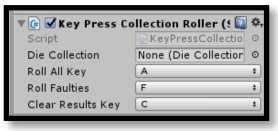
Die Collection (Script)

Script	DieCollection
Rollables	
Size	9
Element 0	D4 (PhysicsDie)
Element 1	D6 (PhysicsDie)
Element 2	D8 (PhysicsDie)
Element 3	D10-X (PhysicsDie
Element 4	D10-00 (PhysicsD
Element 5	D10-0 (PhysicsDie
Element 6	D12 (PhysicsDie)
Element 7	D20 (PhysicsDie)
Element 8	D100 (DieCollecti
Ignore Independent	☐

The DieCollection component allows you to aggregate multiple dice and even other collections into another collection, making it possible to retrieve totals and status for all those dice at once, roll & reroll all dice and even just reroll only the 'faulty' dice; those without an exact value. Dice can be added at editor time through the inspector or at runtime through code.

The 'Ignore Independent Child Events' specifies whether the collection should update its own status if the status of a child die changes due to reasons outside of the collections control. Change this in the sample scene, click a die to roll it and watch the MainDebugUI to see the difference. Keep this flag in mind if collection events are not dispatched the way you expect them to be.

KEYPRESSCOLLECTIONROLLER COMPONENT



Key Press Collection Roller (S

Script	KeyPressCollectio
Die Collection	None (Die Collectior
Roll All Key	A
Roll Faulties	F
Clear Results Key	C

The KeyPressCollectionRoller is just a simple helper component to trigger different methods on either a referenced DieCollection or one attached to the same game object.

DIEDEBUGEVENTLISTENER & DIECOLLECTIONDEBUGEVENTLISTENERCOMPONENT

Debug components similar to the presented UI's but then for use with Debug.Log only.

FOLDER & FILE OVERVIEW

An overview of all folders & files in the InnerDriveStudios/DiceCreator package of note.

Folder/file	**Description**
AdditionalResources	This folder contains a .zip file with additional resources such as the source .blend file for all models, the .psd file with UV's templates, separate .png UV templates, White on Black & Black on White masks and html documentation for the source code.
Fonts	Contains a font used for the DebugUI's and TextPrefabs in combination with the SideUtility (see the Side Utility chapter)
Materials	Contain both generic materials for non-die props, as well as folders containing all die materials, as well as MaterialSet and MaterialSetCollection instances. For more details on the materials please refer to the Materials chapter.
_Generic	Some generic non-die materials
*	folders containing all materials (D2_..., D4_... etc) for a given material set
Meshes	
Collision	Optional meshes to be used as optimized collision meshes for the dice.
Low	Low poly meshes for dice to be used in combination with the provided materials.
(High)	There is no high folder, please check the .blend file in the Additional Resources folder.
PhysicsMaterials	Contains DiePhysicsMaterial that is assigned automatically to every PhysicsDie
Prefabs	Documented in the Prefabs & Components chapter.
Scenes	Documented in the Scenes chapter.
Scripts	Documented in the Source Code chapter.
Shaders	Contains both a single sided text shader and mask shaders that you can use to define your own die materials.
Textures	Textures matching with all materials, including normal maps that you can apply to your own materials.

THE SOURCE CODE

Several source packages have been provided:

- a Core package with scripts already described above allowing you to:
  - detect all sides a die has based on the actual triangles in the mesh you are using.
  - quickly enter data values for all die sides, either simple single values for regular D2, D4, D6, D8, etc dice, but also multiple data values for dice used in a game such as Descent.
  - detect which side is currently 'up', providing info about the side itself (values, normals, etc) but also on whether the match was 'exact' or slighty off so you can choose to reroll the die.
- a Die package built on top of the Core package providing an extensible event based die rolling and result retrieving system including a physics based example of how to use it.
- a DieCollection package built on top of the Die package providing an event based system for rolling a collection of dice and retrieving their (total) values. This includes rolling collection of collections. This allows you for example to roll 3D6 or roll a D100 consisting of two (different) D10's.
- various debug classes (both console and uGUI based) to clearly demonstrate all implemented features.
- a Materials package that provides a fast way to assign a material set (eg GoldFringedBlackMarble, ErodedMarbleWhite, etc) to several dice at once without having to dig through all different materials. This is possible both at editor and at runtime. Through the use of scriptable objects it's also very easy to define your own material sets.

For more information refer directly to the provided source code, or the **zipped** html doxygen documentation provided in the ***AdditionalResources*** folder.

MATERIALS

INTRODUCTION

In this chapter we go over the different ways to create materials for the different dice and how they combine with entering the data for the different die sides. We will also have a look at the existing materials, how they have been created and how MaterialSets and MaterialSetCollections tie into all of this.

VISUALS VS DATA/COMPONENTS AND PREFAB REUSE

Basically, in this package, three things make up a die:

- how the die looks (the textures + material definitions)
- the DieSides component that allows you to enter data for each die side
- the data that has actually been entered for all the die sides and stored by the DieSides component

In this package those three are separate things, but they do have to match each other. In other words: first you define what a die looks like through a texture/material, then you assign that material to a die and then you add the DieSides component so that you can define the data for each die side and you make sure that the data you enter matches the visuals.

These last steps (adding the correct components and defining the data) are discussed in the 'Prefabs & Components overview' chapter and the 'How do I ...' chapter. However, without a good reason to *not* stick to the default layout, the simplest approach is to just reuse the provided Die prefabs as much as possible, meaning you take a die prefab including the side data already stored for that prefab, and then create materials sticking exactly to the number layout in the provided PSD template (or check the .png mask files in case you do not have photoshop) so that the visuals automatically match the data.

Assuming you are using this approach of sticking to the provided (and recommended) layout, this chapter discusses 3 main ways of creating your custom materials, but no matter which way you are using the .PSD template is at the heart of every method.

THE .PSD TEMPLATE

When it comes to creating your own materials/textures, the .PSD template is your friend. It contains folders (and layer compositions) for each die, containing a screenshot of the UV layout for a given die, guides for positioning your content, and textfields at the correct location for each die face (so you can easily change the font for all faces for example).

The location & rotation of the numbers/pips in the .PSD template was the result of three factors:

- the UV layout of the provided models
- the desire to have all opposite die sides for a specific die always add up to the same total
- the desire to have a bit of variation in how the numbers on the die sides are rotated

If you are not familiar with UV's and UV mapping: all 3D models have UV layouts which determine which part of a texture ends up where on your model. Creating materials means creating textures that respect this layout. This UV layout is fixed in the .fbx files and although the .blend file with all models has been provided, changing the UV layout will break everything, so you should have very good reasons to even consider that :).

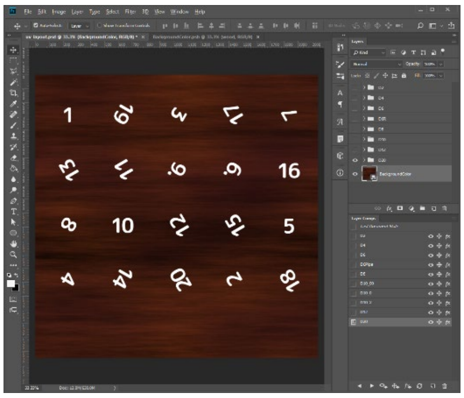
As said the .PSD template comes in handy no matter which method for texture/material creation you choose to use, the topic of the next section.

MATERIAL CREATION METHODS

This section discusses four ways to create your own textures/materials:

1. using the provided .PSD template & Unity's standard shader
2. using the provided .PSD template & the provided mask shaders
3. using Substance Painter (and the provided .PSD template)
4. (using the Side Utility => between brackets, since this is a different approach all together)

CREATING MATERIALS USING THE .PSD TEMPLATE & UNITY'S STANDARD SHADER



This is the most basic method, but also a fairly limited one. The idea is that you can you change the black background in the .psd to a texture or single color, optionally change the font of the die face textfields, maybe even paint details for specific dice and then export all layer compositions to .pngs and use them as textures for your materials. This means by default, you'll only be exporting albedo textures, if you also want to have normal maps, please search for one of the many photoshop tutorials on generating normal maps from grayscale images online.

All the black & white masks provided in the Unity project were generated this way.

Tips:

- if you make any changes in the .psd you would like to keep, be sure to update the layer compositions as well, otherwise all your changes will be reverted the moment you select a specific composition.
- if you only want to edit the background texture, make sure you edit the background smart object, that way all layer compositions will be kept in tact.

After generating the texture, you can simply use the generated textures in the albedo slots of the standard Unity material shader. Also refer to the section on MaterialSets and MaterialCollections, both in this chapter and in the 'Prefabs & Components overview' chapter.

CREATING MATERIALS USING THE PROVIDED .PSD TEMPLATE & THE PROVIDED MASK SHADERS

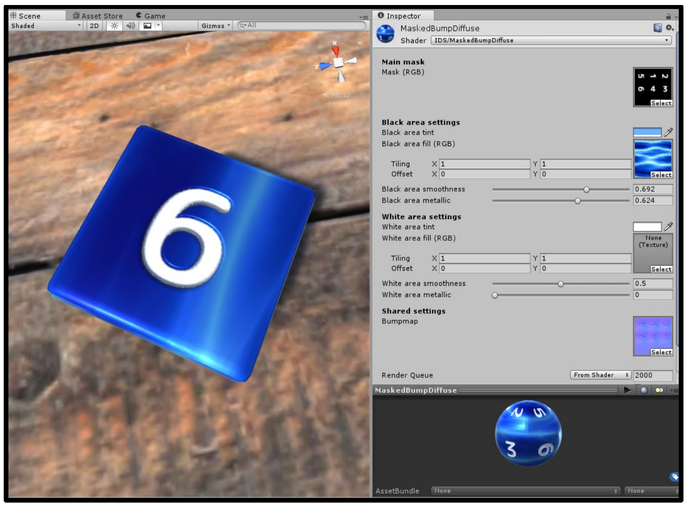
A little bit easier way to create custom materials is to use the provided mask shaders. A mask shader uses 3 main inputs:

- a black & white mask (such as provided with the project)
- a color/texture/etc to use wherever the input mask is black
- a color/texture/etc to use wherever the input mask is white

Which inputs are available exactly, depends on the mask shader you selected. There are three different versions:

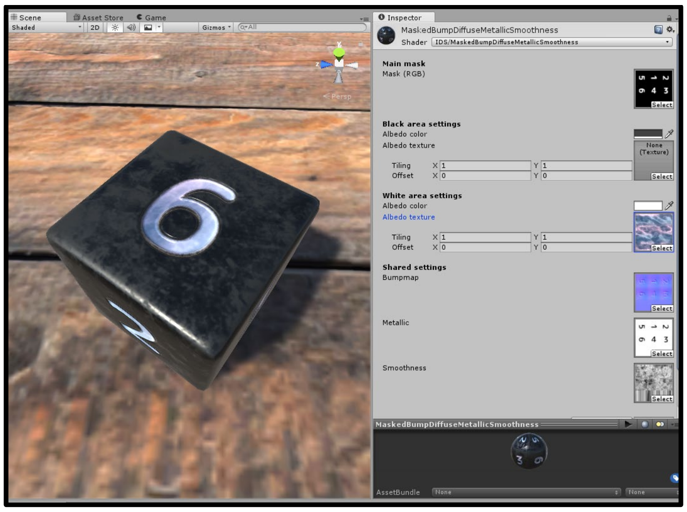
- a MaskedBumpDiffuse shader
- a MaskedBumpDiffuseMetallicSmoothness shader
- an AnimatedMaskedBumpDiffuse shader

Of course there is always a chance you are looking for a combination of settings which isn't covered by these shaders, however if you open them up, you will see they are fairly easy to customize (as far as shaders are ever easy). The provided shaders are discussed on the next page.



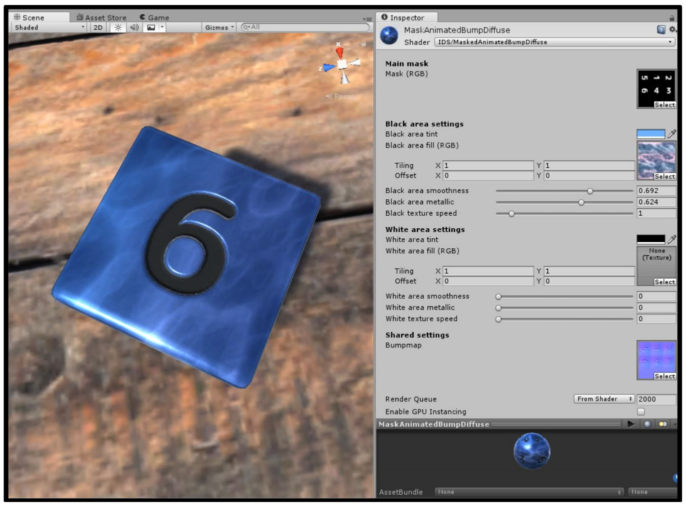
```
Inspector
MaskedBumpDiffuse
Shader  IDS/MaskedBumpDiffuse

Main mask
Mask (RGB)

Black area settings
Black area tint
Black area fill (RGB)
  Tiling   X 1   Y 1
  Offset   X 0   Y 0
Black area smoothness   0.692
Black area metallic     0.624

White area settings
White area tint
White area fill (RGB)                None (Texture)
  Tiling   X 1   Y 1
  Offset   X 0   Y 0
White area smoothness   0.5
White area metallic     0

Shared settings
Bumpmap

Render Queue   From Shader   2000
MaskedBumpDiffuse
AssetBundle  None   None
```

The MaskedBumpDiffuse shader allows you to customize the black & white area tint & texture (including scaling and offset) and you can specify an overall smoothness and metallic setting for both areas. The bumpmap is shared between both areas.



```
Inspector
MaskedBumpDiffuseMetallicSmoothness
Shader  IDS/MaskedBumpDiffuseMetallicSmoothness

Main mask
Mask (RGB)

Black area settings
Albedo color
Albedo texture                       None (Texture)
  Tiling   X 1   Y 1
  Offset   X 0   Y 0

White area settings
Albedo color
Albedo texture
  Tiling   X 1   Y 1
  Offset   X 0   Y 0

Shared settings
Bumpmap

Metallic

Smoothness

MaskedBumpDiffuseMetallicSmoothness
AssetBundle  None   None
```

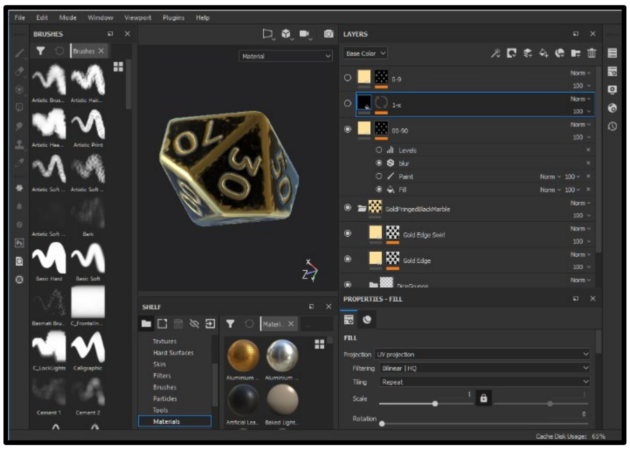
The MaskedBumpDiffuseMetallic-Smoothness is almost the same as the previous shader, but instead of specifying the metallic and smoothness values separately for the white and black areas using a slider, there is now a shared metallic and smoothness texture slot.



```
Inspector
MaskAnimatedBumpDiffuse
Shader  IDS/MaskedAnimatedBumpDiffuse

Main mask
Mask (RGB)

Black area settings
Black area tint
Black area fill (RGB)
  Tiling   X 1   Y 1
  Offset   X 0   Y 0
Black area smoothness   0.692
Black area metallic     0.624
Black texture speed     1

White area settings
White area tint
White area fill (RGB)                None (Texture)
  Tiling   X 1   Y 1
  Offset   X 0   Y 0
White area smoothness   0
White area metallic     0
White texture speed     0

Shared settings
Bumpmap

Render Queue   From Shader   2000
Enable GPU Instancing
MaskAnimatedBumpDiffuse
AssetBundle  None   None
```

The last mask shader that has been provided is very similar to the first, except for the fact that both the black and white area textures can now be animated.

CREATING MATERIALS USING SUBSTANCE PAINTER (AND THE PROVIDED .PSD TEMPLATE)

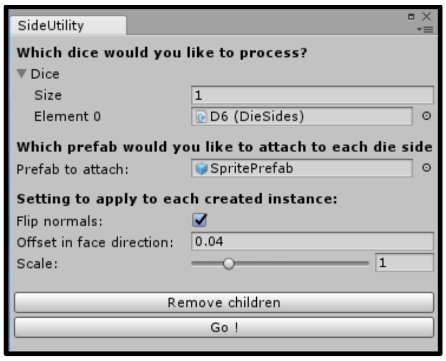


To have the most control over your textures, it is recommended to create them using a procedural texturing tool such as Substance Painter. If you follow one of the many tutorials on the allegorithmic site, you'll be up to speed with this great tool in no time. Everything mentioned in those tutorials that you will need has been provided in the package: high poly models, low poly models & mask textures. On the right you see a screenshot of how Substance Painter looks with the D10_GoldFringedMarble document open. Although you can paint your number masks directly in Substance Painter, it is probably easier to simply adjust the masks in photoshop, export them to bitmaps and reimport them into Substance Painter as bitmap masks.

CREATING MATERIALS USING THE SIDEUTILITY

This package is meant to be used as described in the introduction with dice textures with numbers on them. However for very specific use cases such as where you would like to change the face/value of a die side at runtime, a SideUtility has been provided (in the menu under IDS/DiceCreator/Die side utility). If you want to take this approach, the idea is to assign a 'blank' material to the die (without any numbers/pips etc) on them, and then run the SideUtility to attach a prefab to each die side which contains the number/pips in the form of a sprite/textfield etc (you can also run the SideUtility on multiple dice at once).

Two example prefabs have been provided, one sprite based and one text mesh based. You can also attach a component to the prefab that implements IDieSideAware so that the SideUtility can pass information to the prefab about the DieSide it represents. Please check out the provided prefabs for an example of how this works.



SideUtility

Which dice would you like to process?

▼ Dice

Size	1
Element 0	D6 (DieSides)

Which prefab would you like to attach to each die side

Prefab to attach:	SpritePrefab

Setting to apply to each created instance:

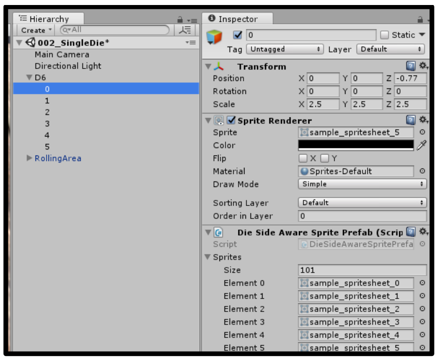
Flip normals:	✓
Offset in face direction:	0.04
Scale:	1

Remove children

Go !

The Side Utility

The result



Hierarchy

Create ▾

▼ 002_SingleDie*
- Main Camera
- Directional Light
- ▼ D6
  - 0
  - 1
  - 2
  - 3
  - 4
  - 5
- ▶ RollingArea

Inspector

✓ 0 — Static

Tag Untagged — Layer Default

Transform

Position	X 0	Y 0	Z -0.77
Rotation	X 0	Y 0	Z 0
Scale	X 2.5	Y 2.5	Z 2.5

✓ Sprite Renderer

Sprite	sample_spritesheet_5
Color	
Flip	X Y
Material	Sprites-Default
Draw Mode	Simple
Sorting Layer	Default
Order in Layer	0

Die Side Aware Sprite Prefab (Scrip

Script	DieSideAwareSpritePrefa
▼ Sprites	
Size	101
Element 0	sample_spritesheet_0
Element 1	sample_spritesheet_1
Element 2	sample_spritesheet_2
Element 3	sample_spritesheet_3
Element 4	sample_spritesheet_4
Element 5	sample_spritesheet_5

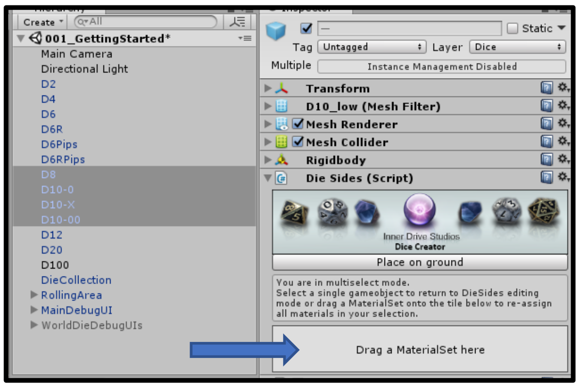
The die with SpritePrefab children

MATERIALSETS & MATERIALSETCOLLECTIONS

There are 8 different dice models, and some of those have variations in textures such as numbers vs pips, 0-based, x-based, 10-based etc, resulting in 12 materials per 'theme' (eg GoldFringedMarble, ErodedMarbleWhite). Assigning all these materials by hand is labor intensive. One option is to combine the atlases for a whole set using a tool like MeshBaker. That is definitely a good option, but not one that has been included in this package. The approach this package uses is to combine all the materials for a theme into a

MaterialSet (which is a ScriptableObject). All MaterialSets in turn can be combined into a MaterialSetCollection, which is used by the MaterialManagerUI at runtime.

At editor time MaterialSets also come in handy, since upon selecting **multiple** dice, an option becomes available to drag and drop a MaterialSet onto the DieSides component, and it will try to figure out which material from the set needs to be assigned. This is based on a simple DX_ matching scheme, where everything in the die name before the _ will be matched with the part of a materialname before the _.



You are in multiselect mode.
Select a single gameobject to return to DieSides editing mode or drag a MaterialSet onto the tile below to re-assign all materials in your selection.

To help in creating MaterialSets, an experimental MaterialSet rebuild script has been added in version 1.2, which works as follows:

- Select a material folder which looks like a good starting point, eg ChalkStoneGreen
- Duplicate the folder, and call it something else, eg ChalkStonePurple
- Now run the Rebuild Material Sets script from the IDS/DiceCreator menu, this will do a couple of things:
    - delete all MaterialSets
    - go through all material folder without an _ in front of the folder name and
        - rename all materials in that folder to DX_<foldername>Material
        - create a MaterialSet for each folder that matches the folder name and contains the materials from that folder
- Now you have both a folder and a MaterialSet called ChalkStonePurple, containing all (appropriately renamed) materials for that theme.
- Select all dice, assign your newly created set and play with the material settings of the new materials so that they actually look purple instead of green.

(Note that this script will only process the InnerDriveStudios/DiceCreator/Materials folder.)

MATERIALSET OVERVIEW

Since version 1.2, nineteen different MaterialSets have been included which are all on display in the 003_MultiMaterialDemo scene. Only the GoldFringedMarbleBlack and the EroredMarbleWhite are substance painter generated texture sets (the core sets as you will). All the other material sets are variations that might or might not reuse some or all of the maps from one of these core sets (almost all materials reuse the normal maps from one of these sets and some reuse the metallic map (which contains the glossiness in the alpha channel)).

For more information, open the aforementioned scene and inspect the materials to see how they've been put together.

HOW DO I ...

... CONSTRUCT A CUSTOM DIE FROM SCRATCH?

If for any reason you do not want to use the provided prefabs, you can also create a die from scratch by following the steps below:

1. Create an empty scene
2. Drag in a die mesh of your choosing
3. Add the 'Die Sides' component and click 'Calculcate die sides' adjusting the cut-off value if necessary.
4. Enter data for all die sides, the easy way is to click the 'R' button for the first die side, fill in a value and then tab through all other values, while the die in the Scene view is being updated as you type.
5. Now your die is finished!

You can interact with the DieSides component through code, requesting info about which side is 'up' or you can add components such as PhysicsDie, PhysicsDieAvoider, MouseClickDieRoller. For more info please check the 'Prefabs & Components overview' chapter or the 'Source Code' chapter.

... RECREATE THE 'GETTING STARTED' SCENE FROM SCRATCH?

The steps below take you through the whole process of setting up a scene with dice and UI's from scratch:

1. Create an empty scene
2. Drag the RollingArea prefab into the hierarchy so it sits at (0,0,0)
3. Drag all the prefabs from the folder PhysicsDice_SingleValue into the hierarchy at the root (they **don't** have to be a parent of the RollingArea)
4. With all the dice are selected move them up above the wooden floor,
   do not worry about aligning them to the floor we will do that later.
5. Select the individual dice and spread them out across the floor.
   If you want you can temporarily switch off 'Show debug info' to hide the display of all side normals.
6. Select all dice again and click the 'Place on ground' button located on the DieSides component.
7. If you want you can play and test what you have so far, the dice should work, but no values are being displayed yet.
8. See the next 'How to ...''s for more info on how to create a D100 and show some debugging info.

... CREATE A DICE LIKE A D100 AS A COMBINATION OF OTHER DICE?

1. Add an empty game object to the scene root and add a DieCollection component to it, optionally rename it
2. Create a D10-0 and D10-00 instance using the provided prefabs and add them to the created collection
3. Optionally add the D100 collection as an element of the larger collection created in the previous how-to

... USE THE MAIN DEBUG UI TO DISPLAY INFO ABOUT A COLLECTION OF DICE?

To use the MainDebugUI in a new (empty) scene, some setup is required:

1. Drag the MainDebugUI prefab into the scene
2. Expand the MainDebugUI/ScreenCanvas/LeftPanel node and ....
   a. select the DieCollectionDebugUI, and drag the die collection instance you want to show info for into the 'Die Collection' slot for the selected DieCollectionDebugUI component
   b. select the MaterialManagerUI and repeat step a for this component, so that you can switch materials for the whole collection at once
3. Select the MainDebugUI instance and configure the Activation Utility prefab, deleting or adding any options as you please.

Of course you can also delete any of the panels under the MainDebugUI/ScreenCanvas/LeftPanel node, just be sure to update the 'Activation Utility' settings accordingly, otherwise you'll run into NullReferenceExceptions.

... USE THE DIE DEBUG UI TO DISPLAY INFO ABOUT A SINGLE DIE?

1. Drag an instance of the DieDebugUI prefab into the scene and select it
2. Drag an instance of a DieSides component (attached to an actual Die) into the 'Die Sides' slot for the selected instance
3. Run the scene

As the 001_GettingStarted scene demonstrates, it is advisable to create a parent node at 0,0,0 to keep your scene clean and organized, adding all DieDebugUI instances to this parent node. In addition you can add this single parent as an item to the Activation Utility (located on the MainDebugUI object) so that you can enable/disable all die debug ui's at runtime.

... FORCE THE D2 AND D6R TO DETERMINE THEIR END RESULT FASTER?

Both the D2 and rounded D6 have rounded edges, which work perfectly but they might roll on longer than you'd like because of those. As an alternative after fully configuring the die, you can remove the MeshCollider and replace it with a BoxCollider to counter this default behaviour.

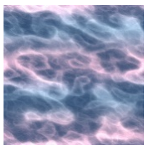
... GET HELP?

If after reading this manual, any of your questions are still unanswered, please contact us at:

info@innerdrivestudios.com

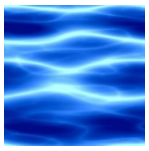
CREDITS & ATTRIBUTIONS

Some of the textures used come from authors on the FilterForge website, listed below, we would like to thank them kindly. Although these textures are free to use, attribution is appreciated.



Ignis Serpentis
https://www.filterforge.com/forum/view_profile.php?UID=16638
https://www.filterforge.com/filters/11336-v7.html

Foxxee
https://www.filterforge.com/forum/view_profile.php?UID=6216
https://www.filterforge.com/filters/6162-v2.html



JE Strange
https://www.filterforge.com/forum/view_profile.php?UID=11347
https://www.filterforge.com/filters/9813.html

CHANGELOG

1.0	• First release
1.1	• Added D2 die • Added rounded D6 die • Updated all textures for D2 and D6 • Updated die side centerpoint calculation to use the correctly weighted average • Improved MainDebugUI layout and structure • Updated DieCollectionDebugUI to be able to handle more than 9 dice • Updated the default Rolling Area to fix the texture stretching
1.2	• Dice layouts have been updated for the D8 and D12 dice so that every die (except the D2 and D4) now follows the "opposite-sides-always-add-up-to-the-same-number" convention. If you have already created your own custom materials for these dice types, make sure your side data still matches. • All prefabs have had their side data recalculated to match the new material layout and benefit from the improved centerpoint calculation • Removed minor glitches from some normal maps • Added a Side Utility that can be used to attach prefabs to all the sides of a given die • Removed the _ from the 6 in the D6 because it didn't make any sense • MaterialSet rebuild script added to allow you to quickly create/update your own material sets (experimental!) • Replaced the double overlay material approach from version 1.0/1.1 with new mask shaders • Added 3 different types of mask shaders to make it easier to define your own look and feel for the dice • Added a bunch of new materialsets (19 in total now!) • Improved the resolution and sharpness of the provided b/w & w/b masks • Added dotted D6 variations (both cubic and rounded) • Added zoom option to the DebugDieUI • Extensive documentation update • Added a Materials chapter with more in depth information on how to create your own materials • Updated wood texture aniso setting for higher quality look and feel under extreme angles