SR Squad Manager Plugin SDK

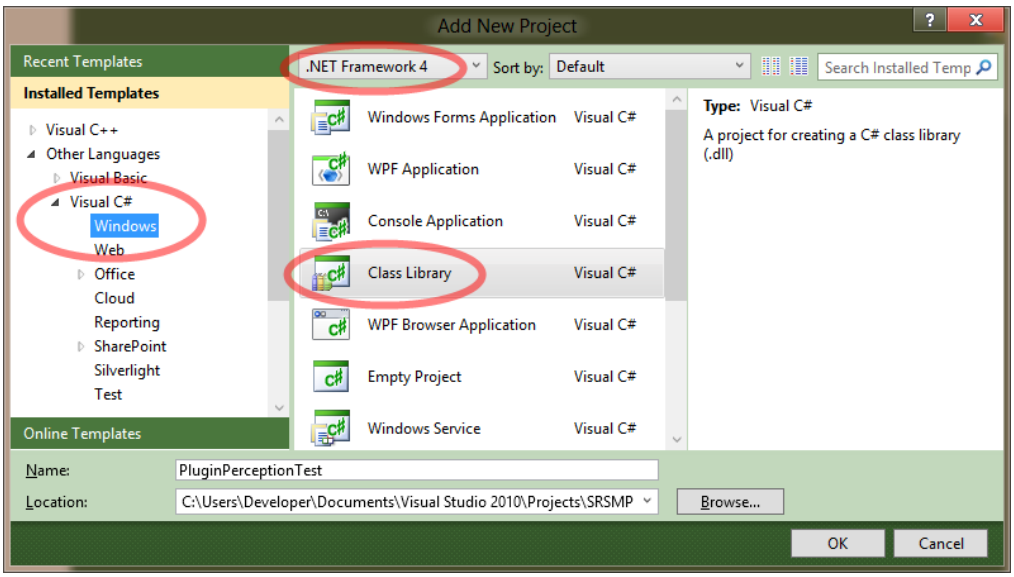
Dokumentation

Um ein Plugin für den SR Squad Manager zu bauen, muss in erster Linie eine C# Klassenbibliothek erstellt werden, die ein bestimmtes Interface implementiert. Dann kann die Bibliothek vom Squad Manager geladen und benutzt werden.

Inhalt

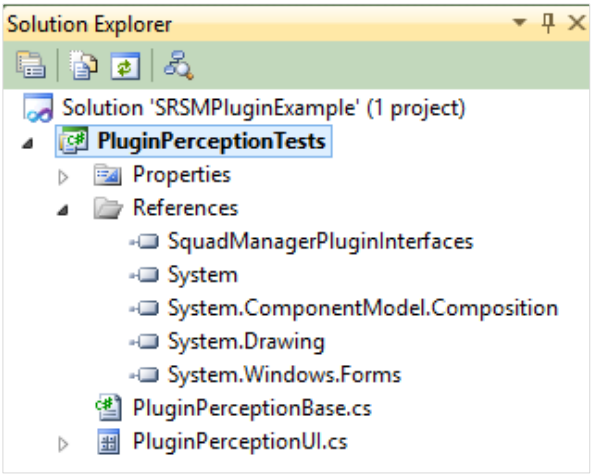
Projekteinstellungen

In Visual Studio (die Beispieldateien sind mit Version 2010/v10 erstellt worden) muss als neues Projekt eine C# Klassenbibliothek für .NET Framework 4 erstellt werden:



Referenzen

Anschließend müssen die Referenzen angepasst werden. Die meisten der vorgegebenen Referenzen können entfernt werden, um die Liste schlank zu halten. Beibehalten werden muss nur eine Referenz, dafür müssen einige neue Referenzen aus dem Angebot von .NET 4.0 ergänzt werden, und natürlich auch die letztlich wichtigste Referenz aus dem Ordner „References“, um aus der Klassenbibliothek ein Squad Manager Plugin zu machen.



Die Projektstruktur sollte dann in etwa so aussehen:

- *SquadManagerPluginInterfaces*
- *System*
- *System.ComponentModel.Composition*
- *System.Drawing*
- *System.Windows.Forms*

Interfaces

Die Assembly SquadManagerPluginInterfaces liefert einige Interfaceklassen, die für die Kommunikation zwischen Plugin und Hauptprogramm genutzt werden. Ein Interface muss implementiert werden, damit das Plugin geladen werden kann, alle weiteren Interfaces werden genutzt um Daten an das Plugin zu geben.

ISquadManagerPlugin

Das Hauptinterface des SDK. Dieses Interface muss von der Hauptklasse des Plugins implementiert werden, damit das Plugin geladen wird.

Die Methode GetLocalizedName wird während des Ladevorgangs einmal aufgerufen, um einen übersetzten Namen des Plugins abzufragen und im Menü anzuzeigen.

Die Methode GetUI wird aufgerufen, um dem Plugin ein Fenster zur Verfügung zu stellen. Hier muss ein gültiges Steuerelement vorbereitet und zurückgegeben werden, wobei Grundfunktionen wie ein größenveränderbares Fenster mit Anzeige des Pluginnamens vom Hauptprogramm gestellt werden.

ReceiveCallbackInterface wird genau einmal aufgerufen, während das Plugin geladen wird. Das übergebene Callback-Objekt sollte im Plugin gespeichert werden, damit später darüber andere Funktionen genutzt werden können (siehe ISquadManagerPluginCallback).

Die beiden Funktionen EventGlitchHappened und EventDamageSuffered werden aufgerufen, wenn ein geladener Charakter einen Patzer gewürfelt bzw. Schaden erlitten hat. Letztere kann auch aufgerufen werden, wenn Schaden geheilt wird. Plugins können die gegebenen Daten direkt verwenden, werden aber abgebrochen, sollte die Berechnungszeit zu lange dauern (derzeit werden minimal 200ms zugesagt). Wenn das Pluginfenster automatisch geöffnet werden soll, muss am Ende der Funktion true zurückgegeben werden, ansonsten false (insbesondere auch, wenn das Plugin auf die Events gar nicht reagieren will).

ISquadManagerPluginDebug

Dieses Interface kann von der Hauptklasse des Plugins zusätzlich zu ISquadManagerPlugin implementiert werden, um Zugriff auf einfache Debug-Funktionen zu bekommen. Dazu wird beim Laden zusätzlich die Funktion ReceiveDebugCallbackInterface aufgerufen, um ein entsprechendes Callback-Objekt zu übergeben (siehe ISquadManagerPluginDebugCallback).

ISquadManagerPluginCallback

Über das Callback Interface hat das Plugin die Möglichkeit, verschiedene Charaktere abzufragen, die Entfernung zwischen zwei Charakteren zu bekommen (sollten beide auf einer kalibrierten Karte platziert sein) oder einen Würfel zu werfen.

Die Funktionen GetSelectedCharacter, GetTwoCharacters und GetAllCharacters geben jeweils einen, zwei oder alle aktuell geladenen Charaktere zurück. Das beinhaltet evtl. auch besondere Akteure, wie Personas, Drohnen oder Geister, nicht zwingend nur metamenschliche Charaktere.

Die Methode GetClosestCharacter liefert zu einem Akteur den nächstgelegenen zurück, unter der Annahme, dass beide Charaktere einen Positionsmarker auf einer kalibrierten Karte haben. Es wird dabei ausschließlich die Direktverbindung verglichen, Rauminformationen, Wände oder geschlossene Türen werden hierbei nicht berücksichtigt. Die Verbindung zwischen zwei Charakteren lässt sich unter den gleichen Voraussetzungen mit GetDistanceBetweenCharacters berechnen.

Mit RollSingleDie lässt sich genau ein einzelner W6 werfen. Es wird direkt die Augenzahl zurückgegeben, alle weiteren Shadowrunregeln (Erfolge, Patzer, Regel der Sechs etc.) müssen vom Plugin selbst behandelt werden.

ISquadManagerPluginDebugCallback

Diese Callback Klasse bietet eine weitere Funktion, Log, um einfache Textzeilen in einem Logfenster auszugeben. Das Logfenster wird automatisch geöffnet, sobald eine neue Meldung ausgegeben wird, und kann nach Bedarf auch geschlossen oder über das Pluginfenster geöffnet werden.

ISquadManagerCharacterInterface

Dieses Interface wird genutzt, um Daten zu Akteuren auszutauschen. Dabei muss es sich nicht immer um klassische, metamenschliche Charaktere handeln, sondern es können auch Agenten, Drohnen oder Geister darüber behandelt werden.

Über die Eigenschaften Name, NPC, Type und Description lassen sich diese einfachen Werte auslesen. Der String Type gibt dabei einen groben Anhaltspunkt darüber, um was für einen Typ von Akteur es sich handelt: der Metatyp (z.B. „Dwarf“) bei metamenschlichen Charakteren oder die Klasse bei anderen Akteuren (z.B. „Ghost“). Der Wert NPC ist immer dann true, wenn es sich um einen Nichtspielercharakter des Spielleiters handelt und false, wenn es ein Spielercharakter ist. Alle diese Eigenschaften sind schreibgeschützt.

Mit GetImage lässt sich ein passendes Bild zum Charakter abfragen, das über Angabe des optionalen Größenparameters auf diese Maximalgröße skaliert wird.

Über GetAttribute, GetEssence, GetSkill, GetDamage und weitere, ähnliche Methoden lassen sich verschiedene, schreibgeschützte Werte des Akteurs abfragen.

Über GetPool lässt sich direkt der aktuelle Würfelpool eines Akteurs zu einer bestimmten Fertigkeit oder beispielsweise zu einem bestimmten Matrixprogramm abfragen, was Modifikatoren wie beispielsweise Schadensabzüge beinhaltet.

Über die verschiedenen GetTestResult Methoden lassen sich Erfolgs- und ausgedehnte Proben werfen lassen, was die volle Auswertung nach SR4A-Regeln beinhaltet.

Manche Werte lassen sich über bestimmte Methoden verändern oder überschreiben. Durch AppendToDescription lässt sich Text an die Charakterbeschreibung anhängen, über SetDamage wird der aktuelle Schadensmonitor beeinflusst. Dabei werden evtl. nicht alle Werte berücksichtigt, so ergibt es z.B. keinen Sinn, geistigen Schaden für ein Fahrzeug festzulegen.

Mit den Funktionen UseEdge, ChangeEdge und BurnEdge lässt sich das aktuelle Edge des Charakters verändern. Die Verbrauchsfunktionen liefern true zurück, falls der Punkt erfolgreich verbraucht wurde oder false, falls kein Edge mehr hierfür zur Verfügung stand. Diese Funktionen sollten natürlich nur sehr vorsichtig eingesetzt, insbesondere bei Spielercharakteren.

Mit Hilfe von SetTag, GetTag und ClearTag lassen sich beliebige Strings in einem Charakter speichern. Die Schlüssel hierfür sind in keiner Weise eingeschränkt, ein Plugin kann also durchaus die Werte eines anderen Plugins auslesen oder überschreiben. Dadurch ist Kommunikation zwischen Plugins generell möglich. Sollte das nicht erwünscht sein, sollte der Schlüsselwert möglichst eindeutig sein, am besten immer mit dem Pluginnamen beginnen. Wenn der Spielleiter den markierten Charakter danach speichert, werden auch alle Schlüssel-Werte-Paare mit gespeichert, es besteht aber keine Garantie dafür, dass der Charakter auch wirklich gespeichert wird.

Parameterklassen

Viele Methoden des ISquadManagerCharacterInterface geben eine bestimmte Klasse zurück oder erwarten eine als Parameter.

Attribute

Beschreibt den natürlichen und verstärkten Wert eines Attributs und gibt eventuelle Modifikatoren an, die auf alle Proben mit diesem Attribut gelten. Die Attribute sind dabei nicht auf die metamenschlichen Werte beschränkt, sondern können beispielsweise auch Matrixattribute darstellen.

Skill

Gibt den Wert einer bestimmten Fertigkeit an und die effektive Höhe des damit verbundenen Attributs, nach Anwendung aller Steigerungen und Modifikatoren. Sollte die Fertigkeit spezialisiert sein, wird auch das hier angegeben.

TestResultSuccess und TestResultExtended

In diesen Klassen werden die Ergebnisse von Erfolgsproben und ausgedehnten Würfelproben gespeichert. Bei normalen Erfolgsproben wird die Anzahl der Erfolge angegeben und die Information, ob der Wurf ein Patzer war oder nicht. Bei ausgedehnten Proben Wird die Anzahl der benötigten Intervalle und die Nettoerfolge angegeben zur Info, ob während der Probe mindestens einmal gepatzt wurde. Sollte die ausgedehnte Probe scheitern (beispielsweise wegen einem kritischen Patzer), werden 0 Intervalle angegeben.

ConditionMonitor

Der vollständige Zustandsmonitor des Charakters speichert jeweils die aktuellen und maximal möglichen Schadenskästchen für geistigen, körperlichen und überzähligen Schaden. Zusätzlich wird der daraus resultierende Würfelpoolmodifikator angegeben und der Hinweis, ob der Charakter „stabil“ ist, oder regelmäßig alle paar Kampfrunden automatischen Schaden erleidet. Bei manchen Akteuren sind einige Werte standardmäßig auf 0, sollten sie anders keinen Sinn ergeben (z.B. geistiger und überzähliger Schaden bei Drohnen und Fahrzeugen). Wenn keiner der vorhandenen Schadensarten passt, gilt der körperliche Schaden als entsprechender Schaden (z.B. statt Matrixschaden für eine KI).

Metadaten

Um ordentlich erkannt und dargestellt zu werden, muss die Hauptklasse des Plugins zumindest zwei Metadaten exportieren: den Namen des Plugins (als String zum Namen „Name“) und den des Autors (als String zum Namen „Author“).

Hauptklasse

Zusammenfassend kann betont werden, dass ein Plugin genau eine Hauptklasse besitzen muss, die zumindest das Interface ISquadManagerPlugin implementiert und diesen Typ und mehrere Metadaten exportiert. Eine minimale Klassendeklaration sieht dann in etwa so aus:

```
[Export(typeof(SquadManagerPluginInterfaces.ISquadManagerPlugin))]
[ExportMetadata("Name", "<Pluginname>")]
[ExportMetadata("Author", "<Autorenname>")]
public class MyPluginBase: SquadManagerPluginInterfaces.ISquadManagerPlugin {
        //...
}
```

Zudem muss das Plugin für die Benutzeroberfläche zumindest ein Element für das Pluginfenster zur Verfügung stellen.