ArkhamHorror Skript

# Allgemeines

## Funktion

AHS ist eine Skriptsprache für ArkhamHorror zur Definition weiterer Spiel-Objekte, wie Waffen, Verbündete, Begegnungen, etc.

Es ist ein Generator für ArkhamHorror’s JavaScript, das vom ArkhamHorror Server verwendet wird. Die Dateien werden intern in JS übersetzt und dann an vom Server verwendet.

Es bietet eine Vereinfachung gegenüber dem Original JS Code.

## Genereller Aufbau

In AHS werden Objekte von bestimmten Klassen definiert. Die verfügbaren Klassen sind fest definiert, und bestimmen, wie mit dem Objekt umgegangen wird. Jedes Objekt hat einen eindeutigen Namen. Gewisse Klassen können anonyme Objekte haben, dann wird der Name automatisch erstellt.

Objekte mancher Klassen können mehrfach im Spiel eingefügt werden (z.B. Gegenstände). Hier wird die Multiplizität nach dem Objektnamen angegeben, durch Doppelpunkt (:) getrennt.

Objekte mancher Klassen können *unendlich* sein. D.h. es werden automatisch so viele Objekte angelegt, wie benötigt werden. Sie können somit niemals ausgehen. Diese Typen werden im Normalfall unendlich erzeugt. Dies kann überschrieben werden, indem eine Multiplizität angegeben wird (z.B. SpecialObjects für dauerhaften Schaden durch Gerüchte, Ancient One, etc. sollten unendlich sein, damit sichergestellt genügend Objekte für alle Spieler vorhanden sind; SpecialObject Deputy Of Arkham sollte einmalig ( „: 1“) sein).

Die Eigenschaften der Objekte werden über Attribute definiert. Jede Klasse legt fest, welche Attribute verfügbar sind, und welche davon verpflichtend sind. Die Attribute werden durch geschwungene Klammern ({ und }) umschlossen.

Attribute haben einen bestimmten Typ. Dieser bestimmt, wie die Werte angegeben werden müssen. Für mache Attribute sind mehrere alternative Typen möglich. Die Interpretation mancher Typen wird durch das Attribut festgelegt.

Attribute haben einen Namen, der mit Doppelpunkt (:) vom Wert getrennt wird. Attributwerte werden mit einem Semikolon (;) abgeschlossen.

Namen haben folgendes Format: Groß-/Kleinbuchstaben, Underscore (\_), Ziffern; dürfen nicht mit Ziffer beginnen.

Manche Attribute werden automatisch vom System vergeben. Diese dürfen nicht angegeben werden.

Bsp.:

CommonItem Food : 2 {  
 Attribute: Value;  
}

## JS-Übersetzung

Bei der Übersetzung von AHS in JS Code werden die AHS-Klassen in JS-Objekte umgewandelt. Dabei wird aus jedem AHS-Attribut eine Property des Objekts. Weitere Properties können durch AHS automatisch eingefügt werden (sind in Kapitel 3 als „automatisch“ gekennzeichnet).

Für jede AHS-Klasse gibt es eine klassenspezifische createXXX und registerXXX JS-Methode, im globalen Objekt game, die aufgerufen werden. Gewisse Register-Funktionen benötigen auch eine Anzahl, die wird automatisch angegeben (durch Multiplizität in AHS).

Für einmalige Objekte wird eine eindeutige ID vergeben (in Kapitel 3 als „ID“ gekennzeichnet), für mehrfache Objekte eine eindeutige Typ-ID (in Kapitel 3 als „TID“ gekennzeichnet). Diese IDs / TIDs setzen sich aus einem Klassen-Typ spezifischen Präfix, und dem Klassennamen zusammen (z.B. "CI\_Dynamite").

Zusätzlich wird für jede AHS-Klasse eine Konstante mit registerConstant erzeug (außer anonyme Objekte), sodass im JS-Code mit {KlassenTyp}.{KlassenName} darauf referenziert werden kann (z.B. CommonItem.Dynamite).

# Attribut Typen

## Kommentar

Kommentare werden ignoriert. Es gibt Zeilenkommentare, startend mit 2 Schrägstrichen (//), die bis zum Zeilenende laufen, und Blockkommentare, die von Schrägstrich-Stern (/\*) bis Stern-Schrägstrich (\*/)laufen (auch über mehrere Zeilen). Bsp.:

Das wird verwendet // Das hier wird ignoriert (Zeilenkommentar)  
Das wird nicht ignoriert /\* Das schon  
und das auch (Blockkommentar) \*/ das nicht mehr

## Boolean

Boolscher Wert. Kann nur die 2 Werte „true“ und „false“ annehmen.

## Ganzzahl (Number)

Zahl. Folge von Ziffern (0-9). Sollte nicht mit „0“ beginnen.

## String

Text. Wird entweder mit Doppelten (") oder Einfachen (') Anführungsstrichen umschlossen.

Wenn in einem String ein Anführungsstrich verwendet wird, muss er mit Backslash (\) escaped werden (z.B. "Test \"mit\" 'Anführungsstrichen'").

Mehrere Strings können per Plus (+) zusammengesetzt werden. Das ermöglicht das Trennen des Strings auf mehrere Zeilen zur besseren Lesbarkeit im Code. Es können Strings mit einfachen und doppelten Anführungsstrichen zusammengesetzt werden. Bsp.:

"Dieser Text ist zu "+  
"lang, um ihn gut in "+  
'einer Zeile im "Code" zu lesen'

## Name

Namen sind Anzeigenamen von Objekten vom Typ String. Wenn kein Name angegeben wird, wird der Objekt-Name verwendet, wobei CamelCase und Underscore in Leerzeichen umgewandelt wird. Bsp.:

Cultes\_desGoules 🡺 Name = "Cultes des Goules"

## Aufzählungswert (Enum)

Für gewisse Attribute sind Attribut-spezifische Aufzählungswerte aus einer Liste möglich (z.B. Bewegungstyp bei Monstern). Bei diesen wird einfach der Wert angegeben. Bsp.:

movementType: Flying;

### JS-Übersetzung

Jedes Enum-Attribut hat einen gewissen Typ (in Kapitel 3 angegeben, Werte in Kapitel 4 gelistet). Für diese Werte gibt es JS-Konstanten, im globalen Objekt Constants.{Type}. Dieser Präfix wird Attribut-spezifisch bei der Übersetzung eingefügt.

## Referenzen (IDRef)

Manche Attribute sind Beziehungen zu anderen Objekten (z.B. Spezialfertigkeiten bei Investigatoren). Hier wird die Objektklasse und der Objektname durch Punkt (.) getrennt angegeben. Bsp.:

specialAbilityId: Object.JennyBarnesSpecialAbility;

### JS-Übersetzung

ID-Referenzen werden als Strings ausgegeben, indem der Klassen-spezifische Präfix und Name zusammengesetzt wird (vgl. 1.3).

## Listen (Array)

Gewisse Attribute erlauben Listen von Aufzählungswerten oder Referenzen. Diese werden von runden Klammern umschlossen (( und )) und durch Komma (,) getrennt. Bsp.:

attributes: ( Unending, Physical\_Resistance );

## Komplex (Complex)

Ein Komplex hat eine Attribut-Spezifische Bedeutung (siehe Attribute unten). Er wird mit eckigen Klammern ([ und ]) umschlossen. Bsp.:

[ /\* Komplexer Wert. Bedeutung Attribut-Abhängig \*/ ]

## Funktion (Function)

Manche Attribute sind Funktionen (siehe Attribute unten, z.B. onMove bei Monstern). Funktionen sind zu komplex, um in AHS abzubilden. Funktionen werden daher direkt in JS Code angegeben.

Eine AHS-Funktion wird mit 3 geschwungenen Klammern umschlossen ({{{ und }}}). Bsp.:

onMove: {{{  
 // JS-Code  
}}};

## Literale (Literal)

Manche Attributwerte sind zu komplex, um sie in AHS abzubilden. Dafür werden Literale verwendet, die 1:1 in den JS-Code übernommen werden. Literale werden mit 3 spitzen Klammern umschlossen (<<< und >>>).

Literale können immer bei Attributwerten verwendet werden. Bsp.:

attribute: <<< /\* JS-Code hier \*/ >>>;

## Geschachtelte Objekte

Bei Attributen, die IDRef Werte benötigen, kann unter umständen das referenzierte Objekt direkt geschachtelt angegeben werden. Derzeit ist das für Optionen (konkret: QuickOption, FieldOption, AEOption und OWEOption) verfügbar. Das geschachtelte Objekt muss mit geschwungenen Klammern umschlossen ({ und }) umschlossen werden.

Das kann auch mit IDRefs kombiniert werden:

options: (  
 Option.SKIP, // IDReferenz  
  
 { // Geschachtelte Option  
 QuickOption {  
 // Definition  
 }  
 },  
  
 {QuickOption { // Geschachtelte Option  
 // Definition  
 }}  
 );

# Klassen

## Action

### Beschreibung

Eine Aktion, die im Spiel etwas verändert. Reine Aktionen werden automatisch durchgeführt. Ein Beispiel ist die Spezialfertigkeit von Jenny Barnes: Sie bekommt in jeder Upkeep Phase automatisch $1.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| id | Automatisch | ID | Eindeutige ID |
| name | Optional | Name | Name |
| description | Optional | String | Beschreibungstext |
| phases | Verpflichtend | Array(Enum): GamePhases | Spielphasen, in denen die Aktion durchgeführt wird |
| activate | Verpflichtend | Function | Aktion der Action |

### JS-Übersetzung:

Create: game.createAction(obj)  
Register: game.registerAction(action)  
ID-Prefix: AC  
Constant-Scope: Action

## Option

### Beschreibung

Eine Spieloption, die dem Spieler angeboten wird. Eine Option hat eine Referenz auf eine Action, die bei Aktivierung durchgeführt wird. Optionen können mit Kosten verbunden sein, oder nur unter gewissen Umständen verfügbar sein. Die Beschreibung wird von der referenzierten Aktion übernommen.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| id | Automatisch | ID | Eindeutige ID |
| name | Optional | Name | Name (Anzeige) |
| isAvailable | Optional | Function | Prüft, ob die Option verfügbar ist. Muss Boolean zurückliefern. Standard: true (immer verfügbar) |
| actionId | Verpflichtend | IDRef: Action | Referenz zur Action |
| continueType | Optional | Enum: ContinueType | Ob der Zug nach der Option beendet ist, oder eine weitere Option gewählt werden kann |
| chooseType | Optional | Enum: ChooseType | Wie die Option gewählt werden kann in Kombination mit anderen Optionen |
| costs | Optional | Complex: Costs | Kosten der Option |

### JS-Übersetzung:

Create: game.createOption(obj)  
Register: game.registerOption(option)  
ID-Prefix: OP  
Constant-Scope: Option

## QuickOption

### Beschreibung

Kombination von Option und Action. Aus den angegebenen Attributen wird eine Action und eine Option dafür erstellt. Kann geschachtelt vorkommen.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| id | Automatisch | ID | Eindeutige ID |
| name | Optional | Name | Name (Anzeige) |
| isAvailable | Optional | Function | Siehe Option |
| continueType | Optional | Enum: ContinueType | Siehe Option |
| chooseType | Optional | Enum: ChooseType | Siehe Option |
| costs | Optional | Complex: Costs | Siehe Option |
| description | Optional | String | Siehe Action |
| phases | Verpflichtend | Array(Enum): GamePhases | Siehe Action |
| activate | Verpflichtend | Function | Siehe Action |

### JS-Übersetzung:

Create: game.quickOption(obj)  
Register: Nicht verfügbar. quickOption registriert die Option automatisch  
ID-Prefix: OP  
Constant-Scope: Option

## FieldOption

### Beschreibung

Option auf einem Spielfeld. Das sind die speziellen Begegnungen in gewissen Feldern (z.B. Heilung in St. Mary’s Hospital). Es entspricht einer QuickOption, plus Feld. Kann geschachtelt vorkommen.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| id | Automatisch | ID | Eindeutige ID |
| name | Optional | Name | Name (Anzeige) |
| isAvailable | Optional | Function | Siehe Option |
| continueType | Optional | Enum: ContinueType | Siehe Option |
| chooseType | Optional | Enum: ChooseType | Siehe Option |
| costs | Optional | Complex: Costs | Siehe Option |
| description | Optional | String | Siehe Action |
| phases | Verpflichtend | Array(Enum): GamePhases | Siehe Action |
| activate | Verpflichtend | Function | Siehe Action |
| field | Verpflichtend | Enum: Field | Feld, auf dem die Option angeboten wird |

### JS-Übersetzung:

Create: game.quickOption(obj)  
Register: addFieldOtion(Field, objId)  
ID-Prefix: OP  
Constant-Scope: Option

## AEOption

### Beschreibung

Option für eine Begegnung in Arkham (siehe 3.16). Es entspricht einer QuickOption, mit Phase fixiert auf ArkhamEncountery. Kann geschachtelt vorkommen.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| id | Automatisch | ID | Eindeutige ID |
| name | Optional | Name | Name (Anzeige) |
| isAvailable | Optional | Function | Siehe Option |
| continueType | Optional | Enum: ContinueType | Siehe Option |
| chooseType | Optional | Enum: ChooseType | Siehe Option |
| costs | Optional | Complex: Costs | Siehe Option |
| description | Optional | String | Siehe Action |
| phases | Automatisch | Array(Enum): GamePhases | ArkhamEncountery |
| activate | Verpflichtend | Function | Siehe Action |

### JS-Übersetzung:

Create: game.quickOption(obj)  
Register: Nicht verfügbar. quickOption registriert die Option automatisch  
ID-Prefix: OP  
Constant-Scope: Option

## OWEOption

### Beschreibung

Option für eine Begegnung in anderen Welten (siehe 3.17). Es entspricht einer QuickOption, mit Phase fixiert auf OtherWorldEncountery. Kann geschachtelt vorkommen.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| id | Automatisch | ID | Eindeutige ID |
| name | Optional | Name | Name (Anzeige) |
| isAvailable | Optional | Function | Siehe Option |
| continueType | Optional | Enum: ContinueType | Siehe Option |
| chooseType | Optional | Enum: ChooseType | Siehe Option |
| costs | Optional | Complex: Costs | Siehe Option |
| description | Optional | String | Siehe Action |
| phases | Automatisch | Array(Enum): GamePhases | OtherWorldEncountery |
| activate | Verpflichtend | Function | Siehe Action |

### JS-Übersetzung:

Create: game.quickOption(obj)  
Register: Nicht verfügbar. quickOption registriert die Option automatisch  
ID-Prefix: OP  
Constant-Scope: Option

## CommonItem

### Beschreibung

Profane Gegenstände, wie Waffen, Bücher, Nahrung, etc. CommonItems können Mehrfach vorkommen.

CommonItems werden in den Stapel für CommonItems gelegt, und können im Spiel gezogen werden.

CommItems können Modifikatoren haben (z.B. Waffen geben einen Physischen Schaden), oder Optionen zur Verfügung stellen (z.B. Buch lesen).

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| tid | Automatisch | TID | Eindeutige Typ ID |
| type | Automatisch | - | Interner Typ |
| name | Optional | Name | Name |
| description | Optional | String | Beschreibung |
| hands | Optional | Number | Anzahl benötigter Hände (Waffen) |
| price | Optional | Number | Preis beim Kauf |
| exhaustable | Optional | Boolean | Ob das Objekt nach der Benutzung getappt wird, und erst in der nächsten Runde verfügbar ist (Standard: false) |
| options | Optional | Array(IDRef): Option | Liste der Optionen, die das Objekt zur Verfügung stellt |
| attributes | Optional | Array(Enum): ObjectAttributes | Objekt-Attribute |
| modifications | Optional | Complex: Mods oder Function | Modifikatoren |
| onAddToInventory | Optional | Function | Funktion, wenn das Objekt erhalten wird |
| onRemoveFromInventory | Optional | Function | Funktion, wenn das Objekt verloren wird |
| properties | Optional | Literal | Dynamische Properties (fortgeschritten) |

### JS-Übersetzung:

Create: game.createObject(obj)  
Register: game.registerMultiObject(count, object)  
ID-Prefix: CI  
Constant-Scope: CommonItem  
Type: CommonItem

## UniqueItem

### Beschreibung

Magische Gegenstände, wie Waffen, magische Steine, , etc. UniqueItems können Mehrfach vorkommen.

UniqueItems werden in den Stapel für UniqueItems gelegt, und können im Spiel gezogen werden.

UniqueItems können Modifikatoren haben (z.B. Waffen geben einen Magischen Schaden), oder Optionen zur Verfügung stellen (z.B. Stein verwenden).

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| tid | Automatisch | TID | Eindeutige Typ ID |
| type | Automatisch | - | Interner Typ |
| name | Optional | Name | Name |
| description | Optional | String | Beschreibung |
| hands | Optional | Number | Anzahl benötigter Hände (Waffen) |
| price | Optional | Number | Preis beim Kauf |
| exhaustable | Optional | Boolean | Ob das Objekt nach der Benutzung getappt wird, und erst in der nächsten Runde verfügbar ist (Standard: false) |
| options | Optional | Array(IDRef): Option | Liste der Optionen, die das Objekt zur Verfügung stellt |
| attributes | Optional | Array(Enum): ObjectAttributes | Objekt-Attribute |
| modifications | Optional | Complex: Mods oder Function | Modifikatoren |
| onAddToInventory | Optional | Function | Funktion, wenn das Objekt erhalten wird |
| onRemoveFromInventory | Optional | Function | Funktion, wenn das Objekt verloren wird |
| properties | Optional | Literal | Dynamische Properties (fortgeschritten) |

### JS-Übersetzung:

Create: game.createObject(obj)  
Register: game.registerMultiObject(count, object)  
ID-Prefix: UI  
Constant-Scope: UniqueItem  
Type: UniqueItem

## Skill

### Beschreibung

Fertigkeiten. Skills können Mehrfach vorkommen.

Skills werden in den Stapel für Skills gelegt, und können im Spiel gezogen werden.

Skills haben Modifikatoren, die die Attribute des Charakters ändern (Luck, Lore, Fight, Will, Speed, Sneak), oder weitere Würfel auf Attribut-Tests geben

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| tid | Automatisch | TID | Eindeutige Typ ID |
| type | Automatisch | - | Interner Typ |
| name | Optional | Name | Name |
| description | Optional | String | Beschreibung |
| options | Optional | Array(IDRef): Option | Liste der Optionen, die das Objekt zur Verfügung stellt |
| modifications | Optional | Complex: Mods oder Function | Modifikatoren |
| onAddToInventory | Optional | Function | Funktion, wenn das Objekt erhalten wird |
| onRemoveFromInventory | Optional | Function | Funktion, wenn das Objekt verloren wird |
| properties | Optional | Literal | Dynamische Properties (fortgeschritten) |

### JS-Übersetzung:

Create: game.createObject(obj)  
Register: game.registerMultiObject(count, object)  
ID-Prefix: SK  
Constant-Scope: Skill  
Type: Skill

## Ally

### Beschreibung

Verbündete. Allies können nur einfach vorkommen.

Allies werden in den Stapel für Allies gelegt, und können im Spiel gezogen werden.

Allies können Modifikatoren haben (z.B. Ignorieren von Monster-Attributen), oder Optionen zur Verfügung stellen.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| tid | Automatisch | TID | Eindeutige Typ ID |
| type | Automatisch | - | Interner Typ |
| name | Optional | Name | Name |
| description | Optional | String | Beschreibung |
| options | Optional | Array(IDRef): Option | Liste der Optionen, die das Objekt zur Verfügung stellt |
| modifications | Optional | Complex:Mods oder Function | Modifikatoren |
| onAddToInventory | Optional | Function | Funktion, wenn das Objekt erhalten wird |
| onRemoveFromInventory | Optional | Function | Funktion, wenn das Objekt verloren wird |
| properties | Optional | Literal | Dynamische Properties (fortgeschritten) |

### JS-Übersetzung:

Create: game.createObject(obj)  
Register: game.registerSingleObject(object)  
ID-Prefix: AL  
Constant-Scope: Ally  
Type: Ally

## Spell

### Beschreibung

Zaubersprüche. Spells können mehrfach vorkommen.

Spells werden in den Stapel für Spells gelegt, und können im Spiel gezogen werden.

Spells müssen durch eine Würfelprobe aktiviert werden. Es gibt Spells, die nur in bestimmten Situationen verfügbar sind und dann eine Option bieten (z.B. Find Gate), oder bieten einen Kampfbonus.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| tid | Automatisch | TID | Eindeutige Typ ID |
| type | Automatisch | - | Interner Typ |
| name | Optional | Name | Name |
| description | Optional | String | Beschreibung |
| hands | Optional | Number | Anzahl benötigter Hände (Angriffszauber) |
| exhaustable | Optional | Boolean | Ob das Objekt nach der Benutzung getappt wird, und erst in der nächsten Runde verfügbar ist (Standard: false) |
| castTarget | Optional | Number | Zielwert bei Würfelprobe (Sandard 1) |
| castCost | Verpflichtend | Number | Sanity-Kosten zur Aktivierung des Spells |
| castAdjustment | Verpflichtend | Number | Anpassung des Lore-Wertes bei Würfelprobe |
| cast | Optional | Function oder Literal | Spell Funktion |
| castIsAvailable | Optional | Function oder Literal | Spell: IsAvailable für Zauber |
| castChooseType | Optional | Enum: ChooseType | Spell: ChooseType |
| castContinueType | Optional | Enum: ContinueType | Spell: ContinueType |
| castPhases | Optional | Array(Enum): GamePhases | Spell: Phases |
| options | Optional | Array(IDRef) | Liste der Optionen, die das Objekt zur Verfügung stellt (Sollte leer sein) |
| modifications | Optional | Complex: Mods oder Function | Modifikatoren (Angriffszauber) |
| onAddToInventory | Optional | Function | Funktion, wenn das Objekt erhalten wird |
| onRemoveFromInventory | Optional | Function | Funktion, wenn das Objekt verloren wird |
| properties | Optional | Literal | Dynamische Properties (fortgeschritten) |

### JS-Übersetzung:

Create: game.createObject(obj)  
Register: game.registerMultiObject(count, object)  
ID-Prefix: SP  
Constant-Scope: Spell  
Type: Spell

## BlessingCurse

### Beschreibung

Segnung / Fluch. Sind normalerweise unendlich. Können mehrfach vorkommen.

Segnung / Fluch sind nicht direkt verfügbar, und können nur durch spezielle Aktionen erhalten werden (z.B Begegnungen, in der Kirche).

Es sollte nur eine Blessing und einen Curse geben, die den Success-Wert für Würfel anpasst.

Ein Spieler kann immer nur 1 Blessing ODER 1 Curse haben.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| tid | Automatisch | TID | Eindeutige Typ ID |
| type | Automatisch | - | Interner Typ |
| name | Optional | Name | Name |
| description | Optional | String | Beschreibung |
| options | Optional | Array(IDRef) | Liste der Optionen, die das Objekt zur Verfügung stellt |
| modifications | Optional | Complex: Mods oder Function | Modifikatoren |
| onAddToInventory | Optional | Function | Funktion, wenn das Objekt erhalten wird |
| onRemoveFromInventory | Optional | Function | Funktion, wenn das Objekt verloren wird |
| properties | Optional | Literal | Dynamische Properties (fortgeschritten) |

### JS-Übersetzung:

Create: game.createObject(obj)  
Register: game.registerMultiObject(count, object)  
ID-Prefix: BC  
Constant-Scope: BlessingCurse  
Type: BlessingCurse

## SpecialObject

### Beschreibung

Spezielle Objekte. Sind normalerweise unendlich. Können mehrfach vorkommen.

Spezielle Objekte sind nicht direkt verfügbar, und können nur durch spezielle Aktionen erhalten werden (z.B. Hilfssherif-Marke, Logen-Mitgliedschaft, Bankanleihe, etc.).

Spezielle Objekte können Modifikatoren haben, oder Optionen und auch Aktionen zur Verfügung stellen. Aktionen werden automatisch ausgeführt, sind also z.B. bei der Bankanleihe in der Upkeep-Phase notwendig, sodass der Spieler automatisch würfeln muss, ob er die Bankanleihe verliert.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| tid | Automatisch | TID | Eindeutige Typ ID |
| type | Automatisch | - | Interner Typ |
| name | Optional | Name | Name |
| description | Optional | String | Beschreibung |
| exhaustable | Optional | Boolean | Ob das Objekt nach der Benutzung getappt wird, und erst in der nächsten Runde verfügbar ist (Standard: false) |
| actions | Optional | Array(IDRef): Action | Liste der Aktionen, die das Objekt zur Verfügung stellt |
| options | Optional | Array(IDRef): Option | Liste der Optionen, die das Objekt zur Verfügung stellt |
| modifications | Optional | Complex: Mods oder Function | Modifikatoren |
| onAddToInventory | Optional | Function | Funktion, wenn das Objekt erhalten wird |
| onRemoveFromInventory | Optional | Function | Funktion, wenn das Objekt verloren wird |
| properties | Optional | Literal | Dynamische Properties (fortgeschritten) |

### JS-Übersetzung:

Create: game.createObject(obj)  
Register: game.registerMultiObject(count, object)  
ID-Prefix: OBJ  
Constant-Scope: SpecialObject  
Type: Special

## SpecialAbility

### Beschreibung

Spezielle Fähigkeiten von Investigatoren, können nur einfach vorkommen.

Spezielle Fähigkeiten können Modifikatoren haben, oder Optionen und auch Aktionen zur Verfügung stellen. Aktionen werden automatisch ausgeführt, sind also z.B. bei der Trust Fund (Jenny Barnse) in der Upkeep-Phase notwendig, sodass der Spieler automatisch $1 erhält.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| tid | Automatisch | TID | Eindeutige Typ ID |
| type | Automatisch | - | Interner Typ |
| name | Optional | Name | Name |
| description | Optional | String | Beschreibung |
| exhaustable | Optional | Boolean | Ob das Objekt nach der Benutzung getappt wird, und erst in der nächsten Runde verfügbar ist (Standard: false) |
| actions | Optional | Array(IDRef): Action | Liste der Aktionen, die das Objekt zur Verfügung stellt |
| options | Optional | Array(IDRef): Option | Liste der Optionen, die das Objekt zur Verfügung stellt |
| modifications | Optional | Complex: Mods oder Function | Modifikatoren |
| onAddToInventory | Optional | Function | Funktion, wenn das Objekt erhalten wird |
| onRemoveFromInventory | Optional | Function | Funktion, wenn das Objekt verloren wird |
| properties | Optional | Literal | Dynamische Properties (fortgeschritten) |

### JS-Übersetzung:

Create: game.createObject(obj)  
Register: game.registerSingleObject(object)  
ID-Prefix: SA  
Constant-Scope: SpecialAbility  
Type: Special

## Monster

### Beschreibung

Ein Monster. Monster können mehrfach vorkommen.

Monster werden in den Monsterpool gelegt und bei Bedarf gezogen.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| tid | Automatisch | TID | Eindeutige Typ ID |
| name | Optional | Name | Name |
| description | Optional | String | Beschreibung |
| mythText | Optional | String | Weiterer Beschreibungstext. |
| movementType | Verpflichtend | Enum(Movement) | Art der Fortbewegung |
| awareness | Verpflichtend | Number | Wachsamkeit. Für Evade/Flight Tests |
| horrorAdjustment | Verpflichtend | Number | Anpassung für Horror-Probe |
| horrorDamage | Verpflichtend | Number | Horror-Schaden |
| combatAdjustment | Verpflichtend | Number | Anpassung für Kampf-Probe |
| combatDamage | Verpflichtend | Number | Kampf-Schaden |
| toughness | Verpflichtend | Number | Ausdauer |
| dimension | Optional | Enum(Dimension) | Heim-Dimension. Wenn leer wird eine zufällige Dimension vergeben |
| attributes | Optional | Array(Enum): MonsterAttributes oder Function | Attribute |
| modifications | Optional | Complex: Mods oder Function | Dynamische Modifikatoren |
| onMove | Optional | Function | Bewegung für Spezielle Bewegung |
| onDamage | Optional | Function(type, chr) | Funktion wenn das Monster einen Charakter angreift |
| onDefeat | Optional | Function | Funktion, wenn das Monster besiegt wurde |

### Anmerkungen

Monster mit movementType = Special müssen eine onMove Funktion haben.

onDamage wird aufgerufen, wenn Horror oder Combat-Schaden verursacht wurde. Der Parameter type gibt die Art des Schadens an (Enum DamageType: Horror, Combat, Nightmare, Overwhelm). Wenn die Funktion false zurückliefert, ist der Kampf automatisch beendet. Wenn die Funktion nichts oder true zurückliefert, wird der Kampf nach Standardregeln fortgesetzt.

onDefeat wird aufgerufen, wenn ein Spieler das Monster besiegt hat. Wenn die Funktion false zurückliefert, wird die das Monster nicht als Monstertrophäe an den Spieler gegeben (bzw. in den Monster-Pool zurückgelegt bei Unending Monstern). Wenn die Funktion true oder nichts zurückliefert, wird das getan (=Standardverhalten)

### JS-Übersetzung:

Create: game.createMonster (obj)  
Register: game.registerMonster(count, object)  
ID-Prefix: MO  
Constant-Scope: Monster

## ArkhamEncounter

### Bescheibung

Begegnung in Arkham. Kann anonym sein. Eine Begegnung wird auf ein Feld gelegt, und bietet Optionen.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| description | Verpflichtend | String | Beschreibungstext |
| field | Verpflichtend | Enum: Field | Feld |
| options | Verpflichtend | Array(IDRef): Option | Optionen |

### Anmerkungen

Die Optionen sollten als AEOption angelegt werden.

Feld muss ein Interior Feld sein.

### JS-Übersetzung:

Create: game.createArkhamEncounter (obj)  
Register: game.registerArkhamEncounter (object)  
ID-Prefix: AE  
Constant-Scope: ArkhamEncounter

## OtherWorldEncounter

### Bescheibung

Begegnung in einer anderen Welt. Kann anonym sein. Eine Begegnung wird auf ein Feld gelegt, und bietet Optionen.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| description | Verpflichtend | String | Beschreibungstext |
| field | Optional | Enum: Field | Feld |
| color | Verpflichtend | Enum: OtherWorld | Farbe |
| options | Verpflichtend | Array(IDRef): Option | Optionen |

### Anmerkungen

Die Optionen sollten als OWEOption angelegt werden.

Feld muss ein OtherWorld Feld sein. Wenn kein Feld angegeben wird, gilt die Encounter für alle Welten dieser Farbe.

### JS-Übersetzung:

Create: game.createOtherWorldEncounter (obj)  
Register: game.registerOtherWorldEncounter (object)  
ID-Prefix: OE  
Constant-Scope: OtherWorldEncounter

## Headline

### Beschreibung

Eine Mythoskarte vom Typ Schlagzeile. Headlines werden direkt ausgeführt und danach verworfen.

Jede Mythoskarte bewegt Monster, erstellt Tore, und platziert Hinweismarker.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| id | Automatisch | ID | Eindeutige ID |
| type | Automatisch | Enum: Mythos | Mythos Art (Headline) |
| name | Optional | Name | Name |
| description | Verpflichtend | String | Beschreibungstext |
| shortDesc | Optional | String | Kurzbeschreibung |
| clueField | Optional | Enum: Field | Feld für Hinweismarker (zufällig wenn fehlt) |
| gateField | Optional | Enum: Field | Feld für Tor (zufällig wenn fehlt) |
| moveBlack | Optional | Array(Enum): Dimension | Dimensionen für Bewegung nach Schwarz (zufällig wenn fehlt) |
| moveWhite | Optional | Array(Enum): Dimension | Dimensionen für Bewegung nach Weiß (zufällig wenn fehlt) |
| properties | Optional | Literal | Dynamische Eigenschaften (fortgeschritten) |
| executeHeadline | Verpflichtend | Function | Funktion der Headline |

### JS-Übersetzung:

Create: game.createMythosCard(obj)  
Register: game.registerMythosCard(object)  
ID-Prefix: MY  
Constant-Scope: Mythos  
Type: Headline

## Rumor

### Beschreibung

Eine Mythoskarte vom Typ Gerücht. Es kann immer nur ein Gerücht aktiv sein. Wenn ein neues Gerücht gezogen wird, wird das neue verworfen.

Gerüchte stellen eine länger laufende Bedrohung dar. In der Regel wird in jeder Mythos Phase das Gerücht weiter fortgeschritten (z.B. Platzieren von Markern), bis ein bestimmter Wert erreicht ist. Dann haben die Spieler einen Misserfolg (Fail), und negative Auswirkungen werden aktiv.

Jedes Gerücht stellt den Spielern eine Möglichkeit, das Gerücht abzuwenden (z.B. durch Abgabe von Gegenständen, etc.), als zusätzliche Option an einem Bestimmten Feld. Wenn diese Bedingung erfüllt ist, haben die Spieler einen Erfolg (Pass), und positive Auswirkungen werden aktiv.

Jede Mythoskarte bewegt Monster, erstellt Tore, und platziert Hinweismarker.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| id | Automatisch | ID | Eindeutige ID |
| type | Automatisch | Enum: Mythos | Mythos Art (Rumor) |
| name | Optional | Name | Name |
| description | Verpflichtend | String | Beschreibungstext |
| shortDesc | Optional | String | Kurzbeschreibung |
| clueField | Optional | Enum: Field | Feld für Hinweismarker (zufällig wenn fehlt) |
| gateField | Optional | Enum: Field | Feld für Tor (zufällig wenn fehlt) |
| moveBlack | Optional | Array(Enum): Dimension | Dimensionen für Bewegung nach Schwarz (zufällig wenn fehlt) |
| moveWhite | Optional | Array(Enum): Dimension | Dimensionen für Bewegung nach Weiß (zufällig wenn fehlt) |
| properties | Optional | Literal | Dynamische Eigenschaften (fortgeschritten) |
| rumorFieldOptionId | Verpflichtend | IDRef: Option | Option für Success-Bedingung |
| rumorField | Verpflichtend | Enum: Field | Feld für Success-Option |
| setupRumor | Verpflichtend | Function | Vorbereitungsfunktion. Wird aufgerufen, wenn das Gerücht gezogen wird |
| teardownRumor | Verpflichtend | Function | Aufräumfunktion. Wird aufgerufen, wenn das Gerücht entfernt wird (Pass & Fail) |
| onMythos | Verpflichtend | Function | Wird in jeder Mythos-Phase aufgerufen, solange das Gerücht aktiv ist |
| onPass | Verpflichtend | Function | Wird aufgerufen, wenn das Gerücht abgewendet wird |
| onFail | Verpflichtend | Function | Wird aufgerufen, wenn die Spieler versagen |

### Anmerkungen

Die onMythos Funktion sollte den Status prüfen. Wenn ein Fail bzw. Pass Status erreicht wird, muss sie this.fail() bzw. this.pass() aufgerufen werden.

Wenn der Success-Status durch die Option festgelegt wird, kann die Option mit game.context.rumor.pass(); das Gerücht abwenden.

### JS-Übersetzung:

Create: game.createMythosCard(obj)  
Register: game.registerMythosCard(object)  
ID-Prefix: MY  
Constant-Scope: Mythos  
Type: Rumor

### Beispiel

Als Beispiel eine vereinfachte Variante des SouthSideStrangler Gerüchts: Anfangs sind 7 Allies vorhanden, jede Mythosphase wird dies um 1 verringert. Wenn kein Ally mehr da ist, Fehlschlag. In Ma’s Boarding House können 5 Hinweismarker ausgegeben werden, um das Gerücht abzuwenden.

QuickOption SSSOption {  
 costs: [ Clue 5 ]; // 5 Hinweismarker für Pass  
 phases: ArkhamEncountery;  
 activate: {{{ game.context.rumor.pass(); }}};  
}  
  
Rumor SSS {  
 name: "South Side Strangler";  
 description: "…";  
 rumorFieldOptionId: Option.SSSOption;  
 rumorField: SS\_MasBoardingHouse;  
 setupRumor: {{{ this.allyCount = 7; }}}; // 7 Runden  
 teardownRumor: {{{ }}}; // Kein Teardown  
  
 onMythos: {{{   
 this.allyCount--;  
 // Fail wenn keine Allies mehr da  
 if (this.allyCount == 0) this.fail();  
 }}};  
  
 onPass: {{{  
 for (var c in game.context.allCharacters) {  
 c.addMoney(5); // Jeder Spieler erhält $5  
 }  
 }}};  
  
 onFail: {{{  
 for (var c in game.context.allCharacters) {  
 // Jeder erhält 1 Santiy & Stamia Schaden  
 c.damageSanity(1);  
 c.damageStamina(1);  
 }  
 }}};  
}

## Environment

### Beschreibung

Eine Mythoskarte vom Typ Umgebung.

Umgebungen haben Einfluss auf die Spielsituation durch Modifikatoren, oder eine Aktivität auf einem bestimmten Feld anbieten. Es kann immer nur 1 Umgebung aktiv sein. Wenn eine neue Umgebung gezogen wird, ersetzt diese die alte.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| id | Automatisch | ID | Eindeutige ID |
| type | Automatisch | Enum: Mythos | Mythos Art (Environment) |
| name | Optional | Name | Name |
| description | Verpflichtend | String | Beschreibungstext |
| shortDesc | Optional | String | Kurzbeschreibung |
| clueField | Optional | Enum: Field | Feld für Hinweismarker (zufällig wenn fehlt) |
| gateField | Optional | Enum: Field | Feld für Tor (zufällig wenn fehlt) |
| moveBlack | Optional | Array(Enum): Dimension | Dimensionen für Bewegung nach Schwarz (zufällig wenn fehlt) |
| moveWhite | Optional | Array(Enum): Dimension | Dimensionen für Bewegung nach Weiß (zufällig wenn fehlt) |
| properties | Optional | Literal | Dynamische Eigenschaften (fortgeschritten) |
| environmentType | Verpflichtend | Enum: EnvironmentType | Umgebungs-Typ |
| environmentFieldOptionId | Optional | IDRef: Option | Aktivitats-Option für Environment |
| environmentField | Optional | Enum: Field | Aktivitäts-Feld |
| environmentModifications | Optional | Complex: Mods | Modifikatoren |
| monsterModifications | Optional | Complex: MonsterMods | Monster-Spezifische Modifikatoren |
| monsterMoveModifications | Optional | Complex: Monster Move Mods | Monster Bewegungsmodifikatoren |

### JS-Übersetzung:

Create: game.createMythosCard(obj)  
Register: game.registerMythosCard(object)  
ID-Prefix: MY  
Constant-Scope: Mythos  
Type: Environment

## Investigator

### Beschreibung

Investigatoren sind die Charaktere, die die Spieler wählen können.

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| id | Automatisch | ID | Eindeutige ID |
| name | Optional | Name | Name |
| occupation | Optional | String | Beruf |
| homeFieldId | Verpflichtend | Enum: Field | Startfeld |
| sanity | Verpflichtend | Number | Geistige Gesundheit (Max) |
| stamina | Verpflichtend | Number | Ausdauer (Max) |
| money | Verpflichtend | Number | Geld (Start) |
| focus | Verpflichtend | Number | Fokus |
| clues | Verpflichtend | Number | Hinweismarker |
| fixedPossessionObjectIds | Verpflichtend | Array(IDRef): Object | Fixer Besitz |
| randomPossessions | Verpflichtend | Complex: Random Possesions | Zufälliger Besitz |
| attributes | Verpflichtend | Complex: Character Attributes | Attribute |
| uniqueAbilityId | Optional | IDRef: Object | Besondere Fähigkeit |

### Anmerkungen

Die besondere Fähigkeit muss eine UniqueAbility sein.

Wenn ein Investigator keinen fixen Besitz hat, muss ein literales leeres Array angegeben werden:

fixedPossessionObjectIds: <<<[]>>>;

### JS-Übersetzung:

Create: game.createInvestigator(obj)  
Register: game.registerInvestigator (object)  
ID-Prefix: IN  
Constant-Scope: Investigator

## AncientOne

### Beschreibung

Ancient Ones sind die Großen Alten, die die Hauptwidersacher der Spieler darstellen. In jedem Spiel ist 1 Ancient One im Spiel. Während des Spiels ist der Ancient One im Schlaf. Durch verschiedene Umstände kann der AncientOne erwachen, wodurch es zum Endkampf kommt.

Ancient Ones können während dem Schlaf globale Modifikatoren haben, und haben Worshippers, die oft durch Monster Modifikatoren beeinflusst werden. Im Endkampf haben sie eine Attacke, deren Schwierigkeit in der Regel mit jeder Kampfrunde steigt.

Die Investigatoren müssen Schaden akkumulieren, um den AncientOne zu besiegen. AncientOnes können hier wie normale Monster Attribute haben (z.B. Physische Resistenz).

### Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Art | Typ | Beschreibung |
| id | Automatisch | ID | Eindeutige ID |
| name | Optional | Name | Name |
| doomTrack | Verpflichtend | Number | Anzahl Doom-Marker |
| worshippersText | Optional | String | Beschreibungstext für Worshippers (links auf Karte) |
| monsterModifications | Optional | Complex: Monster Mods | Monstermodifikatoren für Worshippers |
| monsterMoveModifications | Optional | Complex: Monster Move Mods | Bewegungsmodifikatoren für Worshippers |
| powerTitle | Optional | String | Überschrift für spezielle Fähigkeit (mittig auf Karte) |
| powerText | Optional | String | Beschreibung für spezielle Fähigkeit(mittig auf Karte) |
| slumberModifications | Optional | Complex: Mods | Spezielle Fähigkeiten während des Schlafs (Power) |
| battleStartText | Optional | String | Beschreibung der Aktion wenn der Endkampf beginnt (mittig auf Karte) |
| onAwake | Optional | Function | Funktion wenn der Endkampf beginnt (AO aufwacht) |
| combatAdjustment | Optional | Number | Anpassung für Kampfprobe |
| defenses | Optional | Array(Enum): Monster Attributes | Verteidigungs-Attribute |
| attackText | Optional | String | Beschreibung der Attacke (rechts auf Karte) |
| attackAdjustment | Optional | Number | Anpassung für Attacke. Wird in der Regel jede Kampfrunde um 1 verringert |
| attack | Verpflichtend | Function | Angriff |
| postAttack | Optional | Function | Nach Angriff auf alle Investigators |

### JS-Übersetzung:

Create: game.createIAncientOne(obj)  
Register: game.registerAncientOne (object)  
ID-Prefix: AO  
Constant-Scope: AncientOne

### Beispiel

Hier als Beispiel der AncientOne Hastur:

AncientOne Hastur {  
 doomTrack: 13;  
// Worshippers  
/////////////////////////////////////////////////  
 worshippersText: "Hastur's worshippers ride byakhee mounts "+  
 "that they call with enchanted whistles. "+  
 "<b>Cultists</b> are <em>flying monsters</em> and "+  
 "their combat rating is -2.";  
 // Cultists are Flying  
 monsterMoveModifications: [  
 Cultist Flying  
 ];  
 // Cultists have Combat Adjustment 1. Add -3 => -2  
 monsterModifications: [  
 Cultist { Monster\_CombatAdjustment -3 }  
 ];  
// Slumber / Power  
/////////////////////////////////////////////////  
 powerTitle: "The King in Yellow";  
 powerText: "While Hastur stirs in his slumber, the cost to "+  
 "seal a gate is 8 Clue tokens instead of 5.";  
 // It costs 3 more to seal gates  
 slumberModifications: [  
 Game\_SealClueCost +3  
 ];  
// Battle Start  
///////////////////////////////////////////////////  
 battleStartText:  
 "Combat adjustment is set to the current terror level";  
 // Initial Combat Adjustment: 0  
 combatAdjustment: 0;  
 // Set combat adjustment to Terrorlevel onAwake  
 onAwake: {{{  
 this.setCombatAdjustment(game.context.terrorLevel);  
 }}};  
  
  
// Attack / End fight  
///////////////////////////////////////////////////  
attackText: "Each investigator must pass a Luck (+1) check "+  
 "or lose 2 Sanity.\n"+  
 "This check's modifier decreases by 1 each turn " +  
 "(+0 the 2nd turn, -1 the 3rd turn, etc.)";  
  
 defenses: PhysicalResistance;  
 // Initial Attack Adjustment is 1.   
 // Autmotically decreased each fight round  
 attackAdjustment: 1;  
 attack: {{{  
 // Roll dice:  
 var res = CHAR.skillTest(  
 "Hastur attacks",   
 Constants.Skills.Luck, // Test LUCK  
 this.attackAdjustment, // Adjustment  
 1); // Required Successes  
 // If not succeeded, damage sanity by 2  
 // In Endfight, insane characters are automatically  
 // devoured, and cannot come back  
 if (!res) {  
 CHAR.damageSanity(2);  
 }  
 }}}  
}

# Enums

## Skills

Skills sind die Basisattribute der Charaktere + weitere spezielle Skills. Sie dienen oft als Basis von Würfel-Tests.

JS-Konstanten-Präfix: Constants.Skills

|  |  |
| --- | --- |
| Name | Beschreibung |
| Speed | Attribut Geschwindigkeit |
| Sneak | Attribut Schleichen |
| Fight | Attribut Kampf |
| Will | Attribut Wille |
| Lore | Attribut Wissen |
| Luck | Attribut Glück |
| Evade | Speziell: Ausweichen-Probe |
| Combat | Speziell: Angriff-Probe |
| Horror | Speziell: Horror-Probe |
| Spell | Speziell: Zauber-Probe |

## ObjectType

Objekttypen. In AHS werden diese automatisch aufgrund der Objekt-Klasse festgelegt

JS-Konstanten-Präfix: Constants.ObjectType

|  |  |
| --- | --- |
| Name | Beschreibung |
| CommonItem | Einfache Gegenstände |
| UniqueItem | Besondere Gegenstände |
| Spell | Zauber |
| Skill | Fähigkeit |
| Ally | Verbündeter |
| BlessingCurse | Segnung / Fluch |
| Special | Spezielles Objekt (Anleihe, Besondere Fähigkeit, …) |

## ObjectAttribute

Objekt-Attribute

JS-Konstanten-Präfix: Constants. ObjectAttribute

|  |  |
| --- | --- |
| Name | Beschreibung |
| CannotBeLost | Objekt kann nicht verloren werden bei Bewusstlos/Verrückt |
| DiscardAfterAttack | Muss nach Angriff abgelegt werden |
| DiscardOnEndFight | Muss abgelegt werden, wenn Ancient One aufwacht |

## Option Attribute

Bestimmt, wann Optionen gewählt werden können, und wie nach einer Option weitergespielt wird. Diese Werte werden für ChooseType und ContinueType bei Options und QuickOptions angegeben

JS-Konstanten-Präfix: Constants. Option

|  |  |  |
| --- | --- | --- |
| Name | Typ | Beschreibung |
| CannotContinue | ContinueType | Zug ist nach Wahl dieser Option beendet |
| CanContinue | ContinueType | Weitere Optionen können nach dieser gewählt werden |
| Optional | ChooseType | Spieler ist nicht verpflichtet, diese Option zu wählen |
| Mandatory | ChooseType | Spieler muss diese Option wählen |
| Supplemental | ChooseType | Spieler kann diese Option wählen, auch wenn Mandatory Optionen vorhanden sind |

### Erklärung zu ChooseType

In jeder Phase werden dem Spieler Optionen angeboten. Dabei kann es sein, dass eine Option gewählt werden muss, wenn sie verfügbar ist (z.B. Tor durchschreiten muss gewählt werden, wenn der Spieler auf einem Feld mit Tor ist, das er noch nicht besucht hat).

Wenn Mandatory Optionen vorhanden sind, muss der Spieler eine dieser Optionen wählen. Optional Optionen werden dann nicht angeboten (z.B. Begegnung in Arkham ist nicht verfügbar, wenn ein Tor durchschritten werden muss). Nur wenn keine Mandatory Option verfügbar ist, werden die Optional angeboten.

Supplemental Optionen können immer gewählt werden. Z.B. kann ein Healing-Stone verwendet werden, bevor ein Tor durschritten werden muss.

## GamePhases

Phasen im Spiel. Actions sind immer nur in bestimmten Phasen möglich. Z.B. ist das Lesen von Ancient Tome nur in der Movement Phase möglich.

JS-Konstanten-Präfix: Constants. GamePhases

|  |  |
| --- | --- |
| Name | Beschreibung |
| Upkeep | Unterhalt |
| Movement | Bewegung |
| ArkhamEncountery | Begegnung in Arkham |
| OtherWorldEncountery | Begegnung in anderen Welten |
| Mythos | Mythos |
| AllPhases | Alle oben genannten Phasen |
| DieRollPhase | Spezielle Phase: Verfügbar bei Würfel-Proben (für neuwürfeln) |
| FightPhase | Spezielle Phase: Verfügbar bei Kämpfen (für Angriffs-Zauber) |
| Any | = AllPhases |

## Mythos Typ

Typ von Mythos Karten. Wird in AHS automatisch aufgrund des Mythos-Typs vergeben.

JS-Konstanten-Präfix: Constants. Mythos

|  |  |
| --- | --- |
| Name | Beschreibung |
| Headline | Schlagzeile |
| Environment | Umgebung |
| Rumor | Gerücht |

## Umgebungs-Typ

Typ von Umgebungs-Mythos-Karten

JS-Konstanten-Präfix: Constants. EnvironmentType

|  |  |
| --- | --- |
| Name | Beschreibung |
| Weather | Wetter |
| Urban | Stadt |
| Mystic | Mystisch |

## Payment Item

Basis-Einheiten für Bezahlung von Optionen. Die Kosten für Optionen basieren auf diesen Typen

JS-Konstanten-Präfix: Constants. Costs

|  |  |
| --- | --- |
| Name | Beschreibung |
| None | Nichts |
| Money | Geld ($) |
| Clue | Hinweismarker |
| Stamina | Ausdauer-Punkte |
| Sanity | Geistige Gesundheits-Punkte |
| Movement | Bewegungs-Punkte |
| GateTrophy | Tor-Trophäen |
| MonsterTrophy | Monster-Trophäen |
| MonsterToughness | Monster-Trophäen im Wert von Monster Ausdauer |

## Dimension

Dimensionen für Monster, Tore und Monsterbewegung bei Mythos Karten

JS-Konstanten-Präfix: Constants. Dimension

|  |  |
| --- | --- |
| Name | Beschreibung |
| Square | Quadrat |
| Circle | Kreis |
| Moon | Sichelmond |
| Slash | Schräger Balken |
| Star | Stern |
| Diamond | Karo |
| Hexagon | Sechseck |
| Plus | Plus |
| Triangle | Dreieck |

## Monster Movement

Bestimmt Bewegung von Monstern

JS-Konstanten-Präfix: Constants. Movement

|  |  |
| --- | --- |
| Name | Beschreibung |
| Normal | Normal (1 Feld) |
| Fast | Schnell (2 Felder) |
| Stationary | Unbewegt |
| Flying | Fliegend |
| Special | Spezielle Bewegung (benötigt onMove Function) |

## Monster Attribute

Modifiziert Monster.

JS-Konstanten-Präfix: Constants. Monster

|  |  |
| --- | --- |
| Name | Beschreibung |
| Ambush | Ausweichen/fliehen nicht möglich |
| Endless | Kann nicht als Trophäe gesammelt werden, sondern kommt in den Monster-Pool zurück |
| PhysicalResistance | Physischer Schaden wird halbiert |
| MagicalResistance | Magischer Schaden wird halbiert |
| PhysicalImmunity | Physischer Schaden wird ignoriert |
| MagicalImmunity | Magischer Schaden wird ignoriert |
| Undead | Untot |
| Mask | Masken Monster (müssen speziell aktiviert werden) |
| Nightmarish\_1 | Macht 1 Horrorschaden auch bei bestandener Probe |
| Nightmarish\_2 | Macht 2 Horrorschaden auch bei bestandener Probe |
| Nightmarish\_3 | Macht 3 Horrorschaden auch bei bestandener Probe |
| Overwhelming\_1 | Macht 1 Kampfschaden auch bei bestandener Probe |
| Overwhelming\_2 | Macht 2 Kampfschaden auch bei bestandener Probe |
| Overwhelming\_3 | Macht 3 Kampfschaden auch bei bestandener Probe |

## Andere Welt Farben

OtherWorldEncounters haben eine Farbe, die bestimmt, für welche Andere Welten sie gelten.

JS-Konstanten-Präfix: Constants. OtherWorld

|  |  |
| --- | --- |
| Name | Beschreibung |
| Red | Rot |
| Green | Grün |
| Blue | Blau |
| Yellow | Gelb |

## Modifikatoren (Mods)

Modifikatoren beeinflussen verschiedenste Bereiche des Spiels.

JS-Konstanten-Präfix: Constants. Mods

|  |  |  |
| --- | --- | --- |
| Name | Typ | Beschreibung |
| NoProperty | ? | (Intern) |
| Attr\_Speed | Attribut | Reiner Attributwert Geschwindigkeit |
| Attr\_Sneak | Attribut | Reiner Attributwert Schleichen |
| Attr\_Fight | Attribut | Reiner Attributwert Kampf |
| Attr\_Will | Attribut | Reiner Attributwert Wille |
| Attr\_Lore | Attribut | Reiner Attributwert Wissen |
| Attr\_Luck | Attribut | Reiner Attributwert Glück |
| Skill\_Speed | Skill | Geschwindigkeit |
| Skill\_Sneak | Skill | Schleichen |
| Skill\_Fight | Skill | Kampf |
| Skill\_Will | Skill | Wille |
| Skill\_Lore | Skill | Wissen |
| Skill\_Luck | Skill | Glück |
| Skill\_Evade | Skill | Ausweichen |
| Skill\_Combat | Skill | Angriff |
| Skill\_Horror | Skill | Horror |
| Skill\_Spell | Skill | Zaubern |
| Prop\_MaxStamina | Charakter | Maximale Ausdauer |
| Prop\_MaxSanity | Charakter | Maximale Geistige Gesundheit |
| Prop\_Focus | Charakter | Fokuspunkte |
| Prop\_Movement | Charakter | Bewegungspunkte |
| Prop\_MinSuccessDieRoll | Charakter | Minimaler Wert für Erfolg |
| Prop\_HandCount | Charakter | Anzahl Hände (ausgerüstete Waffen) |
| Prop\_CastAdjustment | Charakter | Anpassung für Zauber |
| Prop\_CastCost | Charakter | Kosten für Zaubersprüche (geistige Gesundheit) |
| DieRoll\_All | ClueBurn | Alle Würfe |
| DieRoll\_Speed | ClueBurn | Geschwindigkeit |
| DieRoll\_Sneak | ClueBurn | Schleichen |
| DieRoll\_Fight | ClueBurn | Kampf |
| DieRoll\_Will | ClueBurn | Wille |
| DieRoll\_Lore | ClueBurn | Wissen |
| DieRoll\_Luck | ClueBurn | Glück |
| DieRoll\_Evade | ClueBurn | Ausweichen |
| DieRoll\_Combat | ClueBurn | Angriff |
| DieRoll\_Horror | ClueBurn | Horror |
| DieRoll\_Spell | ClueBurn | Zaubern |
| Ignore\_MonsterAttributes | Ignore | Monster Attribute ignorieren |
| Ignore\_PhysicalDamage | Ignore | Physischen Schaden ignorieren |
| Ignore\_MagicalDamage | Ignore | Magischen Schaden ignorieren |
| DrawCard\_CommonItem | DrawCard | Einfache Gegenstände |
| DrawCard\_UniqueItem | DrawCard | Besondere Gegenstände |
| DrawCard\_Spell | DrawCard | Zauber |
| DrawCard\_Skill | DrawCard | Fähigkeiten |
| DrawCard\_Ally | DrawCard | Verbündete |
| DrawCard\_ArkhamEncountery | DrawCard | Begegnung in Arkham |
| DrawCard\_OtherWorldEncountery | DrawCard | Begegnung in anderen Welten |
| Damage\_General | Waffe | Genereller Schaden (kann nicht abgewehrt werden) |
| Damage\_Physical | Waffe | Physischer Schaden |
| Damage\_Magical | Waffe | Magischer Schaden |
| Monster\_CombatDamage | Monster | Kampf-Schaden |
| Monster\_CombatAdjustment | Monster | Kampf-Anpassung |
| Monster\_HorrorDamage | Monster | Horror-Schaden |
| Monster\_HorrorAdjustment | Monster | Horror-Anpassung |
| Monster\_Awareness | Monster | Wachsamkeit |
| Monster\_Toughness | Monster | Ausdauer |
| Monster\_Attribute | Monster | Monster Attribute **(TODO: Noch nicht implementiert)** |
| Game\_SealClueCost | Spiel | Anzahl Hinweismarker für Versiegeln |
| Game\_AllowMaskMonster | Spiel | Masken Monster erlauben |
| Game\_MaxGateCount | Spiel | Max. offene Tore, bis AO erwacht |
| Game\_MaxBoardMonsterCount | Spiel | Max. Monster, bis sie in Outskirts gehen |
| Game\_MaxOutskirtsMonsterCount | Spiel | Max. Monster in Outskirts, bis Terror Level steigt |
| Game\_TerrorLevel | Spiel | Anpassung Terror Level (?) |
| Game\_MonsterCountFromGates | Spiel | Anzahl Monster aus Toren |
| Game\_CloseGeneralStoreTerrorLevel | Spiel | TL, wann General Store schließt |
| Game\_CloseCuriosityStoreTerrorLevel | Spiel | TL, wann Curiosity Store schließt |
| Game\_CloseYeOldeMagickShoppeTerrorLevel | Spiel | TL, wann Magick Shop schließt |
| Game\_OverrunArkhamTerrorLevel | Spiel | TL, wann Arkham von Outskirts überrannt wird |
| Game\_SealedGatesToWin | Spiel | Anzahl von versiegelten Toren zum gewinnen |
| Game\_GateTrophiesToWin | Spiel | Anzahl von gesammelten Tortrophäen um zu gewinnen, wenn alle Tore geschlossen sind |

### Hinweise zu Modifikatoren

Objekte gewissen Typs sollten nur gewisse Arten von Modifikatoren verwenden, da sie nur dann gelten.

Es gibt verschiedene Zusammenfassungen:

* Inventar: CommonItem, UniqueItem, Skill, Spell, Ally, SpecialAbility, SpecialObject
* Setting: Environment, Ancient One

#### Attribut

Sollte generell selten verwendet werden, da dies ein Grundattribut verändert.

Wenn verwendet, dann nur in Objekten im Inventar.

#### Skill

Modifikatoren für Attribute gelten automatisch auch für Skills, die auf dem Attribut basieren. Für Spezielle Skills sind dies:

* Evade: Sneak
* Combat: Fight
* Horror: Will
* Spell: Lore

Skill-Modifikatoren können angewandt werden bei:

* Objekten im Inventar
* Setting

#### Charakter

Generelle Modifikatoren für Charakter Eigenschaften. Diese Modifikatoren können angewandt werden in:

* Objekte im Inventar
* Setting

#### ClueBurn

Modifikatoren, wie viele Extra-Würfel beim Ausgeben von Hinweismarkern erhalten werden. Diese Modifikatoren können angewandt werden in:

* Objekte im Inventar (v.a. Skills)
* Setting

#### Ignore

Ignorieren gewisse Auswirkungen für Charaktere. (Für Physisch/Magisch immune Monster siehe Monster Attribute)

Verwendbar in:

* Objekte im Inventar (v.a. Allies)
* Setting

#### DrawCard

Beeinflusst das Ziehen von Karten. Ermöglicht es dem Spieler beim Ziehen von Karten weitere extra Karten zu ziehen und von diesen zu wählen.

Verwendbar in:

* Objekte im Inventar (v.a. SpecialAbility)
* Setting

#### Waffe

Bestimmen den Schaden einer Waffe (oder eher: Eine Waffe ist definiert dadurch, dass sie einen Schaden macht).

Waffen können im Kampf ausgerüstet werden, unter Berücksichtigung Hände. D.h. jedes Objekt, das Schaden macht, sollte auch eine Hand-Zahl haben.

Verwendbar in:

* Objekte im Inventar (v.a. CommonItem, UniqueItem, Spell)

#### Monster

Beeinflusst Eigenschaften von Monstern.

Generell verwendbar in:

* Monster (Hier sollte es eine Function sein, sonst macht es kaum Sinn)
* Setting

AncientOne und Environment können zusätzlich noch Monster-Spezifische Modifikatoren haben.

#### Spiel

Beeinflusst Spiel-Variablen. Normalerweise sind diese durch das Regelwerk mit festen Werten (evtl. abhängig von Spielerzahl) definiert.

Verwendbar in:

* Setting

## Würfel-Optionen

Bestimmen, wie viele und welche Würfel neu geworfen werden dürfen. Dies wird verwendet bei Objekten, die Optionen zum Neu-Werfen von Würfeln in der DieRollPhase anbieten.

JS-Konstanten-Präfix: Constants. Reroll

|  |  |
| --- | --- |
| Name | Beschreibung |
| All | Alle Würfel neu werfen |
| AllFailed | Alle Würfel, die einen Misserfolg darstellen |
| OneFailed | 1 Würfel, der einen Misserfolg darstellt |

## MonsterDamage

Schadens-Typ für onDamage Funktionen bei Monstern

JS-Konstanten-Präfix: Constants. MonsterDamage

|  |  |
| --- | --- |
| Name | Beschreibung |
| Combat | Kampfschaden (Ausdauer) |
| Horror | Horrorschaden (geistige Gesundheit) |
| Overwhelm | Überwältigender Schaden (Ausdauer) |
| Nightmare | Albtraumhafter Schaden (geistige Gesundheit) |

## Feld-Typen

Typen von Spielfeldern

JS-Konstanten-Präfix: Constants.FieldType

|  |  |
| --- | --- |
| Name | Beschreibung |
| Location | Interior Feld (hat Begegnung in Arkham) |
| Street | Straße |
| OtherWorld | Andere Welt (hat Begegnung in Anderer Welt) |
| Sky | Für fliegende Monster |
| Outskirts | Stadtrand |
| SpaceAndTime | Für verloren in Raum und Zeit |

## Felder

Spielfelder.

JS-Konstanten-Präfix: Constants.Fields

|  |  |  |
| --- | --- | --- |
| Name | Typ | Bereich / Farbe |
| NS\_Northside | Street | Northside |
| NS\_TrainStation | Location | Northside |
| NS\_Newspaper | Location | Northside |
| NS\_CuriositieShoppe | Location | Northside |
| DT\_Downtown | Street | Downtown |
| DT\_BankOfArkham | Location | Downtown |
| DT\_ArkhamAsylum | Location | Downtown |
| DT\_IndependenceSquare | Location | Downtown |
| ET\_Easttown | Street | Easttown |
| ET\_HibbsRoadhouse | Location | Easttown |
| ET\_VelmasDiner | Location | Easttown |
| ET\_PoliceStation | Location | Easttown |
| MD\_MerchantDist | Street | Merchant District |
| MD\_UnvisitedIsle | Location | Merchant District |
| MD\_RiverDocks | Location | Merchant District |
| MD\_TheUnnamable | Location | Merchant District |
| RT\_Rivertown | Street | Rivertown |
| RT\_Graveyard | Location | Rivertown |
| RT\_BlackCave | Location | Rivertown |
| RT\_GeneralStore | Location | Rivertown |
| MU\_MiskatonicU | Street | Miskatonic University |
| MU\_ScienceBuilding | Location | Miskatonic University |
| MU\_Administration | Location | Miskatonic University |
| MU\_Library | Location | Miskatonic University |
| FH\_FrenchHill | Street | French Hill |
| FH\_TheWitchHouse | Location | French Hill |
| FH\_SilverTwilightLodge | Location | French Hill |
| UT\_Uptown | Street | Uptown |
| UT\_StMarysHospital | Location | Uptown |
| UT\_YeOldeMagickShoppe | Location | Uptown |
| UT\_Woods | Location | Uptown |
| SS\_Southside | Street | South Side |
| SS\_MasBoardingHouse | Location | South Side |
| SS\_SouthChurch | Location | South Side |
| SS\_HistoricalSociety | Location | South Side |
| OW\_AnotherDimension | Other World | Red, Green, Blue, Yellow |
| OW\_Abyss | Other World | Red, Blue |
| OW\_CityOfGreatRace | Other World | Green, Yellow |
| OW\_Yuggoth | Other World | Blue, Yellow |
| OW\_Celeano | Other World | Blue, Green |
| OW\_TheDreamlands | Other World | Red, Green, Blue, Yellow |
| OW\_PlateauOfLeng | Other World | Red, Green |
| OW\_R\_lyeh | Other World | Red, Yellow |
| Sp\_Sky | Sky |  |
| Sp\_Outskirts | Outskirts |  |
| Sp\_SpaceAndTime | Space And Time |  |

# Komplex

## Allgemein

Komplexe Werte sind spezielle AHS-Konstrukte, die bei bestimmten Attributen verwendet werden. Sie werden in eckigen Klammern ([ und ]) geschrieben.

## Costs

Kosten definieren, was und wie viel ein Spieler ausgeben muss, um eine Option zu aktivieren.

Costs können mehrere Alternativen beinhalten, wovon jede aus mehreren Positionen bestehen kann. Jede Position besteht aus eine Pay-Item Art (definiert mittels Payment Item, siehe 4.8), und Anzahl.

### Syntax

#### EBNF

Costs := "[" {Alternativen} "]"  
Alternativen := Alternative {"," Alternativen }  
Alternative := Positionen  
Positionen := Position { "+" Positionen }  
Position := *paymentItem* *number*

#### Erläuterung

Alternativen werden durch Komma (,) getrennt.

Positionen werden durch Plus (+) verbunden.

Position besteht aus Enum(PaymentItem) und Number.

#### Beispiele

// Hinweismarker  
[ Clue 3 ]

[  
 // 2 Tortrophäen  
 GateTrophy 2,  
 // ODER Monstertrophäen für 10 Ausdauer  
 MonsterToughness 10,  
 // ODER (1 Tortrophäe UND Monstertrophäen für 5 Ausdauer)  
 GateTrophy 1 + MonsterToughness 10  
]

### JS-Übersetzung

Eine Position ist ein Objekt mit Property property für das Payment Item und Property amount für die Menge.

Ein Dimensionales Array solcher Positionen stellen Alternativen dar.

Bei einem Array von Arrays solcher Positionen stellt das äußere Array Alternativen dar, die inneren verbundene Kosten.

Also:

// 3 Hinweismarker  
{ property: Constants.Costs.Clue, amount: 3 }

// 3 Hinweismarker ODER 5 $  
[ { property: Constants.Costs.Clue, amount: 3},  
 { property: Constants.Costs.Money, amount: 5}  
]

// 3 Hinweismarker UND 5 $  
[[ { property: Constants.Costs.Clue, amount: 3},  
 { property: Constants.Costs.Money, amount: 5}  
]]

// 2 Tortrophäen  
// ODER Monstertrophäen für 10 Ausdauer  
// ODER (1 Tortrophäe UND Monstertrophäen für 5 Ausdauer)  
[  
 [{ property: Constants.Costs.GateTrophy, amount: 2} ],  
 [{ property: Constants.Costs.MonsterTrophy, amount: 10} ],  
 [  
 { property: Constants.Costs.GateTrophy, amount: 1},  
 { property: Constants.Costs.MonsterTrophy, amount: 5},  
 ]  
]

## Mods

Komplex für Modifikatoren.

### Syntax

#### EBNF

Mods := "[" Modifikationen "]"  
Modifikationen := Modifikation {"," Modifikationen }  
Modifikation := *modification* *operator* *number*

#### Erläuterung

Eine einzelne Modikation besteht aus der Eigenschaft, die modifiziert wird (definiert über Enum Modifikatoren, siehe 4.13), einem Operator, und einer Zahl.

### Operatoren

Es gibt verschiedene Arten von Operatoren:

|  |  |  |
| --- | --- | --- |
| Name | Symbol | Beschreibung |
| Additiv | + (oder keines) | Fügt die Zahl hinzu |
| Subtraktiv | - | Zieht die Zahl ab |
| Multiplikativ | \* | Multipliziert die Zahl |
| Dividierend auf | ^/ | Dividiert durch die Zahl, aufrundend |
| Dividierend ab | / | Dividiert durch die Zahl, abrundend |
| Setzend | = | Setzt einen fixen Wert |
| Ergänzend | | | Fügt Wert bei Kombinationen hinzu |

In der Regel werden nur Additive und Subtraktive Modifikatoren verwendet.

#### Setzende Modifikatoren

Setzende Modifikatoren (=) sollten wo möglich vermieden werden.

Wenn z.B. durch Hastur die Kosten für das Versiegeln von Toren von 5 auf 8 Hinweismarker erhöht wird, könnte das durch einen Setzenden Modifikator (=8) gelöst werden. Besser ist hier ein Additiver von 3 (+3). Dadurch können andere Effekte, die diese Kosten beeinflussen, neben den Hastur-Effekten zugleich wirksam sein.

#### Ergänzende Modifikatoren

Die Ergänzung ist technisch ein Bit-weises OR für Attribut-Flags. Es hat nur Anwendung auf die Modifikatoren Monster\_Attribute und Ignore\_MonsterAttribute. Diese stellen Werte dar, die eine Kombination von einzelnen Monster Attributen (siehe 4.11). Dabei muss der interne Numerische Wert des Attributes durch OR-Verknüpfung ergänzt werden.

AHS unterstützt diese Operation derzeit nicht direkt. In JS ist dies auch nicht direkt möglich. Mit folgendem Trick kann dies aber in AHS umgesetzt werden:   
|"+Constants.Monster.*Attribut*+"

### Kombination von Modifikatoren

Mehrere Modifikationen können aus mehreren Quellen zugleich auf eine Eigenschaft angewandt werden. In dem Fall werden die einzelnen Modifikatoren in der Reihenfolge oben angewandt.

### Beispiele

// Wille um 2 verringert  
[ Skill\_Will -2 ]

// Sanity-Kosten für Zaubersprüche halbiert (abgerundet)  
[ Prop\_CastCost /2 ]

// 1 Zusatzwürfel pro Hinweismarker bei Glückproben  
// UND Glück um 1 erhöht (Skill-Karte „Luck“)  
[ DieRoll\_Luck +1, Skill\_Luck +1 ]

// Endless ignorieren, sodass diese auch als Trophäen  
// gesammelt werden können (Ally John Legrasse)  
[ Ignore\_MonsterAttribute |"+Constants.Monster.Endless+" ]

### JS-Übersetzung

Für jede Modifikation wird ein Objekt mit Property property als Modifikation, und Property value dem Modifikator als String. Mehrere Modifikationen werden in ein Array gesetzt:

// Wille um 2 verringert  
{ property: Constants.Mods.Skill\_Will, value: "-2" }

// 1 Zusatzwürfel pro Hinweismarker bei Glückproben  
// UND Glück um 1 erhöht (Skill-Karte „Luck“)  
[ { property: Constants.Mods.Skill\_Luck, value: "+1" },   
{ property: Constants.Mods.DieRoll\_Luck, value: "+1" } ]

Somit funktioniert der Ergänzend-Trick auch:

// Endless ignorieren, sodass diese auch als Trophäen  
// gesammelt werden können (Ally John Legrasse)  
{ property: Constants.Mods.Ignore\_MonsterAttribute,   
 value: "|"+Constants.Monster.Endless+"" }

Rot hervorgehoben der Wert, der aus AHS eingefügt wird. Constants.Monster.Endless ist eine Konstante, der der interne numerische Wert für dieses Attribut hinterlegt ist. Dadurch entsteht eine Konkatination von 3 Strings: *"|"* , dem numerischen Wert als String, und ein Leer-String. Dies ergibt dann wieder das Endformat: "|8192".

## Monster Mods

Monster Modifikationen sind Modifikationen von Monster\_XXX, die auf einen bestimmten Monstertyp angewandt werden. Die Modifikatoren können auf alle (\*), einen bestimmten Monster-Typ (z.B. Maniac), oder auf Monster mit bestimmten Attributen (z.B. Undead) angewandt werden.

Für jeden Eintrag werden die Modifikatoren wie oben beschrieben hinterlegt.

### Syntax

#### EBNF

MonsterMods := "[" MonsterMod {"," MonsterMod} "]"  
MonsterMod := MonsterRef "{" Mods "}"  
MonsterRef := *id* | "\*" | "Attribute." *attribute*

#### Erläuterung

Monster werden mit Ihrer ID, einem Attribut (vorangestellt „Attribute.“), oder durch Stern alle gewählt. Für jedes Monster werden beliebige Mods in geschwungenen Klammern ({ und }) angegeben. Mehrere Einträge werden durch Komma (,) getrennt.

### Beispiele

// Maniac Toughness um 2 erhöht  
[ Maniac { Monster\_Toughness +2} ]

// Awareness für alle Monster um 1 verringert (schwerer für Evade/Flight)  
[ \* { Monster\_Awareness -1 } ]

// Horror Schaden für Untote um 1 verringert  
[ Attribute.Undead { Monster\_HorrorDamage -1 } ]

// Byakhee und Cultisten haben um 1 erhöhten Kampfschaden,   
// und um 1 verringerte Kampf Adjustment (schwerer zu besiegen)  
[ Byakhee { Monster\_CombatDamage +1, Monster\_CombatAdjustment -1},  
 Cultist { Monster\_CombatDamage +1, Monster\_CombatAdjustment -1} ]

### JS-Umsetzung

Die Modifikatoren werden wie oben in 5.3.5 beschrieben umgesetzt. Das ganze wird in ein Objekt gesetzt, wobei die Modifikatoren in der Property mod angegeben werden. Wenn \* angegeben wird, wird dieser als String in der Property id gegeben; Wenn ID angegeben wird, wird diese in einen ID-Wert umgewandelt und in der Property id angegeben; bei Attribute in der Property attribute. Das ganze wird in ein Array gesetzt.

// Maniac Toughness um 2 erhöht  
[ id: "MO\_Maniac", mod:   
 { property: Constants.Mods.Monster\_Toughness , value: "+2"} ]

// Awareness für alle Monster um 1 verringert (schwerer für Evade/Flight)  
[ id: "\*", mod:   
 { property: Constants.Mods.Monster\_Awareness, value: "-1" } ]

// Horror Schaden für Untote um 1 verringert  
[ attribute: Constants.Monster.Undead, mod:  
 { property: Constants.Mods.Monster\_HorrorDamage, value: "-1" } ]

// Byakhee und Cultisten haben um 1 erhöhten Kampfschaden,   
// und um 1 verringerte Kampf Adjustment (schwerer zu besiegen)  
[ id: "MO\_Byakhee", mod:  
 [ { property: Constants.Mods.Monster\_CombatDamage, value: "+1" },   
 { property: Constants.Mods.Monster\_CombatAdjustment,value:"-1"}  
 ],  
 id: "MO\_Cultist", mod:  
 [ { property: Constants.Mods.Monster\_CombatDamage, value: "+1" },   
 { property: Constants.Mods.Monster\_CombatAdjustment,value:"-1"}  
 ]  
]

## Monster Move Mods

Verändert die Bewegung von Monstern.

Es kann eine Move Modifikation für Monster per ID angegeben werden, oder ein Bewegungstyp generell durch einen anderen ersetzt werden.

### Syntax

#### EBNF

MonsterMoveMods := "[" MonsterMoveMod {"," MonsterMoveMod }"]"  
MonsterMoveMod := MonsterOrType *movementType*  
MonsterOrType := *id* | "Type." *movementType*

#### Erläuterung

In jeder Monster Move Spezifikation wird entweder eine Monster ID direkt oder ein Bewegungstyp mit vorangestelltem „Type.“ angegeben, gefolgt vom neuen Bewegungstyp. Mehrere Spezifikationen werden durch Komma (,) getrennt.

### Kombination mehrerer Typen

Derzeit können nur AncientOnes und Environment Mythos diesen Modifikator besitzen. Es gilt derzeit, dass ein Environment Vorrang hat.

**TODO: Das muss geändert werden!**

Generell gilt, dass zuerst die Monster-Spezifischen (per ID) Modifikatoren angewandt werden, dann die Bewegungs-Spezifischen Modifikatoren.

So wird z.B. durch AncientOne Hastur ein Cultist zu einem Fliegenden Monster. Durch die Environment „An Evil Fog“ werden Fliegende Monster zu Stationären 🡺 Cultist sind Stationär.

### Beispiele

// Cultisten fliegen  
[ Cultist Flying ]

// Fliegende können sich nicht bewegen  
[ Type.Flying Stationary ]

// Maniacs und Zombies sind schnell  
[ Maniac Fast, Zombie Fast ]

### JS-Übersetzung

Für jeden Eintrag wird ein Objekt erstellt. Monster-ID wird dabei in die Property id geschrieben und in eine Monster-ID umgewandelt; Ursprünglicher Bewegungstyp in die Property type, von der Enum Movement. Der neue Bewegungs-Typ wird in die Property moveAs geschrieben. Mehrere Werte werden in ein Array gepackt.

// Cultisten fliegen  
{ id: "MO\_Cultist", moveAs: Constants.Movement.Flying }

// Fliegende können sich nicht bewegen  
{ type: Constants.Movement.Flying, moveAs: Constants.Movement.Stationary }

// Maniacs und Zombies sind schnell  
[ { id: "MO\_Maniac", moveAs: Constants.Movement.Fast},   
 { id: "MO\_Zombie", moveAs: Constants.Movement.Fast} ]

## Random Possession

Zufälliger Besitz wird bei der Erstellung von Charakteren vergeben. Es gibt verschiedene Arten von Objekten, definiert über Enum ObjectType (siehe 4.2) angegeben

### Syntax

#### EBNF

RandomPossession := "[" RandomPossession "]"  
RandomPossession := RandPos {"," RandPos}  
RandPos := *type* *number*

#### Erläuterung

Für jede Art von zufälligem Objekt wird der Typ und die Anzahl angegeben. Mehrere Stellen werden durch Komma (,) getrennt.

### Beispiel

[ CommonItem 2, UniqueItem 1, Skill 1 ]

### JS-Übersetzung

Jeder Teil wird als Objekt geschrieben, Typ als Property type, und Anzahl als Property amount. Das ganze wird in ein Array gepackt.

[ { type: Contants.ObjectType.CommonItem, amount: 2 },  
 { type: Contants.ObjectType.UniqueItem, amount: 1 },  
 { type: Contants.ObjectType.Skill, amount: 1 }  
]

## Charakter Attribute

Charakter Attribute definieren die möglichen Werte für die Grund-Attribute des Charakters. Es sind immer Paare von Werten, für die Attribute Speed/Sneak, Fight/Will, und Lore/Luck.

Es sollten pro Typ immer 4 Paare angegeben werden, ein Wert sollte dabei steigen, der andere sinken.

### Syntax

#### EBNF

CharacterAttributes :=   
 "[" AttributeLine "," AttributeLine "," AttributeLine "]"  
AttributeLine := AttributeType "{" AttributePairs "}"  
AttributePairs := AttributePair { "," AttributePair }  
AttributePair := "(" *number* "," *number* ")"  
AttributeType := "SpeedSneak" | "FightWill" | "LoreLuck"  
(\* Jeder AttributType muss 1 mal vorkommen \*)

#### Erläuterung

Jeder der Attribut-Paare Speed/Sneak, Fight/Will, und Lore/Luck muss 1 mal vorkommen. Für jeden Typ werden beliebig viele Wertepaare (normalerweise 4, mindestens 1) in geschwungenen Klammern angegeben, die Paare werden durch Komma getrennt. Ein Paar besteht aus 2 Zahlen, durch Komma getrennt, von runden Klammern umschlossen.

### Beispiel

[  
 SpeedSneak {(0,4),(1,3),(2,2),(3,1)},  
 FightWill {(1,5),(2,4),(3,3),(4,2)},  
 LoreLuck {(1,5),(2,4),(3,3),(4,2)}  
]

### JS-Übersetzung

Die Attribut-Typen werden so wiederverwendet, als Properties eines Objekts. Die Paare und Paar-Listen werden in Arrays umgewandelt.

{ FightWill: [  
 [1,5],  
 [2,4],  
 [3,3],  
 [4,2]  
 ],  
 LoreLuck: [  
 [1,5],  
 [2,4],  
 [3,3],  
 [4,2]  
 ],  
 SpeedSneak: [  
 [0,4],  
 [1,3],  
 [2,2],  
 [3,1]  
 ]  
}

# Callback Funktionen

## Allgemenes

Gewisse Klassen haben Attribute vom Typ Funktion. Diese werden mit bestimmten Parametern aufgerufen, und manche haben ein this Objekt. Hier werden diese Funktionen beschrieben.

## Action / QuickOption

Eine Action kann alleine stehen (z.B. bei Begegnungen), oder an ein GameObject (UniqueItem, SpecialAbility, etc.) gebunden sein.

### activate

Aktionsfunktion einer Action.

#### this

Bei alleinstehenden Aktionen ist kein this Objekt vorhanden

Bei Objekt-gebundenen Aktionen ist das this Objekt das ursprüngliche Objekt

#### Parameter

Keine

#### Return

false bei Fehler. Dann gilt die Aktion als nicht durchgeführt, und der Benutzer muss evtl. eine weitere Option wählen, falls diese Aktion durch eine Option ausgelöst wurde.

true / nichts, wenn Ok.

## Option / QuickOption

Eine Option kann alleine stehen (z.B. bei Begegnungen), oder an ein GameObject (UniqueItem, SpecialAbility, etc.) gebunden sein.

### isAvailable

Prüft, ob die Option derzeit zur Verfügung steht.

#### this

Bei alleinstehenden Optionen ist kein this Objekt vorhanden

Bei Objekt-gebundenen Optionen ist das this Objekt das ursprüngliche Objekt

#### Parameter

Keine

#### Return

true, wenn die Option verfügbar ist; false wenn sie nicht verfügbar ist.

## Objekte

Objekte sind alle Typen von Objekten, die im Inventar vorhanden sein können: CommonItem, UniqueItem, Skill, Spell, Ally, SpecialAbility, SpecialObject.

### modifications

Dynamische Modifikatoren. Einfache Modifikatoren können wie oben unter 5.3 beschrieben angegeben werden.

Wenn die Modifikatoren eines Objekts von der Spielsituation abhängt (z.B. CommonItem.Cross hat nur Bonus im Kampf gegen Untote), kann stattdessen eine Funktion angegeben werden. Diese muss Modifikatoren als JS-Objekt zurückgeben.

#### this

Das Objekt

#### Parameter

Keine

#### Return

Modifikatoren, im Format wie unter 5.3.5 dargestellt.

### onAddToInventory

Wird aufgerufen, wenn das Objekt ins Inventar eines Spielers hinzugefügt wird

#### this

Das Objekt

#### Parameter

c: Der Charakter, in dessen Inventar das Objekt hinzugefügt wird

#### Return

false, wenn das Objekt *nicht* hinzugefügt werden soll.

true / nichts, wenn das Objekt hinzugefügt werden soll.

### onRemoveFromInventory

Wird aufgerufen, wenn das Objekt aus dem Inventar eines Spielers entfernt wird

#### this

Das Objekt

#### Parameter

c: Der Charakter, aus dessen Inventar das Objekt entfernt wird

#### Return

false, wenn das Objekt *nicht* entfernt werden soll.

true / nichts, wenn das Objekt entfernt werden soll.

## Spell

Spells haben alle Objektfunktionen.

### cast

Spells können zusätzlich eine cast-Funktion haben, die aufgerufen wird, wenn der Spruch aktiviert wird (nicht bei Waffen-Spells, wenn der Spell im Angriff verwendet wird.)

#### this

Das Objekt (=der Spell)

#### Parameter

Keine

#### Return

Kein

## Monster

### onMove

Monster mit MovementType Special benötigen eine onMove Funktion, die aufgerufen wird, wenn sich das Monster bewegt.

#### this

Das Monster

#### Parameter

Keine

#### Return

Kein

### onDamage

Wird aufgerufen, wenn das Monster einem Charakter Schaden zufügt. Der „Normale“ Schaden wurde zu dem Zeitpunkt bereits zugefügt.

#### this

Das Monster

#### Parameter

* type: Der Schaden-Typ (siehe 4.15)
* chr: Der Charakter, dem der Schaden zugefügt wird

#### Return

false wenn der Kampf abgebrochen wird (z.B. die Damage Funktion hat den Kampf beendet weil der Charakter automatisch Lost In Space And Time ist)

true / nichts, wenn der Kampf normal fortgesetzt wird.

### onDefeat

Wird aufgerufen, wenn das Monster von einem Charakter besiegt wurde.

#### this

Das Monster

#### Parameter

* chr: Der Charakter, von dem das Monster besiegt wurde

#### Return

Keine

### onFlee

Wird aufgerufen, wenn ein Charakter vor dem Monster geflohen ist

#### this

Das Monster

#### Parameter

* chr: Der Charakter, von dem das Monster besiegt wurde

#### Return

Keine

### onEvade

Wird aufgerufen, wenn ein Charakter dem Monster ausgewichen ist

#### this

Das Monster

#### Parameter

* chr: Der Charakter, von dem das Monster besiegt wurde

#### Return

Keine

### modifications

Dynamische Modifikatoren. Einfache Modifikatoren können wie oben unter 5.3 beschrieben angegeben werden.

Wenn die Modifikatoren eines Monsters von der Spielsituation abhängt (z.B. Maniac hat höhere Werte wenn TerrorLevel >= 6), kann stattdessen eine Funktion angegeben werden. Diese muss Modifikatoren als JS-Objekt zurückgeben.

#### this

Das Monster

#### Parameter

Keine

#### Return

Modifikatoren, im Format wie unter 5.3.5 dargestellt.

### attributes

Dynamische Attribute. Einfache Attribute können wie oben unter 4.11 beschrieben angegeben werden.

Wenn die Attribute eines Monsters von der Spielsituation abhängt (z.B. Maniac ist Endless wenn TerrorLevel >= 6), kann stattdessen eine Funktion angegeben werden. Diese muss ein Array von Attributen oder eine Bit-weise OR-Kombination von Attributen zurückliefern

#### this

Das Monster

#### Parameter

Keine

#### Return

Attribute

## Headline

### executeHeadline

Funktion der Headline.

#### this

Die Headline

#### Parameter

Keine

#### Return

Kein

## Rumor

Rumors haben mehrere Funktionen, die den Lebenszyklus des Gerüchts darstellen.

### setupRumor

Vorbereitung. Wird aufgerufen, wenn das Rumor ins Spiel kommt.

#### this

Das Rumor

#### Parameter

Keine

#### Return

Kein

### teardownRumor

Aufräumen. Wird aufgerufen, wenn das Rumor aus dem Spiel genommen wird (nach Pass bzw. Fail)

#### this

Das Rumor

#### Parameter

Keine

#### Return

Kein

### onMythos

Fortschritt. Wird in jeder Mythos-Phase (nach der, in der das Rumor ins Spiel gekommen ist) aufgerufen.

Wenn hier der Fail oder Pass Status erkannt wird, sollte pass bzw. fail auf dem Rumor aufgerufen wreden.

#### this

Das Rumor

#### Parameter

Keine

#### Return

Kein

### onPass

Wird aufgerufen, wenn die Spieler Erfolg hatten.

#### this

Das Rumor

#### Parameter

Keine

#### Return

Kein

### onFail

Wird aufgerufen, wenn die Spieler versagt haben.

#### this

Das Rumor

#### Parameter

Keine

#### Return

Kein

## Ancient One

### onAwake

Vorbereitung, wenn der Große Alte erwacht. Das ist direkt, bevor der Endkampf beginnt. Hier werden z.B. alle Charaktere devoured, die keine Hinweismarker haben (Nyarlathotep), oder verlieren Gegenstände bei nicht bestandenem Würfeltest (Ithaqua).

#### this

Der Ancient One

#### Parameter

Keine

#### Return

Kein

### attack

Angriff des Ancient Ones. Diese Funktion wird für jeden Spieler aufgerufen, der noch im Spiel ist.

Im Allgemeinen wird hier ein Skill-Test abgelegt, dessen Erschwernis sich jede Kampfrunde erhöht. Die Erschwernis wird automatisch in this.attackAdjustment mitgeliefert.

#### this

Der Ancient One

#### Parameter

Keine

#### Return

Kein

### postAttack

Wird in jeder Angriffsrunde aufgerufen, nachdem alle Spieler angegriffen wurden, und der Kampf noch nicht beendet ist.

Hier können Anpassungen vor der nächsten Kampfrunde durchgeführt werden (der attackAdjustment wird nach diesem Callback automatisch erhöht).

#### This

Der Ancient One

#### Parameter

Keine

#### Return

bool: Wenn false ist der Kampf beendet und die Spieler haben verloren. Wenn true / fehlt geht der Kampf normal weiter.

# Library

Arkham Horror bietet zur Interaktion mit dem Spiel verschiedene Objekte, Properties und Funktionen zur Verfügung, die aus den obigen Callback-Funktionen aufgerufen werden können. Dadurch wird die Interaktion aus Spiel und gescripteten Objekten vervollständigt.

## Globale Objekte

### Constants

Dieses Objekt definiert alle Aufzählungswerte, die in AH verwendet werden. Die Enums sind in Bereiche unterteilt. Für Details siehe Kapitel 4.

### ID Konstanten

Mit AHS erstellte Objekte registrieren automatisch eine Konstante. Mit JS erstellte Objekte können dies auch tun (mit game.registerConstant).

Über diese Konstanten können Objekte referenziert werden (z.B. CommonItem.Dynamite), anstatt der textuellen ID ("CI\_Dynamite").

Siehe Kapitel 3, wie die Objekte registriert werden.

### CHAR

Das Objekt CHAR ist ein Shortcut für game.context.character.

### game

Das globale Spiel Objekt. Siehe unten.

### game.context

Das Kontext Objekt. Siehe unten.

## game

Das globale Spiel-Objekt. Es liefert Zugriff auf alle Erstellungs- und Registrierungs-Funktionen (automatisch in AHS behandelt, siehe Kapitel 3), und generelle Funktionen, die nicht von der aktuellen Spielsituation abhängen.

### game.context

Das Kontext Objekt. Siehe unten.

### registerConstant

Registriert eine Konstante im globalen Scope zur Referenzierung von Objekten.

#### Parameter

* scope: der Scope, in dem die Konstante erstellt werden soll
* name: Name der Konstanten
* value: Wert der Konstanten

#### Return

bool: true, wenn die Konstante erstellt wurde, false wenn bereits eine Konstante mit diesem Namen in diesem Scope existiert.

#### Verwendung

// Erstellt Konstante CommonItem.Dynamite mit Wert CI\_Dynamite  
game.registerConstant("ComonItem", "Dynamite", "CI\_Dynamite");

### cardsOnDeck

Ermittelt die Anzahl verbliebener Karten eines Typs auf dem Stapel.

#### Parameter

* type: Typ der Karte (Enum ObjectType, siehe 4.2)

#### Return

int: Anzahl verbliebener Karten auf dem Stapel

#### Verwendung

// Ermittelt Anzahl verbliebener Allies  
var allyCount = game.cardsOnDeck(Constants.ObjectType.Obj\_Ally);

### drawSingleObject

Zieht eine zufällige Objekt-Karte vom angegebenen Typ aus dem Karten-Stapel. Der Stapel wird zuvor gemischt.

Wenn ein Objekt von einem Spieler gezogen wird, um es in sein Inventar hinzuzufügen, sollte stattdessen game.context.drawObject verwendet werden (siehe 7.3.10).

#### Parameter

* type: Typ der Karte (Enum ObjectType, siehe 4.2)

#### Return

GameObject: Das gezogene Objekt, oder null wenn keine Karten mehr im Stapel sind.

#### Verwendung

// Zieht einen zufälligen Zauber  
var sp = game.drawSingleObject(Constants.ObjectType.Obj\_Spell);  
if (sp == null) { /\* Kein Zauber mehr vorhanden \*/ }

### drawSpecificObject

Zieht ein bestimmtes Objekt aus dem Kartenstapel.

#### Parameter

* id: ID des Objekts

#### Return

GameObject: Das gezogene Objekt, oder null wenn keine Karten mehr von diesem Objekt im Stapel sind.

#### Verwendung

// Zieht Duke aus dem Ally-Stapel. Alternativ auch "AL\_Duke" möglich  
var duke = game.drawSpecificObject(Ally.Duke);  
if (duke == null) { /\* Duke wurde bereits gezogen\*/ }

### returnMonstersFromField

Entfernt alle Monster von einem bestimmten Feld. Die Monster landen wieder im Monsterpool.

#### Parameter

* fieldId: ID des Feldes (Enum Field, siehe 4.17)

#### Return

bool: true, wenn mindestens 1 Monster entfernt wurde, false wenn keines entfernt wurde.

#### Verwendung

// Entfernt alle Monster aus dem Straßengebiet von NorthSide  
var removed = game.returnMonstersFromField(Constants.Fields.NS\_NorthSide);  
if (removed) { /\* Monster wurden entfernt \*/ }  
else { /\* Keine Monster wurden entfernt \*/ }

### returnMonstersFromFields

Entfernt alle Monster von mehreren Feldern. Die Monster landen wieder im Monsterpool.

#### Parameter

* fieldIds: Array von IDs der Felder (Enum Field, siehe 4.17)

#### Return

bool: true, wenn mindestens 1 Monster entfernt wurde, false wenn keines entfernt wurde.

#### Verwendung

// Entfernt alle Monster aus NorthSide  
var removed = game.returnMonstersFromFields( [  
 Constants.Fields.NS\_NorthSide,  
 Constants.Fields.NS\_TrainStation,  
 Constants.Fields.NS\_Newspaper,  
 Constants.Fields.NS\_CuriositieShoppe  
] );  
if (removed) { /\* Monster wurden entfernt \*/ }  
else { /\* Keine Monster wurden entfernt \*/ }

### returnMonstersFromFieldType

Entfernt alle Monster von allen Feldern eines bestimmten Typs. Die Monster landen wieder im Monsterpool.

#### Parameter

* fieldType: Art der Felder (Enum FieldType, siehe4.16)

#### Return

bool: true, wenn mindestens 1 Monster entfernt wurde, false wenn keines entfernt wurde.

#### Verwendung

// Entfernt alle Monster aus allen Straßengebieten  
var removed = game.returnMonstersFromFieldType(  
 Constants.FieldType.Street);  
if (removed) { /\* Monster wurden entfernt \*/ }  
else { /\* Keine Monster wurden entfernt \*/ }

### returnMonsterTypeFromBoard

Entfernt alle Monster eines bestimmten Typs vom Spielbrett. Die Monster landen wieder im Monsterpool.

#### Parameter

* typeId: ID des Monster-Typs

#### Return

bool: true, wenn mindestens 1 Monster entfernt wurde, false wenn keines entfernt wurde.

#### Verwendung

// Entfernt alle Dark Youngs aus dem Spiel  
var removed = game.returnMonsterTypeFromBoard(Monster.DarkYoung);  
if (removed) { /\* Monster wurden entfernt \*/ }  
else { /\* Keine Monster wurden entfernt \*/ }

### spawnMonster

Erzeugt ein Monster auf einem Feld.

#### Parameter

* fieldId: ID des Feldes (Enum Field, siehe 4.17). Muss ein Feld vom Typ Interior sein

#### Return

bool: true wenn ein Monster erzeugt wurde, false sonst (ungültiges Feld, keine Monster mehr)

#### Verwendung

// Erzeugt ein Monster in den Woods  
game.spawnMonster(Constants.Fields.UT\_Woods);

### createGate

Erzeugt ein Tor auf einem Feld. Wenn das Feld versiegelt ist, geschiegt nichts. Wenn auf dem Feld bereits ein Tor ist, werden Monster aus allen offenen Toren ausgegeben. Wenn noch kein Tor vorhanden ist, wird eines erzeugt und Monster erscheinen.

#### Parameter

* fieldId: ID des Feldes (Enum Field, siehe 4.17). Muss ein Feld vom Typ Interior sein

#### Return

Kein

#### Verwendung

// Erzeugt ein Gate in den Woods  
game.createGate(Constants.Fields.UT\_Woods);

## game.context

Das Context Objekt liefert Zugriff auf Objekte und Funktionen, die von der aktuellen Spielsituation abhängen, wie aktueller Spieler, oder momentan bekämpftes Monster.

### character

Zugriff auf den aktuellen Spieler. Für verfügbare Funktionen siehe 0.

### allCharacters

Liste aller Charaktere.

#### Verwendung

for (var i = 0; i < game.context.allCharacters.length; ++i) {  
 var c = game.context.allCharacters[i];  
 // Aktionen, die für alle Charaktere ausgeführt werden sollen  
 // Bsp.: 1$ hinzufügen  
 c.addMoney(1);  
}

### monster

Zugriff auf das aktuell bekämpfte Monster. Dies kann auch null sein, wenn kein Monster bekämpft wird.

### gate

Zugriff auf das Tor, auf dem sich der aktuelle Charakter befindet. Dies kann auch null sein, wenn kein Tor vorhanden ist.

Typ: Gate. Für verfügbare Funktionen siehe 7.4.20.

### activeRumor

Gibt das aktuell aktive Gerücht zurück. Dies kann auch null sein, wenn kein Gerücht aktiv ist.

Typ: MythosCard. Für verfügbare Funktionen siehe 7.6.2.

**TODO: sollte das nicht in game verschoben werden? Hängt ja nicht vom Spieler o.Ä. ab…**

### terrorLevel

Gibt den aktuellen Terror Level zurück.

Typ: int

**TODO: sollte das nicht in game verschoben werden? Hängt ja nicht vom Spieler o.Ä. ab… Review zusammen mit TerrorLevel Modifikatoren!**

### ancientOne

Gibt Zugriff auf den Ancient One.

Typ: AncientOne. Für verfügbare Funktionen siehe 7.9

**TODO: sollte das nicht in game verschoben werden? Hängt ja nicht vom Spieler o.Ä. ab…**

### increaseTerrorLevel

Erhöht den Terror Level

**TODO: sollte das nicht in game verschoben werden? Hängt ja nicht vom Spieler o.Ä. ab…**

#### Parameter

* amount: Wert (Optional, default 1)

#### Return

Kein

### decreaseTerrorLevel

Verringert den Terror Level

**TODO: sollte das nicht in game verschoben werden? Hängt ja nicht vom Spieler o.Ä. ab…**

#### Parameter

* amount: Wert (Optional, default 1)

#### Return

Kein

### drawObject

Zieht eine zufällige Objekt-Karte vom angegebenen Typ aus dem Karten-Stapel. Der Stapel wird zuvor gemischt.

Diese Funktion berücksichtigt eventuelle Modifikatoren, die den aktuellen Spieler betreffen, die das Ziehen von Karten beeinflussen (z.B. 2 ziehen und 1 davon wählen).

#### Parameter

* type: Typ der Karte (Enum ObjectType, siehe 4.2)
* descr: Beschreibungstext (Optional)

#### Return

GameObject: Das gezogene Objekt, oder null wenn keine Karten mehr im Stapel sind.

#### Verwendung

// Zieht einen zufälligen Zauber  
var sp = game.context.drawObject(Constants.ObjectType.Obj\_Spell);  
if (sp == null) { /\* Kein Zauber mehr vorhanden \*/ }

// Beispiel mit Beschreibungstext  
var obj = game.context.drawObject(Constants.ObjectType.Obj\_CommonItem,   
 "Receive reward");

### drawMultipleObjects

Zieht mehrere zufällige Objekt-Karte vom angegebenen Typ aus dem Karten-Stapel. Eine gewisse Zahl davon muss der Spieler auswählen. Nicht gewählte Objekte werden automatisch in den Stapel zurückgelegt. Der Stapel wird zuvor gemischt.

Diese Funktion berücksichtigt eventuelle Modifikatoren, die den aktuellen Spieler betreffen, die das Ziehen von Karten beeinflussen (z.B. 2 ziehen und 1 davon wählen).

#### Parameter

* type: Typ der Karte (Enum ObjectType, siehe 4.2)
* descr: Beschreibungstext
* count: Anzahl der Karten, die angeboten werden
* min: Anzahl der Karten, die gewählt werden müssen
* max: Maxximale Anzahl der Karten, die gewählt werden darf

#### Return

Array von GameObject: Die gewählten Objekte.

#### Verwendung

// Ziehe 5 Common Items, wähle 0 – 2 aus  
var objs = game.context.drawMultipleObjects(  
 Constants.ObjectType.Obj\_CommonItem,   
 "Select Items to buy",  
 5,  
 0,  
 2);  
for (var i = 0; i < obj.length; ++i) {  
 // Aktion mit gewählten Objekten, z.B. ins Inventar  
 CHAR.addToInventory(obj[i]);  
}

### drawMixedObjects

Zieht mehrere zufällige Objekt-Karte von mehreren Typen aus den Karten-Stapeln. Eine gewisse Zahl davon muss der Spieler auswählen. Nicht gewählte Objekte werden automatisch in die Stapel zurückgelegt. Die Stapel werden zuvor gemischt.

Diese Funktion berücksichtigt eventuelle Modifikatoren, die den aktuellen Spieler betreffen, die das Ziehen von Karten beeinflussen (z.B. 2 ziehen und 1 davon wählen).

#### Parameter

* descr: Beschreibungstext
* types: Array von Typ und Anzahl der Karten (vgl. Random Possessions bei Investigatoren, siehe 3.21.4)
* min: Anzahl der Karten, die gewählt werden müssen
* max: Maxximale Anzahl der Karten, die gewählt werden darf

#### Return

Array von GameObject: Die gewählten Objekte.

#### Verwendung

// Ziehe 2 Common Items und 1 Unique, wähle 1 aus  
var types = [  
 {type: Constants.ObjectType.CommonItem, amount: 2},  
 {type: Constants.ObjectType.UniqueItem, amount: 1}  
];  
var objs = game.context.drawMixedObject(  
 "Receive Reward", types, 1, 1);

### selectChoice

Bietet dem Spieler mehrere Möglichkeiten zur Auswahl, wovon er 1 wählen muss.

#### Parameter

* desc: Beschreibungstext
* options: Array von Wahlmöglichkeiten
* canCancel: Ob der Benutzer abbrechen kann (Optional, default false)

Die Optionen können einen Namen, eine ID und einen Beschreibungstext haben. Sie können auf verschiedene Arten angegeben werden.

1. Als einfacher String  
   ID = Name
2. Als Array mit 1, 2 oder 3 Strings  
   Bei 1 ist ID = Name
3. Als Objekt mit 3 Properties: id, name, description

#### Return

ID der gewählten Option

#### Verwendung

// Als einfache Strings: Bietet Möglichkeiten Stamina, Sanity und Pay  
// Rückgabe ist Stamina oder Sanity oder Pay. Wahl muss getroffen werden  
var choice = game.context.selectChoice([ "Stamina", "Sanity", "Pay" ]);

// Als Arrays  
// Bietet 3 Möglichkeiten: Sanity, Stamina, Pay; wobei Pay eine weitere  
// Beschreibung hat.  
// Liefert Sanity (hat keine ID), STA oder PAY zurück  
// Der Benutzer kann abbrechen (liefert leeren String zurück)  
var choice = game.context.selectChoice( [  
 [ "Sanity" ],  
 [ "STA", "Stamina" ],  
 [ "PAY", "Pay", "Pay 5 Clue Markers instead ]  
], true );

// Als Objekte: Analog zu oben, nur muss Sanity eine ID haben.  
var choice = game.context.selectChoice( [  
 { id: "SAN", name: "Sanity" },  
 { id: "STA", name: "Stamina" },  
 [ id: "PAY", name: "Pay", description: "..." }  
] );

// Auswertung:  
if (choice == "SAN" ) { /\*... \*/ }  
else if (choice == "STA" ) { /\*... \*/ }  
else if (choice == "PAY" ) { /\*... \*/ }

### spontaneousMonsterFight

Startet einen Kampf mit einem zufälligen Monster.

Wenn das Monster nicht besiegt wird (Spieler besiegt, geflohen, ausgewichen), wird das Monster in die Monsterquelle zurückgelegt.

#### Parameter

Keine

#### Return

bool: false, wenn der Charakter besiegt wurde, true, wenn er das Monster besiegt hat, geflohen oder ausgewichen ist

#### Verwendung

if (!game.context.spontaneousMonsterFight()) {  
 // Charakter wurde besiegt.  
}

### createGateAtCharacterField

Erzeugt ein Tor am Feld des aktuellen Spielers. Es gelten die Regeln für Tore erzeugen (vgl. 7.2.11). Wenn ein Tor erzeugt wurde, wird der Spieler hineingezogen.

#### Parameter

Keine

#### Return

Bool: true wenn ein Tor erzeugt wurde, sonst false

#### Verwendung

// Erzeugt Gate beim aktuellen Spieler  
var res = game.context.createGateAtCharacterField();

### genericDieRollSum

Würfelt mit einer fixen Zahl von Würfeln und liefert die Summe.

#### Parameter

* desc: Beschreibungstext
* dieCount: Anzahl Würfel

#### Return

int: Summe der Würfel

#### Verwendung

// Wirf 1 Würfel, bei ungerade devour  
var res = game.context.genericDieRollSum( "Die on odd number", 1 );  
if ( res % 2 == 1) {  
 // Ungerade  
 CHAR.devour();  
}

### genericDieRollCount

Würfelt mit einer fixen Zahl von Würfeln und liefert die Anzahl an Erfolgen.

Erfolge sind normalerweise 5 und 6. Segnung und Fluch werden berücksichtigt.

#### Parameter

* desc: Beschreibungstext
* dieCount: Anzahl Würfel

#### Return

int: Anzahl der Erfolge

#### Verwendung

// Wirf 3 Würfel. Bei 0 Erfolgen 1 Stamina Schaden,   
// sonst Clues in Anzahl an Erfolgen  
var res = game.context.genericDieRollCount( "Check", 3 );  
if ( res == 0) {  
 // Kein Erfolg  
 CHAR.damageStamina(1);  
} else {  
 // Mind. 1 Erfolg => Clues hinzufügen  
 CHAR.addClue(res);  
}

### genericDieRollTest

Würfelt mit einer fixen Zahl von Würfeln und vergleicht die Anzahl an Erfolgen mit einem Zielwert.

Erfolge sind normalerweise 5 und 6. Segnung und Fluch werden berücksichtigt.

#### Parameter

* desc: Beschreibungstext
* dieCount: Anzahl Würfel
* target: Zielwert

#### Return

bool: true, wenn Zielwert erreicht wurde, false wenn nicht.

#### Verwendung

// Wirf 2 Würfel, mindestens 1 Erfolg  
var res = game.context.genericDieRollTest( "Test", 2, 1 );  
if ( res ) {  
 // Erreicht  
}

### dieRollSkillCount

Würfelt mit einer Skill-Abhängigen Anzahl von Würfeln und liefert die Anzahl an Erfolgen.

Erfolge sind normalerweise 5 und 6. Segnung und Fluch werden berücksichtigt.

Alle Modifikatoren, die den Skill betreffen, werden berücksichtigt.

#### Parameter

* desc: Beschreibungstext
* skill: Skill als Basis. Enum Skill (siehe 4.1)
* adjust: Anpassungswert auf Basis

#### Return

int: Anzahl der Erfolge

#### Verwendung

// Wirf mit Glück als Skill -1. Bei 0 Erfolgen 1 Sanity Schaden,  
// Sonst Anuahl Erfolge-1 $  
var res = game.context.dieRollSkillCount(   
 "Check", Constants.Skills.Luck, -1 );  
if ( res == 0) {  
 // Kein Erfolg  
 CHAR.damageSanity(1);  
} else {  
 // Mind. 1 Erfolg => Geld hinzufügen  
 CHAR.addMoney(res - 1);  
}

### skillTest

Eine Skill-Probe. Würfelt mit einer Skill-Abhängigen Anzahl von Würfeln vergleicht die Anzahl an Erfolgen mit einem Zielwert.

Erfolge sind normalerweise 5 und 6. Segnung und Fluch werden berücksichtigt.

Alle Modifikatoren, die den Skill betreffen, werden berücksichtigt.

#### Parameter

* desc: Beschreibungstext
* skill: Skill als Basis. Enum Skill (siehe 4.1)
* adjust: Anpassungswert auf Basis
* target: Zielwert

#### Return

bool: true, wenn Zielwert erreicht wurde, false wenn nicht.

#### Verwendung

// Eine Lore(+1)[2] Probe  
var res = game.context.skillTest( "Test", Constants.Skills.Lore, +1, 2 );  
if ( res ) {  
 // Erreicht  
}

## Charakter

Ein Spieler. Der aktuelle Spieler kann mit dem globalen Objekt CHAR zugegriffen werden.

### sanity

Aktuelle geistige Gesundheit. Nur lesend.

Typ: int

#### Verwendung

// Prüft ob sanity größer 2 ist  
if (CHAR.sanity > 2) { /\* Tu was \*/ }

### stamina

Aktuelle Ausdauer. Nur lesend

Typ: int

#### Verwendung

// Prüft ob stamina größer 2 ist  
if (CHAR.stamina > 2) { /\* Tu was \*/ }

### field

Spielfeld, auf dem sich der Character befindet.

Typ: Field

#### Verwendung

// Prüft, ob der Spieler auf einem Straßen-Feld ist  
if (CHAR.field.type == Constants.FieldType.Street) { }

### inventory

Liste aller Objekte im Inventar (inklusive Special Ability, Special Objects, Skills, …)

Typ: Liste von GameObject

#### Verwendung

for (var i = 0; i < CHAR.inventory.length; ++i) {  
 var o = CHAR.inventory[i];  
 if (o.type == Constants.ObjectType.CommonItem) { /\* Common Item \*/ }  
}

### getSkillValue

Liefert den Wert eines Skills des Charakters, unter Berücksichtigung aller Modifikatoren.

#### Parameter

* skill: Enum Typ Skills (siehe 4.1)

#### Return

int: Wert des Skills

#### Verwendng

If (CHAR.getSkillValue(Constants.Skills.Will) > 5) {  
 // Aktueller Wille Wert ist größer 5  
}

### addToInventory

Fügt ein Objekt ins Inventar hinzu.

#### Parameter

* obj: Das Objekt (kann auch null sein)

#### Return

Kein

#### Verwendung

// Segnet den Spieler (falls noch Segnungen vorhanden)  
var obj = game.drawSpecificObject(BlessingCurse.Blessing);  
CHAR.addToInventory(obj);

### removeFromInventory

Entfernt ein Objekt aus dem Inventar. Der Spieler muss der Besitzer des Objekts sein.

#### Parameter

* obj: Das Objekt (kann auch null sein)

#### Return

Kein

#### Verwendung

// Als Action eines Objekts: Füge 1 Clue hinzu  
// Siehe auch GameObject, Kapitel 7.4.20  
this.owner.addClue(1);  
this.owner.removeFromInventory(this); // Alternativ: this.discard();

### hasObject

Prüft, ob der Character ein Objekt bestimmten Typs hat.

#### Parameter

* typeId: Der Objekttyp

#### Return

bool: true, wenn er ein Objekt dieses Typs hat, false sonst

#### Verwendung

if (CHAR.hasObject(BlessingCurse.Blessing)) { /\* Ist gesegnet \*/ }

### damageSanity

Fügt geistige Gesundheit Schaden zu.

#### Parameter

* amount: Menge

#### Return

Kein

#### Verwendung

// Fügt jedem Spieler 1 Sanity Schaden zu  
for (var c in game.context.allCharacters) {  
 c.damageSanity(1);  
}

### addSanity

Erhöht aktuelle Geistige Gesundheit um einen Wert, jedoch nicht über das Maximum.

#### Parameter

* amount: Menge

#### Return

Kein

#### Verwendung

// Heilt 2 Sanity  
CHAR.addSanity(2);

### restoreSanity

Füllt geistige Gesundheit zum Maximum auf.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

// Füllt Sanity auf  
CHAR.restoreSanity();

### preventDamageSanity

**TODO: geht das so???**

### damageStamina

Fügt Ausdauer-Schaden zu.

#### Parameter

* amount: Menge

#### Return

Kein

#### Verwendung

// Fügt jedem Spieler 1 Stamina Schaden zu  
for (var c in game.context.allCharacters) {  
 c.damageStamina(1);  
}

### addStamina

Erhöht aktuelle Ausdauer um einen Wert, jedoch nicht über das Maximum.

#### Parameter

* amount: Menge

#### Return

Kein

#### Verwendung

// Heilt 2 Stamina  
CHAR.addStamina (2);

### restoreStamina

Füllt Ausdauer zum Maximum auf.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

// Füllt Stamina auf  
CHAR.restoreStamina();

### preventDamageStamina

**TODO: geht das so???**

### commitDamage

Fügt den akkumulierten Sanity und Stamina Schaden zu, und prüft, ob der Charakter dadurch ausfällt.

Schaden wird im ersten Schritt nicht direkt zugefügt (über damageStamina / damageSanity oder allgemeinen Spielregeln, wie Monster-Schaden), sondern wird nur aufsummiert. Das ist notwendig, da normalerweise Stamina/Sanity automatisch auf 1 gesetzt wird, sobald sie auf 0 fällt; und somit nie die Sanity und Stamina zugleich 0 sein können (🡺 devour).

Diese Methode muss im Normalfall nicht aufgerufen werden, da sie nach jeder Aktion automatisch aufgerufen wird. Sie dient nur, um zu prüfen, ob ein Charakter durch die aktuelle Aktion Bewusstlos/Verrückt /Verschlungen wird.

#### Parameter

Keine

#### Return

bool: true, wenn der Charakter noch weiterspielen kann, false wenn er verrückt/bewusstlos/verschlungen wurde.

#### Verwendung

CHAR.damageSanity(2);  
if (!CHAR.commitDamage()) {  
 // Charakter wurde durch diesen Schaden verrückt  
}

### lostInSpaceAndTime

Setzt den Spieler verloren In Zeit und Raum.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

CHAR.lostInSpaceAndTime();

### devour

Verschlingt den Spieler. Bevor der Ancient One erwacht, kann der Spieler einen neuen Charakter auswählen. Wenn der Ancient One erwacht ist, hat der Spieler verloren.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

CHAR.devour();

### addClue

Fügt Hinweismarker hinzu.

#### Parameter

* amount: Anzahl

#### Return

Kein

#### Verwendung

CHAR.addClue(2);

### addMoney

Fügt Geld hinzu.

#### Parameter

* amount: Anzahl

#### Return

Kein

#### Verwendung

CHAR.addMoney(2);

### loseMoney

Spieler verliert Geld.

Für Bezahlungen für Objekte oder Optionen sollte pay verwendet werden.

#### Parameter

* amount: Anzahl

#### Return

Kein

#### Verwendung

CHAR.loseMoney(2);

### loseHalfMoney

Spieler verliert die Hälfte (abgerundet) seines Geldes.

#### Parameter

Kein

#### Return

Kein

#### Verwendung

CHAR.loseHalfMoney();

### addMovementPoint

Fügt Bewegungspunkte hinzu. Diese Punkte können nur in der aktuellen Bewegungsphase ausgegeben werden.

#### Parameter

* amount: Anzahl

#### Return

Kein

#### Verwendung

CHAR.addMovementPoint(2);

### canPay

Prüft, ob der Spieler gewisse Kosten bezahlen könnte.

#### Parameter

* cost: Kosten. Definition gleich wie bei Kosten für Optionen, siehe 5.2.2

#### Return

bool: true, wenn der Spieler bezahlen könnte, false wenn nicht

#### Verwendung

var costs = [ { type: Constants.Costs.GateTrophy, amount: 1 },  
 { type: Constants.Costs.MonsterToughness, amount: 5 } ];  
if (CHAR.canPay(costs)) {  
 // Charakter könnte 1 Tor Trophöe ODER 5 Monster Ausdauer bezahlen  
}

### pay

Lässt den Spieler Kosten bezahlen. Wenn mehrere Alternativen zur Verfügung stehen, wird der Spieler nachgefragt.

#### Parameter

* cost: Kosten. Definition gleich wie bei Kosten für Optionen, siehe 5.2.2

#### Return

bool: true, wenn der Spieler bezahlt hat, false wenn nicht

#### Verwendung

// Implementierung von General Store  
// 3 Objekte zur Auswahl, Spieler kann 1 wählen  
var objs = game.context.drawMultipleObjects(   
 "Buy", Constants.ObjectType.Obj\_CommonItem, 3, 0, 1);  
if (objs.length > 0) {  
 // Hat eines gewählt:  
 var obj = objs[0];  
  
 // Kosten: Bezahle Preis des Objekts in $  
 var costs = { type: Constants.Costs.Money, amount: obj.price };  
 if (CHAR.pay(costs)) {  
 // Hat bezahlt, bekommt das Objekt  
 CHAR.addToInventory(obj);  
 } else {  
 // Nicht bezahlen können 🡺 Zurück  
 obj.returnToDeck();  
 }  
}

### losePossessions

Spieler verliert eine bestimmte Anzahl an Objekten (nur CommonItem und UniqueItem). Berücksichtigt das Attribut CannotBeLost.

Der Spieler kann die Objekte, die er verliert, auswählen.

#### Parameter

* count: Anzahl an Objekten, die er verliert

#### Return

Kein

#### Verwendung

// Spieler verliert 1 Gegenstand seiner Wahl  
CHAR.losePossesions(1);

### loseHalfPossessions

Spieler verliert die Hälfte (abgerundet) seiner Objekten(nur CommonItem und UniqueItem). Berücksichtigt das Attribut CannotBeLost.

Der Spieler kann die Objekte, die er verliert, auswählen.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

// Spieler verliert Hälfte seiner Gegenstand seiner Wahl  
CHAR.loseHalfPossesions();

### placeOnField

Setzt den Spieler auf das angegebene Feld.

#### Parameter

* field: ID des Feldes (Enum Fields, siehe 4.17)

#### Return

bool: false, wenn das Feld ungültig ist, true ansonsten

#### Verwendung

// Setzt den Spieler ins Science Building  
CHAR.placeOnField(Constants.Fields. MU\_ScienceBuilding);

### returnToArkham

Der Spieler kehrt aus einer Anderen Welt nach Arkham zurück.

**TODO: Derzeit wird ein Tor genommen, Benutzer kann nicht wählen**

#### Parameter

Keine

#### Return

bool: true, wenn der Spieler zurückgekehrt ist, false wenn nicht. Das kann sein, wenn alle Tore zurück nach Arkham aus dieser Anderen Welt geschlossen sind (oder der Spieler nicht in einer anderen Welt ist).

#### Verwendung

CHAR.returnToArkham();

### canCloseGate

Prüft, ob der Spieler das Tor am aktuellen Feld schließen kann. D.h. ob er die andere Welt bereits erforscht hat.

#### Parameter

Keine

#### Return

bool: true, wenn er das Tor schließen kann, false wenn nicht.

#### Verwendung

// Als isAvailable von Objekten, wie z.B. Altes Zeichen  
//// ...  
isAvailable: {{{ return CHAR.canCloseGate(); }}}

### setOut

Lässt den Spieler diese Runde aussetzen. D.h. er wird keine Begegnung in Arkham haben. Der Spieler wird automatisch in der nächsten Upkeep-Phase wieder zurückgesetzt.

#### Parameter

Keine

#### Return

Keine

#### Verwendung

CHAR.setOut();

### delay

Verzögert den Spieler. D.h. dass er sich in der nächsten Movement-Phase nicht bewegen kann und auch keine Movement-Optionen (z.B. durch Objekte) durchführen kann.

Der Spieler wird bei der nächsten Movement-Phase automatisch zurückgesetzt.

#### Parameter

Keine

#### Return

Keine

#### Verwendung

CHAR.delay();

### arrest

Verhaftet den Spieler. D.h. er verliert alle Hinweise, setzt aus, wird verzögert und wird auf das Feld Police Station gesetzt

#### Parameter

Keine

#### Return

Keine

#### Verwendung

CHAR.arrest();

## GameObject

Spielobjekte, die im Inventar der Spieler landen können.

### id

ID des Objekts. Bei Mehrfach-Objekten ist das eine Eindeutige ID, bei Single-Objekten (Ally, SpecialAbility) ist sie gleich der typeId.

### typeId

Typ-Id des Objekts. Das ist die Id mit der das Objekt registriert wurde, z.B. "CI\_Dynamite" bzw. identisch CommonItem.Dynamite.

### type

Art des Objekts. Siehe ObjectType (4.2)

### price

Preis, den ein Spieler bezahlen muss, um das Objekt zu kaufen, in $.

### owner

Besitzer des Objekts. Das ist der Spieler, in dessen Inventar sich das Objekt befindet.

Wenn das Objekt keinen Besitzer hat, ist der owner null.

### exhaust

Erschöpft das Objekt, wenn es exhaustable ist. Es wird in der nächsten Upkeep Phase wieder bereit gemacht.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

// Z.B. Healing Stone für Stamina  
this.owner.addStamina(1);  
this.exhaust();

### returnToDeck

Legt das Objekt wieder in den Kartenstapel. Wenn das Objekt im Besitz eines Spielers war, wird es aus dessen Inventar entfernt.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

// Z.B. nach Verwendung von Objekten  
this.returnToDeck();

### discard

Synonym für returnToDeck.

### removeFromGame

Entfernt das Objekt aus dem Spiel. Wenn das Objekt im Besitz eines Spielers war, wird es aus dessen Inventar entfernt.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

// Z.B. Elder Sign werden nach Verwendung entfernt  
this.removeFromGame();

## Field

Spielfelder.

### type

Typ des Spielfelds. Enum FieldType, siehe 4.16.

#### Verwendung

If (CHAR.field.type == Constants.FieldType.Street) { }

### hasMonsters

Prüft, ob Monster auf dem Feld sind.

#### Parameter

Keine

#### Return

bool: true, wenn Monster auf dem Feld sind, false wenn nicht.

### lockUntilNextRound

Sperrt das Feld bis zur nächsten Runde.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

**TODO: Wie kann ein bestimmtes Feld gesperrt werden? Fehlt da nicht was in game?**

## Gate

Ein Tor zu einer anderen Welt

### close

Schließt das Tor

#### Parameter

Keine

#### Return

Kein

#### Verwendung

game.context.gate.close();

### seal

Versiegelt das Tor

#### Parameter

Keine

#### Return

Kein

#### Verwendung

game.context.gate.seal();

## MythosCard

Eine Mythos Karte. Im Spiel kann nur mit game.context.activeRumor auf das aktive Gerücht zugegriffen werden.

### pass

Schließt das Gerücht erfolgreich ab.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

game.context.activeRumor.pass();

### fail

Schließt das Gerücht mit Misserfolg ab.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

game.context.activeRumor.fail();

## AncientOne

Der Große Alte im Spiel

### attackAdjustment

Die aktuelle Angriffs Erschwernis für Attacken des Großen Alten.

#### Verwendung

// In der Attack Funktion  
attack: {{{  
 CHAR.skillTest("..", Constants.Skill.Luck, this.attackAdjustment, 1);  
 // ...  
}}};

### doomTrack

Der aktuelle Wert auf dem Doom Track

### maxDoomTrack

Der maximale Wert des Doom Tracks

### awake

Erweckt den Großen Alten.

#### Parameter

Keine

#### Return

Kein

#### Verwendung

game.context.ancientOne.awake();

### increaseDoomTrack

Erhöht den Wert der Verderbnisleiste.

#### Parameter

* amount: Menge (Optional, default 1)

#### Return

Kein

#### Verwendung

// Erhöht Doom Track um 1  
game.context.ancientOne.increaseDoomTrack();

### decreaseDoomTrack

Verringert den Wert der Verderbnisleiste.

#### Parameter

* amount: Menge (Optional, default 1)

#### Return

Kein

#### Verwendung

// Verringert Doom Track um 2  
game.context.ancientOne.decreaseDoomTrack(2);

### setCombatAdjustment

Setzt die Erschwernis für Kampfproben.

#### Parameter

* value: Wert der Combat Adjustment

#### Return

Kein

#### Verwendung

// Hastur: Combat Adjustment ist TerrorLevel bei Aufwachen  
onAwake: {{{ this.setCombatAdjustment(game.context.terrorLevel); }}};

## Utility

### Allemeines

Utility Funktionen sind Hilfsfunktionen, die nicht im Arkham Horror Core implementiert sind, sondern als JS-Funktionen. Sie können somit beliebig erweitert werden und Bezug auf gescriptete Objekte nehmen.

Die Utility Funktionen werden über das game.utility Objekt verwendet.

### Definition neuer Funktionen

Neue Funktionen können in beliebigen JS-Dateien zum game.utility Objekt hinzugefügt werden.

Die Syntax ist wie folgt:

game.utility.myFunction = function(/\* params \*/) {   
 /\* Implementierung \*/  
};

### Standardfunktionen

Diese Funktionen werden (normalerweise) standardmäßig angeboten:

#### blessCharacter

Segnet einen bestimmten Charakter

##### Parameter

* chr: Character der gesegnet werden soll
* force: bool. Wenn true, wird der Character gesegnet, auch wenn er verflucht war (Optional, default false)

##### Return

Kein

##### Verwendung

// Segne aktuellen Charakter  
game.utility.blessCharacter(CHAR);  
// Segne aktuellen Charakter, auch wenn er verflucht war  
game.utility.blessCharacter(CHAR, true);

#### curseCharacter

Verflucht einen bestimmten Charakter

##### Parameter

* chr: Character der verflucht werden soll
* force: bool. Wenn true, wird der Character verflucht, auch wenn er gesegnet war (Optional, default false)

##### Return

Kein

##### Verwendung

// Verfluche alle Charaktere  
for (var c in game.context.allCharacters) {  
 game.utility.curseCharacter(c);  
}

#### isCharacterBlessed

Prüft, ob ein Charakter gesegnet ist.

##### Parameter

* chr: Character

##### Return

bool: Ob der Charakter gesegnet ist

##### Verwendung

if (game.utility.isCharacterBlessed(CHAR)) { /\* … \*/ }

#### isCharacterCursed

Prüft, ob ein Charakter verfluchtist.

##### Parameter

* chr: Character

##### Return

bool: Ob der Charakter verflucht ist

##### Verwendung

Beispiel Yig awake: Alle verfluchten Charaktere werden verschlungen, die anderen verflucht

for (var c in game.context.allCharacters) {  
 if (game.utility.isCharacterCursed(c)) {  
 c.devour();  
 } else {  
 game.utility.curseCharacter(c, true);  
 }  
}

# Fortgeschrittene Verwendung

## Dynamic Properties

### Allgemeines

Dynamic Properties sind zusätzliche Objekt-Eigenschaften, für die Benutzung des Objekts relevant sind, und beim Benutzer angezeigt werden sollen.

Ein Beispiel hierfür ist Book of Dzyan: Es kann 2 mal erfolgreich verwendet werden, und die Verwendung soll markiert werden durch einen Token.

Dynamic Properties können in AHS nur über ein Literal definiert werden. Dort werden dann per JS die Properties definiert. Die Properties müssen als ein Objekt angelegt werden, d.h. der Inhalt des properties-Literals muss in geschweifelte Klammern ({ und }) stehen.

Die Properties können dann in den Callback-Funktionen des Objekts verwendet werden. Änderungen durch Scripts werden automatisch an die Client weitergeleitet, der die Anzeige anpasst.

### Definition

#### Einfache Properties

Einfache Properties haben einen Namen und einen Start-Wert:

properties: <<<  
{  
 // Erzeugt eine Property „prop“ mit Wert „test“  
 prop: "test"  
}  
>>>;

#### Komplexe Properties

Komplexe Properties können die Anzeige beim Client beeinflussen. Sie haben einen (internen) Namen für das Scripting, optional einen abweichenden Anzeigenamen für die Oberfläche, optional eine Formatierungs-Funktion für den Wert, und einen Startwert. Diese Daten werden in ein JS-Objekt mit den Keys display, onDisplay und value gesetzt.

Hier als Beispiel Book of Dzyan:

properties: <<< {  
 // Erzeugt eine Property mit Namen „tokens“. In Scripts kann  
 // darauf mit this.tokens zugegriffen werden  
 tokens: {  
 // In der GUI steht „Tokens on Book“  
 display: "Tokens on Book",  
 // Zur Anzeige der aktuelle Wert + „/2“. Z.B. „1 / 2“  
 onDisplay: function() { return this.tokens + " / 2"; },  
 // Start-Wert ist 0  
 value: 0  
 }  
} >>>;

### Verwendung

Die dynamic Properties von Objekten können über deren Namen in Scripts verwendet werden.

Bsp. activate von Book Of Dzyan:

activate: {{{  
 // [... Skilltest + Objekte ziehen]  
 this.tokens++;  
 if (this.tokens >= 2) {  
 this.discard();  
 }  
}}};

### Initialisierung

Dynamic Properties sollten immer initialisiert werden, wenn das Objekt ins Spiel kommt. Der Startwert wird zwar zu Beginn des Spieles gesetzt, das Objekt kann aber mehrmals ins Spiel kommen.

Z.B. kann Book of Dzyan von einem Spieler fertig verwendet werden, und später von einem anderen (oder demselben) nochmal gezogen werden. Dann sollten die Anzahl der Tokens zurückgesetzt werden.

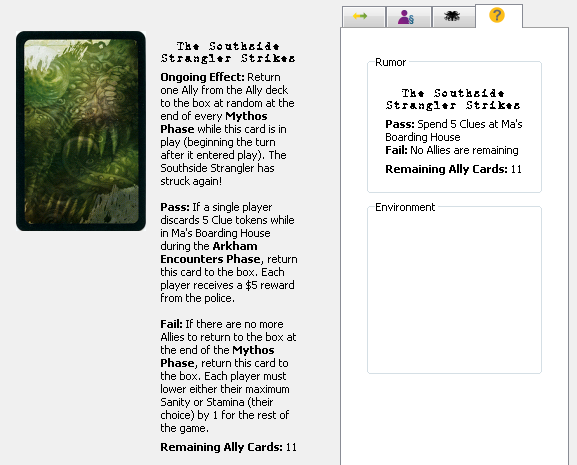
Für Objekte ist die onAddToInventory Funktion geeignet. Für Rumors die setupRumor Funktion.

### Beispiel

Beispiel South Side Strangler: Rumor ist aktiv, solange es Allies gibt. Das ist zwar keine eigene Property, aber sollte in der Oberfläche angezeigt werden.

Rumor SouthSideStriker {  
 // Standard Rumor Felder  
 properties: <<<{  
 allyCount: {  
 // Nur Anzeigename anpassen. Wert ist einfacher Wert  
 display: "Remaining Ally Cards"  
 }  
 }>>>;  
  
 setupRumor: {{{   
 // Initialer Wert = Anzahl Allies  
 this.allyCount = game.cardsOnDeck(Constants.ObjectType.Ally);  
 }}};  
  
 onMythos: {{{  
 var ally = game.drawSingleObject(Constants.ObjectType.Ally);  
 if (ally) {  
 // Entferne Ally aus Spiel  
 ally.removeFromGame();  
 } else {  
 // Kein Ally mehr da => Fail  
 this.fail();  
 }  
 // this.allyCount-- würde normal gezogene Allies  
 // nicht berücksichtigen. => Neu ermitteln  
 this.allyCount = game.cardsOnDeck(Constants.ObjectType.Ally);   
 }}};  
}

Das erzeugt eine weitere Anzeige „Remaining Ally Cards“ auf der Spiel-Oberfläche für das Rumor South Side Striker:



## Cast Objekte