

# **ENTWURFSDOKUMENT**

**(V. 1.1)**

**KNOT<sup>3</sup>**  
**PSE WS 2013/14**

Auftraggeber:  
Karlsruher Institut für Technologie  
Institut für Betriebs- und Dialogsysteme  
Prof. Dr.-Ing. C. Dachsbacher

Betreuer:  
Dipl.-Inf. Thorsten Schmidt  
Dipl.-Inform. M. Retzlaff

Auftragnehmer:  
Tobias Schulz, Maximilian Reuter, Pascal Knodel,  
Gerd Augsburg, Christina Erler, Daniel Warzel

22. Dezember 2013

# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>4</b>
<b>2. Aufbau</b>	<b>5</b>
2.1. Architektur . . . . .	5
2.2. Verwendete Entwurfsmuster . . . . .	6
2.2.1. Model View Controller (MVC) . . . . .	6
2.2.2. Kompositum . . . . .	6
2.2.3. Iterator . . . . .	6
2.2.4. Dekorierer . . . . .	6
2.2.5. Zustand . . . . .	6
2.3. Klassendiagramm . . . . .	7
<b>3. Klassenübersicht</b>	<b>8</b>
3.1. Paket Core . . . . .	8
3.1.1. Klassen . . . . .	8
3.1.2. Schnittstellen . . . . .	21
3.1.3. Enumerationen . . . . .	24
3.2. Paket GameObjects . . . . .	26
3.2.1. Klassen . . . . .	26
3.2.2. Schnittstellen . . . . .	42
3.2.3. Enumerationen . . . . .	43
3.3. Paket Screens . . . . .	43
3.3.1. Klassen . . . . .	43
3.3.2. Schnittstellen . . . . .	56
3.3.3. Enumerationen . . . . .	56
3.4. Paket RenderEffects . . . . .	56
3.4.1. Klassen . . . . .	56
3.4.2. Schnittstellen . . . . .	61
3.4.3. Enumerationen . . . . .	62
3.5. Paket KnotData . . . . .	62
3.5.1. Klassen . . . . .	62
3.5.2. Schnittstellen . . . . .	74
3.5.3. Enumerationen . . . . .	75
3.6. Paket Widgets . . . . .	76
3.6.1. Klassen . . . . .	76
3.6.2. Schnittstellen . . . . .	92
3.6.3. Enumerationen . . . . .	92
<b>4. Abläufe</b>	<b>94</b>
4.1. Sequenzdiagramme . . . . .	94
4.1.1. Ende einer Challenge . . . . .	94
4.1.2. Screenwechsel von dem CreativeMainScreen in den CreativeLoadScreen . . . . .	94
4.1.3. Erstellen der 3D-Objekte nach durchgeführten Kantenverschiebungen . . . . .	95
4.1.4. Verschiebung einer Kante . . . . .	96
<b>Klassenindex</b>	<b>98</b>



# 1. Einleitung

Das Knobel- und Konstruktionsspiel Knot<sup>3</sup> wurde vom IBDS Dachsbacher in Auftrag gegeben. Die Idee und das Konzept entstanden am Institut für Computergrafik in Zusammenarbeit mit der Hochschule für Gestaltung (HfG) in Karlsruhe und werden im Rahmen des Softwareprojekts PSE von Studenten des Karlsruher Instituts für Technologie umgesetzt. Knot<sup>3</sup> wird wie im Pflichtenheft spezifiziert ausgearbeitet.

Als Entwurfsumgebung wurde Microsoft Visual Studio 2013 verwendet. Die Entwicklung soll in der Sprache C-Sharp, aufbauend auf dem Microsoft XNA-Framework, erfolgen.

Der Entwurf besteht aus insgesamt 86 Klassen, 8 Schnittstellen und 7 Enumerationen. Im UML-Diagramm befinden sich etwa 101 Elemente.

## 2. Aufbau

### 2.1. Architektur

Die grundlegende Architektur des Spiels basiert auf der Spielkomponenten-Infrastruktur des XNA-Frameworks, die mit Spielzuständen kombiniert wird. Die abstrakten Klassen `GameScreenComponent` und `DrawableGameScreenComponent` erben von den von XNA bereitgestellten Klassen `GameComponent` und `DrawableGameComponent` und implementieren die Schnittstelle `IGameScreenComponent`. Sie unterscheiden sich von den XNA-Basisklassen dadurch, dass sie immer eine Referenz auf einen bestimmten Spielzustand halten und nur in Kombination mit diesem zu verwenden sind.

Die Spielzustände erben von der abstrakten Basisklasse `GameScreen` und halten eine Liste von `IGameScreenComponent`-Objekten. Wird ein Spielzustand aktiviert, indem von einem anderen Spielzustand aus zu ihm gewechselt wird oder indem er der Startzustand ist, dann weist er seine Liste von `IGameScreenComponent`-Objekten dem `Components`-Attribut der `Knot3Game`-Klasse zu, die von der vom XNA-Framework bereitgestellten abstrakten Klasse `Game` erbt. So ist zu jedem Zeitpunkt während der Laufzeit des Spiels ein Spielzustand aktiv, der die aktuelle Liste von Spielkomponenten verwaltet.

Die Spielkomponenten, die nicht gezeichnet werden und nur auf Eingaben reagieren, haben nur eine `Update()`-Methode und erben von `GameScreenComponent`. Dies sind vor allem verschiedene Input-Handler, welche Tastatur- und Mauseingaben verarbeiten und beispielsweise die Kameraposition und das Kameratarget ändern oder Spielobjekte bewegen.

Spielkomponenten, die neben der `Update()`-Methode auch eine `Draw()`-Methode besitzen, erben von `DrawableGameScreenComponent`. Dies sind vor allem die Elemente, aus denen die grafische Benutzeroberfläche zusammengesetzt ist, deren abstrakte Basisklasse `Widget` darstellt. Dabei gibt es vor allem verschiedene von der abstrakten Klasse `MenuItem` ererbende Klassen, die Einträge in Menüs darstellen, sowie die Klassen `Menu` und `VerticalMenu`, die diese in Menüs anordnen. Dargestellt werden diese entweder direkt in den `GameScreens`, etwa in den verschiedenen `SettingsScreens`, oder sie werden vor einer 3D-Szene gerendert und befinden sich in einem `Dialog`.

Alle Spielobjekte implementieren die Schnittstelle `IGameObject`. Die abstrakte Klasse `GameModel` repräsentiert dabei ein Spielobjekt, das aus einem 3D-Modell besteht, und hält zu diesem Zweck eine Referenz auf ein Objekt der Klasse `Model` aus dem XNA-Framework sowie weitere Eigenschaften wie Position, Drehung und Skalierung.

Spielobjekte sind keine Komponenten, sondern werden in einer Spielwelt zusammengefasst, die durch die Klasse `World` repräsentiert wird. Die Spielwelt ist ein `DrawableGameScreenComponent` und ruft in ihren `Update()`- und `Draw()`-Methoden jeweils die dazugehörigen Methoden aller in ihr enthaltenen Spielobjekte auf.

Shadereffekte werden durch die abstrakte Klasse `RenderEffect` und die von ihr abgeleiteten Klassen gekapselt. Ein `RenderEffect` enthält ein Rendertarget vom Typ `RenderTarget2D` als Attribut und implementiert jeweils eine `Begin()`- und eine `End()`-Methode. In der Methode `Begin()` wird das aktuell von XNA genutzte Rendertarget auf einem Stack gesichert und das Rendertarget des Effekts wird als aktuelles Rendertarget gesetzt.

Nach dem Aufruf von `Begin()` werden alle `Draw()`-Aufrufe von XNA auf dem gesetzten Rendertarget ausgeführt. Es wird also in eine im `RenderTarget2D`-Objekt enthaltene Bitmap gezeichnet. Dabei wird von den `Draw()`-Methoden der `GameModels` die `DrawModel(GameModel)`-Methode des `RenderEffects` aufgerufen, der die Modelle mit bestimmten Shadereffekten in die Bitmap zeichnet.

In der `End()`-Methode wird schließlich das auf dem Stack gesicherte, vorher genutzte Rendertarget wiederhergestellt und das Rendertarget des `RenderEffects` wird, unter Umständen verändert durch Post-Processing-Effekte, auf dieses übergeordnete Rendertarget gezeichnet.

## 2.2. Verwendete Entwurfsmuster

### 2.2.1. Model View Controller (MVC)

Unser Entwurf basiert auf dem Entwurfsmuster Model-View-Controller, bei dem die Aufgaben bei einer Anwendung strikt in drei Einheiten aufgeteilt werden. Diese drei Einheiten sind Datenmodell (Model), Präsentation (View) und Programmsteuerung (Controller).

In das Datenmodell werden die Klassen `Knot`, `Edge` und `Circle` eingeteilt, da diese die darzustellenden Daten enthalten und den Controller über Änderungen informieren. Die Präsentation zeigt die Daten in grafischer Form an und informiert die Programmsteuerung über Benutzereingaben. Sie wird von den Klassen `PipeModel`, `NodeModel` und `ArrowModel` repräsentiert. Der Controller kümmert sich um die Interaktion mit dem Benutzer und entspricht der Klasse `KnotRenderer`.

### 2.2.2. Kompositum

Das Kompositum gehört zu der Kategorie der Varianten-Muster und wird in unserem Entwurf von den Klassen `KnotRenderer` und `EdgeMovement` in Bezug auf `IGameObject`-Objekte, `IGameScreenComponent` in Bezug auf `GameScreen`-Objekte, sowie `Menu` und `MenuItem` dargestellt.

Die Grundidee des Entwurfsmusters Kompositum ist es, in einer abstrakten Klasse sowohl primitive Objekte also auch ihre Behälter zu repräsentieren.

Dies bedeutet konkret, dass `KnotRenderer` und `EdgeMovement` sowohl Iteratoren über `IGameObject`-Objekte sind, als auch selbst die Schnittstelle `IGameObject` implementieren und als solche verwendet werden können, indem sie wie einzelne Spielobjekte in der `World`-Klasse registriert werden. Die Aufrufe von `IGameObject`-spezifische Methoden wie `Draw()`, `Update()` und `Intersects(Ray)` werden dann auf allen in den Containern enthaltenen `IGameObject`-Objekten ausgeführt.

Ein weiteres Beispiel in unserer API stellt die Klasse `Widget` dar. Die von `Widget` zur Verfügung gestellten Properties und Methoden werden von `Menu`, `MenuItem`, `ColorPicker` und `Dialog` implementiert und erweitert.

### 2.2.3. Iterator

Der Iterator gehört zu der Kategorie der Entkopplungsmuster und wird in unserem Entwurf des Öffern verwendet, um auf Elemente einer Menge zuzugreifen. Beispielsweise ist ein Objekt der Klasse `Knot`, das einen Knoten repräsentiert, ein Iterator über die Kanten (`Edge`-Objekte), aus denen der Knoten besteht.

### 2.2.4. Dekorierer

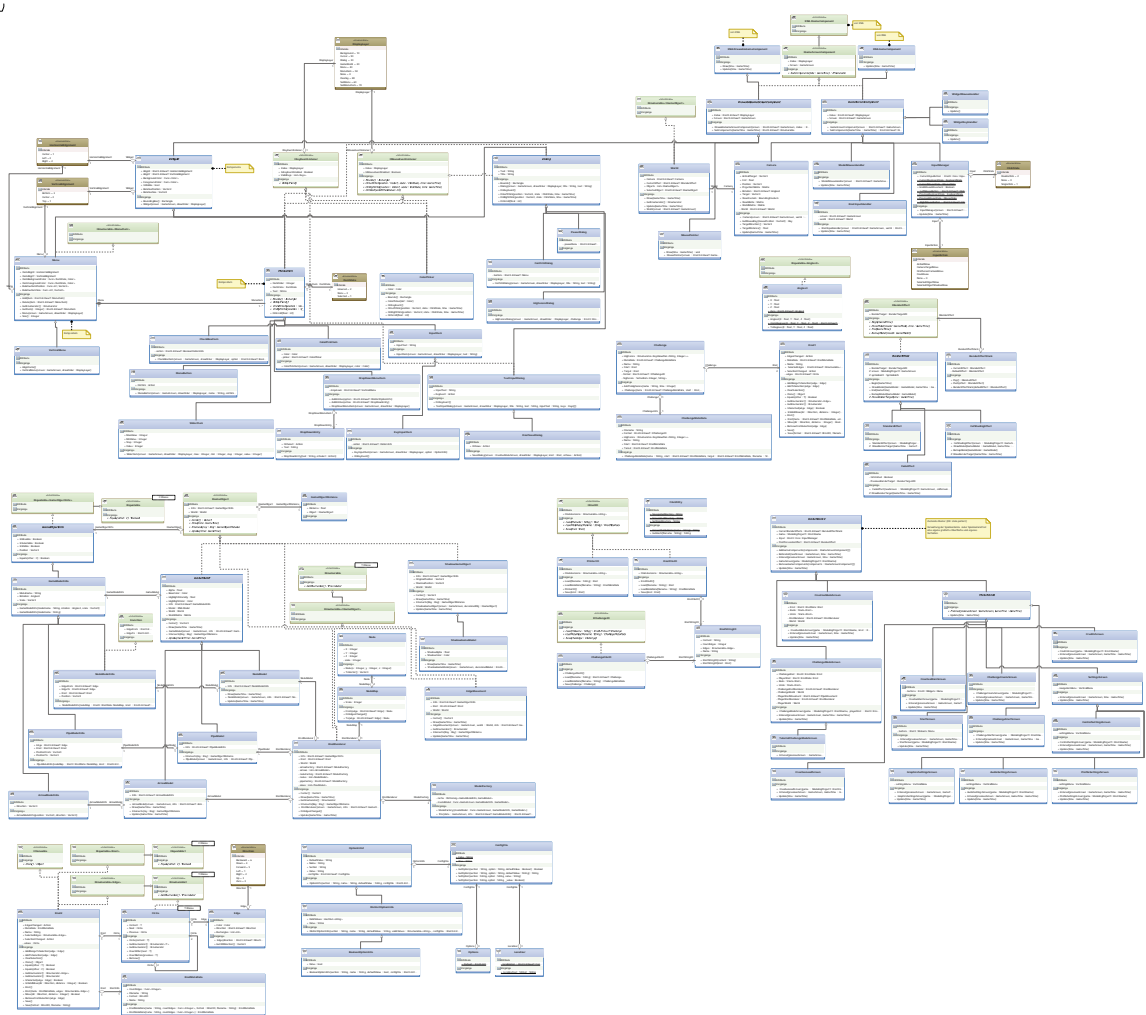
Der Dekorierer gehört zu der Kategorie der Varianten-Muster und wird von der Klasse `ShadowGameObject` dargestellt. `ShadowGameModel` stellt hierbei einen konkreten Dekorierer dar und erweitert die konkrete Komponente `GameModel`.

### 2.2.5. Zustand

Der Zustand gehört zu der Kategorie der Zustandshandhabungs-Muster und wird von der Klasse `GameScreen` repräsentiert. Jeder `GameScreen` hat eine eigene grafische Oberfläche und ein eigenes Verhalten. `Knot3Game` stellt den Kontext dar und verwaltet die verschiedenen `GameScreens`. Bei einem Wechsel zwischen verschiedenen `GameScreens` tauscht `Knot3Game` die betroffenen `GameScreens` aus.

## 2.3. Klassendiagramm

UML-Klassendiagramm

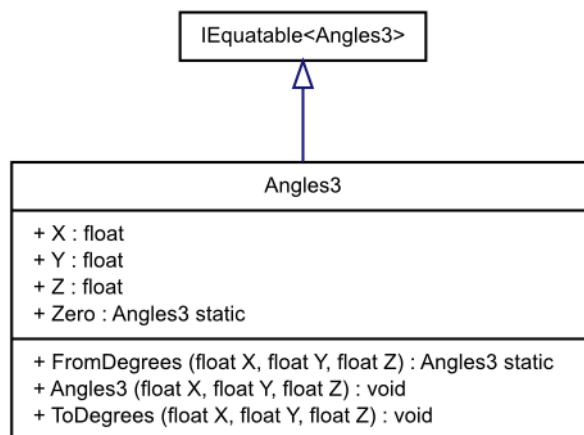


## 3. Klassenübersicht

### 3.1. Paket Core

#### 3.1.1. Klassen

##### 3.1.1.1. Klasse Angles3



#### Beschreibung:

Diese Klasse repräsentiert die Rollwinkel der drei Achsen X, Y und Z. Sie bietet Möglichkeit vordefinierte Winkelwerte zu verwenden, z.B. stellt Zero den Nullvektor dar. Die Umwandlung zwischen verschiedenen Winkelmaßen wie Grad- und Bogenmaß unterstützt sie durch entsprechende Methoden.

#### Eigenschaften:

##### **public float X**

Der Winkel im Bogenmaß für das Rollen um die X-Achse. Siehe statische Methode `Matrix.CreateRotationX(float)` des XNA-Frameworks.

##### **public float Y**

Der Winkel im Bogenmaß für das Rollen um die Y-Achse. Siehe statische Methode `Matrix.CreateRotationY(float)` des XNA-Frameworks.

##### **public float Z**

Der Winkel im Bogenmaß für das Rollen um die Z-Achse. Siehe statische Methode `Matrix.CreateRotationZ(float)` des XNA-Frameworks.

##### **public static Angles3 Zero**

Eine statische Eigenschaft mit dem Wert  $X = 0$ ,  $Y = 0$ ,  $Z = 0$ .



#### Konstruktoren:

**public** **Angles3** (**float** X, **float** Y, **float** Z)

Konstruiert ein neues **Angles3**-Objekt mit drei gegebenen Winkeln im Bogenmaß.

#### Methoden:

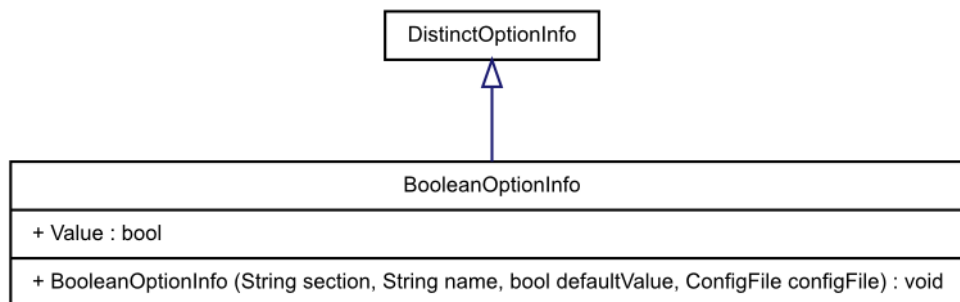
**public static** **Angles3** **FromDegrees** (**float** X, **float** Y, **float** Z)

Eine statische Methode, die Grad in Bogenmaß konvertiert.

**public void** **ToDegrees** (**float** X, **float** Y, **float** Z)

Konvertiert Bogenmaß in Grad.

#### 3.1.1.2. Klasse BooleanOptionInfo



#### Beschreibung:

Diese Klasse repräsentiert eine Option, welche die Werte „Wahr“ oder „Falsch“ annehmen kann.

#### Eigenschaften:

**public bool** Value

Eine Eigenschaft, die den aktuell abgespeicherten Wert zurückgibt.

#### Konstruktoren:

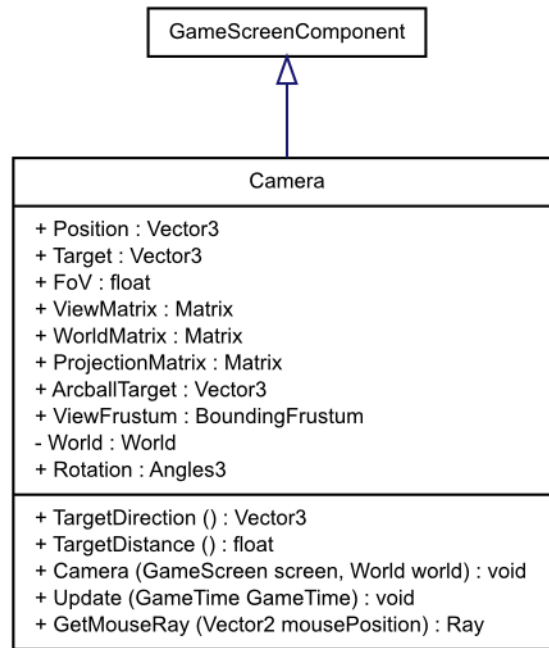
**public** **BooleanOptionInfo** (**String** section, **String** name, **bool** defaultValue, **ConfigFile** configFile)

Erstellt eine neue Option, welche die Werte „Wahr“ oder „Falsch“ annehmen kann. Mit dem angegebenen Namen, in dem angegebenen Abschnitt der angegebenen Einstellungsdatei.

#### 3.1.1.3. Klasse Camera

#### Beschreibung:

Jede Instanz der **World**-Klasse hält eine für diese Spielwelt verwendete Kamera als Attribut. Die Hauptfunktion der Kamera-Klasse ist das Berechnen der drei Matrizen, die für die Positionierung und Skalierung



von 3D-Objekten in einer bestimmten Spielwelt benötigt werden, der View-, [World](#)- und Projection-Matrix. Um diese Matrizen zu berechnen, benötigt die Kamera unter Anderem Informationen über die aktuelle Kamera-Position, das aktuelle Kamera-Ziel und das Field of View.

#### Eigenschaften:

**public Vector3 Position**

Die Position der Kamera.

**public Vector3 Target**

Das Ziel der Kamera.

**public float FoV**

Das Sichtfeld.

**public Matrix ViewMatrix**

Die View-Matrix wird über die statische Methode `CreateLookAt` der Klasse `Matrix` des XNA-Frameworks mit `Matrix.CreateLookAt (Position, Target, Vector3.Up)` berechnet.

**public Matrix WorldMatrix**

Die [World](#)-Matrix wird mit `Matrix.CreateFromYawPitchRoll` und den drei Rotationswinkeln berechnet.

**public Matrix ProjectionMatrix**

Die Projektionsmatrix wird über die statische XNA-Methode `Matrix.CreatePerspectiveFieldOfView` berechnet.

### **public Vector3 ArcballTarget**

Eine Position, um die rotiert werden soll, wenn der User die rechte Maustaste gedrückt hält und die Maus bewegt.

### **public BoundingFrustum ViewFrustum**

Berechnet ein Bounding-Frustum, das benötigt wird, um festzustellen, ob ein 3D-Objekt sich im Blickfeld des Spielers befindet.

### **private World World**

Eine Referenz auf die Spielwelt, für welche die Kamera zuständig ist.

### **public Angles3 Rotation**

Die Rotationswinkel.

#### **Konstruktoren:**

### **public Camera (GameScreen screen, World world)**

Erstellt eine neue Kamera in einem bestimmten GameScreen für eine bestimmte Spielwelt.

#### **Methoden:**

### **public Vector3 TargetDirection ()**

Die Blickrichtung.

### **public float TargetDistance ()**

Der Abstand zwischen der Kamera und dem Kamera-Ziel.

### **public void Update (GameTime gameTime)**

Wird für jeden Frame aufgerufen.

### **public Ray GetMouseRay (Vector2 mousePosition)**

Berechnet einen Strahl für die angegebene 2D-Mausposition.

#### **3.1.1.4. Klasse ConfigFile**

ConfigFile
+ True : String static + False : String static
+ SetOption (String section, String option, String value) : void + GetOption (String section, String option, Boolean defaultValue) : Boolean + GetOption (String section, String option, String defaultValue) : String + SetOption (String section, String option, Boolean _value) : void

### Beschreibung:

Repräsentiert eine Einstellungsdatei.

### Eigenschaften:

**public static String True**

Die Repräsentation des Wahrheitswerts "wahr" als String in einer Einstellungsdatei.

**public static String False**

Die Repräsentation des Wahrheitswerts "falsch" als String in einer Einstellungsdatei.

### Methoden:

**public void SetOption (String section, String option, String value)**

Setzt den Wert der Option mit dem angegebenen Namen in den angegebenen Abschnitt auf den angegebenen Wert.

**public Boolean GetOption (String section, String option, Boolean defaultValue)**

Gibt den aktuell in der Datei vorhandenen Wert für die angegebene Option in dem angegebenen Abschnitt zurück.

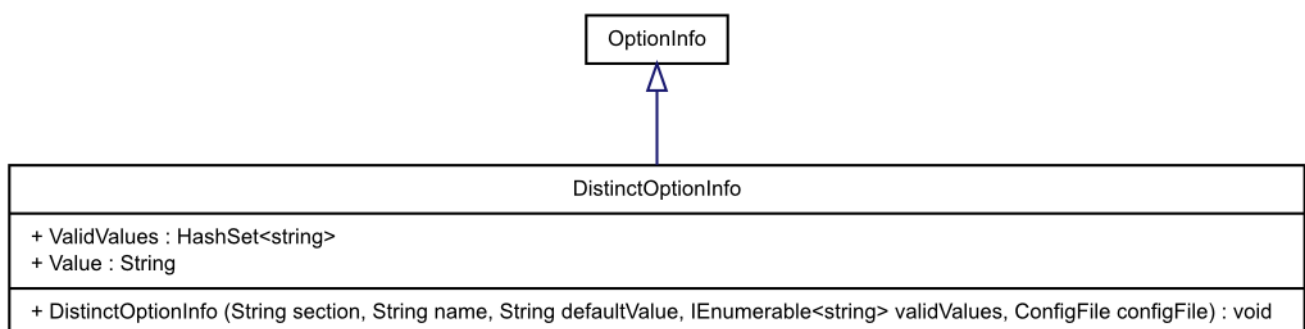
**public String GetOption (String section, String option, String defaultValue)**

Gibt den aktuell in der Datei vorhandenen Wert für die angegebene Option in dem angegebenen Abschnitt zurück.

**public void SetOption (String section, String option, Boolean value)**

Setzt den Wert der Option mit dem angegebenen Namen in den angegebenen Abschnitt auf den angegebenen Wert.

#### 3.1.1.5. Klasse DistinctOptionInfo



### Beschreibung:

Diese Klasse repräsentiert eine Option, die einen Wert aus einer distinkten Werteliste annehmen kann.

### Eigenschaften:

**public** **HashSet**<string> ValidValues

Eine Menge von Texten, welche die für die Option gültigen Werte beschreiben.

**public** **String** Value

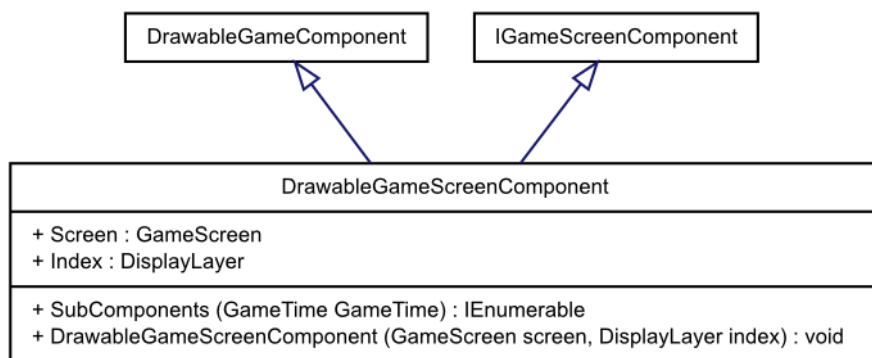
Eine Eigenschaft, die den aktuell abgespeicherten Wert zurück gibt.

### Konstruktoren:

**public** DistinctOptionInfo (**String** section, **String** name, **String** defaultValue, **IEnumerable**<string> validValues, **ConfigFile** configFile)

Erstellt eine neue Option, die einen der angegebenen Werte aus *validValues* annehmen kann, mit dem angegebenen Namen in dem angegebenen Abschnitt der angegebenen Einstellungsdatei.

#### 3.1.1.6. Klasse DrawableGameScreenComponent



### Beschreibung:

Eine zeichenbare Spielkomponente, die in einem angegebenen Spielzustand verwendet wird und eine bestimmte Priorität hat.

### Eigenschaften:

**public** **GameScreen** Screen

Der zugewiesene Spielzustand.

**public** **DisplayLayer** Index

Die Zeichen- und Eingabepriorität.

### Konstruktoren:

**public** DrawableGameScreenComponent (**GameScreen** screen, **DisplayLayer** index)

Erzeugt eine neue Instanz eines `DrawableGameScreenComponent`-Objekts und ordnet dieser ein `GameScreen`-Objekt zu. *index* bezeichnet die Zeichenebene, auf welche die Komponente zu zeichnen ist.

#### Methoden:

**public IEnumerable SubComponents (GameTime gameTime)**

Gibt Spielkomponenten zurück, die in dieser Spielkomponente enthalten sind.

#### 3.1.1.7. Klasse FileUtility

FileUtility
+ SettingsDirectory : String static + SavegameDirectory : String static + ScreenshotDirectory : String static
+ ConvertToFileName (String name) : String static + GetHash (String filename) : String

#### Beschreibung:

Eine Hilfsklasse für Dateioperationen.

#### Eigenschaften:

**public static String SettingsDirectory**

Das Einstellungsverzeichnis.

**public static String SavegameDirectory**

Das Spielstandverzeichnis.

**public static String ScreenshotDirectory**

Das Bildschirmfotoverzeichnis.

#### Methoden:

**public static String ConvertToFileName (String name)**

Konvertiert einen Namen eines Knotens oder einer `Challenge` in einen gültigen Dateinamen durch Weglassen ungültiger Zeichen.

**public String GetHash (String filename)**

Liefert einen Hash-Wert zu der durch *filename* spezifizierten Datei.

#### 3.1.1.8. Klasse GameScreen

#### Beschreibung:

GameScreen
+ Game : Knot3Game + Input : InputManager + PostProcessingEffect : RenderEffect + CurrentRenderEffects : RenderEffectStack
+ Entered (GameScreen previousScreen, GameTime time) : void + BeforeExit (GameScreen nextScreen, GameTime time) : void + Update (GameTime time) : void + GameScreen (Knot3Game game) : void + AddGameComponents (IGameScreenComponent[] components) : void + RemoveGameComponents (IGameScreenComponent[] components) : void

Ein Spielzustand, der zu einem angegebenen Spiel gehört und einen Inputhandler und Rendereffekte enthält.

### Eigenschaften:

**public Knot3Game Game**

Das Spiel, zu dem der Spielzustand gehört.

**public InputManager Input**

Der Inputhandler des Spielzustands.

**public RenderEffect PostProcessingEffect**

Der aktuelle Postprocessing-Effekt des Spielzustands

**public RenderEffectStack CurrentRenderEffects**

Ein Stack, der während dem Aufruf der Draw-Methoden der Spielkomponenten die jeweils aktuellen Rendereffekte enthält.

### Konstruktoren:

**public GameScreen (Knot3Game game)**

Erzeugt ein neues GameScreen-Objekt und initialisiert dieses mit einem Knot3Game-Objekt.

### Methoden:

**public void Entered (GameScreen previousScreen, GameTime time)**

Beginnt mit dem Füllen der Spielkomponentenliste des XNA-Frameworks und fügt sowohl für Tastatur- als auch für Mauseingaben einen Inputhandler für Widgets hinzu. Wird in Unterklassen von GameScreen reimplementiert und fügt zusätzlich weitere Spielkomponenten hinzu.

**public void BeforeExit (GameScreen nextScreen, GameTime time)**

Leert die Spielkomponentenliste des XNA-Frameworks.

**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

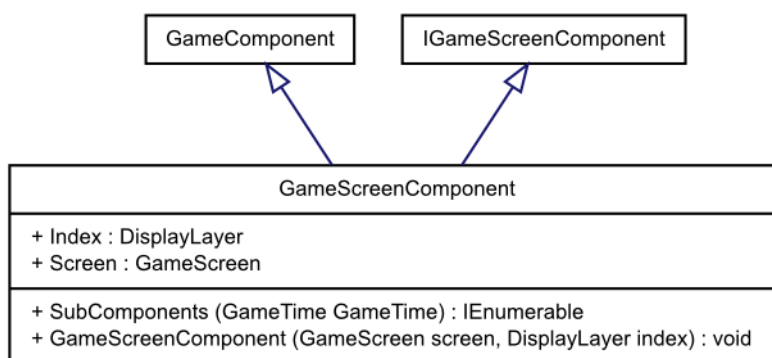
**public void AddGameComponents (IGameScreenComponent[] components)**

Fügt die angegebenen GameComponents in die Components-Liste des Games ein.

**public void RemoveGameComponents (IGameScreenComponent[] components)**

Entfernt die angegebenen GameComponents aus der Components-Liste des Games.

### 3.1.1.9. Klasse GameScreenComponent



#### Beschreibung:

Eine Spielkomponente, die in einem [GameScreen](#) verwendet wird und eine bestimmte Priorität hat.

#### Eigenschaften:

**public DisplayLayer Index**

Die Zeichen- und Eingabepriorität.

**public GameScreen Screen**

Der zugewiesene Spielzustand.

#### Konstruktoren:

**public GameScreenComponent (GameScreen screen, DisplayLayer index)**

Erzeugt eine neue Instanz eines [GameScreenComponent](#)-Objekts und initialisiert diese mit dem zugehörigen [GameScreen](#) und der zugehörigen Zeichenreihenfolge. Diese Spielkomponente kann nur in dem zugehörigen [GameScreen](#) verwendet werden.

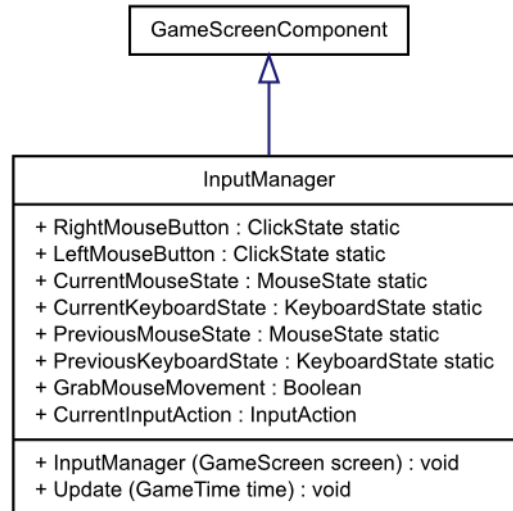
#### Methoden:

**public IEnumerable SubComponents (GameTime gameTime)**



Gibt Spielkomponenten zurück, die in dieser Spielkomponente enthalten sind.

### 3.1.1.10. Klasse InputManager



#### Beschreibung:

Stellt für jeden Frame die Maus- und Tastatureingaben bereit. Daraus werden die nicht von XNA bereitgestellten Mauseingaben berechnet. Zusätzlich wird die aktuelle Eingabeaktion berechnet.

#### Eigenschaften:

**public static ClickState RightMouseButton**

Enthält den Klickzustand der rechten Maustaste.

**public static ClickState LeftMouseButton**

Enthält den Klickzustand der linken Maustaste.

**public static MouseState CurrentMouseState**

Enthält den Mauszustand von XNA zum aktuellen Frames.

**public static KeyboardState CurrentKeyboardState**

Enthält den Tastaturzustand von XNA zum aktuellen Frames.

**public static MouseState PreviousMouseState**

Enthält den Mauszustand von XNA zum vorherigen Frames.

**public static KeyboardState PreviousKeyboardState**

Enthält den Tastaturzustand von XNA zum vorherigen Frames.

**public Boolean GrabMouseMovement**

Gibt an, ob die Mausbewegung für Kameradrehungen verwendet werden soll.

**public InputAction CurrentInputAction**

Gibt die aktuelle Eingabeaktion an, die von den verschiedenen Inputhandlern genutzt werden können.

**Konstruktoren:**

**public InputManager (GameScreen screen)**

Erstellt ein neues InputManager-Objekt, das an den übergebenen Spielzustand gebunden ist.

**Methoden:**

**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

#### 3.1.1.11. Klasse Localizer

Localizer
- localization : ConfigFile static
+ Localize (String text) : String static

**Beschreibung:**

Eine statische Klasse, die Bezeichner in lokalisierten Text umsetzen kann.

**Eigenschaften:**

**private static ConfigFile localization**

Die Datei, welche Informationen für die Lokalisierung enthält.

**Methoden:**

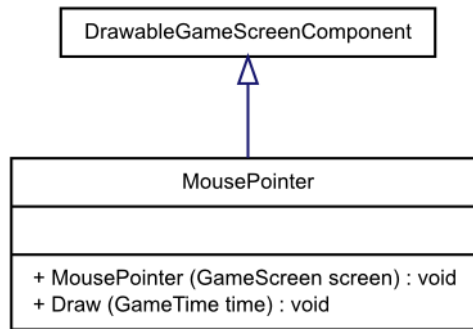
**public static String Localize (String text)**

Liefert zu dem übergebenen Bezeichner den zugehörigen Text aus der Lokalisierungsdatei der aktuellen Sprache zurück, die dabei aus der Einstellungsdatei des Spiels gelesen wird.

#### 3.1.1.12. Klasse MousePointer

**Beschreibung:**

Repräsentiert einen Mauszeiger.



#### Konstruktoren:

**public** MousePointer (**GameScreen** screen)

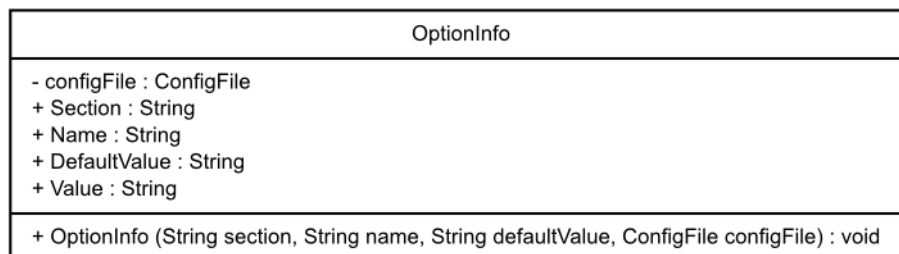
Erstellt einen neuen Mauszeiger für den angegebenen Spielzustand.

#### Methoden:

**public void** Draw (**GameTime** time)

Zeichnet den Mauszeiger.

#### 3.1.1.13. Klasse OptionInfo



#### Beschreibung:

Enthält Informationen über einen Eintrag in einer Einstellungsdatei.

#### Eigenschaften:

**private** **ConfigFile** configFile

Die Einstellungsdatei.

**public** **String** Section

Der Abschnitt der Einstellungsdatei.

**public** **String** Name

Der Name der Option.

**public String DefaultValue**

Der Standardwert der Option.

**public String Value**

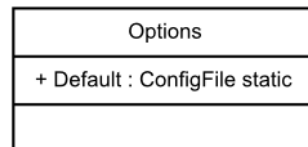
Der Wert der Option.

**Konstruktoeren:**

**public OptionInfo (String section, String name, String defaultValue, ConfigFile configFile)**

Erstellt ein neues OptionsInfo-Objekt aus den übergebenen Werten.

#### 3.1.1.14. Klasse Options



**Beschreibung:**

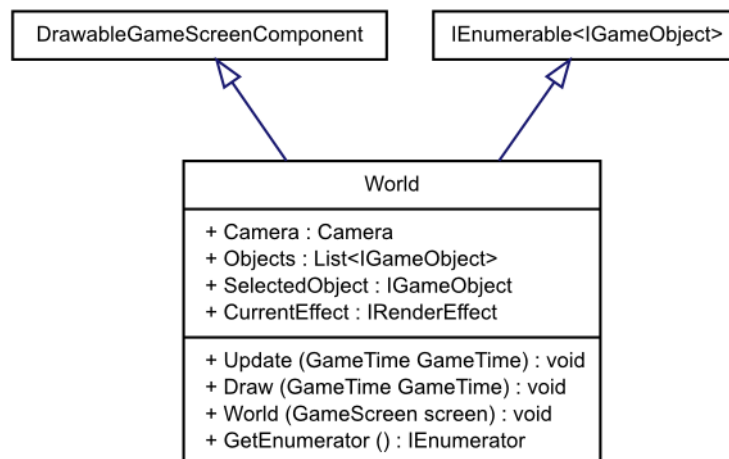
Eine statische Klasse, die eine Referenz auf die zentrale Einstellungsdatei des Spiels enthält.

**Eigenschaften:**

**public static ConfigFile Default**

Die zentrale Einstellungsdatei des Spiels.

#### 3.1.1.15. Klasse World



### **Beschreibung:**

Repräsentiert eine Spielwelt, in der sich 3D-Modelle befinden und gezeichnet werden können.

### **Eigenschaften:**

**public Camera Camera**

Die Kamera dieser Spielwelt.

**public List<IGameObject> Objects**

Die Liste von Spielobjekten.

**public IGameObject SelectedObject**

Das aktuell ausgewählte Spielobjekt.

**public IRenderEffect CurrentEffect**

Der aktuell angewendete Rendereffekt.

### **Konstruktoren:**

**public World (GameScreen screen)**

Erstellt eine neue Spielwelt im angegebenen Spielzustand.

### **Methoden:**

**public void Update (GameTime gameTime)**

Ruft auf allen Spielobjekten die Update()-Methode auf.

**public void Draw (GameTime gameTime)**

Ruft auf allen Spielobjekten die Draw()-Methode auf.

**public IEnumerator GetEnumerator ()**

Liefert einen Enumerator über die Spielobjekte dieser Spielwelt.

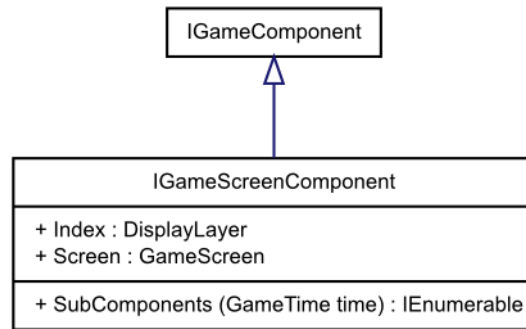
## **3.1.2. Schnittstellen**

### **3.1.2.1. Schnittstelle IGameScreenComponent**

#### **Beschreibung:**

Eine Schnittstelle für eine Spielkomponente, die in einem angegebenen Spielzustand verwendet wird und eine bestimmte Priorität hat.

#### **Eigenschaften:**



**public DisplayLayer Index**

Die Zeichen- und Eingabepriorität.

**public GameScreen Screen**

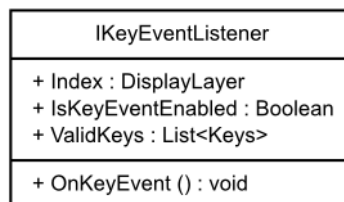
Der zugewiesene Spielzustand.

**Methoden:**

**public IEnumerable SubComponents (GameTime time)**

Gibt Spielkomponenten zurück, die in dieser Spielkomponente enthalten sind.

### 3.1.2.2. Schnittstelle IKeyEventListener



**Beschreibung:**

Eine Schnittstelle, die von Klassen implementiert wird, welche auf Tastatureingaben reagieren.

**Eigenschaften:**

**public DisplayLayer Index**

Die Eingabepriorität.

**public Boolean IsKeyEventEnabled**

Zeigt an, ob die Klasse zur Zeit auf Tastatureingaben reagiert.

**public List<Keys> ValidKeys**

Die Tasten, auf die die Klasse reagiert.

#### Methoden:

**public void OnKeyEvent ()**

Die Reaktion auf eine Tasteneingabe.

#### 3.1.2.3. Schnittstelle **IMouseEventListener**

IMouseEventListener
+ Index : DisplayLayer + IsMouseEventEnabled : Boolean
+ Bounds () : Rectangle + OnLeftClick (Vector2 position, ClickState state, GameTime time) : void + OnRightClick (Vector2 position, ClickState state, GameTime time) : void + OnScroll (int scrollWheelValue) : void

#### Beschreibung:

Eine Schnittstelle, die von Klassen implementiert wird, die auf Maus-Klicks reagieren.

#### Eigenschaften:

**public DisplayLayer Index**

Die Eingabepriorität.

**public Boolean IsMouseEventEnabled**

Ob die Klasse zur Zeit auf Mausklicks reagiert.

#### Methoden:

**public Rectangle Bounds ()**

Die Ausmaße des von der Klasse repräsentierten Objektes.

**public void OnLeftClick (Vector2 position, ClickState state, GameTime time)**

Die Reaktion auf einen Linksklick.

**public void OnRightClick (Vector2 position, ClickState state, GameTime time)**

Die Reaktion auf einen Rechtsklick.

**public void OnScroll (int scrollWheelValue)**

Die Reaktion auf ein Scrollen. Der Wert ist relativ zum letzten Frame.

### 3.1.3. Enumerations

#### 3.1.3.1. Enumeration ClickState

##### Beschreibung:

Eine Wertesammlung der möglichen Klickzustände einer Maustaste.

##### Eigenschaften:

**None = 0**

Wenn der Klickzustand nicht zugeordnet werden konnte. undefiniert.

**SingleClick = 1**

Ein Einzelklick.

**DoubleClick = 2**

Ein Doppelklick.

#### 3.1.3.2. Enumeration DisplayLayer

##### Beschreibung:

Die Zeichenreihenfolge der Elemente der grafischen Benutzeroberfläche.

##### Eigenschaften:

**None = 0**

Steht für die hinterste Ebene bei der Zeichenreihenfolge.

**Background = 10**

Steht für eine Ebene hinter der Spielwelt, z.B. um Hintergrundbilder darzustellen.

**GameWorld = 20**

Steht für die Ebene in der die Spielwelt dargestellt wird.

**Dialog = 30**

Steht für die Ebene in der die Dialoge dargestellt werden. Dialoge werden vor der Spielwelt gezeichnet, damit der Spieler damit interagieren kann.

**Menu = 40**

Steht für die Ebene in der Menüs gezeichnet werden. Menüs werden innerhalb von Dialogen angezeigt, müssen also davor gezeichnet werden, damit sie nicht vom Hintergrund des Dialogs verdeckt werden.



### **MenuItem = 50**

Steht für die Ebene in der Menüeinträge gezeichnet werden. Menüeinträge werden vor Menüs gezeichnet.

### **SubMenu = 60**

Steht für die Ebene in der Untermenüs gezeichnet werden. Untermenüs befinden sich in einer Ebene vor Menüeinträgen.

### **SubMenuItem = 70**

Steht für die Ebene in der Untermenüeinträge gezeichnet werden. Untermenüeinträge befinden sich in einer Ebene vor Untermenüs.

### **Overlay = 80**

Zum Anzeigen zusätzlicher Informationen bei der (Weiter-)Entwicklung oder beim Testen (z.B. ein FPS-Counter).

### **Cursor = 90**

Die Maus ist das Hauptinteraktionswerkzeug, welches der Spieler ständig verwendet. Daher muss die Maus bei der Interaktion immer im Vordergrund sein. Cursor steht für die vorderste Ebene.

## **3.1.3.3. Enumeration InputAction**

### **Beschreibung:**

Repräsentiert die möglichen Eingabeaktionen, wie sie von verschiedenen Inputhandlern berechnet und verwendet werden können. Die aktuelle Eingabeaktion wird zentral in der Klasse **InputManager** gehalten.

### **Eigenschaften:**

#### **None = 0**

Keine Eingabe bzw. undefiniert.

#### **CameraTargetMove**

Der Spieler bewegt mit der Maus oder der Tastatur das Kamera-Ziel.

#### **ArcballMove**

Der Spieler bewegt die Kamera mit der Maus oder der Tastatur wie auf einer Kugel um ein Objekt herum, wobei die Distanz zum Zielobjekt gleich bleibt.

#### **FreeMouse**

Der Spieler bewegt die Maus frei und keine Maustasten sind gedrückt; es sollen keine expliziten Berechnungen auf Basis der absoluten oder relativen Mausposition stattfinden. Über Aktionen, die die Tastatur

betreffen, wird keine Aussage gemacht.

### FirstPersonCameraMove

Der Spieler bewegt die Kamera wie in einem First-Person-Shooter (im Gegensatz zu z.B. einer Arcball-Bewegung).

### SelectedObjectMove

Das in der World-Klasse als selektiert markierte Spielobjekt wird endgültig verschoben, nachdem die Maus losgelassen wurde. Wird immer nach SelectedObjectShadowMove ausgeführt.

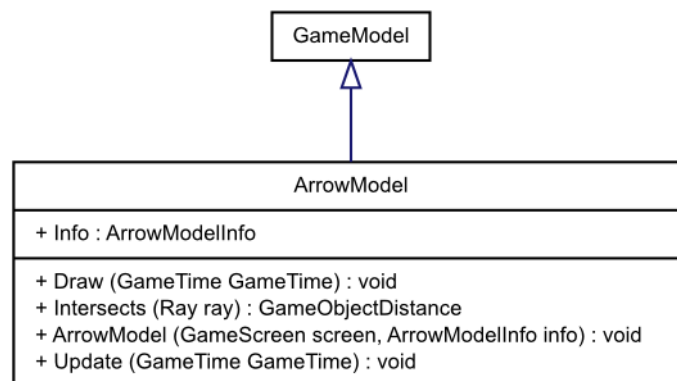
### SelectedObjectShadowMove

Das in der World-Klasse als selektiert markierte Spielobjekt wird bei gedrückt gehaltener Maustaste verschoben. Dabei können die Dekorierer ShadowObject und ShadowModel zum Einsatz kommen. Die Aktion wird vorerst nur visuell ausgeführt und nicht in Datenstrukturen übernommen. Dies kann nach einer eventuell auszuführenden Gültigkeitsprüfung in einem nachfolgenden SelectedObjectMove geschehen.

## 3.2. Paket GameObjects

### 3.2.1. Klassen

#### 3.2.1.1. Klasse ArrowModel



#### Beschreibung:

Diese Klasse [ArrowModel](#) repräsentiert ein 3D-Modell für einen Pfeil, zum Einblenden an selektierten Kanten (s. [Edge](#)).

#### Eigenschaften:

**public ArrowModelInfo Info**

Das Info-Objekt, das die Position und Richtung des [ArrowModel](#)'s enthält.

### Konstruktoren:

**public ArrowModel (GameScreen screen, ArrowModelInfo info)**

Erstellt ein neues Pfeilmodell in dem angegebenen GameScreen mit einem bestimmten Info-Objekt, das Position und Richtung des Pfeils festlegt.

### Methoden:

**public void Draw (GameTime gameTime)**

Zeichnet den Pfeil.

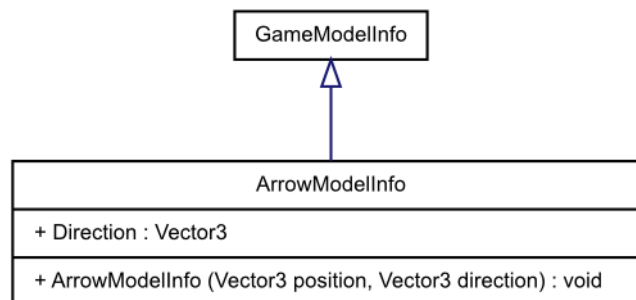
**public GameObjectDistance Intersects (Ray ray)**

Überprüft, ob der Mausstrahl den Pfeil schneidet.

**public void Update (GameTime gameTime)**

Wird für jeden Frame aufgerufen.

### 3.2.1.2. Klasse ArrowModelInfo



### Beschreibung:

Ein Objekt der Klasse ArrowModelInfo hält alle Informationen, die zur Erstellung eines Pfeil-3D-Modelles (s. ArrowModel) notwendig sind.

### Eigenschaften:

**public Vector3 Direction**

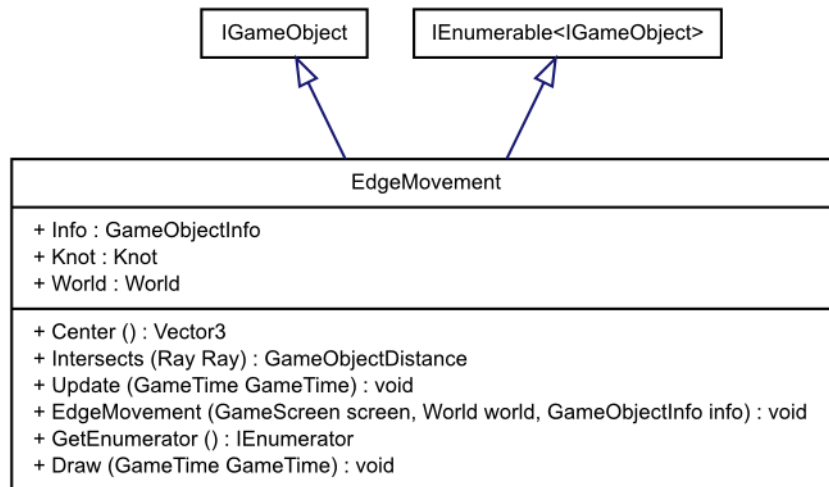
Gibt die Richtung, in die der Pfeil zeigen soll an.

### Konstruktoren:

**public ArrowModelInfo (Vector3 position, Vector3 direction)**

Erstellt ein neues ArrowModelInfo-Objekt an einer bestimmten Position *position* im 3D-Raum. Dieses zeigt in eine durch *direction* bestimmte Richtung.

### 3.2.1.3. Klasse EdgeMovement



#### Beschreibung:

Ein Inputhandler, der für das Verschieben der Kanten zuständig ist.

#### Eigenschaften:

**public GameObjectInfo Info**

Enthält Informationen über die Position des Knotens.

**public Knot Knot**

Der Knoten, dessen Kanten verschoben werden können.

**public World World**

Die Spielwelt, in der sich die 3D-Modelle der Kanten befinden.

#### Konstruktoren:

**public EdgeMovement (GameScreen screen, World world, GameObjectInfo info)**

Erzeugt eine neue Instanz eines **EdgeMovement**-Objekts und initialisiert diese mit ihrem zugehörigen **GameScreen**-Objekt *screen*, der Spielwelt *world* und Objektinformationen *info*.

#### Methoden:

**public Vector3 Center ()**

Gibt den Ursprung des Knotens zurück.

**public GameObjectDistance Intersects (Ray Ray)**

Gibt immer „null“ zurück.

**public void Update (GameTime gameTime)**

Wird für jeden Frame aufgerufen.

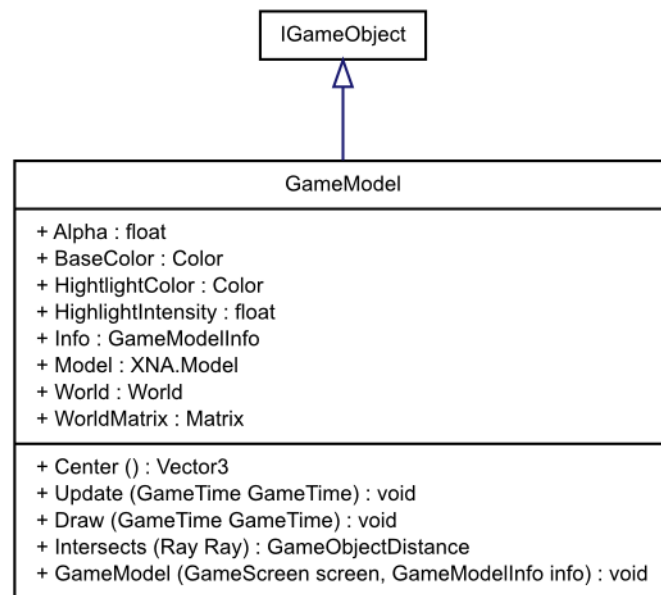
**public IEnumerable GetEnumrator ()**

Gibt einen Enumerator über die während einer Verschiebeaktion dynamisch erstellten 3D-Modelle zurück.

**public void Draw (GameTime gameTime)**

Zeichnet die während einer Verschiebeaktion dynamisch erstellten 3D-Modelle.

#### 3.2.1.4. Klasse GameModel



#### Beschreibung:

Repräsentiert ein 3D-Modell in einer Spielwelt.

#### Eigenschaften:

**public float Alpha**

Die Transparenz des Modells.

**public Color BaseColor**

Die Farbe des Modells.

**public Color HighlightColor**

Die Auswahlfarbe des Modells.

**public float HighlightIntensity**

Die Intensität der Auswahlfarbe.

**public GameModelInfo Info**

Die Modellinformationen wie Position, Skalierung und der Dateiname des 3D-Modells.

**public XNA.Model Model**

Die Klasse des XNA-Frameworks, die ein 3D-Modell repräsentiert.

**public World World**

Die Spielwelt, in der sich das 3D-Modell befindet.

**public Matrix WorldMatrix**

Die Weltmatrix des 3D-Modells in der angegebenen Spielwelt.

**Konstruktoren:**

**public GameModel (GameScreen screen, GameModelInfo info)**

Erstellt ein neues 3D-Modell in dem angegebenen Spielzustand mit den angegebenen Modellinformationen.

**Methoden:**

**public Vector3 Center ()**

Gibt die Mitte des 3D-Modells zurück.

**public void Update (GameTime gameTime)**

Wird für jeden Frame aufgerufen.

**public void Draw (GameTime gameTime)**

Zeichnet das 3D-Modell in der angegebenen Spielwelt mit dem aktuellen Rendereffekt der Spielwelt.

**public GameObjectDistance Intersects (Ray ray)**

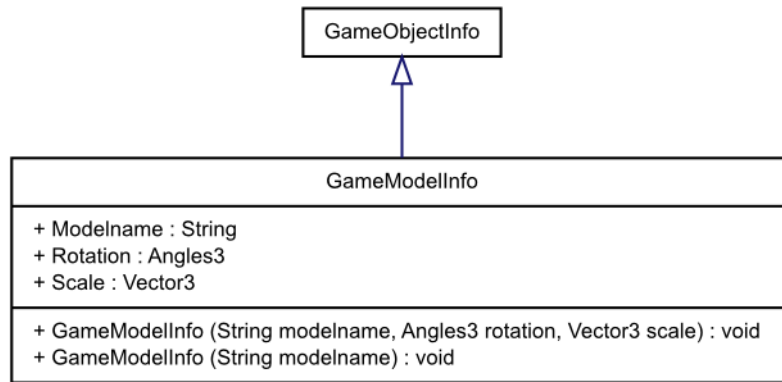
Überprüft, ob der Mausstrahl das 3D-Modell schneidet.

### 3.2.1.5. Klasse GameModelInfo

**Beschreibung:**

Enthält Informationen über ein 3D-Modell wie den Dateinamen, die Rotation und die Skalierung.

**Eigenschaften:**



**public String Modelname**

Der Dateiname des Modells.

**public Angles3 Rotation**

Die Rotation des Modells.

**public Vector3 Scale**

Die Skalierung des Modells.

**Konstruktoren:**

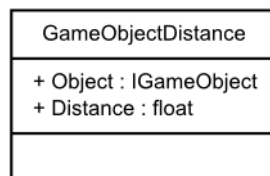
**public GameModelInfo (String modelname, Angles3 rotation, Vector3 scale)**

Erstellt ein neues Informations-Objekt eines 3D-Modells mit den angegebenen Informationen zu Dateiname, Rotation und Skalierung.

**public GameModelInfo (String modelname)**

Erzeugt eine neue Instanz eines `GameModelInfo`-Objekts. In `modelname` wird der Name der Datei angegeben, welche das Model repräsentiert.

### 3.2.1.6. Klasse GameObjectDistance



**Beschreibung:**

Die Klasse `GameObjectDistance` beschreibt den Abstand zu einem Spielobjekt. Dies ist z.B. hilfreich, um Schnittstellen, Verdeckungen, ... zu berechnen.

### Eigenschaften:

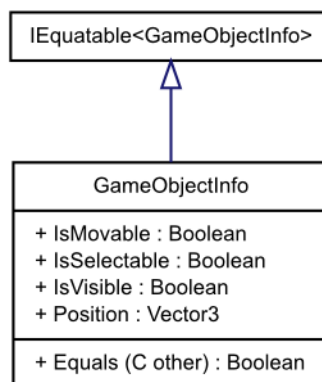
**public IGameObject Object**

Ein Spielobjekt.

**public float Distance**

Distance hält den Abstand als Gleitkommawert.

### 3.2.1.7. Klasse GameObjectInfo



### Beschreibung:

Enthält Informationen über ein 3D-Objekt wie die Position, Sichtbarkeit, Verschiebbarkeit und Auswählbarkeit.

### Eigenschaften:

**public Boolean IsMovable**

Die Verschiebbarkeit des Spielobjektes.

**public Boolean IsSelectable**

Die Auswählbarkeit des Spielobjektes.

**public Boolean IsVisible**

Die Sichtbarkeit des Spielobjektes.

**public Vector3 Position**

Die Position des Spielobjektes.

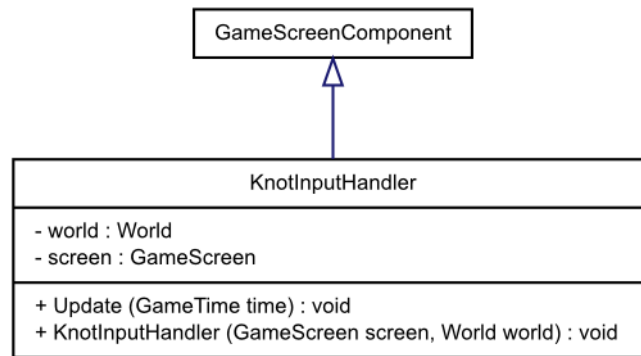
### Methoden:

**public Boolean Equals (C other)**

Vergleicht zwei Informationsobjekte für Spielobjekte.



### 3.2.1.8. Klasse KnotInputHandler



#### Beschreibung:

Verarbeitet die Maus- und Tastatureingaben des Spielers und modifiziert die Kamera-Position und das Kamera-Ziel.

#### Eigenschaften:

**private World world**

Die Spielwelt.

**private GameScreen screen**

Der Spielzustand.

#### Konstruktoren:

**public KnotInputHandler (GameScreen screen, World world)**

Erstellt einen neuen KnotInputHandler für den angegebenen Spielzustand und die angegebene Spielwelt.

#### Methoden:

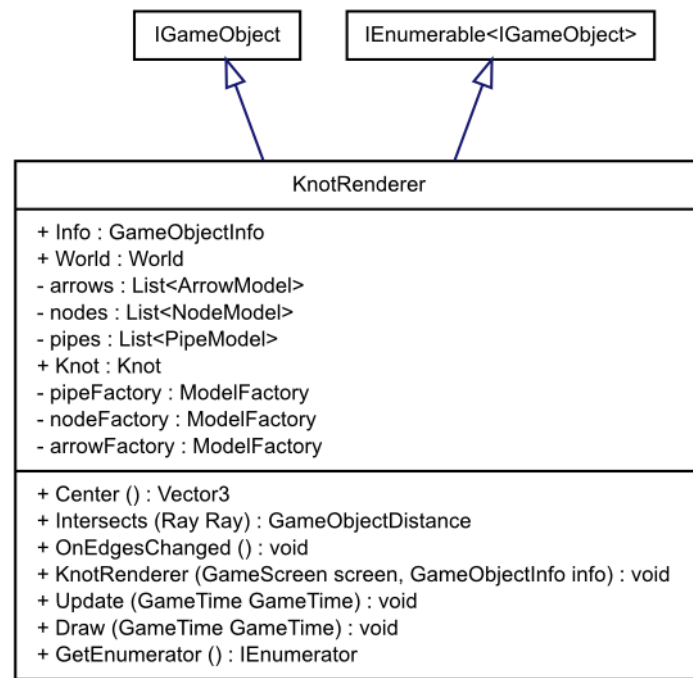
**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

### 3.2.1.9. Klasse KnotRenderer

#### Beschreibung:

Erstellt aus einem Knoten-Objekt die zu dem Knoten gehörenden 3D-Modelle sowie die 3D-Modelle der Pfeile, die nach einer Auswahl von Kanten durch den Spieler angezeigt werden. Ist außerdem ein IGameObject und ein Container für die erstellten Spielobjekte.



### Eigenschaften:

**public GameObjectInfo Info**

Enthält Informationen über die Position des Knotens.

**public World World**

Die Spielwelt, in der die 3D-Modelle erstellt werden sollen.

**private List<ArrowModel> arrows**

Die Liste der 3D-Modelle der Pfeile, die nach einer Auswahl von Kanten durch den Spieler angezeigt werden.

**private List<NodeModel> nodes**

Die Liste der 3D-Modelle der Kantenübergänge.

**private List<PipeModel> pipes**

Die Liste der 3D-Modelle der Kanten.

**public Knot Knot**

Der Knoten, für den 3D-Modelle erstellt werden sollen.

**private ModelFactory pipeFactory**

Der Zwischenspeicher für die 3D-Modelle der Kanten. Hier wird das Fabrik-Entwurfsmuster verwendet.

**private ModelFactory nodeFactory**

Der Zwischenspeicher für die 3D-Modelle der Kantenübergänge. Hier wird das Fabrik-Entwurfsmuster verwendet.

**private ModelFactory arrowFactory**

Der Zwischenspeicher für die 3D-Modelle der Pfeile. Hier wird das Fabrik-Entwurfsmuster verwendet.

**Konstruktoren:**

**public KnotRenderer (GameScreen screen, GameObjectInfo info)**

Erstellt ein neues KnotRenderer-Objekt für den angegebenen Spielzustand mit den angegebenen Spielobjekt-Informationen, die unter Anderem die Position des Knotenursprungs enthalten.

**Methoden:**

**public Vector3 Center ()**

Gibt den Ursprung des Knotens zurück.

**public GameObjectDistance Intersects (Ray Ray)**

Ruft die Intersects(Ray)-Methode der Kanten, Übergänge und Pfeile auf und liefert das beste Ergebnis zurück.

**public void OnEdgesChanged ()**

Wird mit dem EdgesChanged-Event des Knotens verknüpft.

**public void Update (GameTime gameTime)**

Ruft die Update()-Methoden der Kanten, Übergänge und Pfeile auf.

**public void Draw (GameTime gameTime)**

Ruft die Draw()-Methoden der Kanten, Übergänge und Pfeile auf.

**public IEnumerator GetEnumerator ()**

Gibt einen Enumerator der aktuell vorhandenen 3D-Modelle zurück.

### 3.2.1.10. Klasse ModelFactory

ModelFactory
- cache : Dictionary<GameModelInfo, GameModel> - createModel : Func<GameScreen, GameModelInfo, GameModel>
+ this (GameScreen state, GameModelInfo info) : GameModel + ModelFactory (GameModelInfo, GameModel>, Func<GameScreen createModel) : void

### Beschreibung:

Ein Zwischenspeicher für 3D-Modelle.

### Eigenschaften:

**private Dictionary<GameModelInfo, GameModel> cache**

Die Zuordnung zwischen den Modellinformationen zu den 3D-Modellen.

**private Func<GameScreen, GameModelInfo, GameModel> createModel**

Ein Delegate, das beim Erstellen eines Zwischenspeichers zugewiesen wird und aus den angegebenen Modellinformationen und dem angegebenen Spielzustand ein 3D-Modell erstellt.

### Konstruktoren:

**public ModelFactory (GameModelInfo, GameModel>, Func<GameScreen createModel)**

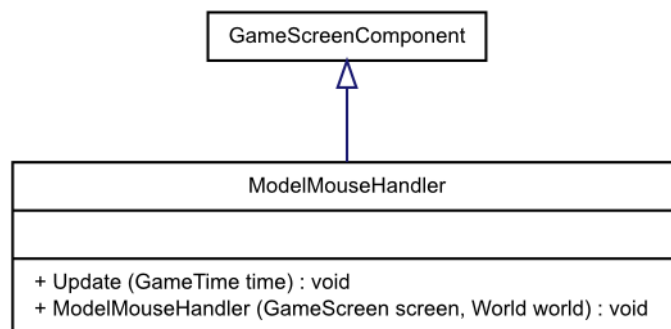
Erstellt einen neuen Zwischenspeicher.

### Methoden:

**public GameModel this (GameScreen state, GameModelInfo info)**

Falls das 3D-Modell zwischengespeichert ist, wird es zurückgegeben, sonst mit createModel() erstellt.

### 3.2.1.11. Klasse ModelMouseHandler



### Beschreibung:

Ein Inputhandler, der Mauseingaben auf 3D-Modellen verarbeitet.

### Konstruktoren:

**public ModelMouseHandler (GameScreen screen, World world)**

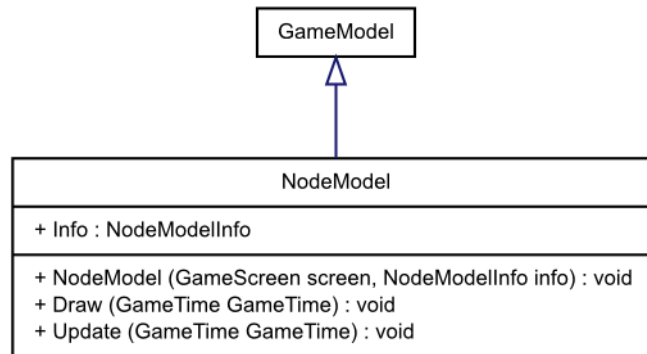
Erzeugt eine neue Instanz eines `ModelMouseHandler`-Objekts und ordnet dieser ein `GameScreen`-Objekt *screen* zu, sowie eine Spielwelt *world*.

#### Methoden:

**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

#### 3.2.1.12. Klasse NodeModel



#### Beschreibung:

Ein 3D-Modell, das einen Kantenübergang darstellt.

#### Eigenschaften:

**public NodeModelInfo Info**

Enthält Informationen über den darzustellende 3D-Modell des Kantenübergangs.

#### Konstruktoren:

**public NodeModel (GameScreen screen, NodeModelInfo info)**

Erstellt ein neues 3D-Modell mit dem angegebenen Spielzustand und dem angegebenen Informationsobjekt.

#### Methoden:

**public void Draw (GameTime gameTime)**

Zeichnet das 3D-Modell mit dem aktuellen Rendereffekt.

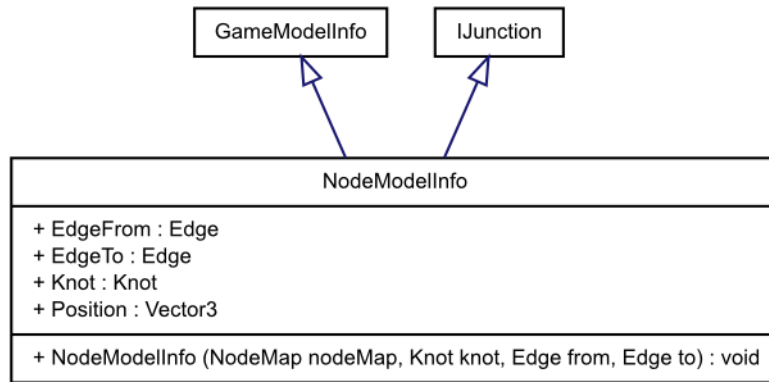
**public void Update (GameTime gameTime)**

Wird für jeden Frame aufgerufen.

#### 3.2.1.13. Klasse NodeModelInfo

#### Beschreibung:

Enthält Informationen über ein 3D-Modell, das einen Kantenübergang darstellt.



### Eigenschaften:

**public Edge EdgeFrom**

Die Kante vor dem Übergang.

**public Edge EdgeTo**

Die Kante nach dem Übergang.

**public Knot Knot**

Der Knoten, der die Kanten enthält.

**public Vector3 Position**

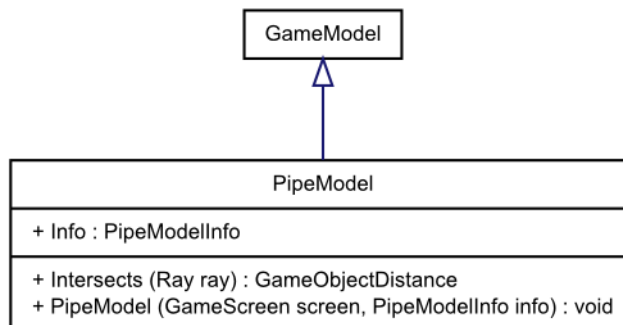
Die Position des Übergangs.

### Konstruktoren:

**public NodeModelInfo (NodeMap nodeMap, Knot knot, Edge from, Edge to)**

Erstellt ein neues Informationsobjekt für ein 3D-Modell, das einen Kantenübergang darstellt.

### 3.2.1.14. Klasse PipeModel



### Beschreibung:

Ein 3D-Modell, das eine Kante darstellt.

#### Eigenschaften:

**public PipeModelInfo Info**

Enthält Informationen über die darzustellende Kante.

#### Konstruktoren:

**public PipeModel (GameScreen screen, PipeModelInfo info)**

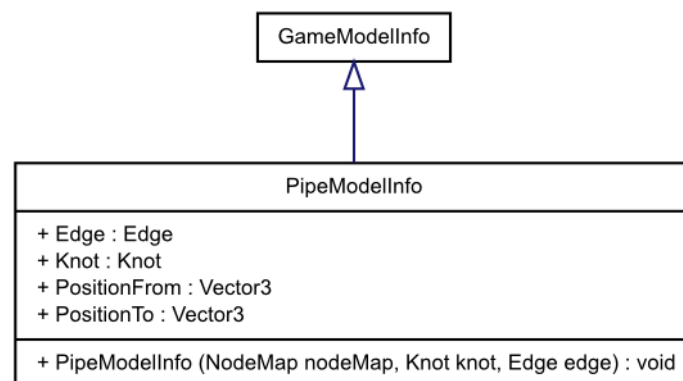
Erstellt ein neues 3D-Modell mit dem angegebenen Spielzustand und den angegebenen Spielinformationen.

#### Methoden:

**public GameObjectDistance Intersects (Ray ray)**

Prüft, ob der angegebene Mausstrahl das 3D-Modell schneidet.

#### 3.2.1.15. Klasse PipeModelInfo



#### Beschreibung:

Enthält Informationen über ein 3D-Modell, das eine Kante darstellt.

#### Eigenschaften:

**public Edge Edge**

Die Kante, die durch das 3D-Modell dargestellt wird.

**public Knot Knot**

Der Knoten, der die Kante enthält.

**public Vector3 PositionFrom**

Die Position, an der die Kante beginnt.

**public Vector3 PositionTo**

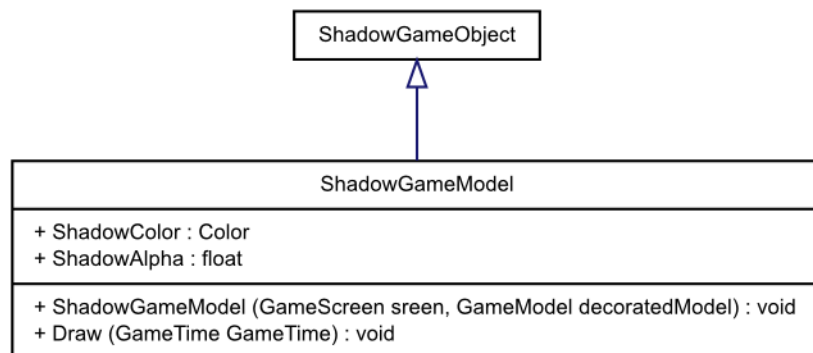
Die Position, an der die Kante endet.

#### Konstruktoren:

**public PipeModelInfo (NodeMap nodeMap, Knot knot, Edge edge)**

Erstellt ein neues Informationsobjekt für ein 3D-Modell, das eine Kante darstellt.

#### 3.2.1.16. Klasse ShadowGameModel



#### Beschreibung:

Die 3D-Modelle, die während einer Verschiebung von Kanten die Vorschau Modelle repräsentieren.

#### Eigenschaften:

**public Color ShadowColor**

Die Farbe der Vorschau Modelle.

**public float ShadowAlpha**

Die Transparenz der Vorschau Modelle.

#### Konstruktoren:

**public ShadowGameModel (GameScreen sreen, GameModel decoratedModel)**

Erstellt ein neues Vorschau Modell in dem angegebenen Spielzustand für das angegebene zu dekorierende Modell.

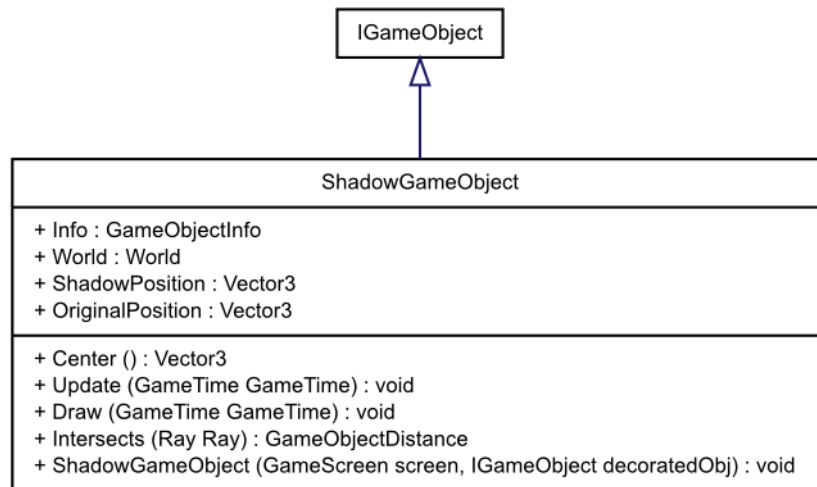
#### Methoden:

**public void Draw (GameTime gameTime)**

Zeichnet das Vorschau Modell.



### 3.2.1.17. Klasse ShadowGameObject



#### Beschreibung:

Eine abstrakte Klasse, die ein Vorschau-Spielobjekt darstellt.

#### Eigenschaften:

**public GameObjectInfo Info**

Enthält Informationen über das Vorschau-Spielobjekt.

**public World World**

Eine Referenz auf die Spielwelt, in der sich das Spielobjekt befindet.

**public Vector3 ShadowPosition**

Die Position, an der das Vorschau-Spielobjekt gezeichnet werden soll.

**public Vector3 OriginalPosition**

Die Position, an der sich das zu dekorierende Objekt befindet.

#### Konstruktoren:

**public ShadowGameObject (GameScreen screen, IGameObject decoratedObj)**

Erstellt ein neues Vorschauobjekt in dem angegebenen Spielzustand für das angegebene zu dekorierende Objekt.

#### Methoden:

**public Vector3 Center ()**

Die Position, an der das Vorschau-Spielobjekt gezeichnet werden soll.

**public void Update (GameTime gameTime)**

Wird für jeden Frame aufgerufen.

**public void Draw (GameTime gameTime)**

Zeichnet das Vorschau-Spielobjekt.

**public GameObjectDistance Intersects (Ray ray)**

Prüft, ob der angegebene Mausstrahl das Vorschau-Spielobjekt schneidet.

### 3.2.2. Schnittstellen

#### 3.2.2.1. Schnittstelle IGameObject

IGameObject
+ Info : GameObjectInfo + World : World
+ Center () : Vector3 + Update (GameTime time) : void + Draw (GameTime time) : void + Intersects (Ray ray) : GameObjectDistance

#### Beschreibung:

Diese Schnittstelle repräsentiert ein Spielobjekt und enthält eine Referenz auf die Spielwelt, in der sich dieses Game befindet, sowie Informationen zu dem Game.

#### Eigenschaften:

**public GameObjectInfo Info**

Informationen über das Spielobjekt, wie z.B. die Position.

**public World World**

Eine Referenz auf die Spielwelt, in der sich das Spielobjekt befindet.

#### Methoden:

**public Vector3 Center ()**

Die Mitte des Spielobjektes im 3D-Raum.

**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

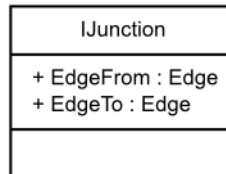
**public void Draw (GameTime time)**

Zeichnet das Spielobjekt.

**public GameObjectDistance Intersects (Ray ray)**

Überprüft, ob der Mausstrahl das Spielobjekt schneidet.

#### 3.2.2.2. Schnittstelle **IJunction**



#### **Beschreibung:**

Repräsentiert einen Übergang zwischen zwei Kanten.

#### **Eigenschaften:**

**public Edge EdgeFrom**

Die Kante vor dem Übergang.

**public Edge EdgeTo**

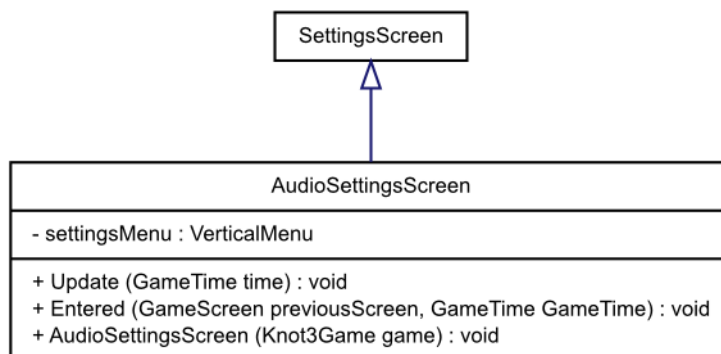
Die Kante nach dem Übergang.

#### 3.2.3. Enumerationen

### 3.3. Paket Screens

#### 3.3.1. Klassen

##### 3.3.1.1. Klasse **AudioSettingsScreen**



### Beschreibung:

Die Klasse `AudioSettingsScreen` steht für den Spielzustand, der die Audio-Einstellungen repräsentiert.

### Eigenschaften:

`private VerticalMenu settingsMenu`

Das Menü, das die Einstellungen enthält.

### Konstruktoren:

`public AudioSettingsScreen (Knot3Game game)`

Erzeugt ein neues `AudioSettingsScreen`-Objekt und initialisiert dieses mit einem `Knot3Game`-Objekt.

### Methoden:

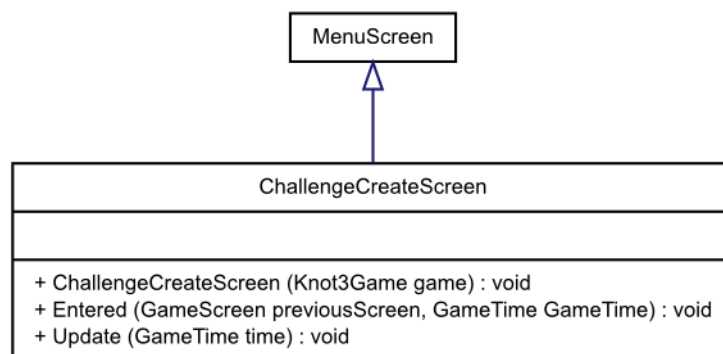
`public void Update (GameTime time)`

Wird für jeden Frame aufgerufen.

`public void Entered (GameScreen previousScreen, GameTime gameTime)`

Fügt das Menü mit den Einstellungen in die Spielkomponentenliste ein.

### 3.3.1.2. Klasse ChallengeCreateScreen



### Beschreibung:

### Konstruktoren:

`public ChallengeCreateScreen (Knot3Game game)`

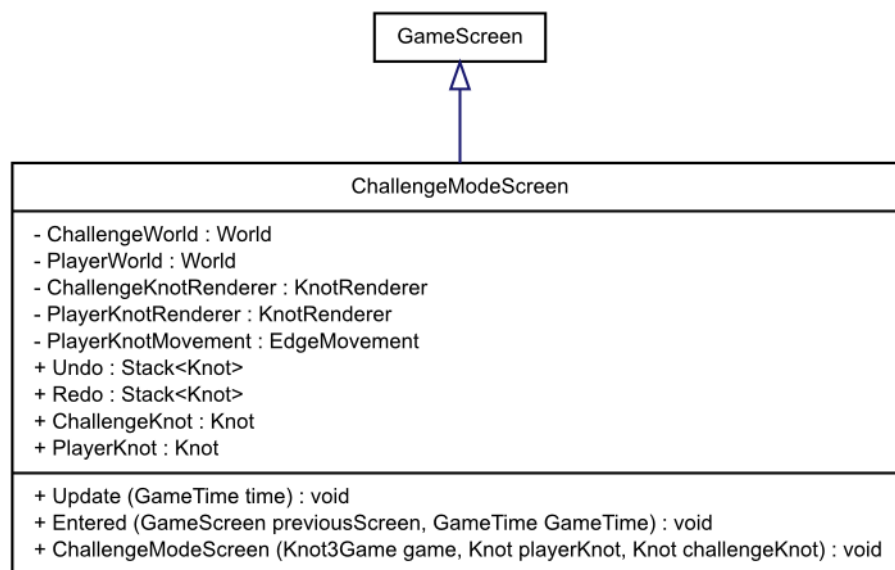
Erzeugt eine neue Instanz eines `ChallengeCreateScreen`-Objekts und initialisiert diese mit einem `Knot3Game`-Objekt *game*.

Methoden:

```
public void Entered (GameScreen previousScreen, GameTime gameTime)
```

```
public void Update (GameTime time)
```

### 3.3.1.3. Klasse ChallengeModeScreen



**Beschreibung:**

Der Spielzustand, der während dem Spielen einer Challenge aktiv ist und für den Ausgangs- und Referenzknoten je eine 3D-Welt zeichnet.

**Eigenschaften:**

```
private World ChallengeWorld
```

Die Spielwelt in der die 3D-Modelle des dargestellten Referenzknotens enthalten sind.

```
private World PlayerWorld
```

Die Spielwelt in der die 3D-Modelle des dargestellten Spielerknotens enthalten sind.

```
private KnotRenderer ChallengeKnotRenderer
```

Der Controller, der aus dem Referenzknoten die 3D-Modelle erstellt.

```
private KnotRenderer PlayerKnotRenderer
```

Der Controller, der aus dem Spielerknoten die 3D-Modelle erstellt.

**private EdgeMovement PlayerKnotMovement**

Der Inputhandler, der die Kantenverschiebungen des Spielerknotens durchführt.

**public Stack<Knot> Undo**

Der Undo-Stack.

**public Stack<Knot> Redo**

Der Redo-Stack.

**public Knot ChallengeKnot**

Der Referenzknoten.

**public Knot PlayerKnot**

Der Spielerknoten, der durch die Transformation des Spielers aus dem Ausgangsknoten entsteht.

**Konstruktoren:**

**public ChallengeModeScreen (Knot3Game game, Knot playerKnot, Knot challengeKnot)**

Erzeugt eine neue Instanz eines ChallengeModeScreen-Objekts und initialisiert diese mit einem Knot3Game-Objekt, einem Spielerknoten *playerKnot* und dem Knoten *challengeKnot*, den der Spieler nachbauen soll.

**Methoden:**

**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

**public void Entered (GameScreen previousScreen, GameTime gameTime)**

Fügt die 3D-Welten und den Inputhandler in die Spielkomponentenliste ein.

#### 3.3.1.4. Klasse ChallengeStartScreen

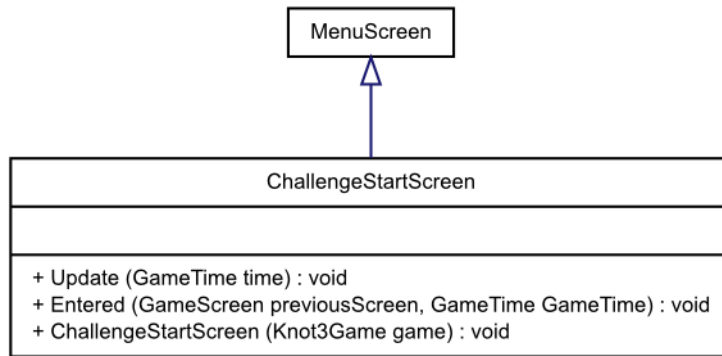
**Beschreibung:**

Der Spielzustand, der den Ladebildschirm für Challenges darstellt.

**Konstruktoren:**

**public ChallengeStartScreen (Knot3Game game)**

Erstellt eine neue Instanz eines ChallengeStartScreen-Objekts und initialisiert diese mit einem Knot3Game-Objekt.



#### Methoden:

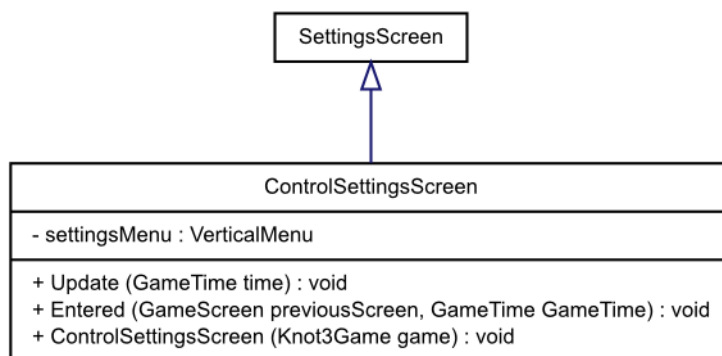
**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

**public void Entered (GameScreen previousScreen, GameTime GameTime)**

Fügt das Menü mit den Spielständen in die Spielkomponentenliste ein.

#### 3.3.1.5. Klasse ControlSettingsScreen



#### Beschreibung:

Der Spielzustand, der die Steuerungs-Einstellungen darstellt.

#### Eigenschaften:

**private VerticalMenu settingsMenu**

Das Menü, das die Einstellungen enthält.

#### Konstruktoren:

**public ControlSettingsScreen (Knot3Game game)**

Erzeugt ein neues ControlSettingsScreen-Objekt und initialisiert dieses mit einem Knot3Game-Objekt.

#### Methoden:

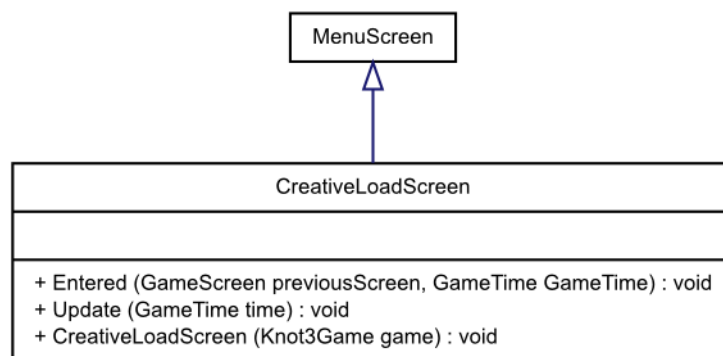
**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

**public void Entered (GameScreen previousScreen, GameTime gameTime)**

Fügt das Menü mit den Einstellungen in die Spielkomponentenliste ein.

#### 3.3.1.6. Klasse CreativeLoadScreen



#### Beschreibung:

Der Spielzustand, der den Ladebildschirm für Knoten darstellt.

#### Konstruktoren:

**public CreativeLoadScreen (Knot3Game game)**

Erzeugt ein neues CreativeLoadScreen-Objekt und initialisiert dieses mit einem Knot3Game-Objekt.

#### Methoden:

**public void Entered (GameScreen previousScreen, GameTime gameTime)**

Fügt das Menü mit den Spielständen in die Spielkomponentenliste ein.

**public void Update (GameTime time)**

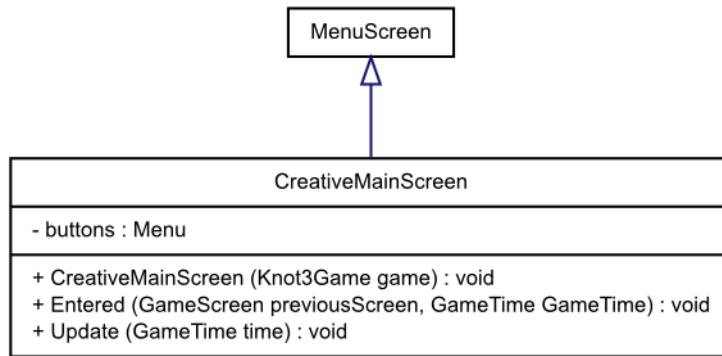
Wird für jeden Frame aufgerufen.

#### 3.3.1.7. Klasse CreativeMainScreen

#### Beschreibung:

In diesem Menü trifft der Spieler die Wahl, ob er im Creative-Modus einen neuen Knoten erstellen, einen Knoten laden oder eine neue Challenge erstellen möchte.





#### Eigenschaften:

**private Menu buttons**

Ein Menü aus Schaltflächen, welche den Spielwunsch des Spielers weiterleiten.

#### Konstruktoren:

**public CreativeMainScreen (Knot3Game game)**

Erzeugt ein neues **CreativeMainScreen**-Objekt und initialisiert dieses mit einem **Knot3Game**-Objekt.

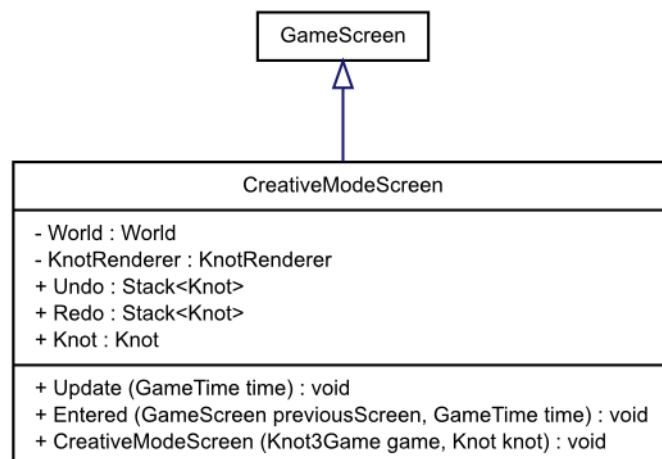
#### Methoden:

**public void Entered (GameScreen previousScreen, GameTime gameTime)**

**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

#### 3.3.1.8. Klasse CreativeModeScreen



### Beschreibung:

Der Spielzustand, der während dem Erstellen und Bearbeiten eines Knotens aktiv ist und für den Knoten eine 3D-Welt zeichnet.

### Eigenschaften:

**private World World**

Die Spielwelt in der die 3D-Objekte des dargestellten Knotens enthalten sind.

**private KnotRenderer KnotRenderer**

Der Controller, der aus dem Knoten die 3D-Modelle erstellt.

**public Stack<Knot> Undo**

Der Undo-Stack.

**public Stack<Knot> Redo**

Der Redo-Stack.

**public Knot Knot**

Der Knoten, der vom Spieler bearbeitet wird.

### Konstruktoren:

**public CreativeModeScreen (Knot3Game game, Knot knot)**

Erzeugt eine neue Instanz eines `CreativeModeScreen`-Objekts und initialisiert diese mit einem `Knot3Game`-Objekt *game*, sowie einem Knoten *knot*.

### Methoden:

**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

**public void Entered (GameScreen previousScreen, GameTime time)**

Fügt die 3D-Welt und den Inputhandler in die Spielkomponentenliste ein.

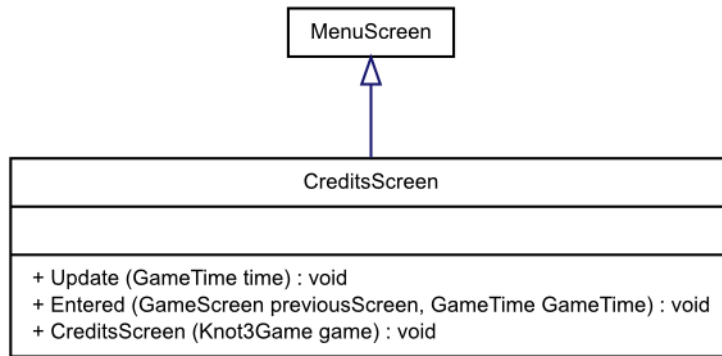
#### 3.3.1.9. Klasse CreditsScreen

### Beschreibung:

Der Spielzustand, der die Auflistung der Mitwirkenden darstellt.

### Konstruktoren:

**public CreditsScreen (Knot3Game game)**



Erzeugt ein neues `CreditsScreen`-Objekt und initialisiert dieses mit einem `Knot3Game`-Objekt.

#### Methoden:

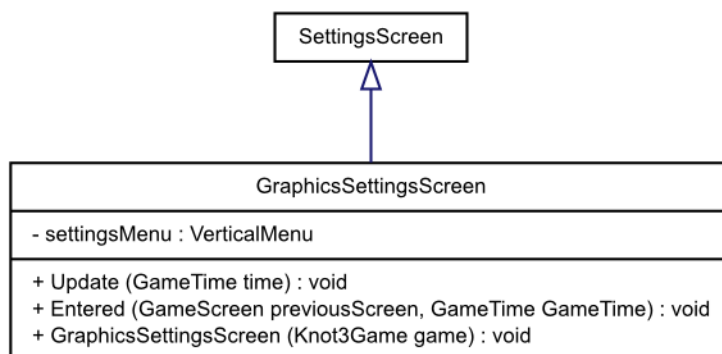
**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

**public void Entered (GameScreen previousScreen, GameTime gameTime)**

Fügt das Menü mit den Mitwirkenden in die Spielkomponentenliste ein.

#### 3.3.1.10. Klasse GraphicsSettingsScreen



#### Beschreibung:

Der Spielzustand, der die Grafik-Einstellungen darstellt.

#### Eigenschaften:

**private VerticalMenu settingsMenu**

Das Menü, das die Einstellungen enthält.

#### Konstruktoren:

**public GraphicsSettingsScreen (Knot3Game game)**

Erzeugt ein neues `GraphicsSettingsScreen`-Objekt und initialisiert dieses mit einem `Knot3Game`-Objekt.

#### Methoden:

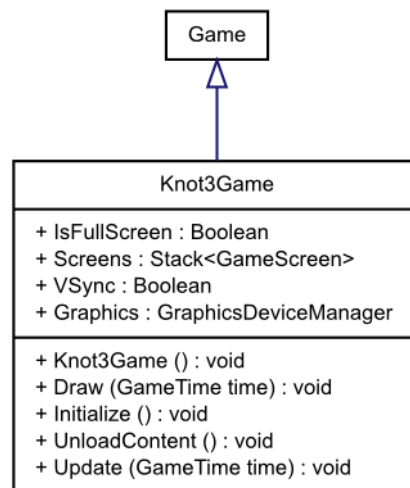
**public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

**public void Entered (GameScreen previousScreen, GameTime gameTime)**

Fügt das Menü mit den Einstellungen in die Spielkomponentenliste ein.

#### 3.3.1.11. Klasse Knot3Game



#### Beschreibung:

Die zentrale Spielklasse, die von der „Game“-Klasse des XNA-Frameworks erbt.

#### Eigenschaften:

**public Boolean IsFullScreen**

Wird dieses Attribut ausgelesen, dann gibt es einen Wahrheitswert zurück, der angibt, ob sich das Spiel im Vollbildmodus befindet. Wird dieses Attribut auf einen Wert gesetzt, dann wird der Modus entweder gewechselt oder beibehalten, falls es auf denselben Wert gesetzt wird.

**public Stack<GameScreen> Screens**

Enthält als oberste Element den aktuellen Spielzustand und darunter die zuvor aktiven Spielzustände.

**public Boolean VSync**

Dieses Attribut dient sowohl zum Setzen des Aktivierungszustandes der vertikalen Synchronisation, als auch zum Auslesen dieses Zustandes.

## **public GraphicsDeviceManager Graphics**

Der aktuelle Grafikgeräteverwalter des XNA-Frameworks.

### **Konstruktoren:**

#### **public Knot3Game ()**

Erstellt ein neues zentrales Spielobjekt und setzt die Auflösung des BackBuffers auf die in der Einstellungsdatei gespeicherte Auflösung oder falls nicht vorhanden auf die aktuelle Bildschirmauflösung und wechselt in den Vollbildmodus.

### **Methoden:**

#### **public void Draw (GameTime time)**

Ruft die Draw()-Methode des aktuellen Spielzustands auf.

#### **public void Initialize ()**

Initialisiert die Attribute dieser Klasse.

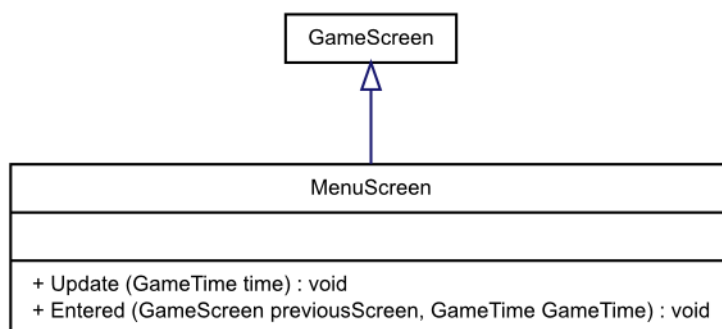
#### **public void UnloadContent ()**

Macht nichts. Das Freigeben aller Objekte wird von der automatischen Speicherbereinigung übernommen.

#### **public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

### **3.3.1.12. Klasse MenuScreen**



### **Beschreibung:**

Eine abstrakte Klasse, von der alle Spielzustände erben, die Menüs darstellen.

### **Methoden:**

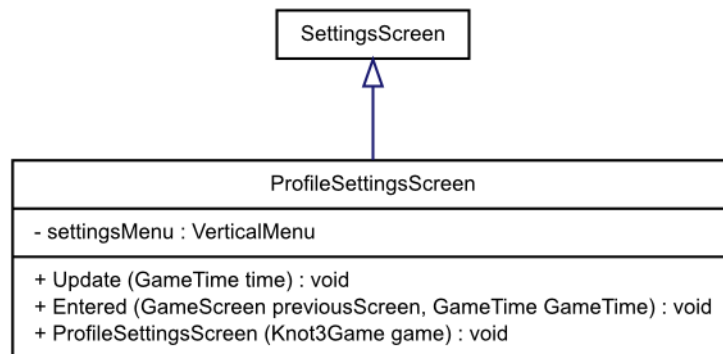
#### **public void Update (GameTime time)**

Wird für jeden Frame aufgerufen.

**public void Entered (GameScreen previousScreen, gameTime gameTime)**

Wird aufgerufen, wenn in diesen Spielzustand gewechselt wird.

### 3.3.1.13. Klasse ProfileSettingsScreen



#### Beschreibung:

Der Spielzustand, der die Profil-Einstellungen darstellt.

#### Eigenschaften:

**private VerticalMenu settingsMenu**

Das vertikale Menü wo die Einstellungen anzeigt. Hier nimmt der Spieler Einstellungen vor.

#### Konstruktoren:

**public ProfileSettingsScreen (Knot3Game game)**

Erzeugt eine neue Instanz eines ProfileSettingsScreen-Objekts und initialisiert dieses mit einem Knot3Game-Objekt.

#### Methoden:

**public void Update (GameTime time)**

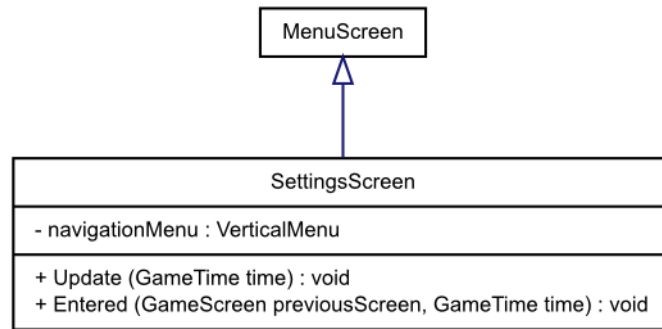
Wird für jeden Frame aufgerufen.

**public void Entered (GameScreen previousScreen, gameTime gameTime)**

Fügt das Menü mit den Einstellungen in die Spielkomponentenliste ein.

### 3.3.1.14. Klasse SettingsScreen

#### Beschreibung:



Ein Spielzustand, der das Haupt-Einstellungsmenü zeichnet.

#### Eigenschaften:

**private VerticalMenu** navigationMenu

#### Methoden:

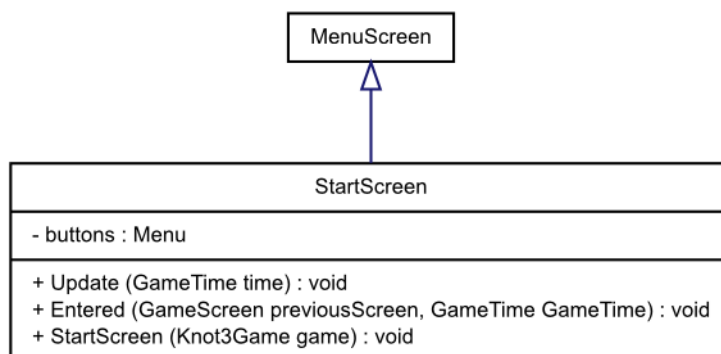
**public void** Update (**GameTime** time)

Wird für jeden Frame aufgerufen.

**public void** Entered (**GameScreen** previousScreen, **GameTime** time)

Fügt das Haupt-Einstellungsmenü in die Spielkomponentenliste ein.

#### 3.3.1.15. Klasse StartScreen



#### Beschreibung:

Der Startbildschirm.

#### Eigenschaften:

**private Menu** buttons

Die Schaltflächen des Startbildschirms.

#### Konstruktoren:

**public** StartScreen (Knot3Game game)

Erzeugt eine neue Instanz eines StartScreen-Objekts und initialisiert diese mit einem Knot3Game-Objekt.

#### Methoden:

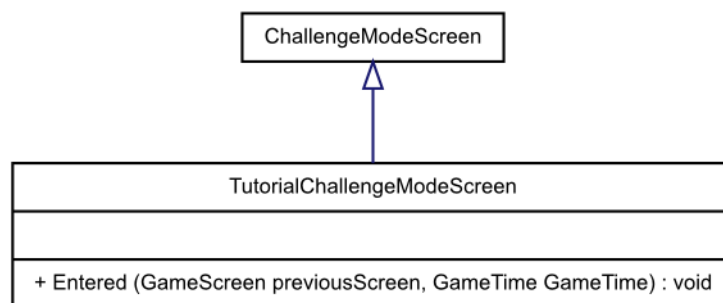
**public void** Update (GameTime time)

Wird für jeden Frame aufgerufen.

**public void** Entered (GameScreen previousScreen, GameTime gameTime)

Fügt die das Menü in die Spielkomponentenliste ein.

#### 3.3.1.16. Klasse TutorialChallengeModeScreen



#### Beschreibung:

Eine Einführung in das Spielen von Challenges. Der Spieler wird dabei durch Anweisungen an das Lösen von Challenges herangeführt.

#### Methoden:

**public void** Entered (GameScreen previousScreen, GameTime gameTime)

Fügt die Tutoriellanzeige in die Spielkomponentenliste ein.

#### 3.3.2. Schnittstellen

#### 3.3.3. Enumerationen

### 3.4. Paket RenderEffects

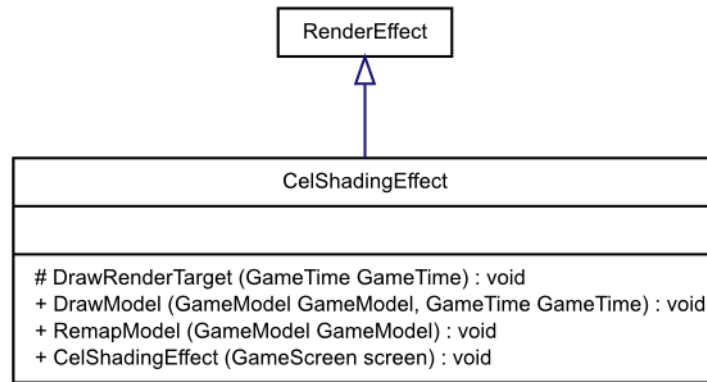
#### 3.4.1. Klassen

##### 3.4.1.1. Klasse CelShadingEffect

#### Beschreibung:

Ein Cel-Shading-Effekt.





#### Konstruktoren:

**public CelShadingEffect (GameScreen screen)**

Erstellt einen neuen Cel-Shading-Effekt für den angegebenen GameScreen.

#### Methoden:

**protected void DrawRenderTarget (GameTime GameTime)**

Zeichnet das Rendertarget.

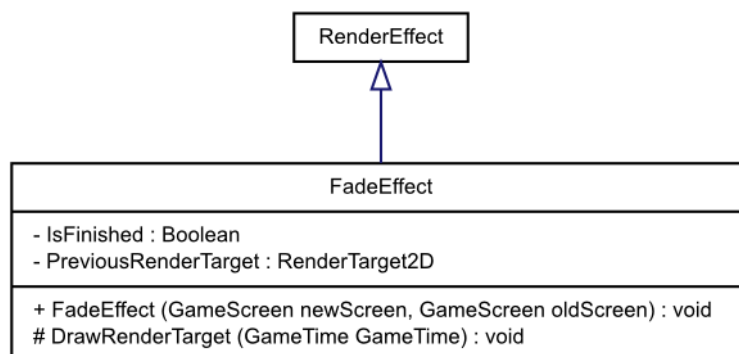
**public void DrawModel (GameModel GameModel, GameTime GameTime)**

Zeichnet das Spielmodell model mit dem Cel-Shading-Effekt. Eine Anwendung des NVIDIA-Toon-Shaders.

**public void RemapModel (GameModel GameModel)**

Weist dem 3D-Modell den Cel-Shader zu.

#### 3.4.1.2. Klasse FadeEffect



#### Beschreibung:

Ein Postprocessing-Effekt, der eine Überblendung zwischen zwei Spielzuständen darstellt.

#### Eigenschaften:

**private Boolean IsFinished**

Gibt an, ob die Überblendung abgeschlossen ist und das RenderTarget nur noch den neuen Spielzustand darstellt.

**private RenderTarget2D PreviousRenderTarget**

Der zuletzt gerenderte Frame im bisherigen Spielzustand.

**Konstruktoren:**

**public FadeEffect (GameScreen newScreen, GameScreen oldScreen)**

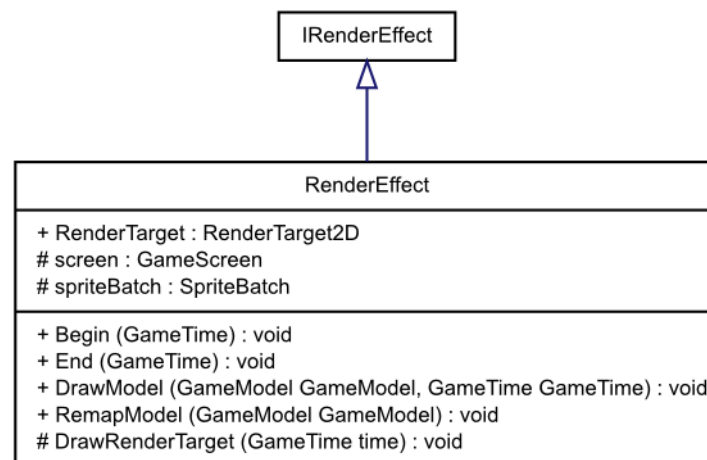
Erstellt einen Überblende-Effekt zwischen den angegebenen Spielzuständen.

**Methoden:**

**protected void DrawRenderTarget (GameTime gameTime)**

Zeichnet das Rendertarget.

### 3.4.1.3. Klasse RenderEffect



**Beschreibung:**

Eine abstrakte Klasse, die eine Implementierung von `IRenderEffect` darstellt.

**Eigenschaften:**

**public RenderTarget2D RenderTarget**

Das Rendertarget, in das zwischen dem Aufruf der `Begin()`- und der `End()`-Methode gezeichnet wird, weil es in `Begin()` als primäres Rendertarget des XNA-Frameworks gesetzt wird.

**protected GameScreen screen**

Der Spielzustand, in dem der Effekt verwendet wird.

### **protected SpriteBatch spriteBatch**

Ein Spritestapel (s. Glossar oder <http://msdn.microsoft.com/en-us/library/bb203919.aspx>), der verwendet wird, um das Rendertarget dieses Rendereffekts auf das übergeordnete Rendertarget zu zeichnen.

#### **Methoden:**

##### **public void Begin (GameTime)**

In der Methode Begin() wird das aktuell von XNA genutzte Rendertarget auf einem Stack gesichert und das Rendertarget des Effekts wird als aktuelles Rendertarget gesetzt.

##### **public void End (GameTime)**

Das auf dem Stack gesicherte, vorher genutzte Rendertarget wird wiederhergestellt und das Rendertarget dieses Rendereffekts wird, unter Umständen in Unterklassen verändert, auf dieses übergeordnete Rendertarget gezeichnet.

##### **public void DrawModel (GameModel GameModel, GameTime GameTime)**

Zeichnet das Spielmodell model mit diesem Rendereffekt.

##### **public void RemapModel (GameModel GameModel)**

Beim Laden des Modells wird von der XNA-Content-Pipeline jedem ModelMeshPart ein Shader der Klasse BasicEffect zugewiesen. Für die Nutzung des Modells in diesem Rendereffekt kann jedem ModelMeshPart ein anderer Shader zugewiesen werden.

##### **protected void DrawRenderTarget (GameTime time)**

Zeichnet das Rendertarget.

#### **3.4.1.4. Klasse RenderEffectStack**

RenderEffectStack
+ CurrentEffect : IRenderEffect - DefaultEffect : IRenderEffect
+ Pop () : IRenderEffect + Push (IRenderEffect effect) : void + RenderEffectStack (IRenderEffect defaultEffect) : void

#### **Beschreibung:**

Ein Stapel, der während der Draw-Aufrufe die Hierarchie der aktuell verwendeten Rendereffekte verwaltet und automatisch das aktuell von XNA verwendete Rendertarget auf das Rendertarget des obersten Rendereffekts setzt.

#### **Eigenschaften:**

**public IRenderEffect CurrentEffect**

Der oberste Rendereffekt.

**private IRenderEffect DefaultEffect**

Der Standard-Rendereffekt, der verwendet wird, wenn der Stapel leer ist.

**Konstruktoren:**

**public RenderEffectStack (IRenderEffect defaultEffect)**

Erstellt einen neuen Rendereffekt-Stapel.

**Methoden:**

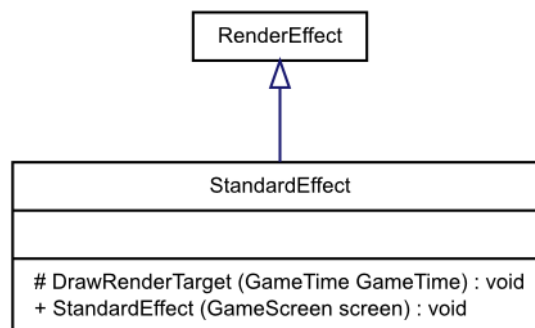
**public IRenderEffect Pop ()**

Entfernt den obersten Rendereffekt vom Stapel.

**public void Push (IRenderEffect effect)**

Legt einen Rendereffekt auf den Stapel.

#### 3.4.1.5. Klasse StandardEffect



**Beschreibung:**

Ein Rendereffekt, der 3D-Modelle mit dem von der XNA-Content-Pipeline standardmäßig zugewiesenen BasicEffect-Shader zeichnet und keinen Post-Processing-Effekt anwendet.

**Konstruktoren:**

**public StandardEffect (GameScreen screen)**

Erstellt einen neuen Standardeffekt.

**Methoden:**

**protected void DrawRenderTarget (GameTime gameTime)**

Zeichnet das Rendertarget.

### 3.4.2. Schnittstellen

#### 3.4.2.1. Schnittstelle IRenderEffect

IRenderEffect
+ RenderTarget : RenderTarget2D
+ Begin (GameTime) : void + End (GameTime) : void + DrawModel (GameModel model, GameTime time) : void + RemapModel (GameModel model) : void

##### Beschreibung:

Stellt eine Schnittstelle für Klassen bereit, die Rendereffekte ermöglichen.

##### Eigenschaften:

**public RenderTarget2D RenderTarget**

Das Rendertarget, in das zwischen dem Aufruf der Begin()- und der End()-Methode gezeichnet wird, weil es in Begin() als primäres Rendertarget des XNA-Frameworks gesetzt wird.

##### Methoden:

**public void Begin (GameTime)**

In der Methode Begin() wird das aktuell von XNA genutzte Rendertarget auf einem Stapel gesichert und das Rendertarget des Effekts wird als aktuelles Rendertarget gesetzt.

**public void End (GameTime)**

Das auf dem Stapel gesicherte, vorher genutzte Rendertarget wird wiederhergestellt und das Rendertarget dieses Rendereffekts wird, unter Umständen in Unterklassen verändert, auf dieses übergeordnete Rendertarget gezeichnet.

**public void DrawModel (GameModel model, GameTime time)**

Zeichnet das Spielmodell *model* mit diesem Rendereffekt.

**public void RemapModel (GameModel model)**

Beim Laden des Modells wird von der XNA-Content-Pipeline jedem ModelMeshPart ein Shader der Klasse BasicEffect zugewiesen. Für die Nutzung des Modells in diesem Rendereffekt kann jedem ModelMeshPart ein anderer Shader zugewiesen werden.

### 3.4.3. Enumerationen

## 3.5. Paket KnotData

### 3.5.1. Klassen

#### 3.5.1.1. Klasse Challenge

Challenge
+ Start : Knot + Target : Knot - highscore : SortedList<Integer, String> - format : IChallengeIO + Highscore : IEnumerable<KeyValuePair<String, Integer>> + MetaData : ChallengeMetaData + Name : String
+ Challenge (ChallengeMetaData meta, Knot start, Knot target) : void + AddToHighscore (String name, Integer time) : void

#### Beschreibung:

Ein Objekt dieser Klasse repräsentiert eine [Challenge](#).

#### Eigenschaften:

**public Knot Start**

Der Ausgangsknoten, den der Spieler in den Referenzknoten transformiert.

**public Knot Target**

Der Referenzknoten, in den der Spieler den Ausgangsknoten transformiert.

**private SortedList<Integer, String> highscore**

Eine sortierte Bestenliste.

**private IChallengeIO format**

Das Speicherformat der [Challenge](#).

**public IEnumerable<KeyValuePair<String, Integer>> Highscore**

Ein öffentlicher Enumerator, der die Bestenliste unabhängig von der darunterliegenden Datenstruktur zugänglich macht.

**public ChallengeMetaData MetaData**

Die Metadaten der [Challenge](#).

**public String Name**

Der Name der Challenge.

#### Konstruktoren:

**public Challenge (ChallengeMetaData meta, Knot start, Knot target)**

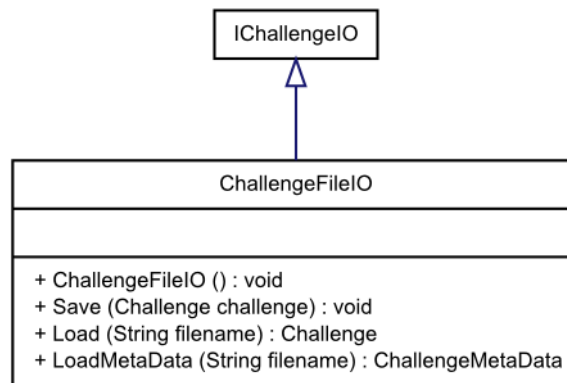
Erstellt ein Challenge-Objekt aus einem gegebenen Challenge-Metadaten-Objekt. Erstellt ein Challenge-Objekt aus einer gegebenen Challenge-Datei.

#### Methoden:

**public void AddToHighscore (String name, Integer time)**

Fügt eine neue Bestzeit eines bestimmten Spielers in die Bestenliste ein.

#### 3.5.1.2. Klasse ChallengeFileIO



#### Beschreibung:

Implementiert das Speicherformat für Challenges.

#### Konstruktoren:

**public ChallengeFileIO ()**

Erstellt ein ChallengeFileIO-Objekt.

#### Methoden:

**public void Save (Challenge challenge)**

Speichert eine Challenge in dem Dateinamen, der in dem Challenge-Objekt enthalten ist.

**public Challenge Load (String filename)**

Lädt eine Challenge aus einer angegebenen Datei.

**public ChallengeMetaData LoadMetaData (String filename)**

Lädt die Metadaten einer [Challenge](#) aus einer angegebenen Datei.

### 3.5.1.3. Klasse ChallengeMetaData

ChallengeMetaData
+ Name : String + Start : KnotMetaData + Target : KnotMetaData + Format : IChallengeIO + Filename : String + Highscore : IEnumerable<KeyValuePair<String, Integer>>
+ ChallengeMetaData (String name, KnotMetaData start, KnotMetaData target, String filename, IChallengeIO format) : void

#### Beschreibung:

Enthält Metadaten zu einer [Challenge](#).

#### Eigenschaften:

**public String Name**

Der Name der [Challenge](#).

**public KnotMetaData Start**

Der Ausgangsknoten, den der Spieler in den Referenzknoten transformiert.

**public KnotMetaData Target**

Der Referenzknoten, in den der Spieler den Ausgangsknoten transformiert.

**public IChallengeIO Format**

Das Format, aus dem die Metadaten der [Challenge](#) gelesen wurden oder null.

**public String Filename**

Der Dateiname, aus dem die Metadaten der [Challenge](#) gelesen wurden oder in den sie abgespeichert werden.

**public IEnumerable<KeyValuePair<String, Integer>> Highscore**

Ein öffentlicher Enumerator, der die Bestenliste unabhängig von der darunterliegenden Datenstruktur zugänglich macht.

#### Konstruktoren:

**public ChallengeMetaData (String name, KnotMetaData start, KnotMetaData target, String filename, IChallengeIO format)**



Erstellt ein **Challenge**-Metadaten-Objekt mit einem gegebenen Namen und den Metadaten des Ausgangs- und Referenzknotens.

#### 3.5.1.4. Klasse Edge

Edge
+ Color : Color + Direction : Direction + Rectangles : List<int>
+ Edge (Direction direction) : void + Get3DDirection () : Vector3

##### Beschreibung:

Eine Kante eines Knotens, die aus einer Richtung und einer Farbe, sowie optional einer Liste von Flächennummern besteht.

##### Eigenschaften:

**public Color** Color

Die Farbe der Kante.

**public Direction** Direction

Die Richtung der Kante.

**public List<int>** Rectangles

Die Liste der Flächennummern, die an die Kante angrenzen.

##### Konstruktoren:

**public Edge (Direction direction)**

Erstellt eine neue Kante mit der angegebenen Richtung.

##### Methoden:

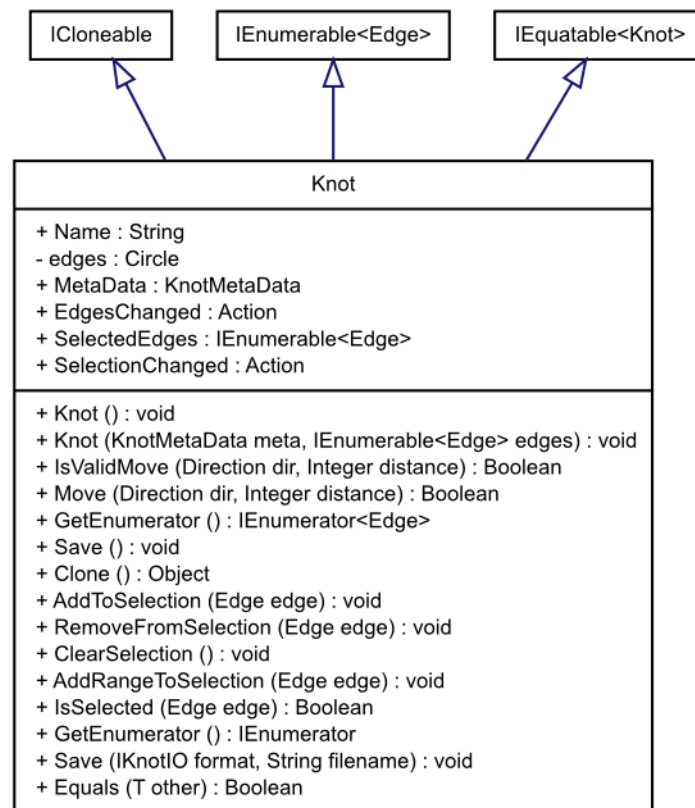
**public Vector3** Get3DDirection ()

Gibt die Richtung als normalisierten Vektor3 zurück.

#### 3.5.1.5. Klasse Knot

##### Beschreibung:

Diese Klasse repräsentiert einen Knoten, bestehend aus einem Knoten-Metadaten-Objekt und einer doppelt-verketteten Liste von Kanten. Ein Knoten ist eine zyklische Kantenfolge, bei der keine zwei Kanten Kanten den gleichen Raum einnehmen.



## Eigenschaften:

### public String Name

Der Name des Knotens, welcher auch leer sein kann. Beim Speichern muss der Nutzer in diesem Fall zwingend einen nichtleeren Namen wählen. Der Wert dieser Eigenschaft wird aus der „Name“-Eigenschaft des Metadaten-Objektes geladen und bei Änderungen wieder in diesem gespeichert. Beim Ändern dieser Eigenschaft wird automatisch auch der im Metadaten-Objekt enthaltene Dateiname verändert.

### private Circle edges

Das Startelement der doppelt-verketteten Liste, in der die Kanten gespeichert werden.

### public KnotMetaData MetaData

Die Metadaten des Knotens.

### public Action EdgesChanged

Ein Ereignis, das in der Move-Methode ausgelöst wird, wenn sich die Struktur der Kanten geändert hat.

### public IEnumerable<Edge> SelectedEdges

Enthält die aktuell vom Spieler selektierten Kanten in der Reihenfolge, in der sie selektiert wurden.

### public Action SelectionChanged

## Konstrukturen:

### **public** Knot ()

Erstellt einen minimalen Standardknoten. Das Metadaten-Objekt enthält in den Eigenschaften, die das Speicherformat und den Dateinamen beinhalten, den Wert „null“.

### **public** Knot (KnotMetaData meta, IEnumerable<Edge> edges)

Erstellt einen neuen Knoten mit dem angegebenen Metadaten-Objekt und den angegebenen Kanten, die in der doppelt verketteten Liste gespeichert werden. Die Eigenschaft des Metadaten-Objektes, die die Anzahl der Kanten enthält, wird auf ein Delegate gesetzt, welches jeweils die aktuelle Anzahl der Kanten dieses Knotens zurückgibt.

## Methoden:

### **public** Boolean IsValidMove (Direction dir, Integer distance)

Prüft, ob eine Verschiebung der aktuellen Kantenauswahl in die angegebene Richtung um die angegebene Distanz gültig ist.

### **public** Boolean Move (Direction dir, Integer distance)

Verschiebt die aktuelle Kantenauswahl in die angegebene Richtung um die angegebene Distanz.

### **public** IEnumerator<Edge> GetEnumerator ()

Gibt die doppelt-verkettete Kantenliste als Enumerator zurück.

### **public** void Save ()

Speichert den Knoten unter dem Dateinamen in dem Dateiformat, das in dem Metadaten-Objekt angegeben ist. Enthalten entweder die Dateiname-Eigenschaft, die Dateiformat-Eigenschaft oder beide den Wert „null“, dann wird eine IOException geworfen.

### **public** Object Clone ()

Erstellt eine vollständige Kopie des Knotens, inklusive der Kanten-Datenstruktur und des Metadaten-Objekts.

### **public** void AddToSelection (Edge edge)

Fügt die angegebene Kante zur aktuellen Kantenauswahl hinzu.

### **public** void RemoveFromSelection (Edge edge)

Entfernt die angegebene Kante von der aktuellen Kantenauswahl.

### **public** void ClearSelection ()

Hebt die aktuelle Kantenauswahl auf.

**public void AddRangeToSelection (Edge edge)**

Fügt alle Kanten auf dem kürzesten Weg zwischen der zuletzt ausgewählten Kante und der angegebenen Kante zur aktuellen Kantenauswahl hinzu. Sind beide Wege gleich lang, wird der Weg in Richtung der ersten Kante ausgewählt.

**public Boolean IsSelected (Edge edge)**

Prüft, ob die angegebene Kante in der aktuellen Kantenauswahl enthalten ist.

**public IEnumerator GetEnumerator ()**

Gibt die doppelt-verkettete Kantenliste als Enumerator zurück.

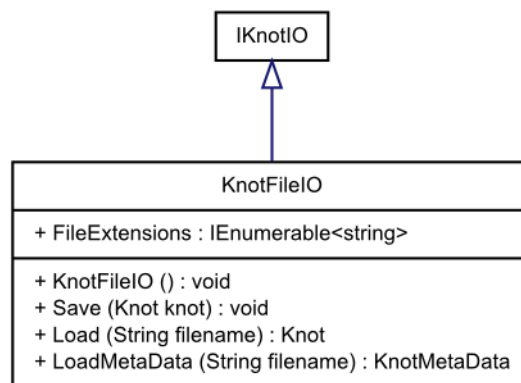
**public void Save (IKnotIO format, String filename)**

Speichert den Knoten unter dem angegebenen Dateinamen in dem angegebenen Datei`format`.

**public Boolean Equals (T other)**

Prüft, ob die räumliche Struktur identisch ist, unabhängig von dem Startpunkt und der Richtung der Datenstruktur.

#### 3.5.1.6. Klasse KnotFileIO



#### Beschreibung:

Implementiert das Speicherformat für Knoten.

#### Eigenschaften:

**public IEnumerable<string> FileExtensions**

Die für eine Knoten-Datei gültigen Dateierendungen.

#### Konstruktoren:

**public KnotFileIO ()**

Erstellt ein `KnotFileIO`-Objekt.

#### Methoden:

**public void Save (Knot knot)**

Speichert einen Knoten in dem Dateinamen, der in dem `Knot`-Objekt enthalten ist.

**public Knot Load (String filename)**

Lädt einen Knoten aus einer angegebenen Datei.

**public KnotMetaData LoadMetaData (String filename)**

Lädt die Metadaten eines Knotens aus einer angegebenen Datei.

#### 3.5.1.7. Klasse KnotMetaData

KnotMetaData
+ Name : String + Format : IKnotIO + CountEdges : Func<Integer> + Filename : String
+ KnotMetaData (String name, Func<Integer> countEdges, IKnotIO format, String filename) : KnotMetaData + KnotMetaData (String name, Func<Integer> countEdges) : KnotMetaData

#### Beschreibung:

Enthält Metadaten eines Knotens, die aus einer Spielstand-Datei schneller eingelesen werden können, als der vollständige Knoten. Dieses Objekt enthält keine Datenstruktur zur Repräsentation der Kanten, sondern nur Informationen über den Namen des Knoten und die Anzahl seiner Kanten. Es kann ohne ein dazugehöriges Knoten-Objekt existieren, aber jedes Knoten-Objekt enthält genau ein Knoten-Metadaten-Objekt.

#### Eigenschaften:

**public String Name**

Der Anzeigename des Knotens, welcher auch leer sein kann. Beim Speichern muss der Spieler in diesem Fall zwingend einen nichtleeren Namen wählen. Wird ein neuer Anzeigename festgelegt, dann wird der Dateiname ebenfalls auf einen neuen Wert gesetzt, unabhängig davon ob er bereits einen Wert enthält oder „null“ ist. Diese Eigenschaft kann öffentlich gelesen und gesetzt werden.

**public IKnotIO Format**

Das Format, aus dem die Metadaten geladen wurden. Es ist genau dann „null“, wenn die Metadaten nicht aus einer Datei gelesen wurden. Nur lesbar.

**public Func<Integer> CountEdges**

Ein Delegate, das die Anzahl der Kanten zurückliefert. Falls dieses Metadaten-Objekt Teil eines Knotens ist, gibt es dynamisch die Anzahl der Kanten des Knoten-Objektes zurück. Anderenfalls gibt es eine statische Zahl zurück, die beim Einlesen der Metadaten vor dem Erstellen dieses Objektes gelesen wurde. Nur lesbar.

**public String Filename**

Falls die Metadaten aus einer Datei eingelesen wurden, enthält dieses Attribut den Dateinamen, sonst „null“.

**Konstruktoren:**

**public KnotMetaData (String name, Func<Integer> countEdges, IKnotIO format, String filename)**

Erstellt ein neues Knoten-Metadaten-Objekt mit einem angegebenen Knoten*namen* und einer angegebenen Funktion, welche eine Kantenanzahl zurück gibt. Zusätzlich wird der Dateiname oder das Speicher*format* angegeben, aus dem die Metadaten gelesen wurden.

**public KnotMetaData (String name, Func<Integer> countEdges)**

Erstellt ein neues Knoten-Metadaten-Objekt mit einem angegebenen Knoten*namen* und einer angegebenen Funktion, welche eine Kantenanzahl zurück gibt.

### 3.5.1.8. Klasse KnotStringIO

KnotStringIO
+ Name : String + Edges : IEnumerable<Edge> + CountEdges : Integer + Content : String
+ KnotStringIO (String content) : void + KnotStringIO (Knot knot) : void

**Beschreibung:**

Diese Klasse repräsentiert einen Parser für das Knoten-Austauschformat und enthält die eingelesenen Informationen wie den Namen des Knotens und die Kantenliste als Eigenschaften.

**Eigenschaften:**

**public String Name**

Der Name der eingelesenen Knotendatei oder des zugewiesenen Knotenobjektes.

**public IEnumerable<Edge> Edges**

Die Kanten der eingelesenen Knotendatei oder des zugewiesenen Knotenobjektes.

### **public Integer CountEdges**

Die Anzahl der Kanten der eingelesenen Knotendatei oder des zugewiesenen Knotenobjektes.

### **public String Content**

Erstellt aus den „Name“ - und „Edges“ -Eigenschaften einen neue Zeichenkette, die als Dateinhalt in einer Datei eines Spielstandes einen gültigen Knoten repräsentiert.

#### **Konstruktoren:**

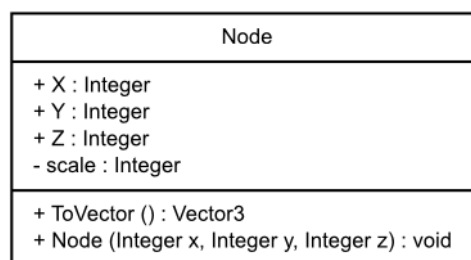
### **public KnotStringIO (String content)**

Liest das in der angegebenen Zeichenkette enthaltene Dateiformat ein. Enthält es einen gültigen Knoten, so werden die „Name“ - und „Edges“ -Eigenschaften auf die eingelesenen Werte gesetzt. Enthält es einen ungültigen Knoten, so wird eine IOException geworfen und das Objekt wird nicht erstellt.

### **public KnotStringIO (Knot knot)**

Erstellt ein neues Objekt und setzt die „Name“ - und „Edge“ -Eigenschaften auf die im angegebenen Knoten enthaltenen Werte.

#### **3.5.1.9. Klasse Node**



#### **Beschreibung:**

Eine Position im 3D-Raster. Die Werte für alle drei Koordinaten sind Integer, wobei 1 die Breite der Raster-Abschnitte angibt. Eine Skalierung auf Koordinaten im 3D-Raum und damit einhergehend eine Konvertierung in ein Vector3-Objekt des XNA-Frameworks kann mit der Methode ToVector() angefordert werden.

#### **Eigenschaften:**

### **public Integer X**

X steht für eine x-Koordinate im dreidimensionalen Raster.

### **public Integer Y**

Y steht für eine y-Koordinate im dreidimensionalen Raster.

### **public Integer Z**

Z steht für eine z-Koordinate im dreidimensionalen Raster.

**private Integer scale**

Ein Skalierungswert.

**Konstruktoren:**

**public Node (Integer x, Integer y, Integer z)**

Erzeugt eine neue Instanz eines **Node**-Objekts und initialisiert diese mit Werten für die *x*-, *y*- und *z*-Koordinate.

**Methoden:**

**public Vector3 ToVector ()**

Liefert die *x*-, *y*- und *z*-Koordinaten im 3D-Raum als ein Vektor3 der Form (*x*, *y*, *z*).

#### 3.5.1.10. Klasse NodeMap

NodeMap
+ Scale : Integer
+ From (Edge edge) : Node + To (Edge edge) : Node + OnEdgesChanged () : void

**Beschreibung:**

Eine Zuordnung zwischen Kanten und den dreidimensionalen Rasterpunkten, an denen sich die Kantenübergänge befinden.

**Eigenschaften:**

**public Integer Scale**

Die Skalierung, die bei einer Konvertierung in einen Vector3 des XNA-Frameworks durch die ToVector()-Methode der **Node**-Objekte verwendet wird.

**Methoden:**

**public Node From (Edge edge)**

Gibt die Rasterposition des Übergangs am Anfang der Kante zurück.

**public Node To (Edge edge)**

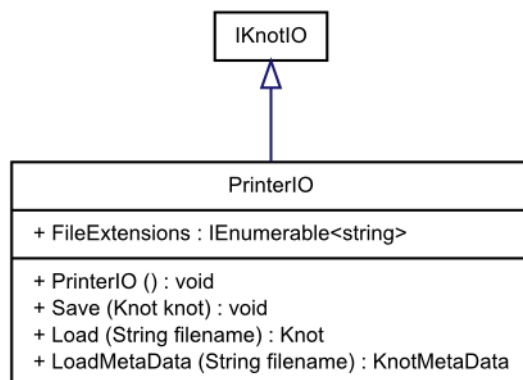
Gibt die Rasterposition des Übergangs am Ende der Kante zurück.



**public void OnEdgesChanged ()**

Aktualisiert die Zuordnung, wenn sich die Kanten geändert haben.

### 3.5.1.11. Klasse PrinterIO



#### Beschreibung:

Ein Exportformat für 3D-Drucker.

#### Eigenschaften:

**public IEnumerable<string> FileExtensions**

Die gültigen Dateiendungen für das 3D-Drucker-Format.

#### Konstruktoren:

**public PrinterIO ()**

Erstellt ein neues `PrinterIO`-Objekt.

#### Methoden:

**public void Save (Knot knot)**

Exportiert den Knoten in einem gültigen 3D-Drucker-Format.

**public Knot Load (String filename)**

Gibt eine `IOException` zurück.

**public KnotMetaData LoadMetaData (String filename)**

Gibt eine `IOException` zurück.

IChallengeIO
+ Save (Challenge challenge) : void + Load (String filename) : Challenge + LoadMetaData (String filename) : ChallengeMetaData

### 3.5.2. Schnittstellen

#### 3.5.2.1. Schnittstelle IChallengeIO

##### Beschreibung:

Diese Schnittstelle enthält Methoden, die von Speicherformaten für Challenges implementiert werden müssen.

##### Methoden:

**public void Save (Challenge challenge)**

Speichert eine Challenge.

**public Challenge Load (String filename)**

Lädt eine Challenge.

**public ChallengeMetaData LoadMetaData (String filename)**

Lädt die Metadaten einer Challenge.

#### 3.5.2.2. Schnittstelle IKnotIO

IKnotIO
+ FileExtensions : IEnumerable<string>
+ Save (Knot knot) : void + Load (String filename) : Knot + LoadMetaData (String filename) : KnotMetaData

##### Beschreibung:

Diese Schnittstelle enthält Methoden, die von Speicherformaten für Knoten implementiert werden müssen.

##### Eigenschaften:

**public IEnumerable<string> FileExtensions**

Aufzählung der Dateierweiterungen.

## Methoden:

**public void Save (Knot knot)**

Speichert einen Knoten.

**public Knot Load (String filename)**

Lädt einen Knoten.

**public KnotMetaData LoadMetaData (String filename)**

Lädt die Metadaten eines Knotens.

## 3.5.3. Enumerationen

### 3.5.3.1. Enumeration Direction

#### Beschreibung:

Eine Wertesammlung der möglichen Richtungen in einem dreidimensionalen Raum. Wird benutzt, damit keine ungültigen Kantenrichtungen angegeben werden können.

#### Eigenschaften:

**Left = 1**

Links.

**Right = 2**

Rechts.

**Up = 3**

Hoch.

**Down = 4**

Runter.

**Forward = 5**

Vorwärts.

**Backward = 6**

Rückwärts.

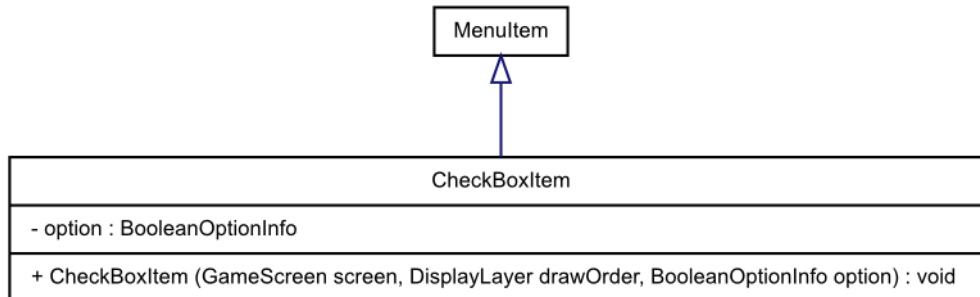
**Zero = 0**

Keine Richtung.

## 3.6. Paket Widgets

### 3.6.1. Klassen

#### 3.6.1.1. Klasse CheckBoxItem



#### Beschreibung:

Ein Menüeintrag, der einen Auswahlkasten darstellt.

#### Eigenschaften:

**private BooleanOptionInfo option**

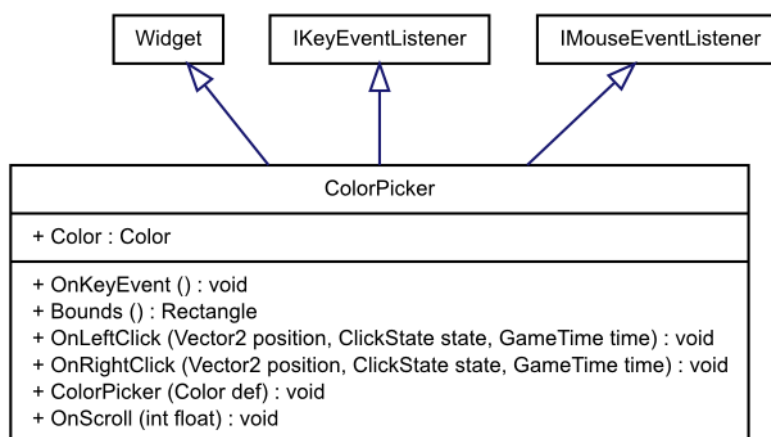
Die Option, die mit dem Auswahlkasten verknüpft ist.

#### Konstruktoren:

**public CheckBoxItem (GameScreen screen, DisplayLayer drawOrder, BooleanOptionInfo option)**

Erzeugt ein neues **CheckBoxItem**-Objekt und initialisiert dieses mit dem zugehörigen **GameScreen**-Objekt. Zudem sind Angaben zur Zeichenreihenfolge und der Auswahl~~option~~ Pflicht.

#### 3.6.1.2. Klasse ColorPicker



### Beschreibung:

Ein Steuerelement der grafischen Benutzeroberfläche, das eine Auswahl von Farben ermöglicht.

### Eigenschaften:

**public Color Color**

Die ausgewählte Farbe.

### Konstruktoren:

**public ColorPicker (Color def)**

Erzeugt eine neue Instanz eines **ColorPicker**-Objekts und initialisiert diese mit der Farbe, auf welche der Farbwähler beim Aufruf aus Sicht des Spielers zeigt.

### Methoden:

**public void OnKeyEvent ()**

Reagiert auf Tastatureingaben.

**public Rectangle Bounds ()**

Gibt die Ausmaße des ColorPickers zurück.

**public void OnLeftClick (Vector2 position, ClickState state, gameTime time)**

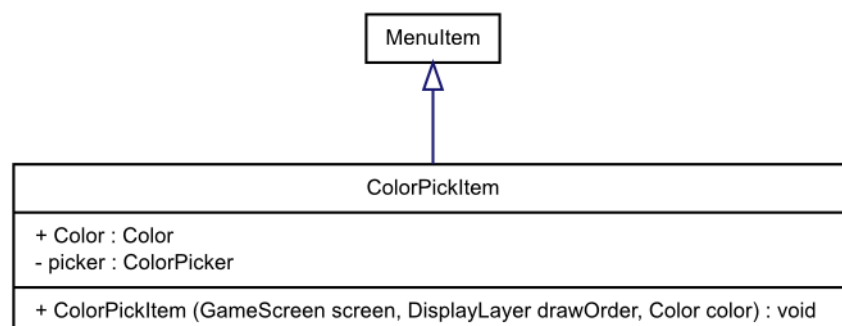
Bei einem Linksklick wird eine Farbe ausgewählt und im Attribut Color abgespeichert.

**public void OnRightClick (Vector2 position, ClickState state, gameTime time)**

Bei einem Rechtsklick geschieht nichts.

**public void OnScroll (int float)**

### 3.6.1.3. Klasse ColorPickItem



### Beschreibung:

Ein Menüeintrag, der eine aktuelle Farbe anzeigt und zum Ändern der Farbe per Mausklick einen `ColorPicker` öffnet.

### Eigenschaften:

`public Color color`

Die aktuelle Farbe.

`private ColorPicker picker`

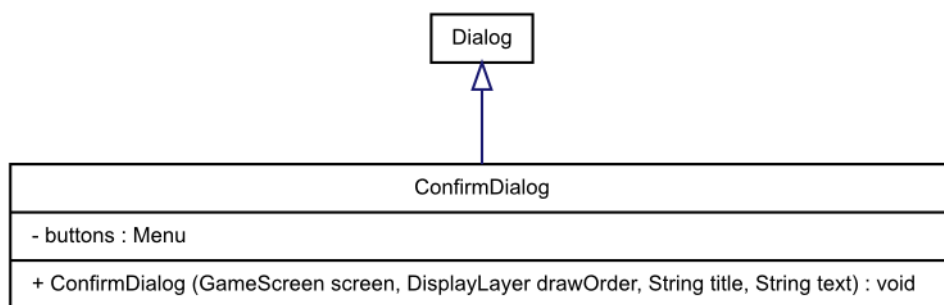
Der `ColorPicker`, der bei einem Mausklick auf den Menüeintrag geöffnet wird.

### Konstruktoren:

`public ColorPickItem (GameScreen screen, DisplayLayer drawOrder, Color color)`

Erzeugt ein neues `ColorPickItem`-Objekt und initialisiert dieses mit dem zugehörigen `GameScreen`-Objekt. Zudem sind Angaben zur Zeichenreihenfolge und der Auswahloption Pflicht.

#### 3.6.1.4. Klasse `ConfirmDialog`



### Beschreibung:

Ein `Dialog`, der Schaltflächen zum Bestätigen einer Aktion anzeigt.

### Eigenschaften:

`private Menu buttons`

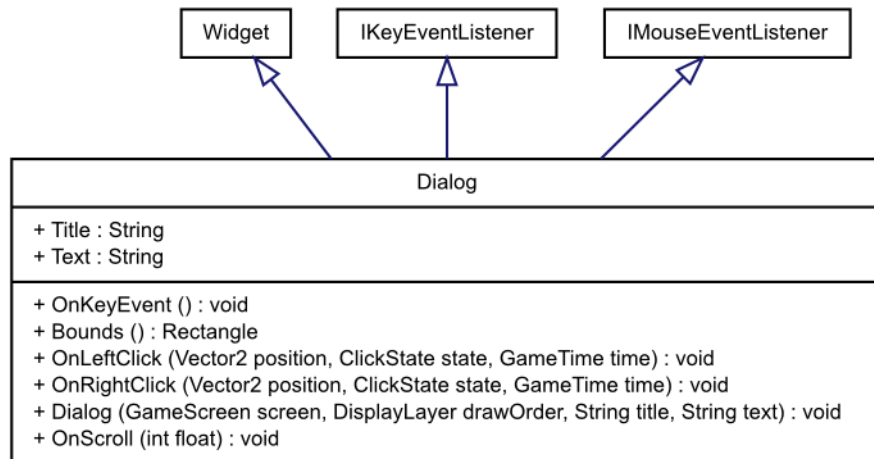
Das Menü, das Schaltflächen enthält.

### Konstruktoren:

`public ConfirmDialog (GameScreen screen, DisplayLayer drawOrder, String title, String text)`

Erzeugt ein neues `ConfirmDialog`-Objekt und initialisiert dieses mit dem zugehörigen `GameScreen`-Objekt. Zudem sind Angaben zur Zeichenreihenfolge, einer Zeichenkette für den Titel und für den eingeblendeten Text Pflicht.

### 3.6.1.5. Klasse Dialog



#### Beschreibung:

Ein **Dialog** ist ein im Vordergrund erscheinendes Fenster, das auf Nutzerinteraktionen wartet.

#### Eigenschaften:

**public String Title**

Der Fenstertitel.

**public String Text**

Der angezeigte Text.

#### Konstruktoren:

**public Dialog (GameScreen screen, DisplayLayer drawOrder, String title, String text)**

Erzeugt ein neues **Dialog**-Objekt und initialisiert dieses mit dem zugehörigen **GameScreen**-Objekt. Zudem sind Angaben zur Zeichenreihenfolge, einer Zeichenkette für den Titel und für den eingeblendeten Text Pflicht.

#### Methoden:

**public void OnKeyEvent ()**

Durch Drücken der Entertaste wird die ausgewählte Aktion ausgeführt. Durch Drücken der Escape-Taste wird der **Dialog** abgebrochen. Mit Hilfe der Pfeiltasten kann zwischen den Aktionen gewechselt werden.

**public Rectangle Bounds ()**

Gibt die Ausmaße des Dialogs zurück.

**public void OnLeftClick (Vector2 position, ClickState state, gameTime time)**

Bei einem Linksklick geschieht nichts.

**public void OnRightClick (Vector2 position, ClickState state, GameTime time)**

Bei einem Rechtsklick geschieht nichts.

**public void OnScroll (int float)**

#### 3.6.1.6. Klasse DropDownEntry

DropDownEntry
+ Text : String + OnSelect : Action
+ DropDownEntry (String text, Action onSelect) : void

##### Beschreibung:

Repräsentiert einen Eintrag in einem Dropdown-Menü.

##### Eigenschaften:

**public String Text**

Der Text des Eintrags.

**public Action OnSelect**

Die Aktion, welche bei der Auswahl ausgeführt wird.

##### Konstruktoren:

**public DropDownEntry (String text, Action onSelect)**

Erzeugt eine neue Instanz eines DropDownEntry-Objekts. *text* bezeichnet den Text für den Eintrag, *onSelect* ist die Aktion, welche bei der Auswahl des Eintrags auszuführen ist (s. Action).

#### 3.6.1.7. Klasse DropDownMenuItem

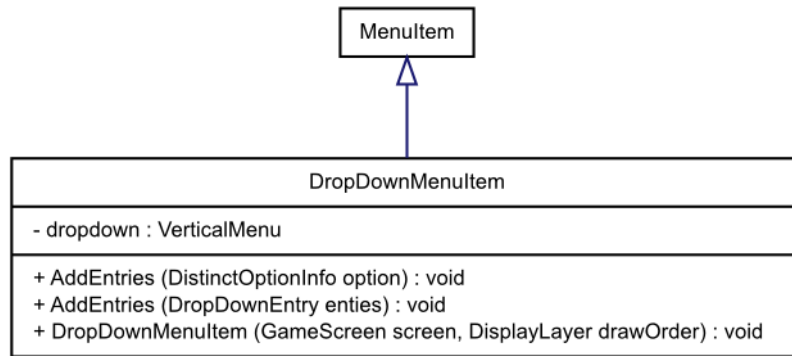
##### Beschreibung:

Ein Menüeintrag, der den ausgewählten Wert anzeigt und bei einem Linksklick ein Dropdown-Menü zur Auswahl eines neuen Wertes ein- oder ausblendet.

##### Eigenschaften:

**private VerticalMenu dropdown**





Das Dropdown-Menü, das ein- und ausgeblendet werden kann.

#### Konstruktoren:

**public DropDownMenuItem (GameScreen screen, DisplayLayer drawOrder)**

Erzeugt ein neues `ConfirmDialog`-Objekt und initialisiert dieses mit dem zugehörigen `GameScreen`-Objekt. Zudem ist die Angabe der Zeichenreihenfolge Pflicht.

#### Methoden:

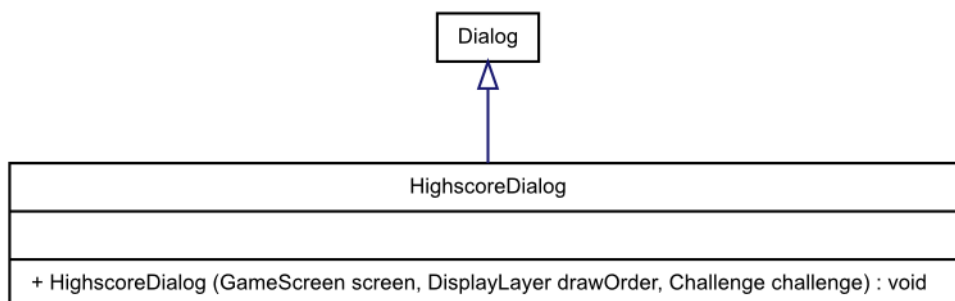
**public void AddEntries (DistinctOptionInfo option)**

Fügt Einträge in das Dropdown-Menü ein, die auf Einstellungs`optionen` basieren. Fügt Einträge in das Dropdown-Menü ein, die nicht auf Einstellungs`optionen` basieren.

**public void AddEntries (DropDownEntry enties)**

Fügt Einträge in das Dropdown-Menü ein, die auf Einstellungsoptionen basieren. Fügt Einträge in das Dropdown-Menü ein, die nicht auf Einstellungsoptionen basieren.

#### 3.6.1.8. Klasse HighscoreDialog

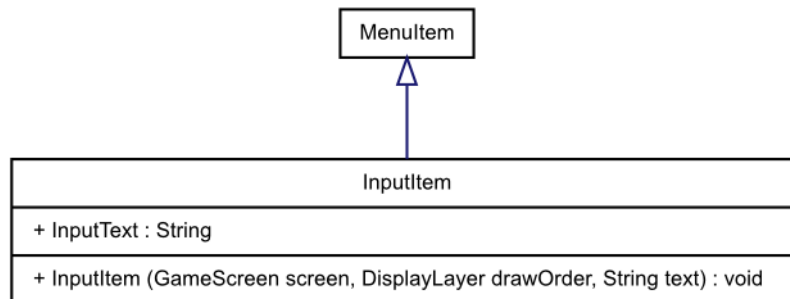


#### Beschreibung:

#### Konstruktoren:

**public** HighscoreDialog (**GameScreen** screen, **DisplayLayer** drawOrder, **Challenge** challenge)

#### 3.6.1.9. Klasse InputItem



#### Beschreibung:

Ein Menüeintrag, der Texteingaben vom Spieler annimmt.

#### Eigenschaften:

**public String** InputText

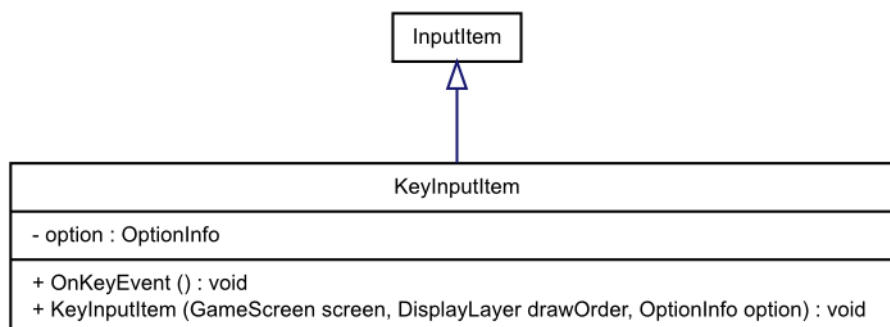
Beinhaltet den vom Spieler eingegebenen Text.

#### Konstruktoren:

**public** InputItem (**GameScreen** screen, **DisplayLayer** drawOrder, **String** text)

Erzeugt ein neues **InputItem**-Objekt und initialisiert dieses mit dem zugehörigen **GameScreen**-Objekt. Zudem sind Angaben zur Zeichenreihenfolge und für evtl. bereits vor-eingetragenen Text Pflicht.

#### 3.6.1.10. Klasse KeyInputItem



### Beschreibung:

Ein Menüeintrag, der einen Tastendruck entgegennimmt und in der enthaltenen Option als Zeichenkette speichert.

### Eigenschaften:

**private** **OptionInfo** option

Die Option in einer Einstellungsdatei.

### Konstruktoren:

**public** KeyInputItem (**GameScreen** screen, **DisplayLayer** drawOrder, **OptionInfo** option)

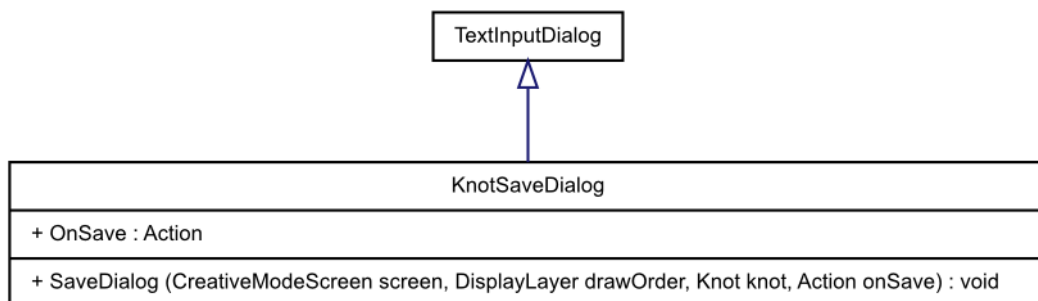
Erzeugt ein neues **CheckBoxItem**-Objekt und initialisiert dieses mit dem zugehörigen **GameScreen**-Objekt. Zudem sind Angaben zur Zeichenreihenfolge und der Eingabe~~option~~ Pflicht.

### Methoden:

**public void** OnKeyEvent ()

Speichert die aktuell gedrückte Taste in der Option.

### 3.6.1.11. Klasse KnotSaveDialog



### Beschreibung:

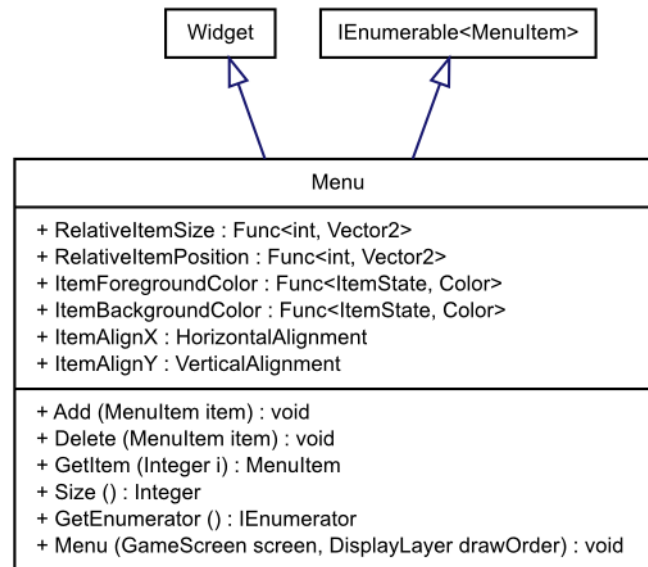
### Eigenschaften:

**public** **Action** OnSave

### Methoden:

**public void** SaveDialog (**CreativeModeScreen** screen, **DisplayLayer** drawOrder, **Knot** knot, **Action** onSave)

### 3.6.1.12. Klasse Menu



#### Beschreibung:

Ein Menü enthält Bedienelemente zur Benutzerinteraktion. Diese Klasse bietet Standardwerte für Positionen, Größen, Farben und Ausrichtungen der Menüeinträge. Sie werden gesetzt, wenn die Werte der Menüeinträge „null“ sind.

#### Eigenschaften:

**public Func<int, Vector2> RelativItemSize**

Die von der Auflösung unabhängige Größe in Prozent.

**public Func<int, Vector2> RelativItemPosition**

Die von der Auflösung unabhängige Position in Prozent.

**public Func<ItemState, Color> ItemForegroundColor**

Die vom Zustand des Menüeintrags abhängige Vordergrundfarbe des Menüeintrags.

**public Func<ItemState, Color> ItemBackgroundColor**

Die vom Zustand des Menüeintrags abhängige Hintergrundfarbe des Menüeintrags.

**public HorizontalAlignment ItemAlignX**

Die horizontale Ausrichtung der Menüeinträge.

**public VerticalAlignment ItemAlignY**

Die vertikale Ausrichtung der Menüeinträge.

#### Konstruktoren:

**public Menu (GameScreen screen, DisplayLayer drawOrder)**

Erzeugt ein neues Menu-Objekt und initialisiert dieses mit dem zugehörigen GameScreen-Objekt. Zudem ist die Angabe der Zeichenreihenfolge Pflicht.

#### Methoden:

**public void Add (MenuItem item)**

Fügt einen Eintrag in das Menü ein. Falls der Menüeintrag „null“ oder leere Werte für Position, Größe, Farbe oder Ausrichtung hat, werden die Werte mit denen des Menüs überschrieben.

**public void Delete (MenuItem item)**

Entfernt einen Eintrag aus dem Menü.

**public MenuItem GetItem (Integer i)**

Gibt einen Eintrag des Menüs zurück.

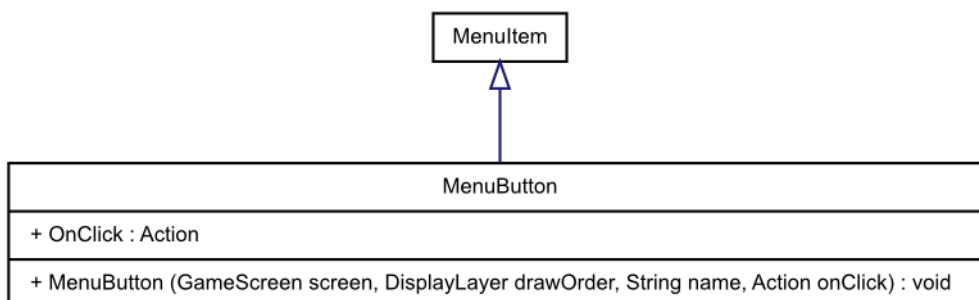
**public Integer Size ()**

Gibt die Anzahl der Einträge des Menüs zurück.

**public IEnumerator GetEnumerator ()**

Gibt einen Enumerator über die Einträge des Menüs zurück.

#### 3.6.1.13. Klasse MenuButton



#### Beschreibung:

Eine Schaltfläche, die eine Zeichenkette anzeigt und auf einen Linksklick reagiert.

#### Eigenschaften:

**public Action OnClick**

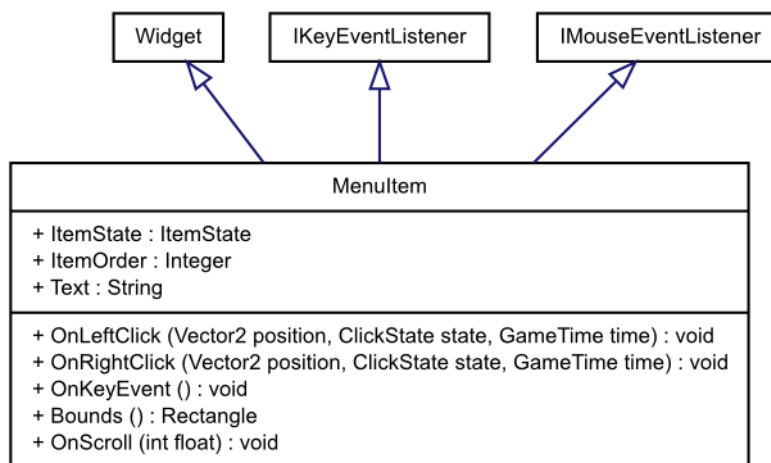
Die Aktion, die ausgeführt wird, wenn der Spieler auf die Schaltfläche klickt.

**Konstruktoren:**

**public MenuButton (GameScreen screen, DisplayLayer drawOrder, String name, Action on-Click)**

Erzeugt ein neues MenuButton-Objekt und initialisiert dieses mit dem zugehörigen GameScreen-Objekt. Zudem sind Angabe der Zeichenreihenfolge, einer Zeichenkette für den Namen der Schaltfläche und der Aktion, welche bei einem Klick ausgeführt wird Pflicht.

#### 3.6.1.14. Klasse MenuItem



**Beschreibung:**

Ein abstrakte Klasse für Menüeinträge, die

**Eigenschaften:**

**public ItemState ItemState**

Gibt an, ob die Maus sich über dem Eintrag befindet, ohne ihn anzuklicken, ob er ausgewählt ist oder nichts von beidem.

**public Integer ItemOrder**

Die Zeichenreihenfolge.

**public String Text**

Der Anzeigetext der Schaltfläche.

**Methoden:**

**public void OnLeftClick (Vector2 position, ClickState state, GameTime time)**

Reaktionen auf einen Linksklick.

```
public void OnRightClick (Vector2 position, ClickState state, GameTime time)
```

Reaktionen auf einen Rechtsklick.

```
public void OnKeyEvent ()
```

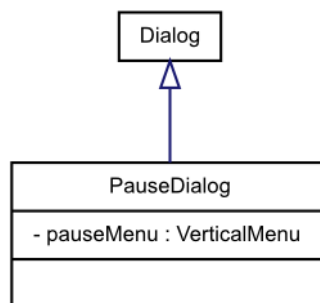
Reaktionen auf Tasteneingaben.

```
public Rectangle Bounds ()
```

Gibt die Ausmaße des Eintrags zurück.

```
public void OnScroll (int float)
```

#### 3.6.1.15. Klasse PauseDialog



##### Beschreibung:

Pausiert ein Spieler im Creative- oder Challenge-Modus das Spiel, wird dieser Dialog über anderen Spielkomponenten angezeigt.

##### Eigenschaften:

```
private VerticalMenu pauseMenu
```

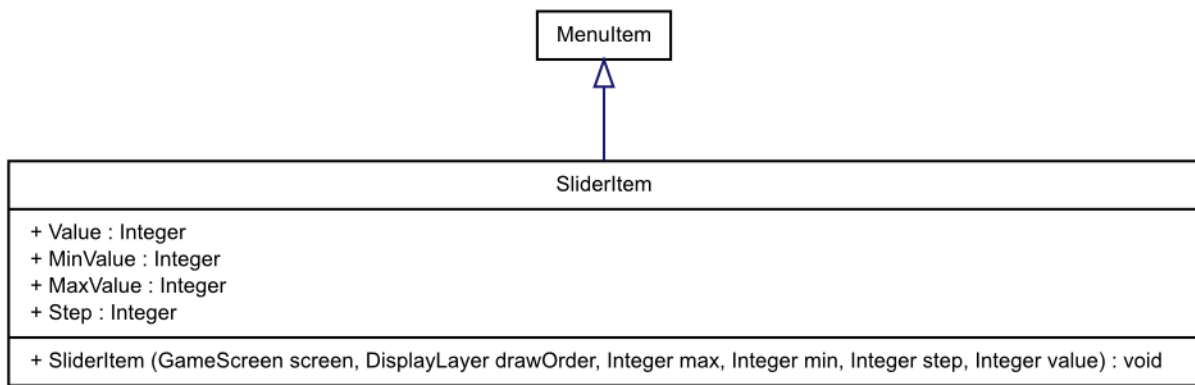
Das Menü, das verschiedene Schaltflächen enthält.

#### 3.6.1.16. Klasse SliderItem

##### Beschreibung:

Ein Menüeintrag, der einen Schieberegler bereitstellt, mit dem man einen Wert zwischen einem minimalen und einem maximalen Wert über Verschiebung einstellen kann.

##### Eigenschaften:



**public Integer Value**

Der aktuelle Wert.

**public Integer MinValue**

Der minimale Wert.

**public Integer MaxValue**

Der maximale Wert.

**public Integer Step**

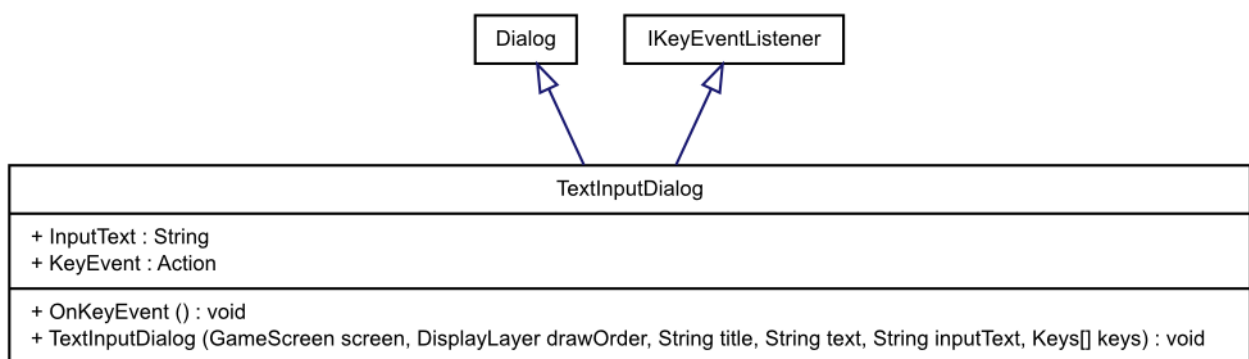
Schrittweite zwischen zwei einstellbaren Werten.

**Konstruktoren:**

**public SliderItem (GameScreen screen, DisplayLayer drawOrder, Integer max, Integer min, Integer step, Integer value)**

Erzeugt eine neue Instanz eines `SliderItem`-Objekts und initialisiert diese mit dem zugehörigen `GameScreen`-Objekt. Zudem ist die Angabe der Zeichenreihenfolge, einem *minimalen* einstellbaren Wert, einem *maximalen* einstellbaren Wert und einem Standardwert Pflicht.

### 3.6.1.17. Klasse TextInputDialog





### Beschreibung:

Ein [Dialog](#), der eine Texteingabe des Spielers entgegennimmt.

### Eigenschaften:

**public String InputText**

Der Text, der durch den Spieler eingegeben wurde.

**public Action KeyEvent**

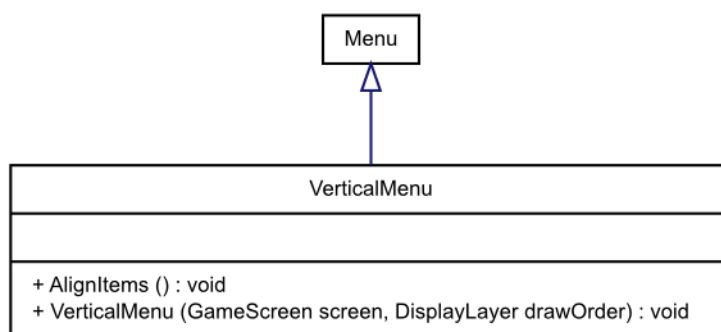
### Konstruktoren:

**public TextInputDialog (GameScreen screen, DisplayLayer drawOrder, String title, String text, String inputText, Keys[] keys)**

### Methoden:

**public void OnKeyEvent ()**

### 3.6.1.18. Klasse VerticalMenu



### Beschreibung:

Ein Menü, das alle Einträge vertikal anordnet.

### Konstruktoren:

**public VerticalMenu (GameScreen screen, DisplayLayer drawOrder)**

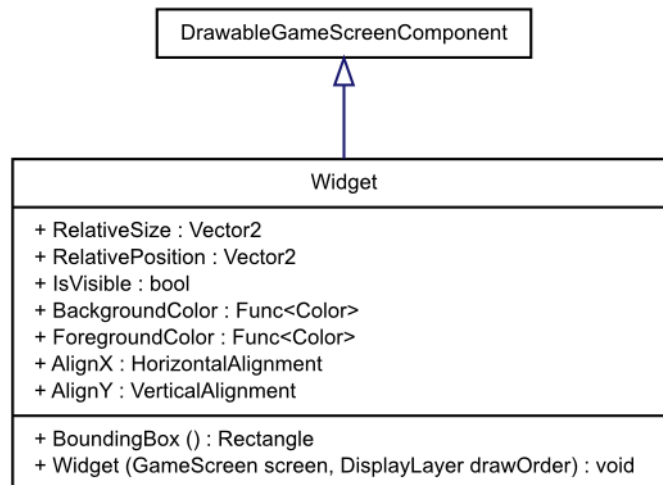
Erzeugt eine neue Instanz eines [VerticalMenu](#)-Objekts und initialisiert diese mit dem zugehörigen [GameScreen](#)-Objekt. Zudem ist die Angaben der Zeichenreihenfolge Pflicht.

## Methoden:

**public void AlignItems ()**

Ordnet die Einträge vertikal an.

### 3.6.1.19. Klasse Widget



## Beschreibung:

Eine abstrakte Klasse, von der alle Element der grafischen Benutzeroberfläche erben.

## Eigenschaften:

**public Vector2 RelativeSize**

Die von der Auflösung unabhängige Größe in Prozent.

**public Vector2 RelativePosition**

Die von der Auflösung unabhängige Position in Prozent.

**public bool IsVisible**

Gibt an, ob das grafische Element sichtbar ist.

**public Func<Color> BackgroundColor**

Die Hintergrundfarbe.

**public Func<Color> ForegroundColor**

Die Vordergrundfarbe.

**public HorizontalAlignment AlignX**

Die horizontale Ausrichtung.

**public VerticalAlignment AlignY**

Die vertikale Ausrichtung.

#### Konstruktoren:

**public Widget (GameScreen screen, DisplayLayer drawOrder)**

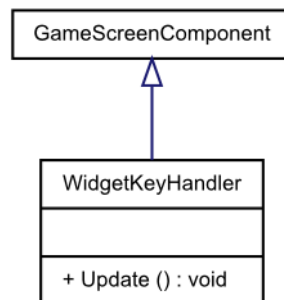
Erstellt ein neues grafisches Benutzerschnittstellenelement in dem angegebenen Spielzustand mit der angegebenen Zeichenreihenfolge.

#### Methoden:

**public Rectangle BoundingBox ()**

Die Ausmaße des grafischen Elements

#### 3.6.1.20. Klasse WidgetKeyHandler



#### Beschreibung:

Ein Inputhandler, der Tastatureingaben auf Widgets verarbeitet.

#### Methoden:

**public void Update ()**

Wird für jeden Frame aufgerufen.

#### 3.6.1.21. Klasse WidgetMouseHandler

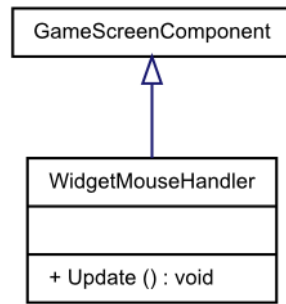
#### Beschreibung:

Ein Inputhandler, der Mauseingaben auf Widgets verarbeitet.

#### Methoden:

**public void Update ()**

Wird für jeden Frame aufgerufen.



### 3.6.2. Schnittstellen

### 3.6.3. Enumerationen

#### 3.6.3.1. Enumeration HorizontalAlignment

##### Beschreibung:

Eine horizontale Ausrichtung.

##### Eigenschaften:

**Left** = 0

Links.

**Center** = 1

Mittig.

**Right** = 2

Rechts.

#### 3.6.3.2. Enumeration ItemState

##### Beschreibung:

Der Zustand eines Menüeintrags.

##### Eigenschaften:

**Selected** = 1

Ausgewählt.

**Hovered** = 2

Die Maus wurde direkt über den Menüeintrag navigiert und verweilt dort.

**None** = 0

Ein undefinierter Zustand.

### 3.6.3.3. Enumeration VerticalAlignment

#### Beschreibung:

Die vertikale Ausrichtung.

#### Eigenschaften:

**Top** = 1

Oben.

**Center** = 0

Mittig.

**Bottom** = 2

Unten.

## 4. Abläufe

### 4.1. Sequenzdiagramme

#### 4.1.1. Ende einer Challenge

Der in der Challenge vom Spieler bearbeitete Knoten vom Typ `Knot` ruft bei Kantenänderungen durch den Spieler eine Methode auf, die ihm beim Hinzufügen zum `ChallengeModeScreen` zugewiesen wurde. Diese prüft die beiden Knoten auf Gleichheit, indem sie die Knoten sich über die Methode `Equals(Knot)` vergleichen lässt. Sind beide `Knoten` gleich, wird die benötigte Zeit ermittelt und ein neuer `Dialog` erstellt, der die Zeit anzeigt und nach einem Benutzernamen fragt.

Dazu wird aus `Options.Default` der Benutzernamen abgefragt und als Standardwert eingetragen. Wird ein neuer Benutzername eingegeben, wird dieser auch in `Options.Default` geändert. Mit Enter bestätigt wird eine zugewiesene Methode aufgerufen, die Benutzername und Zeit dem `Challenge`-Objekt mit `AddToHighscore(name, time)` übergibt, was die Werte richtig einordnet und gegebenenfalls (wenn die Zeit gut genug ist) speichert.

Daraufhin öffnet diese Methode einen neuen `Dialog`, der, soweit vorhanden, die besten zehn Einträge sowie zwei Buttons anzeigt, von denen einer die `Challenge` neu startet und ein anderer den `StartScreen` aufruft. Obwohl nur die besten zehn Highscore-Einträge angezeigt werden, sind trotzdem alle bisher hinzugefügten Einträge sowohl in dem `Challenge`-Objekt als auch in dem Speicherformat der Challenge enthalten, da sie zur Berechnung der durchschnittlichen Zeit benötigt werden, die u.A. im `ChallengeStartScreen` zu jeder Challenge angezeigt werden kann.

#### 4.1.2. Screenwechsel von dem `CreativeMainScreen` in den `CreativeLoadScreen`

Wenn man in dem Menü, das durch den `CreativeMainScreen` dargestellt wird, auf den Button "Load" klickt, wird der Wechsel zu `CreativeLoadScreen` eingeleitet. Dabei wird ein neues `CreativeLoadScreen` erstellt und oben auf den Stack in `Knot3Game` gelegt, in dem die Spielstände abgespeichert werden.

Beim nächsten `Update()`-Call des XNA-Frameworks wird der `PostProcessingEffect` von `CreativeLoadScreen` für ein Überblenden der Rendertargets auf `FadeEffect` gesetzt und der `CreativeMainScreen` wird deaktiviert, d.h. alle `IGameScreenComponents` von `CreativeMainScreen` werden aus der Liste der aktuell in XNA registrierten `IGameScreenComponent` gelöscht. Außerdem wird die `BeforeExit()`-Methode von `CreativeMainScreen` aufgerufen, in der eventuell anfallende Aufräumarbeiten ausgeführt werden können.

Dann wird `CreativeLoadScreen` aktiviert und dessen `Entered()`-Methode aufgerufen, die dessen `IGameScreenComponents` mittels der in `GameScreen` implementierten `AddGameComponents()`-Methode in die `IGameComponent`-Liste des XNA-Frameworks einträgt. Diese Operation ist rekursiv, d.h. die über die `SubComponents()`-Methode jedes `IGameScreenComponents` ermittelten Unterkomponenten werden von `AddGameComponents()` ebenfalls erfasst.

In diesem Beispiel wäre eine direkt von `CreativeLoadScreen` hinzugefügte Komponente das Spielstand-Menü, das von der Klasse `VerticalMenu` repräsentiert wird. Dieses gibt über die `SubComponents()`-Methode alle in dem Menü enthaltenen `MenuItems` zurück, damit diese von `AddGameComponents()` ebenfalls als `IGameScreenComponent` registriert und dargestellt werden können.

Daraufhin durchsucht `CreativeLoadScreen` das Spielstand-Verzeichnis und lädt die Metadaten der Speicherstände aller zwischengespeicherten Knoten. Noch nicht zwischengespeicherte Knoten werden auf Gültigkeit überprüft und wenn für Gültig befunden in den Zwischenspeicher geschrieben als Hash.

Anschließend werden die gefundenen gültigen Spielstände in ein Menü eingetragen, welches dann beim nächsten `Draw()`-Call von XNA auf den Bildschirm gezeichnet wird. Dieses Menü ist ein `VerticalMenu`, welches einen `MenuButton` für jeden gültigen gefundenen Spielstand enthält. Durch einen Klick auf den jeweiligen Button startet man mit dem Spielstand im `CreativeModeScreen`.

#### 4.1.3. Erstellen der 3D-Objekte nach durchgeführten Kantenverschiebungen

Sowohl im `CreativeModeScreen` als auch im `ChallengeModeScreen` wird ein `Knoten` vom Spieler bearbeitet, auf Basis dessen mittels einer Instanz von `KnotRenderer` 3D-Modell-Objekte, die von `GameModel` und `IGameObject` erben, erstellt werden. Dazu wird die `OnEdgesChanged()`-Methode der `KnotRenderer`-Klasse dem `EdgesChanged()`-Delegate des `Knot`-Objekts zugewiesen, sodass der `KnotRenderer` bei allen Kantenänderungen benachrichtigt wird.

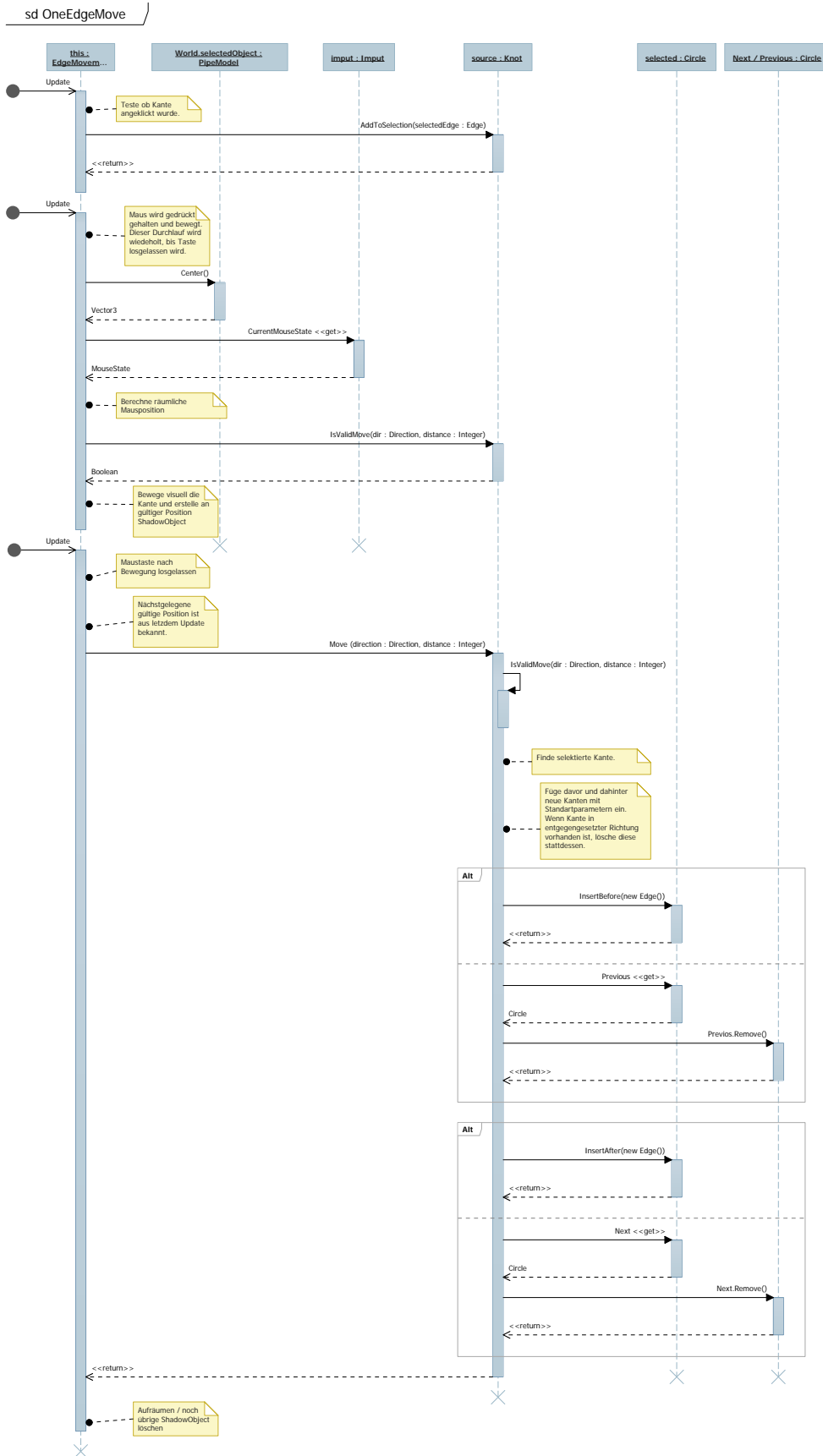
Wurde eine Kantenänderung durchgeführt und `OnEdgesChanged()` auf dem `KnotRenderer` aufgerufen, so leert dieser zunächst seine drei Listen mit den bisher vorhandenen `PipeModels` (als Röhren dargestellte Kanten), `NodeModels` (Übergänge zwischen zwei Kanten) und `ArrowModels` (Pfeile an selektierten Kanten).

Nun ruft der `KnotRenderer` die `OnEdgesChanged()`-Methode auf einem von ihm gehaltenen `NodeMap`-Objekt auf, um für alle Kanten des `Knots` eine Zuordnung zwischen dem `Edge`-Objekt und der jeweiligen Position im 3D-Raster zu erstellen, welche von `Node` repräsentiert wird. Ein Objekt der Klasse `Node` hält dabei einen Rasterpunkt, wobei der Abstand zwischen den zwei `Nodes`, an denen eine Kanten beginnt und endet, den normalisierten Abstand 1 hat. Die Konvertierung in 3D-Koordinaten in Form von `Vector3`-Objekten erfolgt später in den über die `ToVector()`-Methode der `Node`-Objekte.

Daraufhin werden für alle drei Kategorien neue Modell-Informationen erstellt, welche von Objekten der Klassen `PipeModelInfo`, `NodeModelInfo` und `ArrowModelInfo` repräsentiert werden. Dabei wird jeweils ein als Attribut der Klasse `KnotRenderer` vorhandenes `ModelFactory`-Objekt verwendet, um bereits vor der Verschiebeaktion vorhandene Modell-Objekte wiederzuverwenden und nur bisher nicht vorhandene Modell-Objekte neu zu erstellen. Die Klasse `ModelFactory` implementiert dabei einen Cache, der über eine Hashtabelle beliebige `GameModelInfo`-Objekte daraus generierten `GameModel`-Objekten zuordnet. Die Erstellung neuer Objekte erfolgt dabei durch ein Delegate, das bei der Erstellung einer `ModelFactory` ihrem Konstruktor übergeben wird.

Die Erstellung der Modell-Objekte erfolgt wie nachfolgend beschrieben.

## 4.1.4. Verschiebung einer Kante





**Kanten** Um für alle aktuell im **Knoten** vorhandenen Kanten eine jeweils Röhre (**PipeModel**) zu erstellen, wird über die im **Knot**-Objekt enthaltenen **Edge**-Objekte iteriert. Für jede **Edge** wird ein **PipeModelInfo** erstellt, dem im Konstruktor der **Knoten**, die **Edge** sowie die **NodeMap** übergeben werden. Die **PipeModelInfo**-Instanz berechnet daraus dann die Position der Röhre sowie weitere zur Darstellung notwendige Daten. Mit Hilfe dieser Modellinformationen werden dann jeweils über die zuständige **ModelFactory** die bereits vorhandenen **PipeModel**-Objekte abgefragt bzw. neu erstellt und der **PipeModel**-Liste des **KnotRenderers** zugewiesen.

**Kantenübergänge** Da die 3D-Modelle, die an den Kantenübergängen dargestellt werden, sich abhängig von der Anzahl der an dem Rasterpunkt angrenzenden Kanten und ihrer Positionen unterscheiden, wird für die Bestimmung der 3D-Modelle zunächst eine Zuordnung zwischen den Rasterpunkten (**Node**) und einer Liste von einzelnen Kantenübergängen (**IJunction**) durch Iteration über die **Edges** bzw. den zugehörigen **Nodes** erstellt. Das Interface **IJunction** enthält Informationen über die eingehende und ausgehende **Edge** und wird von der Klasse **NodeModelInfo** implementiert, die im Konstruktor diese beiden Objekte sowie eine Referenz auf die **NodeMap** erwartet.

Danach wird für alle Rasterpunkte entschieden, wie viele und welche 3D-Modelle zur Darstellung des Kantenübergangs benötigt werden. Dabei werden zwischen null und drei **NodeModelInfo**-Objekte mit teilweise unterschiedlichen Modell-Dateien, Drehungen und Skalierungen erstellt, aus denen mit Hilfe der für **NodeModel**-Objekte zuständigen **ModelFactory**-Instanz die entsprechenden **NodeModel**-Objekte abgefragt bzw. erstellt werden. Diese werden dann, analog zu dem Vorgehen bei den **PipeModel**-Objekten, in die **NodeModel**-Liste des **KnotRenderers** eingefügt.

**Pfeile** Zur Erstellung der Modellinformationen der **ArrowModels** wird zunächst über die Liste der selektierten Kanten iteriert, um die Kante zu finden, die sich in der Mitte der selektierten Kantenfolge befindet. An dieser Position soll, unter Umständen um einen noch zu bestimmenden Abstand in Richtung der Kameraposition verschoben, für jede gültige Richtung, in welche die selektierten Kanten verschoben werden können, ein Pfeil angezeigt werden. Dabei wird die Distanz der Verschiebung nicht in der Berechnung berücksichtigt; es reicht aus, wenn mindestens eine Distanz existiert, in die dieser Zug gültig wäre.

Für die ermittelten gültigen Pfeilrichtungen wird jeweils ein **ArrowModelInfo**-Objekt erstellt, das zur Abfrage und Erstellung von **ArrowModel**-Objekten über die zugehörige **ModelFactory** verwendet wird. Die erstellten **ArrowModel**-Objekte werden in die **ArrowModel**-Liste des **KnotRenderers** eingefügt. Falls keine Kanten selektiert sind, ist die Liste der Pfeilmodelle leer.

Dieses Erstellen von **ArrowModel**-Objekten bei einer Selektion von Kanten kann in den Einstellungen des Spiels über eine Option deaktiviert werden. In diesem Fall werden Verschiebeoperationen nur direkt durch Anklicken und Ziehen an den Kanten ausgeführt; diese Möglichkeit ist dem Spieler auch gegeben, wenn die Pfeil-Anzeige aktiviert ist.

Damit die **ArrowModel**-Objekte nicht nur nach Verschiebe-Aktionen, sondern auch nach einer Änderung der Kantenauswahl mit dem beschriebenen Algorithmus neu erstellt werden, wird der pfeilbezogene Teil des beschriebenen Algorithmus in eine private Methode ausgelagert. Diese wird einerseits nach dem Erstellen der **PipeModel**- und **NodeModel**-Objekte über die **OnEdgesChanged()**-Methode aufgerufen, andererseits wird sie bei der Zuweisung eines **Knotens** zur aktuellen **KnotRenderer**-Instanz auch in das **SelectionChanged()**-Delegate des **Knoten** eingefügt, um ebenfalls bei allen Auswähleränderungen benachrichtigt zu werden.

# Klassenindex

## Core

- Angles3, 8
- BooleanOptionInfo, 9
- Camera, 9
- ConfigFile, 11
- DistinctOptionInfo, 12
- DrawableGameScreenComponent, 13
- FileUtility, 14
- GameScreen, 14
- GameScreenComponent, 16
- IGameScreenComponent, 21
- IKeyEventListener, 22
- IMouseEventListener, 23
- InputManager, 17
- Localizer, 18
- MousePointer, 18
- OptionInfo, 19
- Options, 20
- World, 20

## GameObjects

- ArrowModel, 26
- ArrowModelInfo, 27
- EdgeMovement, 28
- GameModel, 29
- GameModelInfo, 30
- GameObjectDistance, 31
- GameObjectInfo, 32
- IGameObject, 42
- IJunction, 43
- KnotInputHandler, 33
- KnotRenderer, 33
- ModelFactory, 35
- ModelMouseHandler, 36
- NodeModel, 37
- NodeModelInfo, 37
- PipeModel, 38
- PipeModelInfo, 39
- ShadowGameModel, 40
- ShadowGameObject, 41

## KnotData

- Challenge, 62
- ChallengeFileIO, 63
- ChallengeMetaData, 64
- Edge, 65
- IChallengeIO, 74

- IKnotIO, 74
- Knot, 65
- KnotFileIO, 68
- KnotMetaData, 69
- KnotStringIO, 70
- Node, 71
- NodeMap, 72
- PrinterIO, 73

#### RenderEffects

- CelShadingEffect, 56
- FadeEffect, 57
- IRenderEffect, 61
- RenderEffect, 58
- RenderEffectStack, 59
- StandardEffect, 60

#### Screens

- AudioSettingsScreen, 43
- ChallengeCreateScreen, 44
- ChallengeModeScreen, 45
- ChallengeStartScreen, 46
- ControlSettingsScreen, 47
- CreativeLoadScreen, 48
- CreativeMainScreen, 48
- CreativeModeScreen, 49
- CreditsScreen, 50
- GraphicsSettingsScreen, 51
- Knot3Game, 52
- MenuScreen, 53
- ProfileSettingsScreen, 54
- SettingsScreen, 54
- StartScreen, 55
- TutorialChallengeModeScreen, 56

#### Widgets

- CheckBoxItem, 76
- ColorPicker, 76
- ColorPickItem, 77
- ConfirmDialog, 78
- Dialog, 79
- DropDownEntry, 80
- DropDownMenuItem, 80
- HighscoreDialog, 81
- InputItem, 82
- KeyInputItem, 82
- KnotSaveDialog, 83
- Menu, 84
- MenuButton, 85
- MenuItem, 86
- PauseDialog, 87
- SliderItem, 87
- TextInputDialog, 88
- VerticalMenu, 89

Widget, 90  
WidgetKeyHandler, 91  
WidgetMouseHandler, 91

## A. Glossar

Bounding-Frustum	Ein Frustum, das z.B. aus der View- und Projection-Matrix berechnet werden kann und dabei hilft zu bestimmen, ob ein 3D-Modell in dem für den Spieler sichtbaren Bereich des 3D-Raums liegt. <sup>1</sup>
Einstellungsdatei	Eine Datei im INI-Format, die die Einstellungen des Spiels speichert. Die zentrale Einstellungsdatei von Knot <sup>3</sup> liegt unter Windows in einem Unterordner des Dokumente-Ordners des aktuellen Windows-Benutzers, beispielsweise in <i>C:\Users\Name\Documents\Knot3\config.ini</i> . Wird Knot <sup>3</sup> unter Linux in Verbindung mit der offenen XNA-Implementierung Monogame gespielt, wird eine Datei aus dem Unterverzeichnis ".knot3" des Homeverzeichnisses ausgelesen, etwa <i>/home/name/.knot3/config.ini</i> .
Inputhandler	Ein Inputhandler ist eine Möglichkeit um Benutzereingaben zu verarbeiten. Der Inputhandler stellt beispielsweise Informationen über die Position der Maus bereit.
Rendereffekt	Ein Effekt, der auf 3D-Modelle oder Sprites angewandt wird und durch Shader realisiert wird.
Rendertarget	Ein Rendertarget ist ein Pixel-Buffer.
Spielkomponentenliste	Die Sammlung von GameComponent-Elementen, die im Besitz des Spiels sind. Quelle: <a href="http://msdn.microsoft.com/de-de/library/microsoft.xna.framework.game.components(v=xnagamestudio.40).aspx">http://msdn.microsoft.com/de-de/library/microsoft.xna.framework.game.components(v=xnagamestudio.40).aspx</a>
Sprite	Ein Sprite (engl. unter anderem für ein Geistwesen, Kobold) ist ein Grafikobjekt, das von der Grafikhardware über das Hintergrundbild bzw. den restlichen Inhalt der Bildschirmanzeige eingeblendet wird. Die Positionierung wird dabei komplett von der Grafikhardware erledigt. Quelle: <a href="http://de.wikipedia.org/wiki/Sprite_(Computergrafik)">http://de.wikipedia.org/wiki/Sprite_(Computergrafik)</a>
Spritestapel	Ermöglicht das Zeichnen einer Gruppe von Sprites mithilfe derselben Einstellungen. Quelle: <a href="http://msdn.microsoft.com/de-de/library/microsoft.xna.framework.graphics.spritebatch(v=xnagamestudio.40).aspx">http://msdn.microsoft.com/de-de/library/microsoft.xna.framework.graphics.spritebatch(v=xnagamestudio.40).aspx</a>

---

<sup>1</sup>Genauere Definition: [http://msdn.microsoft.com/de-de/library/microsoft.xna.framework.boundingfrustum\(v=xnagamestudio.40\).aspx](http://msdn.microsoft.com/de-de/library/microsoft.xna.framework.boundingfrustum(v=xnagamestudio.40).aspx)

Widget

Ein Widget ist eine Komponente eines grafischen Fenstersystems. Das Widget besteht zum einen aus dem Fenster, einem sichtbaren Bereich, der Maus- und/o-der Tastaturereignisse empfängt, und zum anderen aus dem nicht sichtbaren Objekt, das den Zustand der Komponente speichert und über bestimmte Zeichenoperationen den sichtbaren Bereich verändern kann.

Quelle: <http://de.wikipedia.org/wiki/Widget>

XNA-Content-Pipeline

Die XNA Game Studio-Inhalts-Pipeline ist eine Gruppe von Prozessen, die beim Build eines Spiels mit Grafikassets angewendet werden. Der Prozess beginnt mit einem Grafikasset, das ursprünglich als Datei vorliegt. Dieses Grafikasset wird dann in Daten transformiert, die über die XNA Framework-Klassenbibliothek abgerufen und in einem XNA Game Studio-Spiel verwendet werden können.

Quelle: [http://msdn.microsoft.com/de-de/library/bb447745.aspx#fbid=cm5w\\_zXCQc2](http://msdn.microsoft.com/de-de/library/bb447745.aspx#fbid=cm5w_zXCQc2)