Zellomat3D

Design Applikation

Projekt:	3D Cellular Automata Simulator – Diplomarbeit – SS/2005	
Auftraggeber:	Hochschule Rapperswil HSR	
Betreuer:	Eduard Glatz – Prof. Dipl. Ing. ETH	eglatz@hsr.ch
Mitarbeiter:	Michael Florin	loop@loop.li
	Andreas Weinmann	a.weinmann@gmx.ch
Ablage:	DesignApplikation - 11042005.doc	

Inhaltsverzeichnis

1. Einführung

Zweck

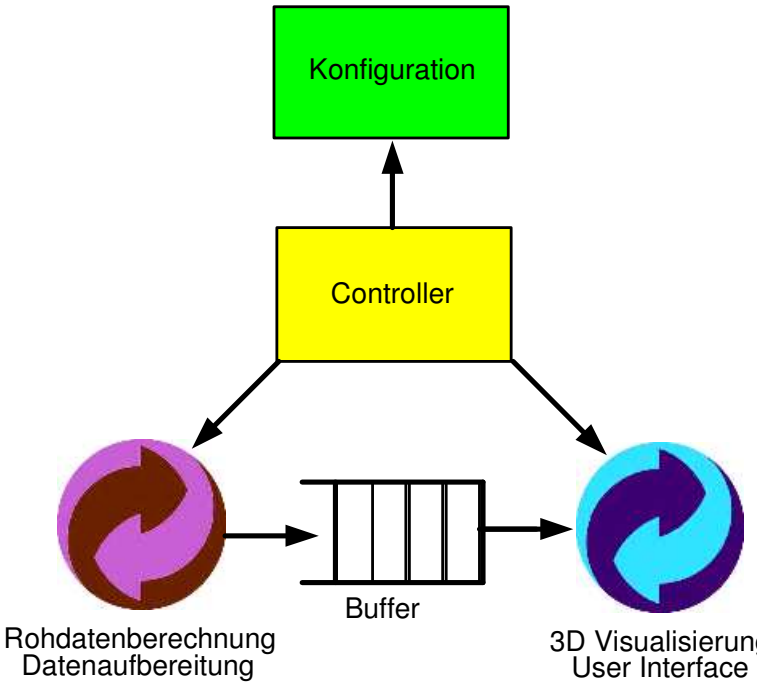
Dieses Dokument beschreibt die Architektur und das Design der Applikation. Es erleichtert einem Entwickler den Einstieg in den Programmcode, damit dieser die Funktionsweisen der Applikation nachvollziehen kann. Des Weiteren ist daraus ersichtlich, warum dieses Design und diese Architektur verwendet wurden.
Zu Beginn wird die Architektur der Applikation beschrieben, danach folgen das Klassendiagramm, die Klassen im Detail und zum Schluss einige Sequenz- und Kollaborationsdiagramme, die den dynamischen Aspekt der Applikation dokumentieren.

Gültigkeitsbereich

Dieses Dokument gilt für die Diplomarbeit "Zellomat3D", welche im SS/2005 an der Hochschule Rapperswil HSR durchgeführt wurde.

2. Architekturmodel Zellomat3D

Architektur



Beschreibung

Die Applikation kann grob in zwei getrennte Kreisläufe unterteilt werden: Die Berechnung der Rohdaten und derern grafische Aufbereitung sowie die Visualisierung der errechneten Daten. Dazwischen fungiert ein Buffer für die Daten als Trennung zwischen den zwei Kreisläufen.
Der Controller ist für die Steuerung der ganzen Applikation verantwortlich. Das Konfigurationsmodul ist für das Parsen resp. Initialisieren der Daten, die aus den Beschreibungsdateien der ZA gelesen werden, zuständig.

Kopplungen

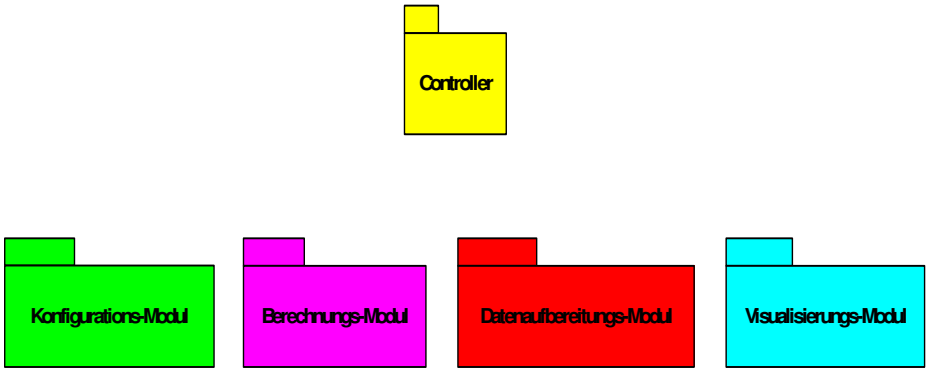
Die einzelnen Teilbereiche sind untereinander lose gekoppelt, einzig der Controller hat feste Verbindungen zu allen Programmmodulen. Innerhalb der zwei Kreisläufe ist unter den einzelnen Teilmodulen eine hohe Kopplung vorhanden.

Kreislauf Berechnung: Um eine möglichst performante Berechnung zu erreichen, sind die einzelnen Module innerhalb des Kreislaufes eng miteinander gekoppelt.

Kreislauf Visualisierung: Auch hier sind einzelne Teile eng miteinander verbunden. Dies weil die 3D-Darstellung innerhalb eines Applikationsfensters geschieht und somit viele direkte Abhängigkeiten zwischen diesen zwei Teilen bestehen.

3. Klassendiagramm Zellomat3D

Struktur der Klassenkategorien



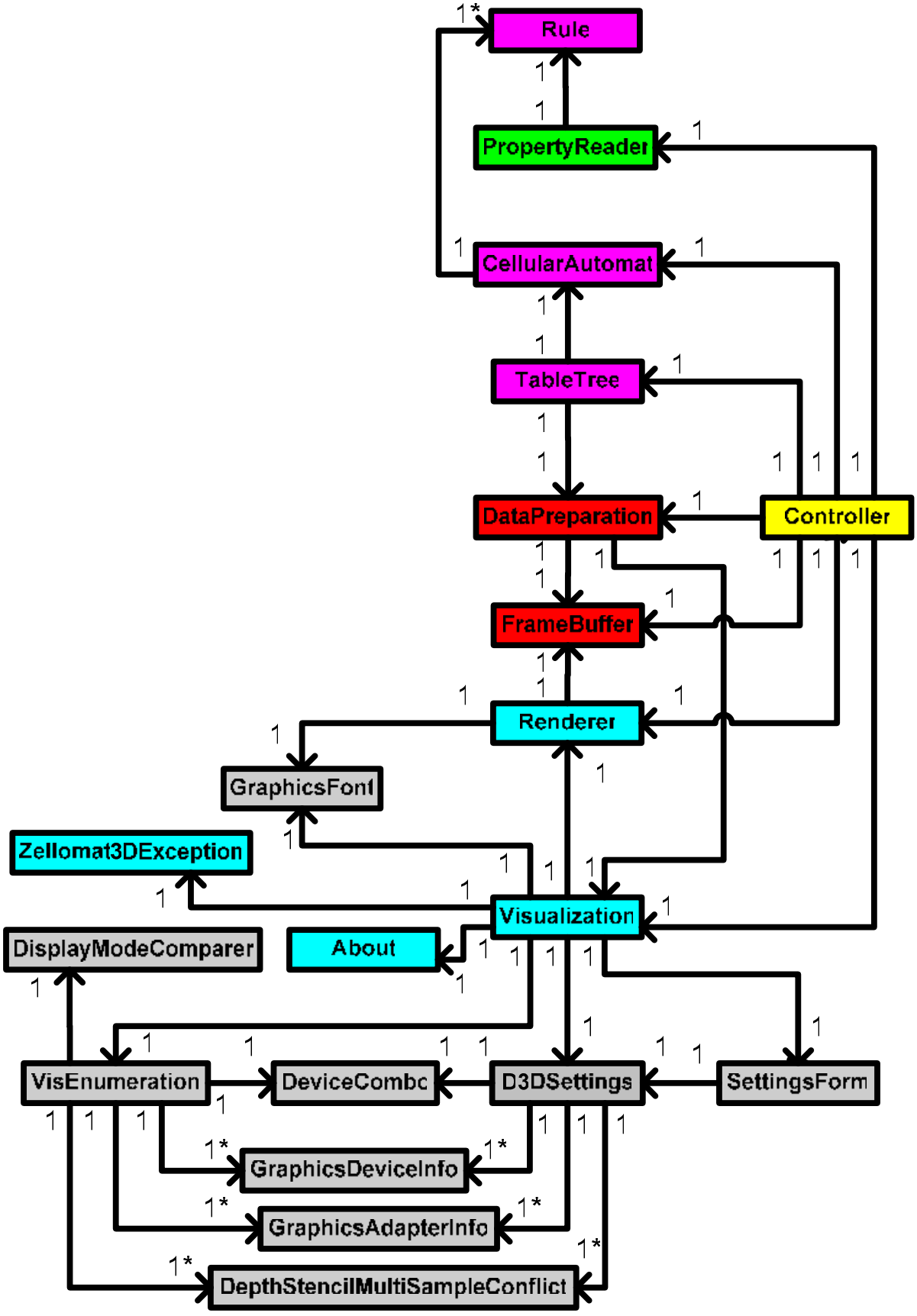
Im nachfolgenden detaillierten Klassendiagramm zeigen die Farben der einzelnen Klassen ihre Zugehörigkeit zu den fünf logischen Bestandteilen.
Die FrameBuffer-Klasse könnte auch Teil des Visualisierungs-Modul sein. Sie entkoppelt den Berechnungskreislauf von der Visualisierung. Die Verbindung zwischen der DataPreparation- und Visualization-Klasse scheint nicht in die Architektur zu passen. Sie ist aber zwingend notwendig, denn die DataPreparation braucht das Objekt der Grafikkarte (Device), um die Frames darauf anzupassen.
Das Device kann auch nicht übergeben werden, denn bei jeder Änderung der Grafikkarteneinstellungen (Bildschirmauflösung, Rechengenauigkeit, Fenstergrösse usw.) wird das alte Device-Objekt zerstört und anschliessend ein Neues erzeugt.

Helferklassen

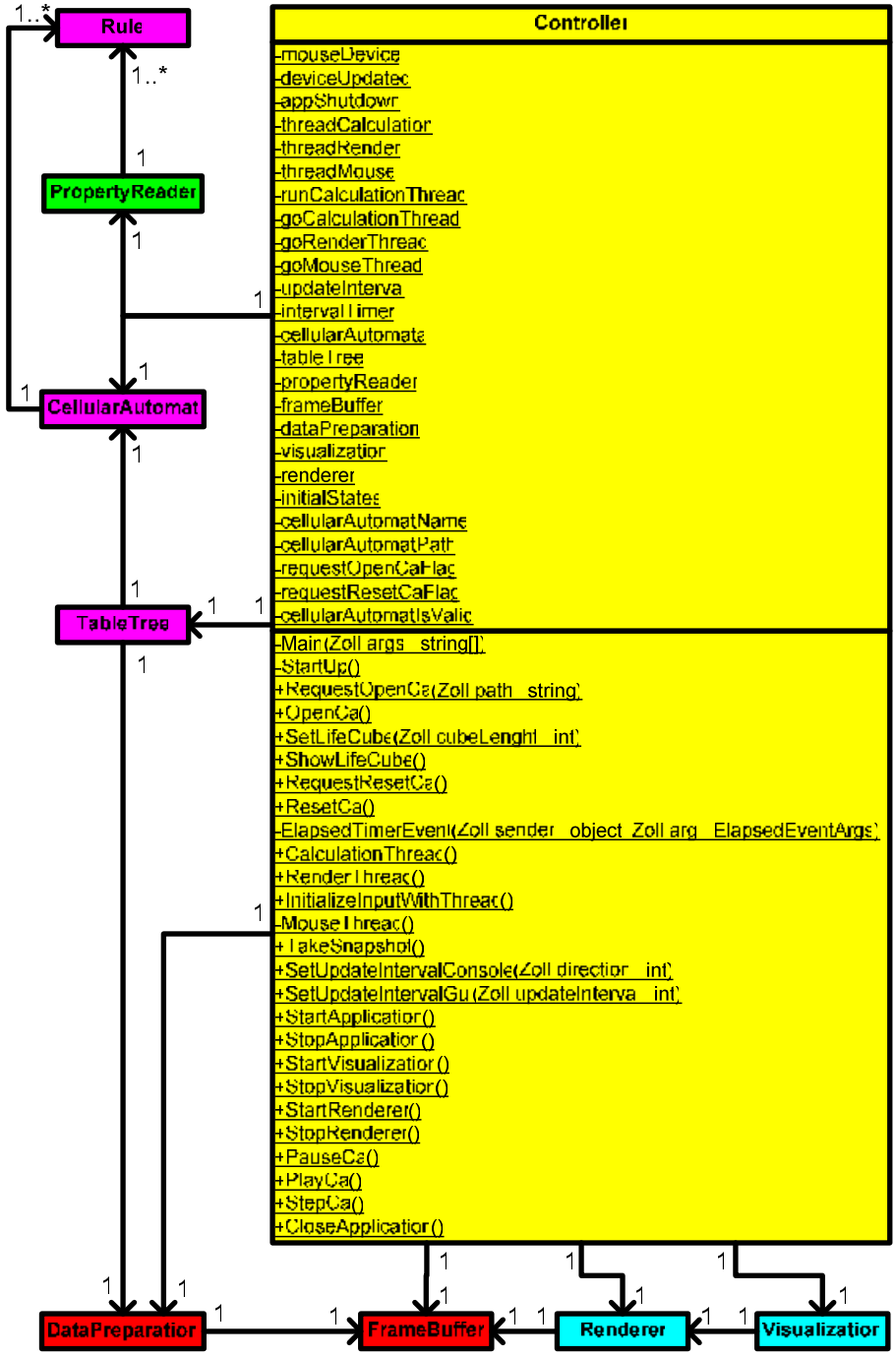
Zur Unterstützung der DirectX 3D Visualisierung sind ein paar Helferklassen[1] in das Programm miteingebunden worden. Diese Klassen sind im Allgemeinen verantwortlich für das Handling des Grafikkarten Device. Mit Hilfe dieser Klassen wird das Programm um die Möglichkeit der Konfiguration verschiedenster Parameter der 3D-Visualisierung wie zum Beispiel die Auflösung, Refreshrate oder allgemeine Parameter, die die Erzeugung der 3D-Frames beeinflussen.

[1] Stammen aus dem Managed DirectX 9.0 SDK

Klassendiagramm



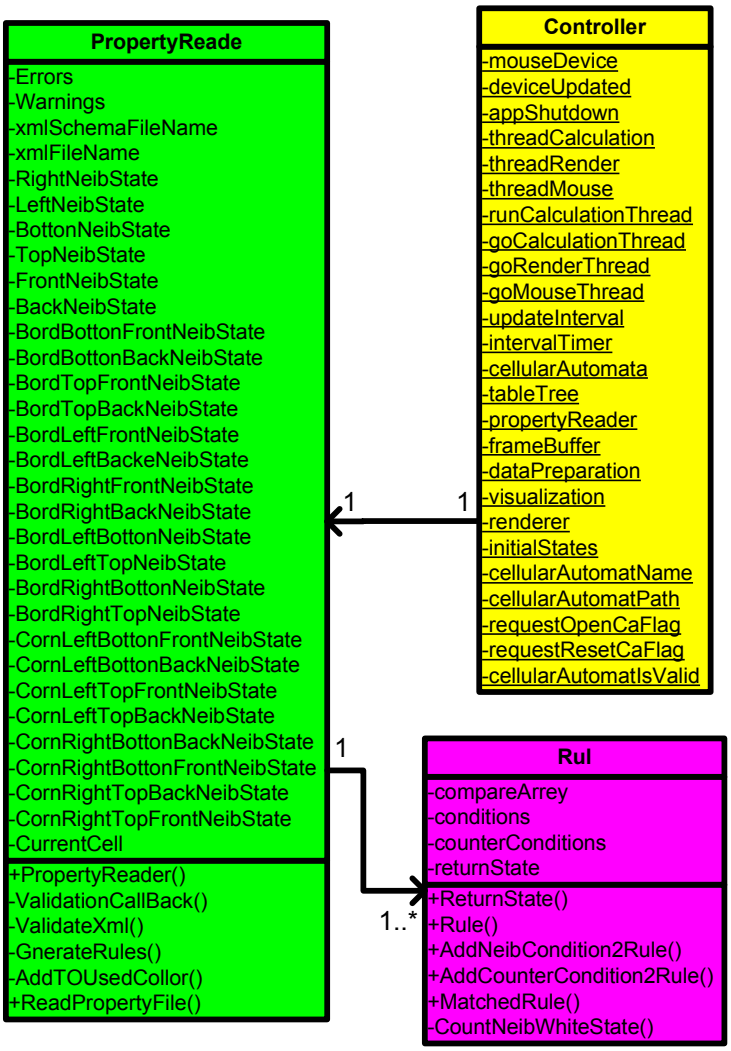
Modul Controller



Beschreibung

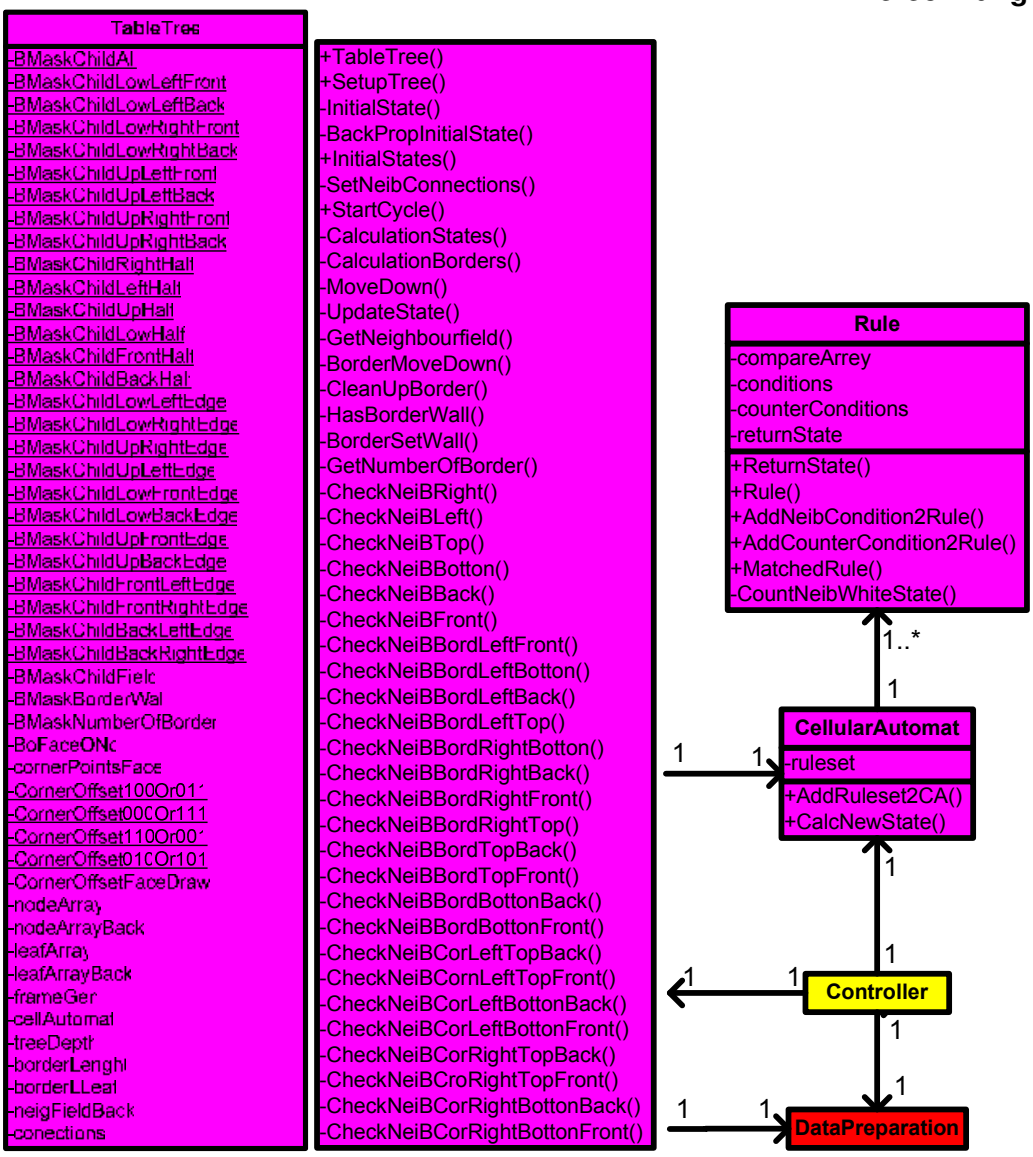
Wie hier ersichtlich ist, übernimmt der Controller eine zentrale Position in der Klassenhierarchie. Er steuert den modulübergreifenden Programmablauf und verarbeitet Benutzereingaben, welche er per Events erhält.
Der Controller erstellt beim Programmstart alle Objekte, welche nicht reine Bestandteile eines Moduls sind und dessen Verwaltung obliegen.
Ebenso ist er auch für das Handling der verschiedenen Threads verantwortlich.

Konfigurations-Modul



Das Konfigurations-Modul validiert und parst das ZA spezifische XML- File. Es generiert sogleich die Regeln und leitet alle eingelesenen Parameter dem Controller zurück.

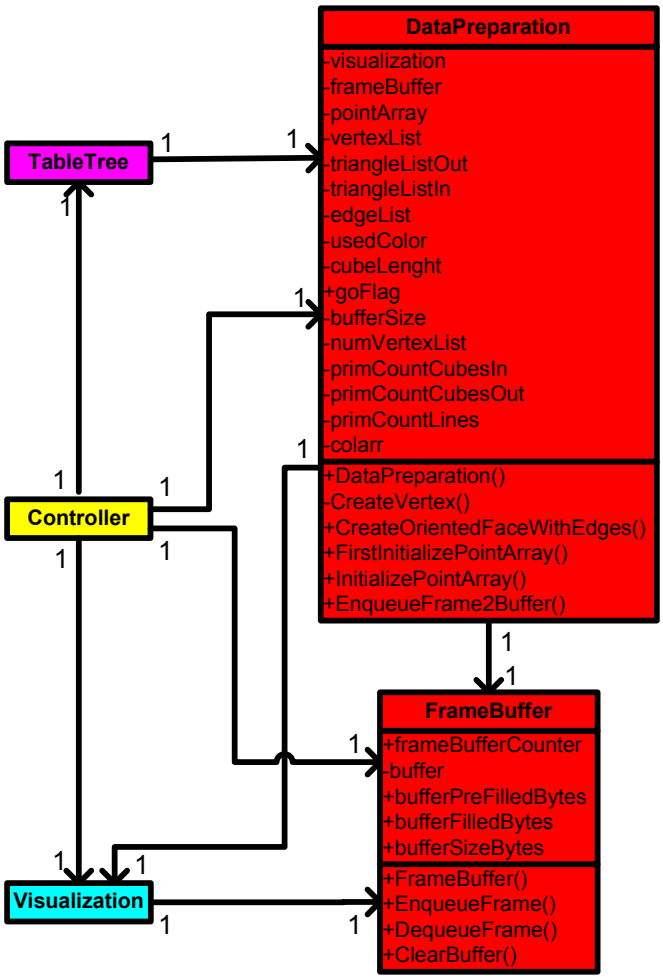
Modul Rohdaten-Berechnung



Beschreibung

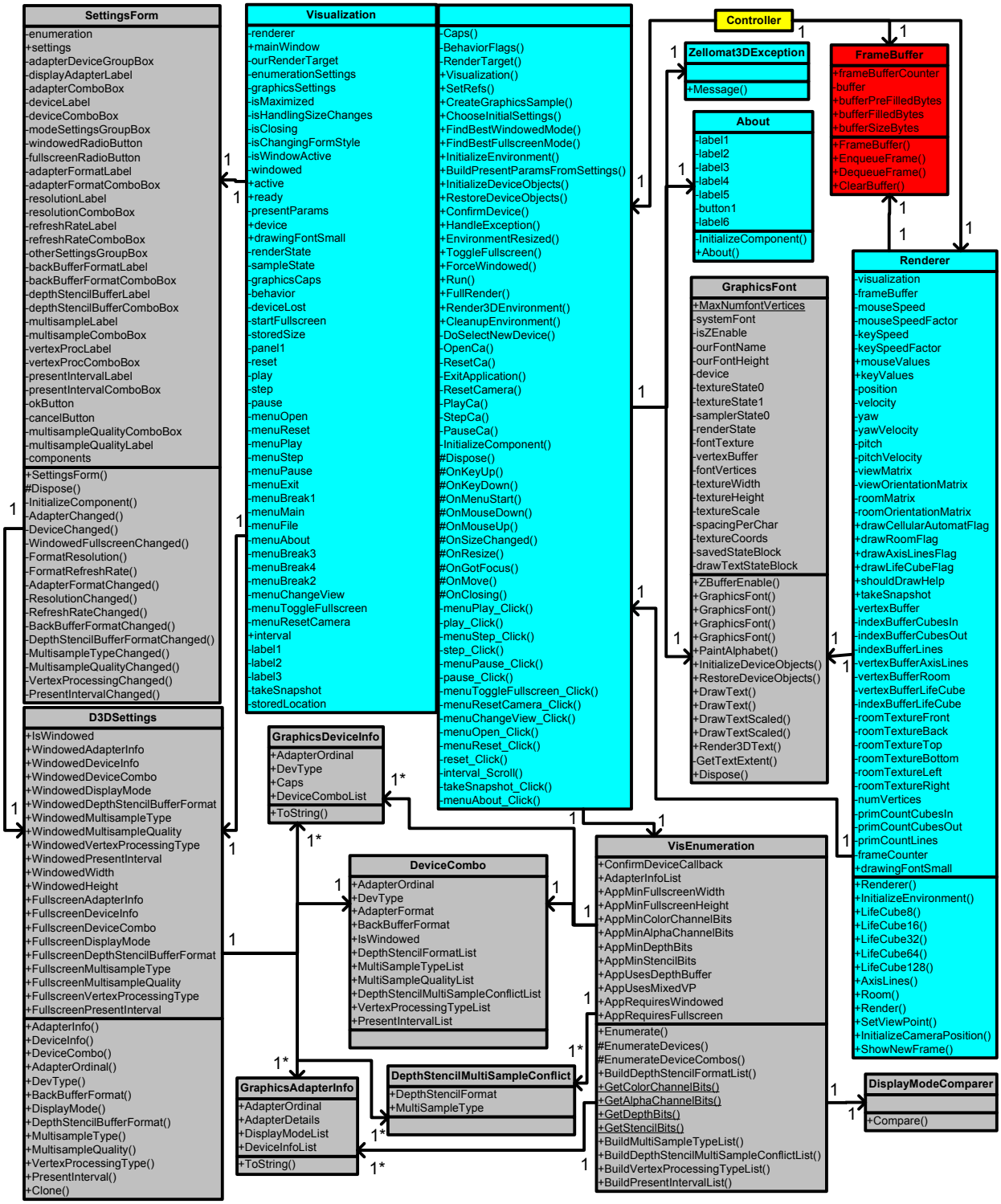
Dieses Modul verfügt über eine Datenhaltungsstruktur, welche den aktuellen sowie den neue Zyklus beinhalten. Diese Struktur wird jeweils nach dem Einlesen des ZA durch den Controller automatisch initialisiert. Mit der Methode CycleStart() wird ein Berechnungsvorgang eines einzelnen Zyklus' gestartet. Die Berechnungen der Zustände der Zellen werden durch den CellularAutomat durchgeführt. Dieser vergleicht das momentane Umfeld der Zelle mit den aktuellen Regeln. Bei einer Übereinstimmung übernimmt die Zelle den in der Regel definierten Zustand.
Nach jedem Berechnungszyklus wird bestimmt welche Wände zwischen den Zellen zu zeichnen sind. Für diese Generierung der Wände wird das Datenaufbereitungs-Modul aufgerufen.

Datenaufbereitungs -Modul



Das Datenaufbereitungs-Modul erzeugt aus den übergebenen Parameter alle Kuben der entsprechenden Zellen eines Zyklus'. Das fertige Frame wird im FrameBuffer bis zur Anzeige zwischengelagert.

Modul Visualisierung



Beschreibung

Das Modul Visualisierung beinhaltet verschiedenste Teilbereiche der visuellen Ausgabe. Sie ist für die Fenster der Applikation sowie für die für das Handling der 3D-verantwortlichen Klassen zuständig. Die Klasse Visualization ist der Kern des ganzen Modules. In ihr arbeiten die 2D- und 3D-Komponenten eng zusammen, da zwischen dem Handling des Hauptfensters und der in ihr angezeigten 3D-Ausgabe ein hoher Abhängigkeitsgrad besteht. Dies aus dem Grund, dass, wenn sich die Fenstergrösse ändert, auch die Einstellungen für das 3D Rendering auf der Grafikkarte dem Fenster angeglichen werden müssen.

4. Klasse Controller

Allgemeine Beschreibung

Der Controller ist für das Handling mehrerer Events verantwortlich. Er delegiert die verschiedenen Aufgaben an die entsprechenden Objekte, ist für das Starten der Anwendung zuständig, hat die Kontrolle über die Threads und ist für das Behandeln von Benutzerinteraktionen zuständig.

Main (string []args)

Main Methode, hier beginnt dir Applikation

StartUp ()

StartUp() ist zuständig für das starten der ganzen Applikation und das Initialisieren und Starten der einzelnen Threads

RequestOpenCa (string path)

Wird vom OpenFileDialog aufgerufen, bittet den Controller rür eine Möglichkeit, einen neuen ZA zu laden

OpenCa ()

Erstellt einen neuen ZA.

SetLifeCube (int cubeLenght)

Initialisiert im Renderer den dem neuen ZA entsprechenden Lebensraumkubus.

ShowLifeCube ()

Toggle Anzeige des Lebensraumkubus

RequestResetCa ()

Controller wird um einen Reset des aktuellen ZA angefragt

ResetCa ()

Reset des aktuellen ZA

ElapsedTimerEvent (object sender, ElapsedEventArgs arg)

Event, der aufgerufen wird, wenn der Interval zwischen 2 Frames abgelaufen ist. Er geigt an, dass ein neues Frame aus dem Buffer zu holen ist.

CalculationThread ()

Dieser Thread ist für die Berechnung der Zyklen eines ZA zuständig

RenderThread ()

Dieser Thread ist für die Visualisation (Renderer) verantwortlich

InitializeInputWithThread ()

Initialisierung des Maus-Threads, reagiert auf die Maus-Events sowie wenn die Applikation geschlossen wird.

MouseThread ()

Dieser Thread ist dafür verantwortlich die Maus-Events einzufangen, die Koordinaten der Position zu ermitteln und diese der Kamerasteuerung zur Verfügung zu stellen

TakeSnapshot ()

Fordert einen Schnappschuss des aktuell am Bildschirm dargestellten Zyklus an

SetUpdateIntervalConsole (int direction)

Reagiert auf Eingaben über die Tastatur und setzt das UpdateInterval des Timers.

SetUpdateIntervalGui (int updateInterval)

Reagiert auf Eingaben über das GUI und setzt das UpdateInterval des Timers.

StartApplication ()

Öffnet den Buffer für weitere hinzukommende Frames, lässt den 3D Renderer laufen.

StopApplication ()

Stoppt den Buffer für weitere hinzukommende Frames, stoppt den 3D Renderer, stoppt das weitere herausholen der Frames aus dem Buffer für die Anzeige.

StartVisualization ()

Startet den 3D Renderer

StopVisualization ()

Stoppt das weitere Herausholen der Frames aus dem Buffer für die Anzeige, stoppt den 3D Renderer.

StartRenderer ()

Startet den 3D Renderer.

StopRenderer ()

Stoppt den 3D Renderer.

PauseCa ()

Stoppt das weitere herausholen der Frames aus dem Buffer für die Anzeige.

PlayCa ()

Startet das Herausholen der Frames aus dem Buffer für die Anzeige.

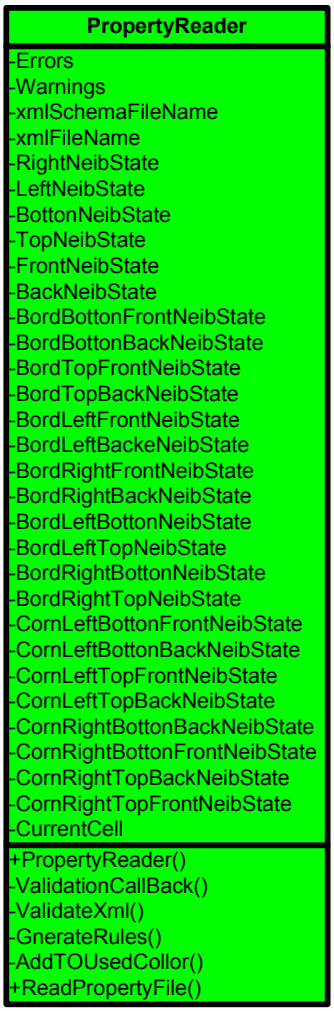
StepCa ()

Stoppt das weitere herausholen der Frames aus dem Buffer für die Anzeige, holt ein Frame aus dem Buffer für die Anzeige heraus.

CloseApplication ()

Beendet die Applikation und räumt auf.

5. Klasse PropertyReader



Allgemeine Beschreibung

Diese Klasse ist für das Einlesen des ZA-Files zuständig. Sie validiert das XML mittels Schema und parst alle im XML angegebenen Werte. Danach erzeugt sie die Regeln und übergibt alles dem "Controller".

PropertyReader ()

Der Konstruktor initialisiert die Errorliste und Warnungenliste.

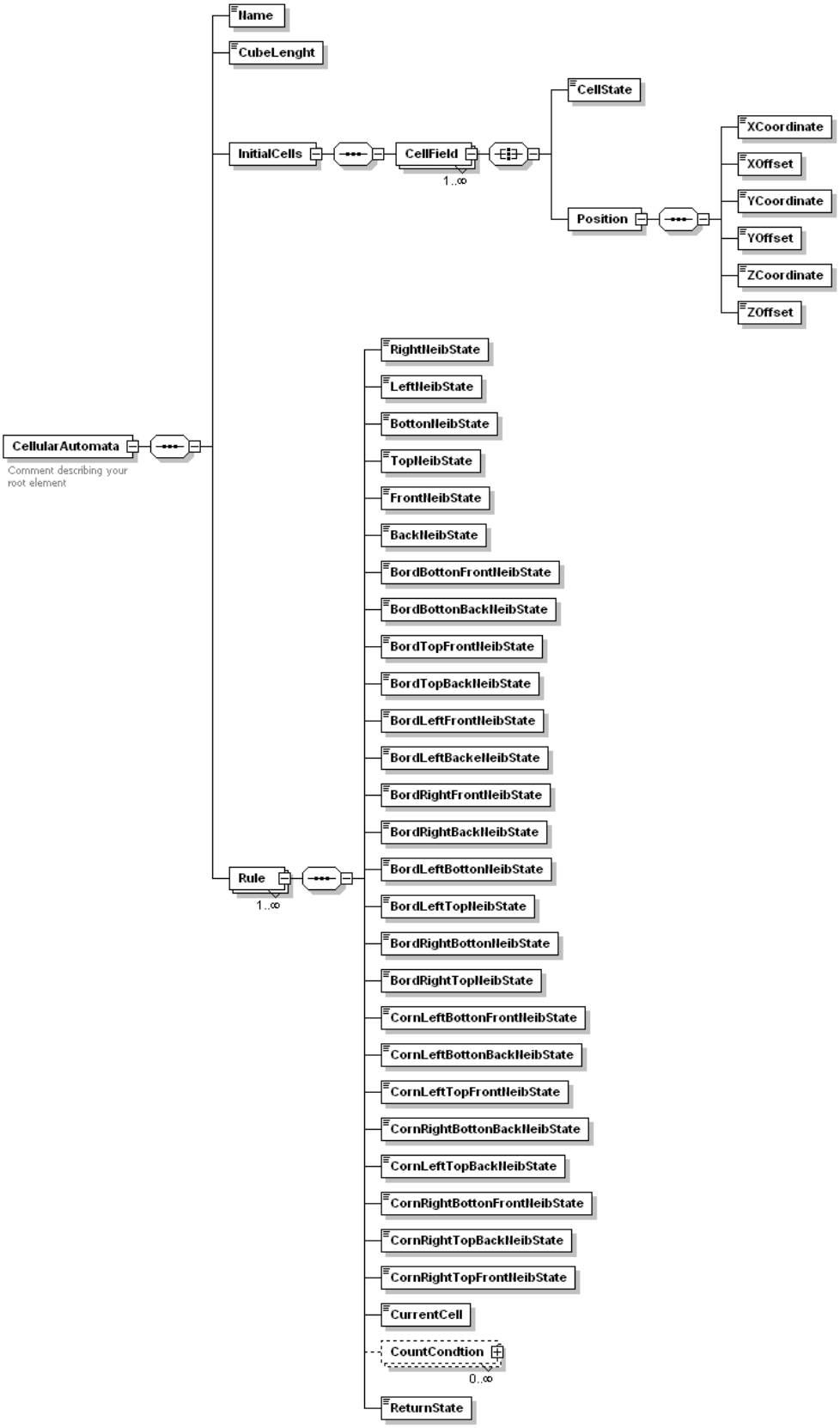
ValidationCallBack (object sender, ValidationEventArgs args)

Die Fehler, die beim Validieren gefunden werden, werden von hier aufgefangen.

ValidateXml ()

Die Methode validiert das CA XML-File anhand eines erstellten XML-Schema.

Schemaübersicht



Die Beschreibung der einzelnen Tags mit Ihren gültigen Werten stehen im Benutzerhandbuch im Kapitel ZA erstellen.

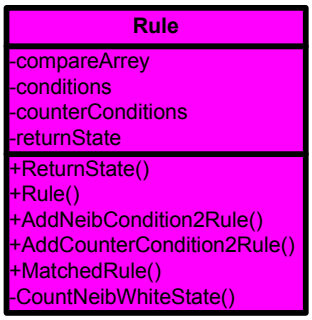
**GenerateRules (out ArrayList initialStates, out ArrayList ruleset,
ref bool []neibDirections, ref string CAName,
ref int borderLenght, ref ArrayList usedColor)**

Das Property File wird geparst und die Regeln werden generiert. Die Regeln und alle ZA spezifischen Parameter werden an den "Controller" weiter geleitet.

public bool ReadPropertyFile (string xmlFileName, out ArrayList initialStates, out ArrayList ruleset, ref bool []neibDirections, ref string CAName, ref int borderLenght, ref ArrayList usedColor, ref ArrayList warnings, ref ArrayList errors)

Das Validieren und danach das Parsen des Propertyfiles werden aufgerufen.

6. Klasse Rule



Allgemiene Beschreibung

Diese Klasse repräsentiert eine Regel des ZA. Sie beinhaltet Methoden zum Setzen einer Regel sowie zum Überprüfen, ob die Regel bei einer Berechnung der Zelle greift.

Rule ()

Ist der Konstruktor der Klasse. Er înitialisiert ein Array für die Vergleichszustände.

AddNeibCondition2Rule (int []neighbour, byte state)

Eine Nachbarbedingung wird der Regel hinzugefügt.

AddCounterCondition2Rule (int stateComparator, byte countableState, int referenceComparator, int referenceValue)

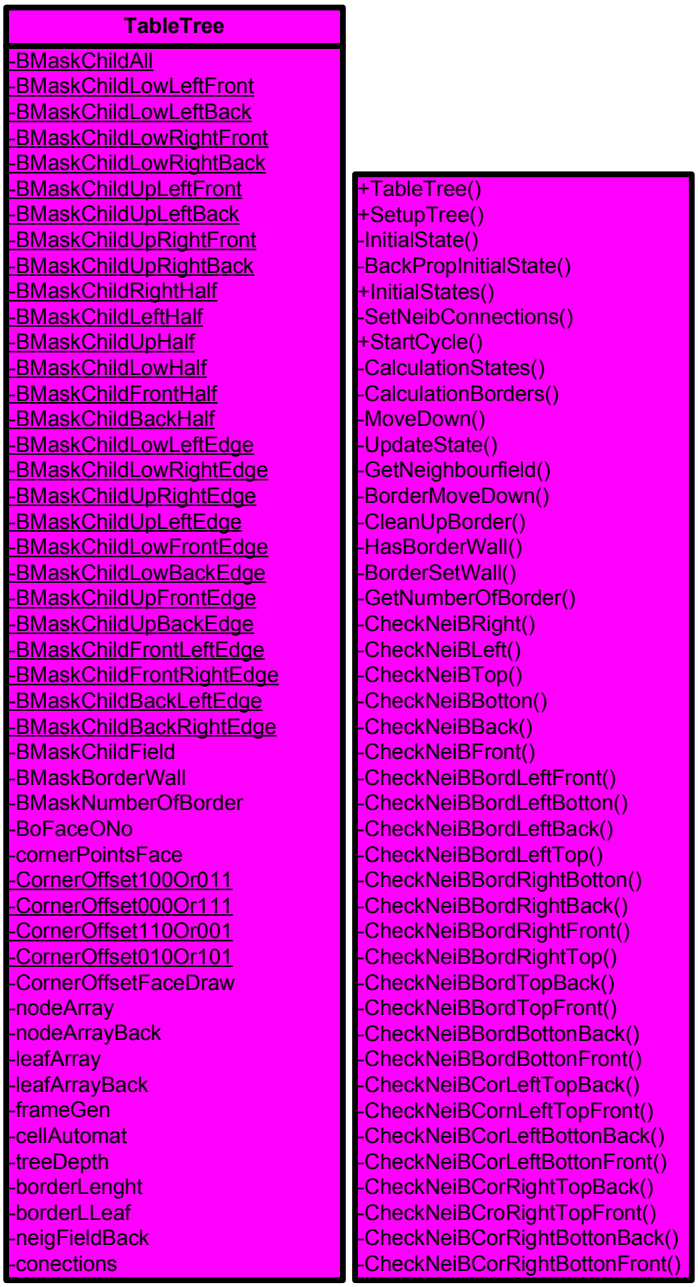
Eine Zählbedingung wird der Regel hinzugefügt.
MatchedRule (byte [,,] neigField)

Alle Bedingungen der Regel werden nacheinander auf ihr Zutreffen überprüft, indem das CompareArray mit dem Nachbarzustandsfeld an den im NachbarArray gesetzten Positionen verglichen wird. Sobald eine Bedingung nicht zutrifft, wird „false“ zurückgegeben. Wenn alle Bedingungen zutreffen, wird „true“ zurückgegeben.

CountNeibWhiteState (byte state, int comparator, byte[,,] neigField)

Alle Zustände des Nachbarschaftsfeldes werden gezählt, die grösser als, kleiner als oder gleich dem angegebenen Zustand sind.

7. Klasse TableTree



Allgemeine Beschreibung

Die Klasse TableTree enthält die Datenhaltung der Zellen. Sie Ist für das Finden der Zellen verantwortlich, welche neu berechnet werden müssen. Sie liefert alle Zustände der Nachbarn für die Berechnung. Nach einem Berechnungszyklus bestimmt sie alle Zellwände, die gezeichnet werden müssen.

TableTree (DataPreparation dataPreparation, CellularAutomat cellAutomat)

Dies ist der Konstruktor des TableTree. Er benötig ein DataPreparation-Objekt für die Frameerzeugung und ein CellularAutomat-Objekt, welches den Zustand einer Zelle berechnet

SetupTree (int cubeLenght, bool []neibDirections)

Die für den aktuellen ZA benötigten Einstellungen werden gesetzt. Die Kantenlänge des Lebensraumes bestimmt die Grösse des Baumes. Es werden auch die benötigten Nachbarschaftsrichtungen gesetzt.

InitialState (int x, int y, int z, byte state)

Eine Zelle mit den Koordinaten X, Y, Z wird auf den mit "state" angegebenen Zustand gesetzt. Danach wird die BackPropInitialState Methode aufgerufen, um die Knoten im Baum richtig zu setzen.

BackPropInitialState (int x, int y, int z, int i, int j, int k, int l)

Der gesetzte Zustand einer Zelle wird den Baum hinauf propagiert. Die Methode benötigt die Koordinaten des Vaters und die Indexe zum Bestimmung des richtigen Maskierungsfelds. Sie ruft sich rekursive wieder auf, bis die Wurzel des Baumes erreicht ist.

InitialStates (ArrayList initialStates)

Der Baum wird erzeugt und für alle gesetzten Initialzustände wird die InitialState Methode aufgerufen.

SetNeibConnections (bool []direction)

Die Richtungen in den Baumknoten werden gesetzt, in denen geprüft werden muss, ob sich eine der Zellen, die zum Teilbaum des Nachbars gehören, geändert hat.

StartCycle ()

Die Berechnung und Datenaufbereitung eines Frames werden gestartet. Wenn Zellwände existieren, wird das Frame in den Buffer geladen.

CalculationStates ()

Es wird zuerst ein Backup des Baumes erstellt. Danach wird die MoveDown Methode aufgerufen. Wenn der Berechnungszyklus zu Ende ist, wird das Backup wieder freigegeben.

CalculationBorders ()

Startet die Bestimmung der zu zeichnenden Wände der Zellen und generiert ein Frame. Dazu wird zuerst InitializePointArray() der DataPreparation aufgerufen. Danach wird die BorderMoveDown Methode gestartet.

MoveDown (int l, int x, int y, int z, ref byte isBorder, ref byte changed)

Sie entscheidet welch Kinder des aktuellen Knotens weiterverfolgt werden müssen. Danach ruft sie sich rekursiv mit den Koordinaten und der Baumtiefe der entsprechenden Kinder immer wieder auf, bis eine Zelle (Blatt des Baumes) erreicht ist. Für jede ereichte Zelle wir eine Update gestartet. Ergibt die Berechnung einer Zelle eine Zutstandsänderung, wird dies allen Vätern mitgeteilt.

UpdateState (ref bool changed, ref byte isBorder, int x, int y, int z)

Die Berechnung einer Zelle wird aufgerufen. Das erhaltene Resultat wird gespeichert und bei einer Änderung des Zellzustandes werden die Werte "changed" = true und "isBorder" = 1 gesetzt. Hat eine Zelle nach der Berechnung immer noch den gleichen Zustand wie vor der Berechnung, wird "changed" = false gesetzt. "isBorder" wird in diesem Fall nur dann auf 1 gesetzt, wenn die Zelle schon vor der Neuberechnung eine zu zeichnende Wand besessen hat.

GetNeighbourfield (int x, int y, int z)

Ein Feld mit den Zuständen aller Nachbarzellen wird gebildet.

BorderMoveDown (int l, int x, int y, int z, ref byte isBorder)

Diese rekursive Methode dient zur Bestimmung der Zellen die eine oder mehrere Wände haben. Sie springt dabei von einem Knoten zu alle Kindern des Knotens, die weiter verfolgt werden müssen. Wenn sie eine Zelle (Blatt) ereicht hat startet sie die "CleanUpBorder" Methode

.

CleanUpBorder (int x, int y, int z, ref byte isBorder)

Alle nicht benötigten Wände einer Zelle mit den Koordinaten X, Y, Z werden gelöscht und die benötigten Wände der Datenaufbereitung übergeben. Eine Wand zwischen zwei Zellen, die den gleichen Zustand haben, wird nicht benötigt.

HasBorderWall (byte borderField, int index)

Aus einem Wert "boundaryBitField" wird herausgelesen, ob auf der mit dem Wert "index" angegebene Seite eine zuzeichnende Wand existiert.

BorderSetWall (ref byte borderField, int index, bool wall)

Eine Wand wird auf der mit "index" angegebenen Seite der Zelle gesetzt oder entfernt.

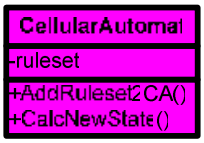
GetNumberOfBorder (byte borderField)

Die Anzahl der Wände einer Zelle wird ermittelt.

CheckNeiB… (int xo, int yo, int zo, int borderLenght, int l, bool [,,] childs)

Es wird geprüft, ob irgendeines der Kinder des entsprechenden Nachbarn des aktuellen Knotens, die sich auf der ihm zugewandten Seite dieses Nachbarn befinden, ein "changed" aufweisen. Trifft dies zu, werden die entsprechenden "childs" auf "true" gesetzt.

8. Klasse CellularAutomat



Allgemeine Beschreibung

Die Klasse CellularAutomat berechnet den Zustand einer Zelle.

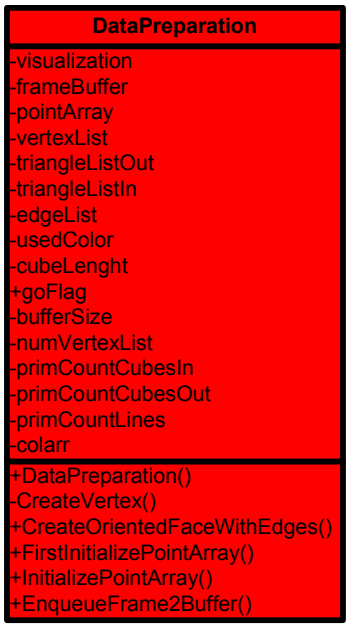
AddRuleset2CA (ArrayList ruleset)

Die Regeln werden in Form einer ArrayList dem ZA hinzugefügt.

CalcNewState (byte[,,] neigField)

Alle Regeln werden der Reihe nach mit einem Feld von Zuständen verglichen. Dieses Feld beinhaltet die Zustände der Zelle und aller ihrer Nachbarn. Bei einer Übereinstimmung wird der in der Regel definierte Zustandswert zurückgegeben. Gibt es keine Regel, die greift, wird der alte Zustandswert wieder zurückgegeben.

9. Klassse DataPreparation



Allgemeine Beschreibung

Erstellt aus den übergebenen Wänden ein Frame (vertexBuffer, indexBufferCubesIn, indexBufferCubesOut, indexBufferLines). Nach dem Ende eines Berechnungszyklus reiht er das Frame in den FrameBuffer ein.

DataPreparation (Visualization visualization, FrameBuffer frameBuffer)

Der Konstruktor initialisiert vertexList, triangleListOut, und edgeList. Ebenso speichert er das erhaltene "Visualization" und "FrameBuffer" Objekt.

CreateVertex (int x, int y, int z, int color, int pointArrayIndex)

Die übergebenen Punkte werden, wenn sie noch nicht existieren, erstellt. Danach wird die Referenz des Punktes im PointArray gespeichert.

CreateOrientedFaceWithEdges (int [,] cornerPoints, int colorMe, int colorNb)

Zuerst werden die Punkte für das Gitternetz erstellt. Danach werden die Vertexes für die Seiten erstellt. Dabei muss die CreateEdges-Methode für die farbgebenden Punkte aufgerufen werden.

FirstInitializePointArray (int cubeLenght, ArrayList usedColor)

Alle ZA-abhängigen Parameter werden gesetzt, die Kantenlänge des Lebensraumes und eine Liste mit den gebrauchten Farben.

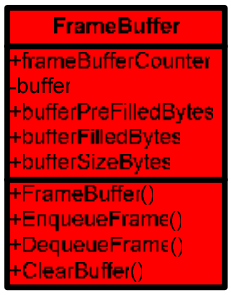
InitializePointArray ()

Initialisiert das PointArray, in welchem die schon erstellten Punkte gespeichert werden.

EnqueueFrame2Buffer ()

Die generierten Daten werden in Vertex- und IndexBuffer umgewandelt und in die Queue eingefügt.

10. Klasse FrameBuffer



Allgemeine Beschreibung

Diese Klasse regelt die Interaktionen mit dem Buffer, in dem die berechneten Framen bis zu deren Anzeige auf dem Bildschirm abgelegt sind.

public FrameBuffer()

Das ist der Konstruktor der Klasse. Er Initialisiert die Frame Queue.

EnqueueFrame (int numVertices, int primCountCubesIn, int primCountCubesOut, int primCountLines, VertexBuffer vertexBuffer, IndexBuffer indexBufferCubesIn, IndexBuffer indexBufferCubesOut, IndexBuffer indexBufferLines)

Hier werden die vorproduzierten Vertex- und IndexBuffer der Reihe nach der Queue hinzugefügt.

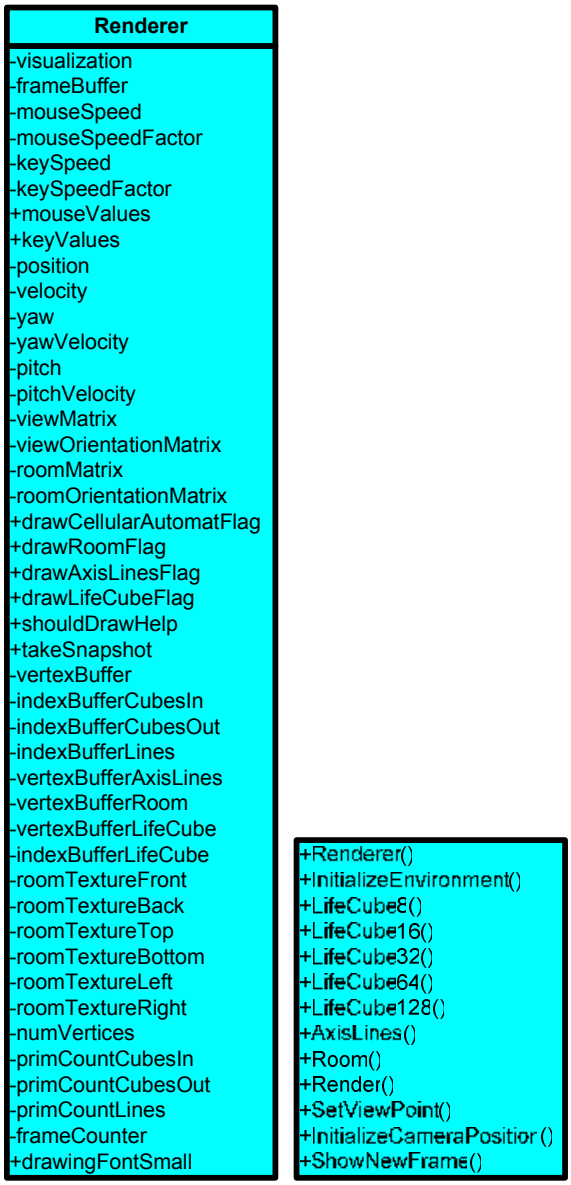
DequeueFrame (ref bool drawCellularAutomatFlag, ref int numVertices, ref int primCountCubesIn, ref int primCountCubesOut, ref int primCountLines, ref VertexBuffer vertexBuffer, ref IndexBuffer indexBufferCubesIn, ref IndexBuffer indexBufferCubesOut, ref IndexBuffer indexBufferLines)

Hier werden die gepufferten Frames aus dem Framebuffer genommen.

ClearBuffer()

Aufräumen der Parameter.

11. Klasse Renderer



Allgemeine Beschreibung

Diese Klasse ist für das Zeichnen der 3D-Objekte verantwortlich, Dazu gehören der Lebensraumkubus, die XYZ-Achsenlinien, der Hintergrundraum und zuletzt der aktuelle Zyklus des zellulären Automaten. Des Weiteren ist sie auch für das Initialisieren des Lebensraumkubus, der XYZ-Achsenlinien und des Hintergrundraumes zuständig. Die Kamerasteuerung befindet sich ebenfalls in dieser Klasse.

Renderer (Visualization visualization, FrameBuffer frameBuffer)

Konstruktor des Renderers.

InitializeEnvironment ()

Die Umgebung wird initialisiert, d.h. die benötigten 3D Objekte werden initialisiert.

LifeCube8 ()

Lebensraumkubus der Grösse 8x8x8.

LifeCube16 ()

Lebensraumkubus der Grösse 16x16x16.

LifeCube32 ()

Lebensraumkubus der Grösse 32x32x32.

LifeCube64 ()

Lebensraumkubus der Grösse 64x64x64.

LifeCube128 ()

Lebensraumkubus der Grösse 128x128x128.

AxisLines ()

Die XYZ-Achsen werden hier initialisiert.

Room ()

Hier wird die Umgebung, in der der zelluläre Automat „lebt," initialisiert

Render ()

Dieser wird einmal pro Frame aufgerufen, hier werden die effektiven Renderbefehle abgesetzt und die 3D Objekte auf der Grafikkarte gerendert und auf dem Bildschirm ausgegeben. Setzt die RenderStates, löscht den BackBuffer, rendert die Szene, bewegt die Kamera.

SetViewPoint ()

Berechnet den neuen Standort & Blickrichtung der Kamera. Des Weiteren wird die Verschiebung des Umgebungskubuses mit der Kameraposition errechnet.

InitializeCameraPosition ()

Diese Methode bewirkt ein Reset der Kameraposition und die Blickrichtung auf den Initialwert

ShowNewFrame ()

Hier werden die Daten für das Anzeigen des nächsten Zyklusstandes des ZA aus dem FrameBuffer herausgeholt.

12. Klasse Visualization

Allgemeine Beschreibung

In dieser Klasse befinden sich das User Interface sowie die Verarbeitung der DirectX-Komponenten des Zellomat3D. Da zwischen dem GUI und der 3D-Visualisierung der zellulären Automaten eine enge direkte Beziehung besteht, sind diese zwei Teilbereiche in dieser Klasse zusammengefasst.

Visualization ()

Konstruktor

SetRefs (Renderer renderer)

Direktzugriff der Visualisierung auf den Renderer (für den Aufruf der Render Methode)

CreateGraphicsSample ()

Wählt automatisch das geeignetste Grafikdevice aus und initialisiert es.

ChooseInitialSettings ()

Wählt die Initialzustände für die Applikation

FindBestWindowedMode (bool doesRequireHardware, bool doesRequireReference)

Setup der graphicsSettings mit dem besten zur Verfügung stehenden Windowmode, basierend auf den doesRequireHardware und doesRequireReference Einschränkungen

FindBestFullscreenMode (bool doesRequireHardware, bool doesRequireReference)

Setup der graphicsSettings mit dem besten zur Verfügung stehenden fullscreen-Mode, basierend auf den doesRequireHardware und doesRequireReference Einschränkungen

InitializeEnvironment ()

Initialisierung der grafischen Einstellungen / Umgebung

BuildPresentParamsFromSettings ()

Präsentation des Parameters aufgrund der aktuellen Einstellungen ändern

InitializeDeviceObjects ()

Initialisierung der Schriften für die 3D-Textausgabe

RestoreDeviceObjects (System.Object sender, System.EventArgs e)

Stellt die Einstellungen wieder her, nachdem das Device geresetet wurde. Ein Reset kommt dann vor, wenn Änderungen an der Auflösung der Ausgabe erfolgen.

ConfirmDevice (Caps caps, VertexProcessingType vertexProcessingType, Format adapterFormat, Format backBufferFormat)

Wird während der Device Initialisierung aufgerufen. Sie überprüft das Device auf die minimalen Funktionalitäten.

HandleException (Exception e, ApplicationMessage Type)

Zeigt dem Benutzer die aufgetretene Exception an.

EnvironmentResized (object sender, System.ComponentModel.CancelEventArgs e)

Event wenn das Fenster in der Grösse geändert wird.

ToggleFullscreen ()

Wird aufgerufen, wenn der Benutzer zwischen Fullscreen- und Windowsmodus wechselt.

ForceWindowed ()

Wechsel zu Windowsmodus (erzwingen), auch wenn das bedeutet, ein neues Device wieder zuerstellen.

Run ()

Applikationsfenster erstellen, und kontinuierlich Events behandeln

FullRender ()

Die Hauptrender Methode, zuständig für das Rendern der visuellen Objekte. Zuerst werden aber ein paar Bedingungen abgeklärt, ob das Rendern stattfinden kann.

Render3DEnvironment ()

Zuständig für das Behandeln von Vorbedingungen, bevor die 3D Objekte gerendert werden können.

CleanupEnvironment ()

Aufräumen der Applikation.

DoSelectNewDevice ()

Zeigt einen Dialog an, in dem der Benutzer diverse Einstellungen, wie einen neuen Adapter (Device), Anzeigemodus, Auflösung usw. selektieren kann. Anschliessend wird den Einstellungen entsprechend das 3DEnvironment neu initialisiert.

OpenCa ()

Öffnet einen Dialog um einen ZA auszuwählen.

ResetCa ()

Reset des Zellulären Automaten.

ExitApplication (object sender, EventArgs e)

Die Applikation wir beendet.

ResetCamera ()

Die Kameraposition wird auf den Anfangszustand gesetzt.

PlayCa ()

Die Zyklen des zellulären Automaten werden nacheinander angezeigt.

StepCa ()

Jeweils nur der nächste Zyklus wird angezeigt.

PauseCa ()

Keine Anzeige der nachfolgenden Zyklen.

InitializeComponent ()

Komponenten des Fensters initialisieren.

Dispose (bool disposing)

Aufräumen der Applikation.

OnKeyUp (System.Windows.Forms.KeyEventArgs e)

Event, wenn einen Keyboardtaste losgelassen wird.

OnKeyDown (System.Windows.Forms.KeyEventArgs e)

Event, wenn eine Taste gedrückt wird.

OnMenuStart (System.EventArgs e)

Wenn das Menü erscheint, pausiert die Applikation-

OnMouseDown (System.Windows.Forms.MouseEventArgs e)

Event, wenn eine Maustaste gedrückt wird.

OnMouseUp (System.Windows.Forms.MouseEventArgs e)

Event, wenn eine Maustaste losgelassen wird.

OnSizeChanged (System.EventArgs e)

Event wenn das Fenster in der Grössen geändert wurde

OnResize (System.EventArgs e)

Event, wenn das Fenster in der Grösse geändert wurde.

OnGotFocus (System.EventArgs e)

Wenn das Fenster wieder den Fokus erhält, werden Parameter für die Behandlung von Fenstergrösse und das Rendern gesetzt.

OnMove (System.EventArgs e)

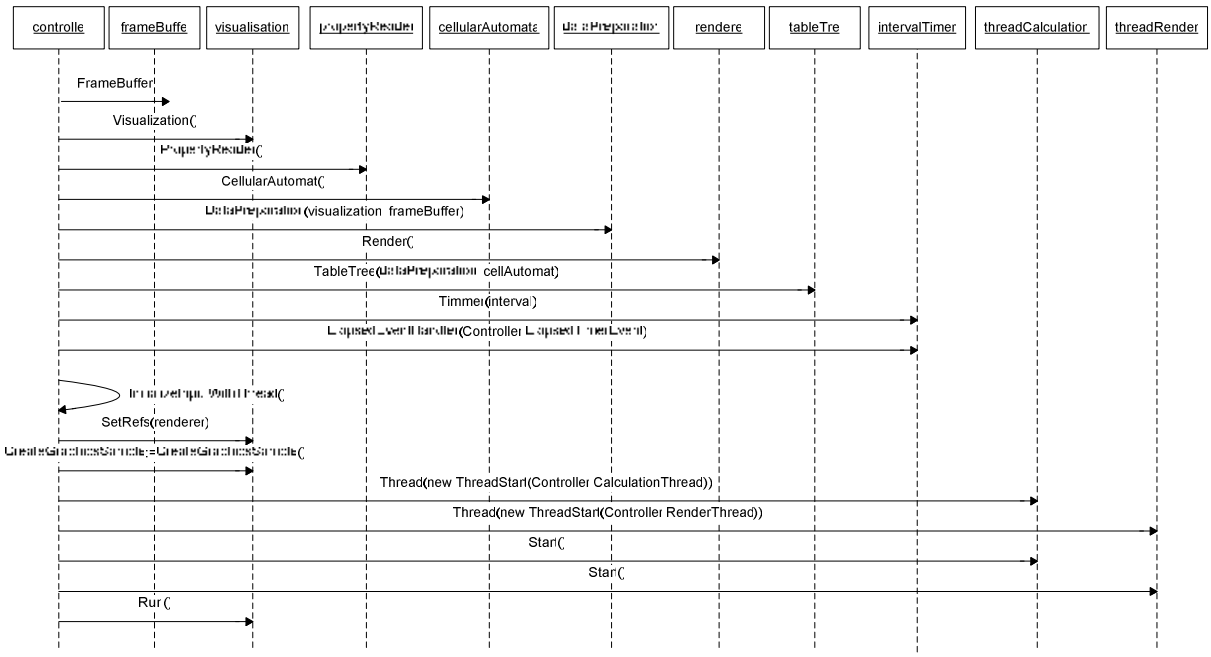
Event, wenn das Fenster herumgeschoben wird.

OnClosing (System.ComponentModel.CancelEventArgs e)

Event, wenn das Fenster geschlossen wird.

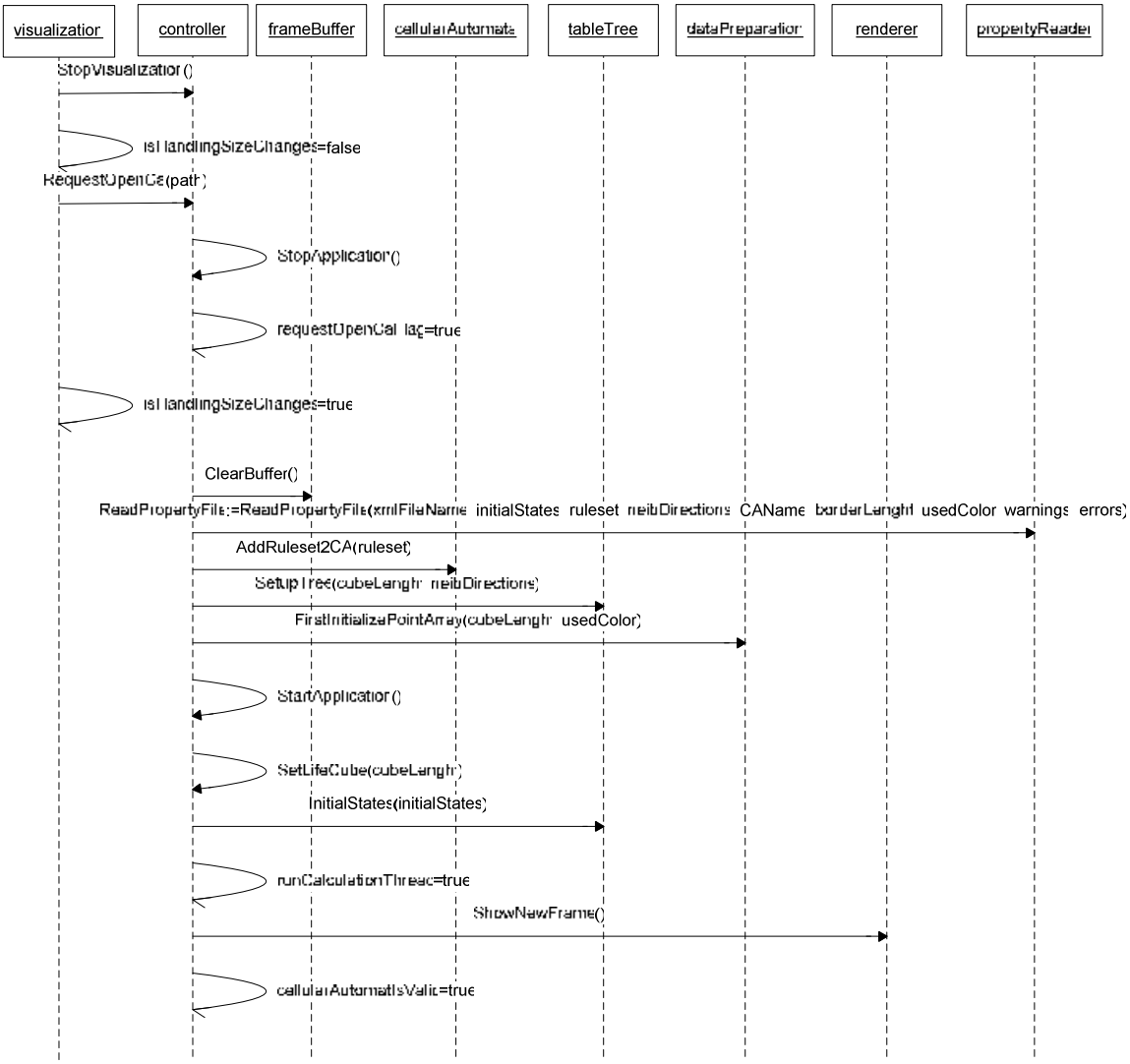
13. Sequenzdiagramme

Programmstart



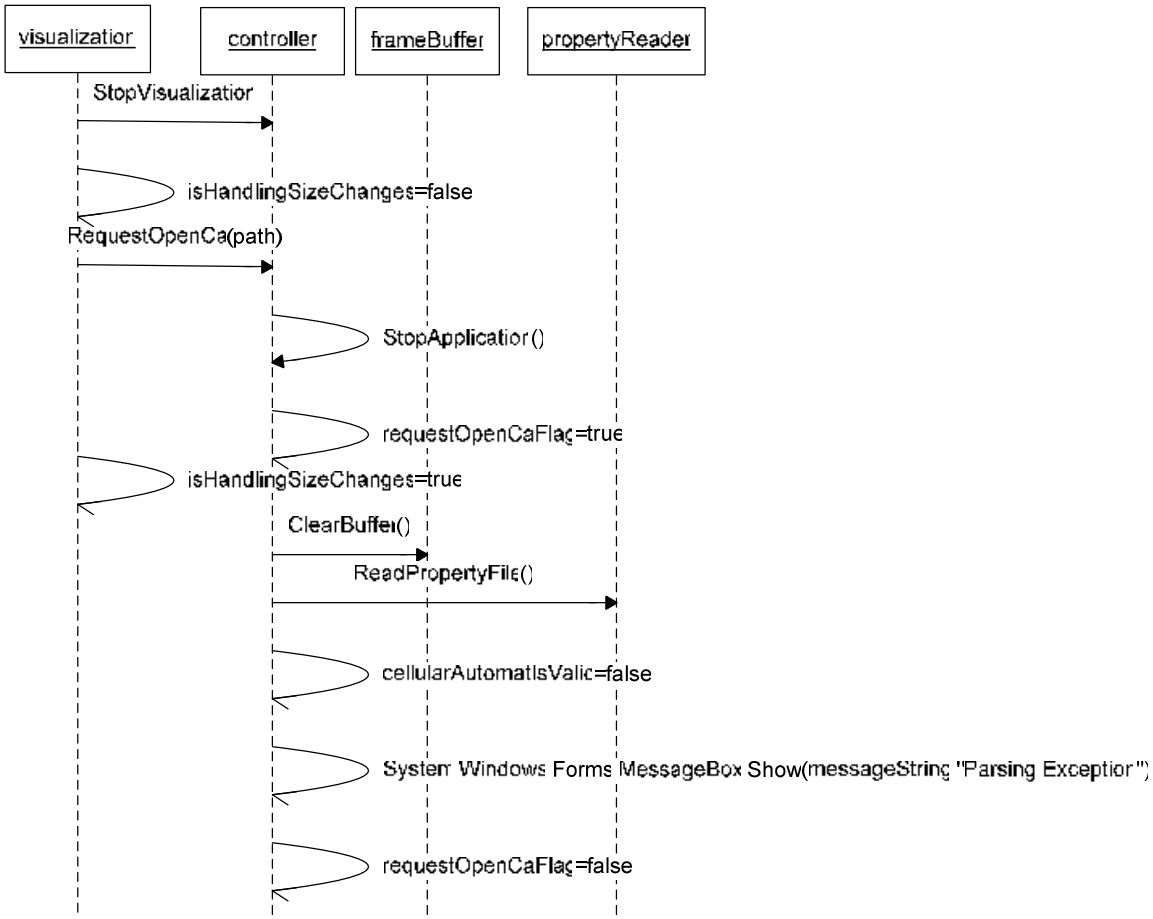
Beim Starten des Programms werden zuerst alle Objekte initialisiert. Danach werden die Threads für die Maussteuerung, das zyklische EventHandling des GUIs, den Rendervorgang und die Berechnung gestartet. Zum Schluss wird das Fenster der Applikation aktiviert.

ZA laden (ohne Errors)



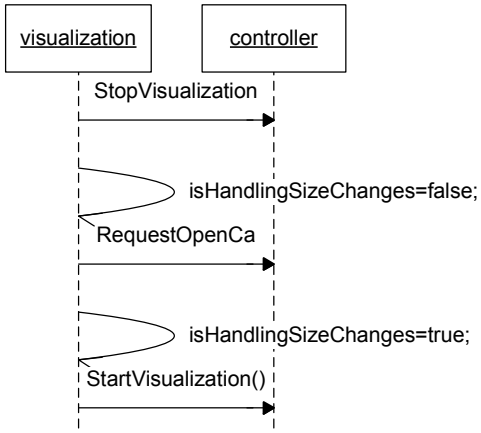
Dieses Diagramm beschreibt den Vorgang des Ladens eines ZA. Er wird gestartet, wenn der Benutzer einen neuen ZA im Menü anwählt. Bevor ein ZA geladen werden kann, wird sicher gestellt, dass der Rendervorgang sowie das Berechnen gestoppt sind. Daraufhin wird der FrameBuffer geleert. Nun ist das Programm für das Laden eines neuen ZA bereit. Um einen ZA zu laden, wird die Methode ReadPropertyFile aufgerufen. Sie hat als Übergabeparameter den Pfad des ZA. Die ReadPropertyFile-Methode validiert zuerst das File. Danach werden die einzelnen Teile des Files geparst. Die ausgelesenen Parameter werden dem Controller zurückgegeben. Der Controller setzt diese bei den Klassen, die sie benötigen. Bevor die Berechnung wieder gestartet wird, werden noch die Anfangszustände gesetzt und angezeigt.

ZA laden mit Parsingfehler



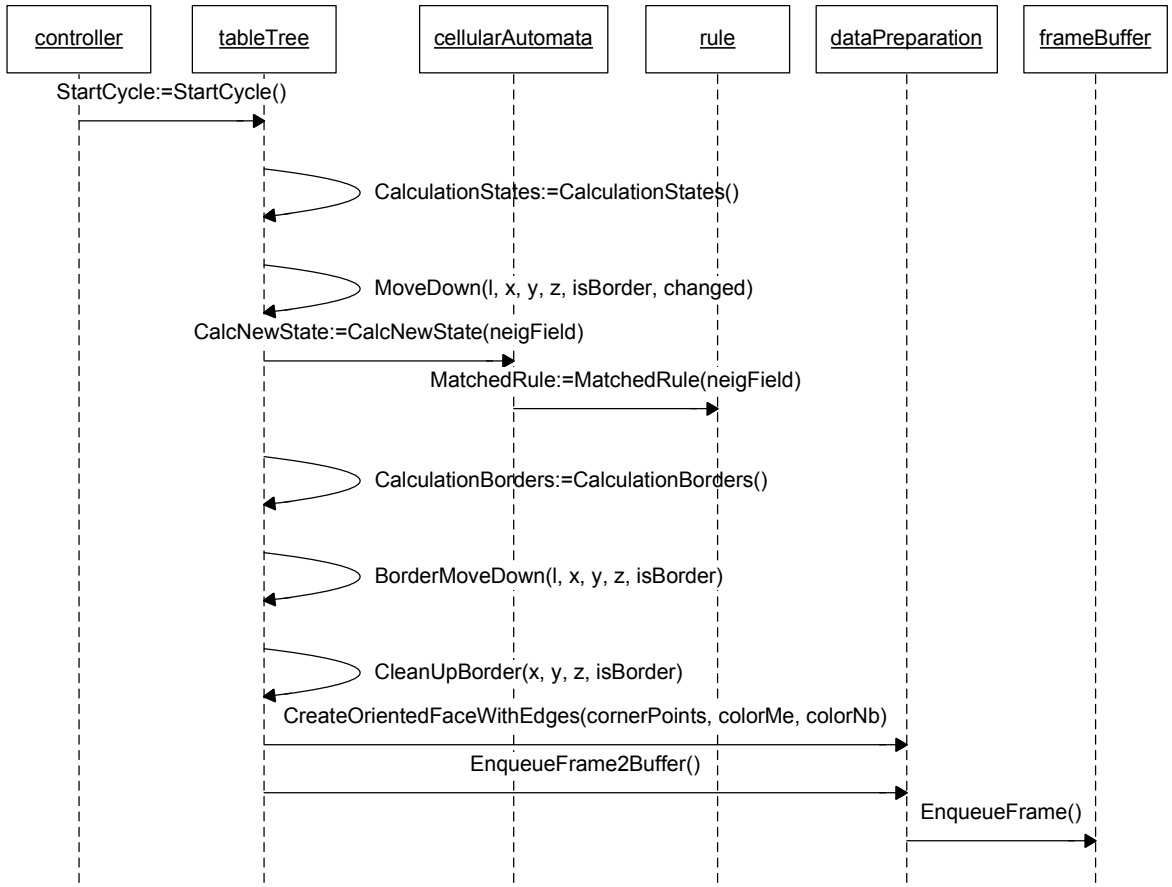
Ist das einzulesende ZA-File fehlerhaft, werden Berechnung und der Renderervorgang nicht fortgesetzt. Statt dessen wird eine MassageBox angezeigt, welche die Fehler des ZA-Files beschreibt.

Abbruch beim Laden des ZA



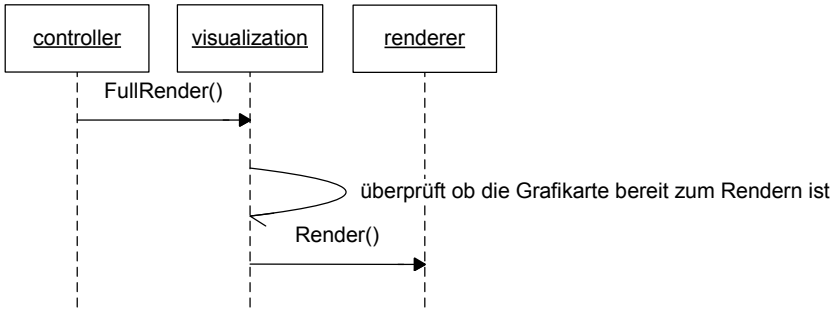
Wird der Ladevorgang durch Drücken des Cancel-Knopfes abgebrochen, werden Anzeige sowie Berechnung weiter geführt.

Berechnung



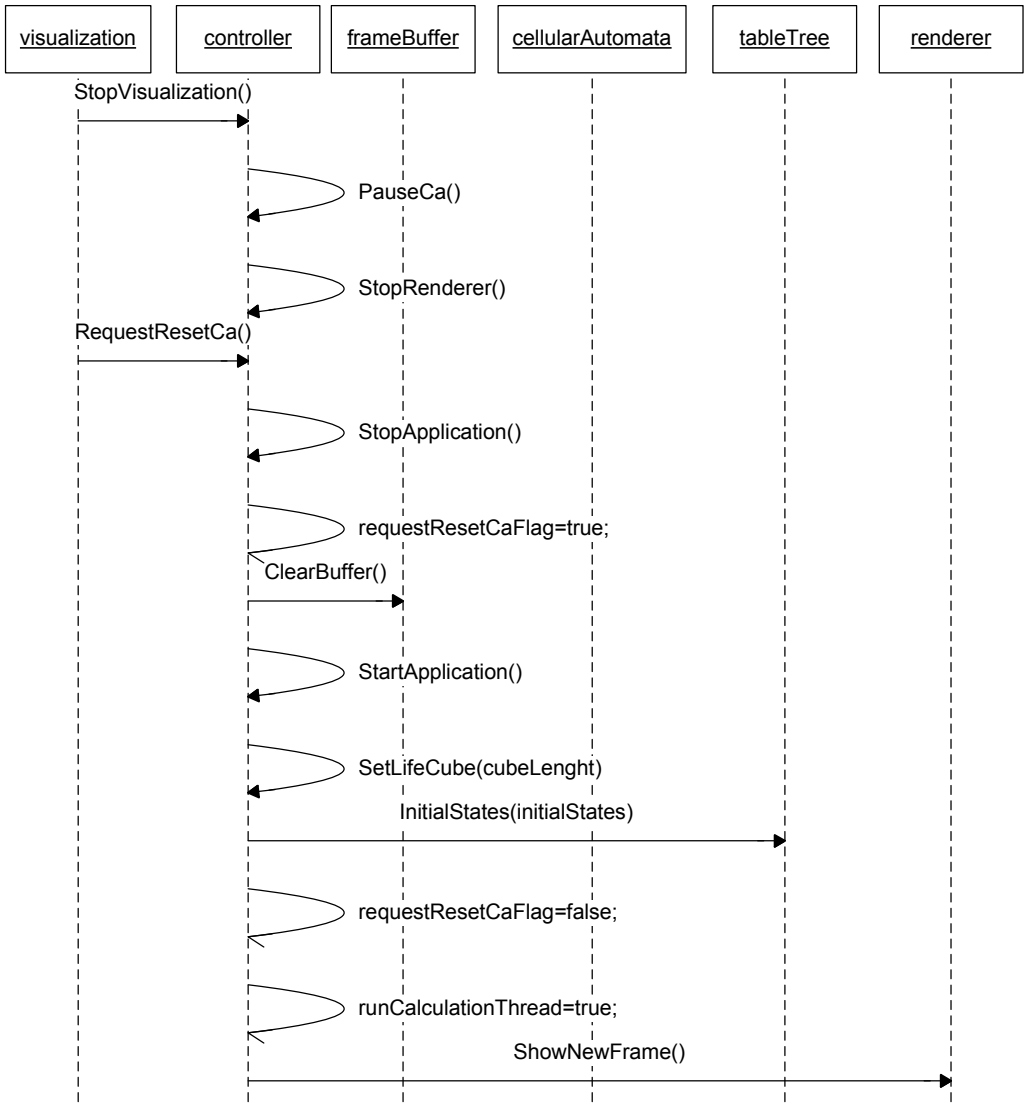
Die Berechnung der Zustände läuft in einem eigenständigen Thread ab. Der Controller ruft für jeden Zyklus die StartCycle Methode des TableTree auf. Diese Methode startet zuerst den Berechnungszyklus der neuen Zustände. Die MoveDown Methode findet die zu berechnenden Zellen. Alle diese Zellen werden durch den CellularAutomat neu berechnet. Der CellularAutomat verwendet die erhaltenen Rules für die Berechnung. Ist der Berechnungszyklus abgeschlossen, startet die StartCycle Methode die Bestimmung der Grenzen zwischen den Zellen. Jede gefunden Grenze wird der Datenaufbereitung übergeben. Nach dem Ende des Grenzbestimmungszyklus wird das Frame in den FrameBuffer gelegt.

Rendern



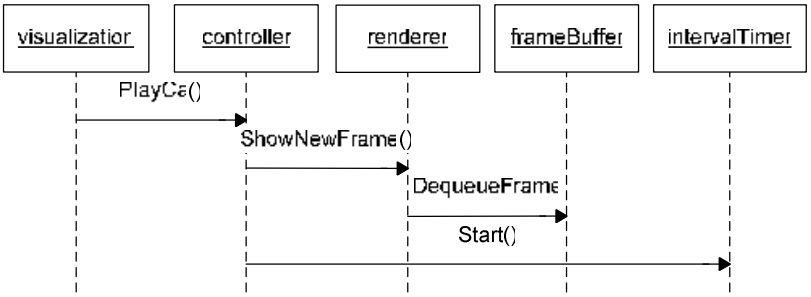
Der Rendervorgang ist ein eigenständiger Thread. Dieser Thread wird im Controller gestartet. Es wird die FullRendere-Methode der Visualization angesprungen. Dort wird geprüft, ob die Grafikkarte bereit ist. Nur wenn die Prüfung die Bereitschaft der Grafikkarte bestätigt hat, wird der Rendervorgang gestartet. Nach Abschluss des Rendervorgang wird der Thread für 30 Millisekunden „schlafen gelegt“ danach startet er von Neuem.

Event Reset ZA



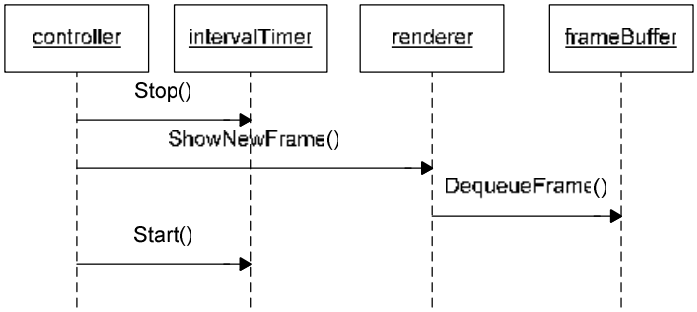
Dieser Event wird direkt durch den Benutzer ausgelöst. Er ist ausserdem ein Bestandteil der Umschaltung auf den Vollbildmodus und wieder zurück.
Wie beim Laden des ZA werden die Berechnung und die Anzeige gestoppt und der Inhalt des Frame Buffer wird gelöscht. Danach können die Anfangszustände wieder gesetzt und angezeigt werden.

ZA-Play Event



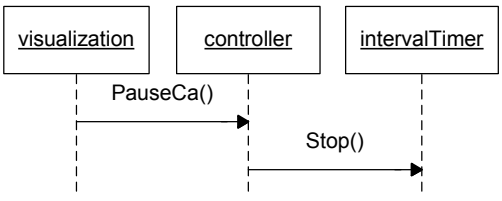
Dieser Envent wird vom Benutzer ausgelöst, wenn er die Taste X betätigt, den Play-Knopf drückt oder im Menü auf Play klickt. Dieser Event wird von der Visualization-Klasse an die Controller Klasse weitergeleitet. Der Controller befiehlt dem Rendere,r sich eine Referenz auf das nächste Frame zu holen. Dieses Frame wird nun solange im Rendervorgang angezeigt, bis sich der Renderer wieder eine Referenz auf das nächste Frame geholt hat. Am Schluss dieses Events wird der IntervalTimmer gestartet.

Event Intervall Timmer abgelaufen



Dieser Event wird ausgelöst, wenn die eingestellte Zeit des IntervalTimers abgelaufen ist. Zuerst wird der IntervalTimer gestoppt. Danach wird dem Renderer befohlen, sich eine Referenz auf das nächste Frame zu holen. Am Schluss wird der intervalTimer wieder gestartet.

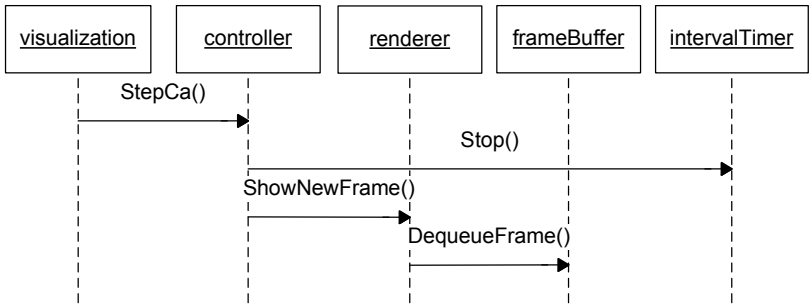
ZA Pause Event



Wenn der Benutzer über Taste, Knopf oder Menü die Anzeige

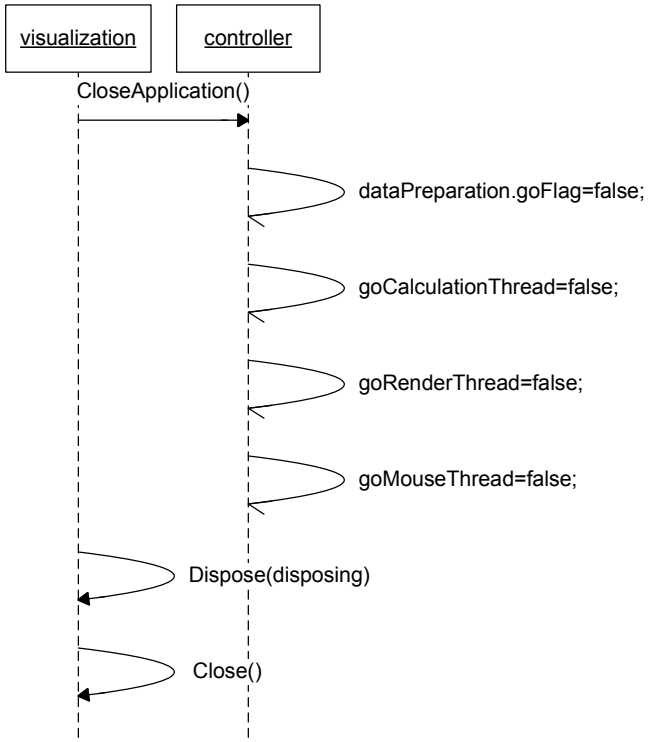
stoppt, wird einfach der intervalTimer angehalten.

Event Step



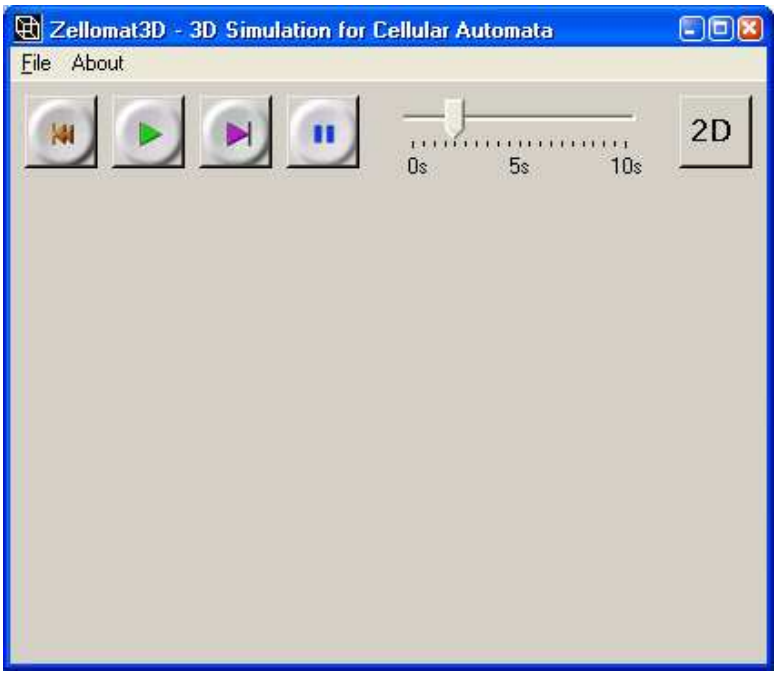
Bei dem Step Event wird auch zuerst der intervalTimer gestoppt. Dann holt sich der Renderer ein neues Frame.

Applikation beenden



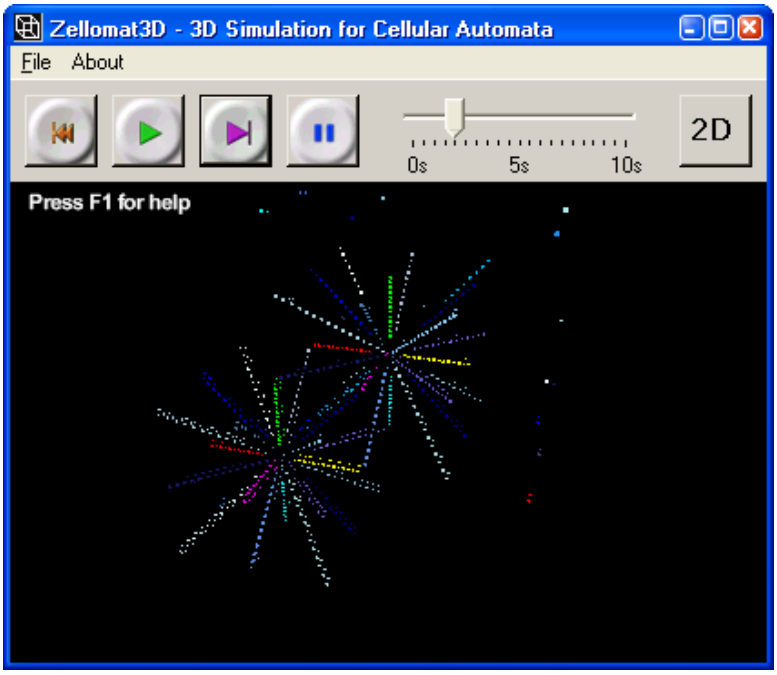
Der von der Visualization aufgefangene Befehl wird an den Controller weiter geleitet. Dort werden alle Flags der Threads so gesetzt, dass sie nicht mehr weiter laufen. Danach schliesst die Visualization das Fenster.

14. Externes Design



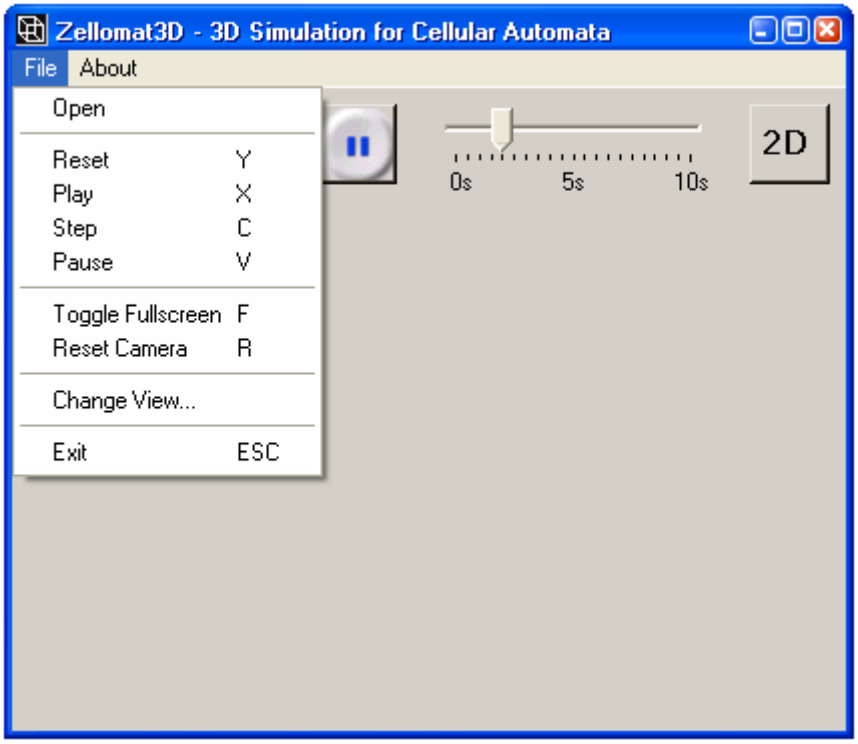
Hauptfenster

Nach dem Starten des Programms erscheint dieses Fenster. Es ist das Hauptfenster des Zellomat3D.



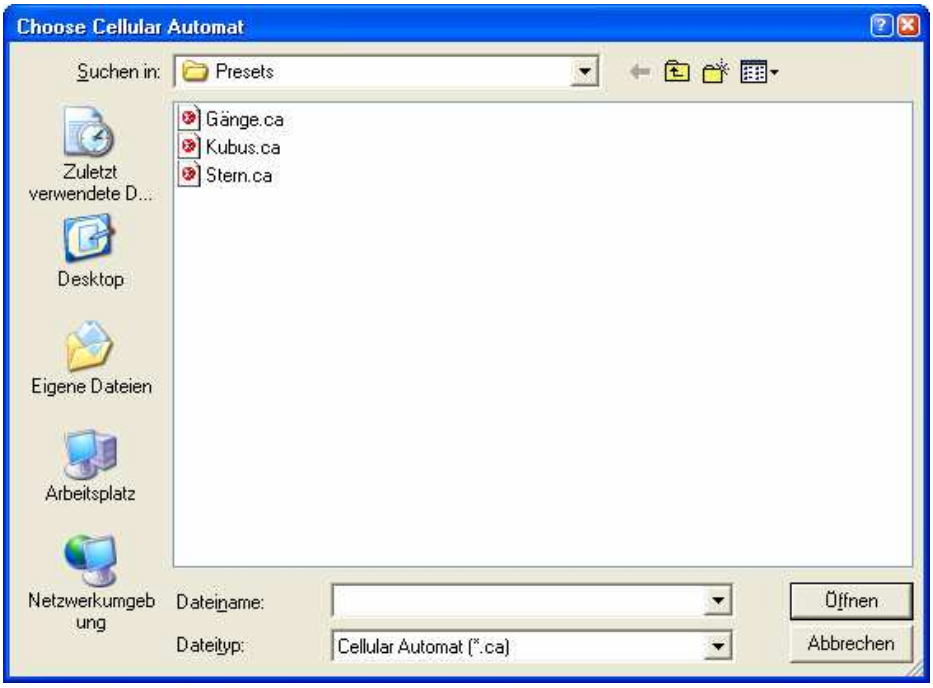
Auf diesem Bild ist ersichtlich, dass der untere Teil des Fensters die Ausgabe der dreidimensionalen Darstellung des ZA enthält. Mit den Knöpfen oberhalb dieser Darstellung lässt sich die Ausgabe steuern.

Menü Auswahl

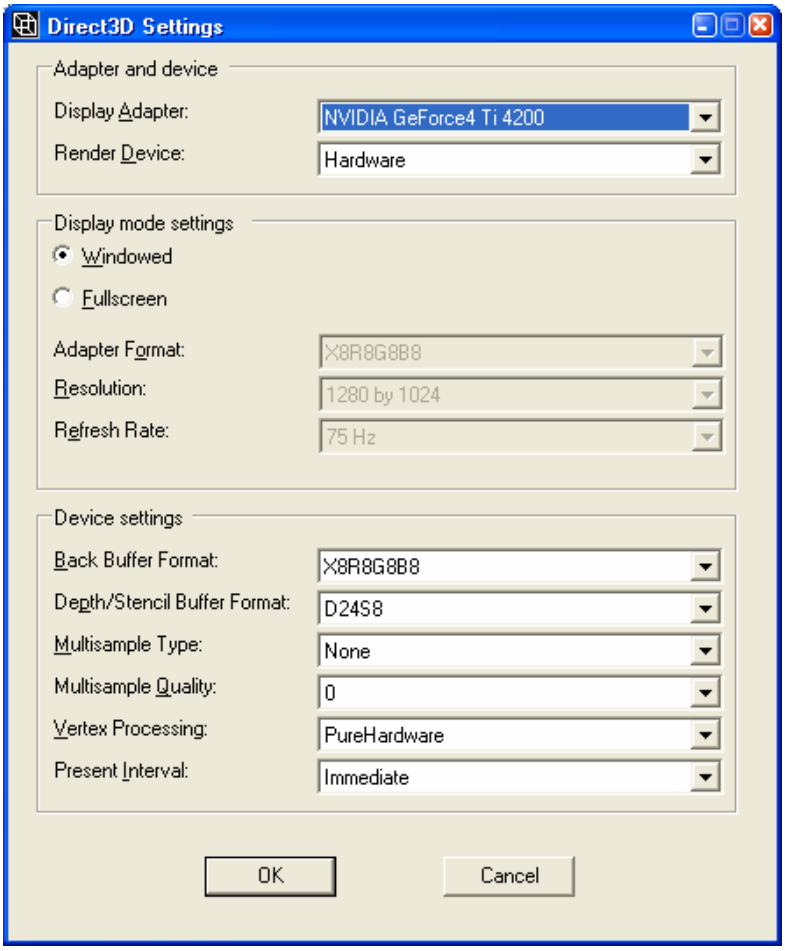


Dieses Bild zeigt die verschiedenen Menüpunkte, die zur Auswahl stehen.

Open Dialog



Dieses Fenster erschein nach dem Betätigen des Menüpunktes File Open. Es dient zum Öffnen eins ZA. Es werden nur Dateien mit der Endung „*.ca“ angezeigt.



Direct3D Settings

Nach dem Anwählen von File Change View erscheint diese Fenster. In diesem Fenster können Einstellungen der Grafikkarte gemacht werden. Alle Änderungen werden nach dem Drücken des OK Knopfs übernommen.

About

Dieses Fenster öffnet sich nach dem Klicken des About-Menüpunktes. Es beinhaltet den Auftraggeber sowie die Kontaktadressen der Erbauer.