Pflichtenheft

StandUp
VCIC02
20.12.2012

1 Management- und Dokumentationsattribute

Dokumentationsattribute	
Autor	Hillebrand, Khalifa, Ferchland
Eindeutige Teamnummer	VCIC02
Quelle	
Version	1.0
Bearbeitungsstatus	Abgeschlossen
Managementattribute	
Priorität	hoch
Stabilität	gefestigt
Kritikalität	hoch
Entwicklungsrisiko	hoch

2 Visionen und Ziele

/PV009/ *„StandUp" ist ein Weckersystem mit innovativer Mensch-Maschine-Schnittstelle.*
/PV010/ *Das System soll die Nutzer wecken, ohne dass er ein Eingabegerät benutzt.*
/PV020/ *Das System ist komplett in den Haushalt/Raum integriert.*
/PV021/ *Das System erlangt umfangreiche Kenntnisse über die Situation des Nutzers und den Raum.*
/PV022/ *Das Produkt kann Signale an externe Geräte (Kaffeemaschine, Radio, Toaster,…) senden.*
/PV030/ *Das System ermöglicht einen angenehmen Start in den Tag.*
/PV040/ *Die Weckfunktion wird vollautomatisch deaktiviert.*
/PV050/ *Das Produkt ist ansprechend gestaltet.*
/PV060/ *Der Nutzer kann schnell und mühelos die aktuelle Uhrzeit vom System darstellen lassen.*
/PV065/ *Die Weckzeit lässt sich durch den Nutzer über ein ergonomisch gestaltetes UI einstellen.*

/PZ010/ *Das System soll den Nutzer durch ein akustisches Signal sanft aufwecken und kann am Verhalten des Nutzers feststellen ob dieser den „Wach"-Zustand erreicht hat.*
/PZ020/ *Das System erkennt eine oder mehrere Personen und deren Schlafplätze.*
/PZ025/ *Das System soll zwischen einer schlafenden, aufgestandenen, schlummernden (wieder hingelegt) und wachen Person unterscheiden können. Dazu wird eine stationäre Kamera verwendet.*

/PZ030/ *Es gibt einen „Schlummer"-Modus, der entweder durch erneutes hinlegen oder, wenn der Benutzer noch im liegenden Zustand ist, durch eine vorher definierte Geste, aktiviert wird.*
/PZ035/ *Falls der Benutzer innerhalb einer gewissen Zeitspanne vor dem Wecksignal bereits aufgestanden ist, wird das Wecken unterlassen.*
/PZ037/ *Das Wecksignal ist gut hörbar, aber hat einen für den Menschen angenehmen Klang.*
/PZ040/ *Innerhalb einer vordefinierten Zeitspanne t0 nach dem Wecken wartet die Software auf das Aufstehen des Benutzers und stellt dann das Wecksignal ein.*
/PZ045/ *Nach dem Aufstehen wird eine weitere Zeit t1 abgewartet. Wenn der Benutzer sich während dieser Zeit nicht wieder hinlegt, wird die gesamte Wecker-Funktionalität inkl. „Schlummer"-Modus deaktiviert. Der Benutzer gilt als vollständig wach.*
/PZ050/ *Der Wecker verfügt über ein Display.*
/PZ051/ *Auf dem Display wird die Uhrzeit in verschiedenen abwechslungsreichen dreidimensionalen Visualisierungen dargestellt.*
/PZ052/ *Das System kann den Raum mit angenehmen Farben ausleuchten.*
/PZ053/ *Das Farbschema kann abhängig von der Uhrzeit verändert werden.*
/PZ060/ *Uhrzeit, Weckzeit und Restschlafzeit können in der Grundansicht des Displays einfach abgelesen werden.*
/PZ061/ *Die Steuerung erfolgt alternativ über Gesten, Touchscreen, Tasten sowie Spracherkennung.*
/PZ064/ *Die verschiedenen Ansichten befinden sich auf den Seiten eines Würfels.*
/PZ065/ *Es gibt drei Ansichten: Visualisierung der Uhr (Grundansicht), Einstellung (Weckzeit/Uhr) und Videostream (Konfiguration/Kalibrierung).*

3 Rahmenbedingungen

/PR010/ *Das Produkt soll als Haushaltsgegenstand hauptsächlich an Schlafplätzen genutzt werden.*
/PR020/ *Das Produkt richtet sich an alle Personengruppen außer bewegungsunfähige Menschen.*
/PR030/ *Das Produkt darf von maximal zwei Personen gleichzeitig benutzt werden.*
/PR040/ *Die Nutzung des Produkts erfolgt im Dauerbetrieb.*
/PR050/ *Der Produkt läuft auf einem Personal Computer mit Windows.*
/PR060/ *Das System an dem das Softwareprodukt genutzt wird verfügt über eine Plug-and-Play Kamera und ausreichende Leistung für die Echtzeitverarbeitung.*
/PR070/ *Das Produkt operiert unter einer gleichbleibenden Helligkeit.*
/PR080/ *Es ist davon auszugehen, dass das Gesicht und der Körper des Nutzers nur in einem kleinen Teil der aufgenommen Videoframes direkt sichtbar sind, da er von der Bettdecke verdeckt sein kann oder nicht der Kamera zugewandt ist.*

4 Kontext und Überblick

Die relevante Systemumgebung (Kontext) und Überblick über das Softwareprodukt.

/PK010/ Die Software ist in C/C++ implementiert und soll unter Windows ausführbar sein.
/PK015/ Das System verfügt über einen Bildschirm, eine Kamera (YUY2_320x240), Lautsprecher, Maus und Tastatur.
/PK020/ Die zugrundeliegende Hardware muss OpenGL fähig sein.

5 Funktionale Anforderungen

/PF010/ *Das System muss zu einer bestimmten Uhrzeit (Weckzeit) ein akustisches Signal (Wecksignal) abspielen.*
/PF011/ *Das System muss dem Nutzer die Möglichkeit bieten die Weckzeit einszustellen*
/PF020/ *Das System muss Bewegungen in verschiedenen Arealen erkennen können, um den Zustand der Objekte(schlafend, schlummernd, aufgestanden, wach, etc.) zu ermitteln. Hierzu werden Bildverarbeitungsalgorithmen eingesetzt.*
/PF030/ *Das System muss auf jeden erkannten Zustand der Person entsprechend reagieren*
/PF031/ *Das System muss mit dem Personenzustand „schlafend“ initialisiert werden, wenn die aktuelle Zeit mit der Weckzeit übereinstimmt und der Alarm aktiviert wurde*
/PF032/ *Ist der Zustand der Person „schlafend“ und wurde die Weckzeit erreicht, soll das Wecksignal ertönen.*
/PF033/ *Das System muss den in den Personenzustand „aufgestanden“ wechseln, wenn Bewegungen im oberen Bereich des des beobachteten Areals registriert werden*
/PF034/ *Ist der Zustand der Person „aufgestanden“ wird das Wecksignal pausiert*
/PF035/ *Das System muss in den Personenzustand „schlummernd“ wechseln, wenn der Personenzustand zuvor „aufgestanden“ war und nur Bewegungen im unteren Bereich (Bett) des beobachteten Areals registriert werden*
/PF036/ *Ist der Zustand der Person „schlummernd“ wird das Wecksingal wieder abgespielt*
/PF037/ *Das System muss in den Personenzustand „wach“ wechseln, wenn der Personenzustand zuvor „aufgestanden“ war und in einer vordefinierten Zeitspanne keine Bewegungen registriert wurden*
/PF038/ *Ist der Zustand der Person „wach“ und wird das Wecksignal gestoppt*
/PF040/ *Das System muss ein intuitiv bedienbares Steuerelement darstellen, um die Alarmzeit einzustellen*
/PF050/ *Der Benutzer hat in jedem Zustand des Systems die Möglichkeit die Uhrzeit abzulesen*
/PF053/ *Das Farbschema kann abhängig von der Uhrzeit verändert werden.*
/PF060/ *Uhrzeit und Weckzeit können in der Alarmansicht des Systems einfach abgelesen werden.*
/PF064/ *Die verschiedenen Ansichten des Systems befinden sich auf den Seiten eines Würfels.*
/PF065/ *Es gibt drei Ansichten: Visualisierung der Uhr (Grundansicht), Einstellung (Weckzeit/Uhr) und Videostream (Konfiguration/Kalibrierung).*

6 Qualitätsanforderungen

Systemqualität	Sehr gut	Gut	Normal	Nicht relevant
Funktionalität	X			
Zuverlässigkeit	X			
Benutzbarkeit	X			
Effizienz		X		
Wartbarkeit			X	
Portabilität		X		

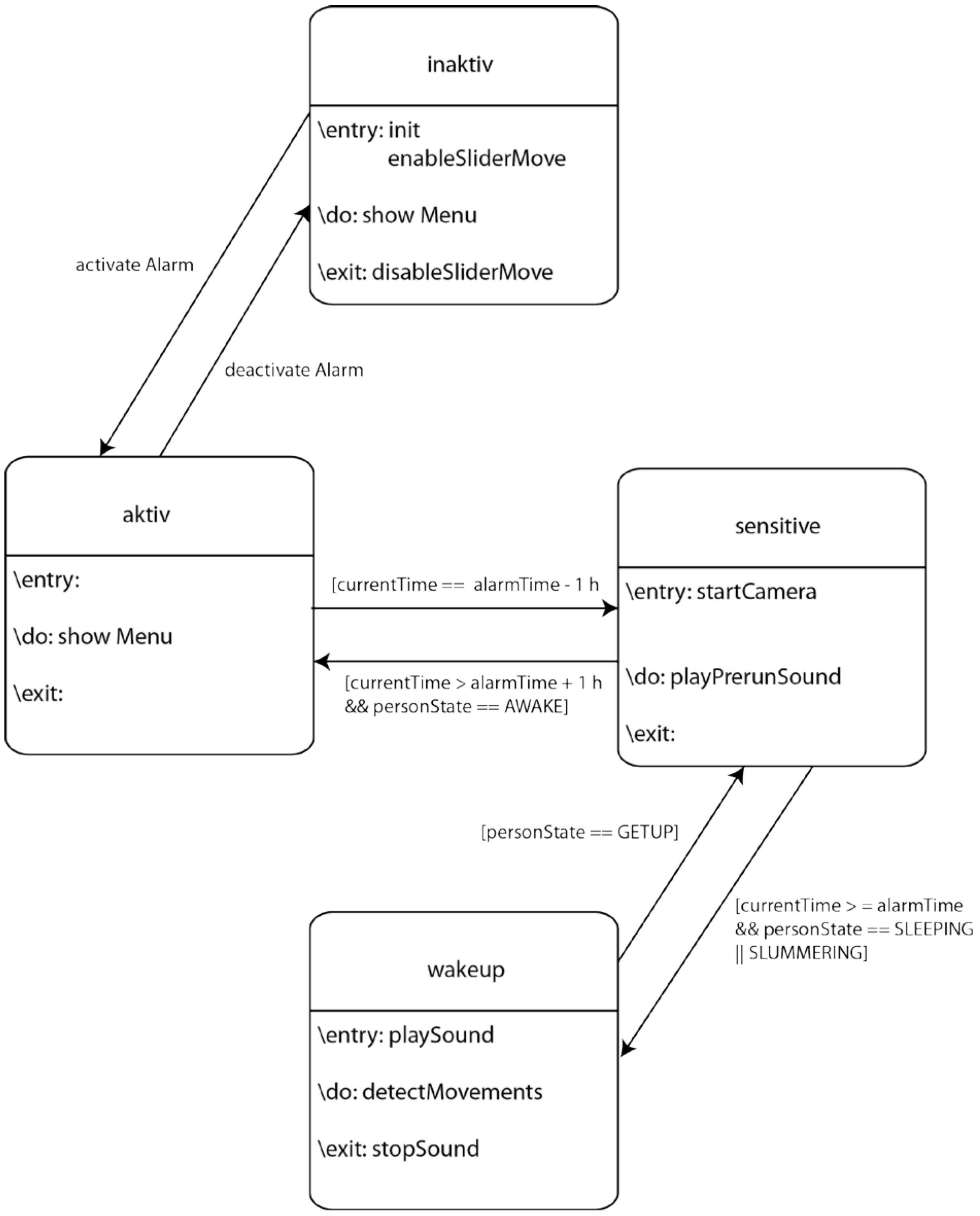
Tabelle : Qualitätsanforderungen

/PQBE10/ *Die Weckzeit lässt sich über ein Steuerelement (Zeitstrahl) einstellen.*
/PQFU10/ *Der Lautstärkepegel des Wecksignals kann über einen bestimmten Zeitraum ansteigen, um ein sanftes Weckerlebnis zu leisten*
/PQFU20/ *Das Wecksignal ist zwar gut hörbar, aber hat einen für den Menschen angenehmen Klang.*

7 Abnahmekriterien

/PAK010/ *Das System muss nach der Aktivierung der Alarmzeit das Steuerelement sperren, sodass die Weckzeit nicht mehr verstellt werden kann.*
/PAK020/ *Das System muss das Wecksignal beim Erreichen der Weckzeit abspielen.*
/PAK030/ *Das System muss das Wecksignal beenden wenn die Person aufgestanden ist.*
/PAK040/ *Die Uhrzeit muss korrekt dargestellt werden.*

8 Subsystem



/ZD010/ Zustandsdiagramm des Weckersystems

9 Glossar

Personenstatus – Der Zustand, in dem sich die zu weckende Person befindet. Es gibt 4 Zustände: schlafend, aufgestanden, schlummernd, wach.
Weckzeit – Die Uhrzeit, zu der das Wecksignal gestartet werden soll.

10 Literatur

Wenn Sie Literatur oder andere Quellen verwendet haben, dann führen Sie diese in diesem Abschnitt auf und verweisen an entsprechender Stelle in diesem Dokument darauf.

10.1 Hinweis zu dieser Vorlage

Die Vorlage für dieses Pflichtenheft wurde Balzert (2009), S. 492 ff. entnommen.

10.2 Literaturliste

Balzert, Helmut (2009). Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering. 3. Auflage. Heidelberg: Spektrum, Seite 492 ff.

11 Testdokumentation

Testfall	Sperren des Steuerelements bei Aktivierung		
Verantwortlich	Hady Khalifa		
Vorraussetzungen			
1.	Das Programm ist gestartet (Dialog1)		
2.	Alarm ist deaktiviert		
Schritte		**Erwartetes Ergebnis**	**Erhaltenes Ergebnis**
1.	Maus in den linken Bereich des Displays bewegen	Unsichtbarer Button erscheint und ist orange	wie erwartet
2.	Linksklick auf den Button	Animation startet: Yrotation und Rechtsbewegung; Dialog2 erscheint	wie erwartet
3.	Stellen der Weckzeit über die Sliderkomponente	Der Button wird unsichtbar; Alarmzeit bewegt sich mit dem Slider-Thumb; RestSchlafzeit wird dargestellt;	wie erwartet
4.	Linksklick auf Aktivierungs-Checkbox	Checkbox wird mit "X" gefüllt; unteres Textfeld ändert sich zu aktiviert	wie erwartet
5.	Stellen der Weckzeit über die Sliderkomponente	Slider-Thumb ist fixiert; lässt sich nicht einstellen	wie erwartet
Testergebnis	passed		
Testdurchführung			
von	Hady Khalifa		
Datum, Uhrzeit	20.12.2012, 16:05 Uhr		

Testfall	Hintergrund zeigt richtige Farbe je nach Tageszeit an		
Verantwortlich	Roman Hillebrand		
Vorraussetzungen			
1.	Programm ist gestartet		
2.	Faktor für Zeitraffer ist eingestellt (Beschleunigung des Testverlaufs, 1 min entspricht 1 Tag)		
Schritte		**Erwartetes Ergebnis**	**Erhaltenes Ergebnis**
1.	Änderung der Farbe beobachten	Stufenloser, sich wiederholender Übergang	wie erwartet
2.	Angezeigte Farbe mit vorher festgelegter Farbe vergleichen	Gleicher Farbton	wie erwartet
Testergebnis	passed		
Testdurchführung			
von	Roman Hillebrand		
Datum, Uhrzeit	19.12.2012, 12:00 Uhr		

Testfall	Wecker verhält sich nach Vorgabe		
Verantwortlich	Roman Hillebrand		
Vorraussetzungen			
1.	[Programm ist gestartet		
2.	Weckzeit ist eingestellt		
3.	Wecker ist aktiviert		
4.	Vorlaufzeit auf 5 sek. Und Snooze auf 30 sek. Eingestellt		
Schritte		**Erwartetes Ergebnis**	**Erhaltenes Ergebnis**
1.	Warten auf Anfang der Vorlaufzeit	Event wird 5 sek. vor Weckzeit ausgelöst	Event wird ca. 30 sek. zu spät ausgelöst
2.	Warten auf Wecksignal	Weck-Event wird 5 sek. später ausgelöst	wie erwartet
3.	Signal senden: Person steht auf	Event "Wecken stoppen" wird sofort ausgelöst	wie erwartet
4.	Warten auf Ablauf der Snooze-Zeit	Event "alles abgeschlossen" wird ausgelöst	wie erwartet
		Zustand des Weckers springt zurück auf "aktiv"	wie erwartet
Testergebnis	failed		
Testdurchführung			
von	Roman Hillebrand		
Datum, Uhrzeit	19.12.2012, 14:00 Uhr		

Testfall	3d-Uhr verhält sich korrekt		
Verantwortlich	Roman Hillebrand		
Vorraussetzungen			
1.	[Programm ist gestartet		
2.	Uhr zum Vergleich steht bereit		
Schritte		**Erwartetes Ergebnis**	**Erhaltenes Ergebnis**
1.	Beobachten der Sekunden-, Minuten- und Stundenzeiger	stimmt mit Vergleichswert überein	wie erwartet
2.	Drehung der einzelnen Komponenten über den Mittelpunkt beobachten	stimmt mit Vorgabe überein	wie erwartet
Testergebnis	passed		
Testdurchführung			
von	Roman Hillebrand		
Datum, Uhrzeit	19.12.2012, 16:00 Uhr		

Testfall	Matlab Bewegungserkennung		
Verantwortlich	Hans Ferchland		
Vorraussetzungen			
1.	Programm gestartet		
2.	Person liegt/schläft		
3.	Szene genügend ausgeleuchtet		
4.	Kamera steht senkrecht zur Bettkante und ist ausgerichtet		
Schritte		**Erwartetes Ergebnis**	**Erhaltenes Ergebnis**
1.	Aufstehen, kurz warten (ca. 2s)	Status auf "up"	wie erwartet
2.	Aus dem Bild, vom Bett weg gehen, warten (ca. 4s)	Status auf "out"	wie erwartet
3.	Wieder ins Bild gehen, vor der Bettkante stehen	Status auf "in"	wie erwartet
4.	Auf die Bettkante setzen	Status auf "down"	wie erwartet
Testergebnis	passed		
Testdurchführung			
von	Hans Ferchland		
Datum, Uhrzeit	20.12.2012, 19:51		

12 Installation und Start des Programms

Eine Installation ist nicht erforderlich.

Zum Starten des Programms muss die „standup_cv.exe" ausgeführt werden.

Verwendete Bibliotheken

Ogre3D wird als Grafikbibliothek verwendet. Ogre3D benötigt die boost-Bibliothek für einige Funktionen.

Für die Darstellung der GUI wird CEGUI verwendet. Die für den gleichzeitigen Einsatz von OGRE und CEGUI nötige Bibliothek wurde aus den Quelldateien erzeugt.

CEGUI verwendet freetype für die Darstellung von Schriften und zlib für das Laden von Bilddateien.

Verwendete Algorithmen aus der Computergrafik

Render-to-texture

Render-to-texture wird eingesetzt, um die Balkendarstellung der Uhr auf einer frei im Raum positionierbaren Fläche zu ermöglichen.

Dazu wird der zur Balkendarstellung gehörige Szenegraph nicht - wie üblich - in den Backbuffer geschrieben, sondern in die Rendertextur.

Diese kann anschließend auf ein Objekt im GUI-Szenegraphen abgebildet werden.

Szenegraph

Der Szenegraph der dreidimensionalen Uhrdarstellung ist, vereinfacht dargestellt, wie folgt aufgebaut:

Umgebung

|

Stundenzeiger

|

Minutenzeiger

|

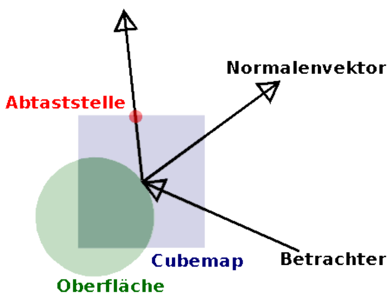
Sekundenzeiger

Die Transformation ist eine Drehung um einen gemeinsamen Mittelpunkt.

Da die Drehung der Objekte weiter unten im Szenegraph von der der weiter oben liegenden abhängt, ergibt sich ein schöner dreidimensionaler Effekt.

Cubemaps

Für eine schönere Darstellung der dreidimensionalen Uhr werden für die Oberflächen der Uhr Cubemap-Reflexionen verwendet.



Cubemap-Reflexionen

Eine Cubemap kann man sich als ein Konstrukt aus 6 Texturen vorstellen. Diese Texturen sind Seitenflächen eines Würfels angebracht.

Die Cubemap wird nicht, wie bei einer einfachen cubemap üblich, mit Hilfe von u-v-Koordinaten abgetastet, sondern über einen Vektor mit 3 Komponenten.

Dieser Vektor zeigt von der Mitte des Würfels nach aussen. Die Stelle, an der dieser Vektor den Würfel schneidet, markiert den Bildpunkt, der abgetastet wird.

Für den Zweck der Darstellung einer Reflexion an einer Oberfläche wird in der Cubemap ein 360°-Panorama der Umgebung gespeichert.

Der Vektor von der Kamera zu dem zu zeichnenden Punkt auf der Oberfläche wird an dem Normalenvektor der Oberfläche gespiegelt.

Das Resultat wird als Parameter für die Abtastfunktion der Cubemap verwendet.

Skybox

Die Skybox stellt weit entfernte Objekte auf als Texturen auf einem Würfel dar, dessen Mittelpunkt sich immer genau auf dem Betrachtungspunkt befindet.

Für den Betrachter erzeugt dies den Eindruck einer dreidimensionalen Darstellung des Hintergrunds.

Eine Skybox ist eine sehr performante und einfache Möglichkeit, einer Szene einen Hintergrund zu geben.

Zusammen mit den Cubemap-Reflexionen kann ein sehr glaubwürdiges Bild erzeugt werden.

Stencil Shadows

Um die Schatten der Uhr auf dem Boden sowie die Schatten von Komponten der Uhr auf darunterliegenden Komponenten darstellen zu können, wird der "stencil shadow"-Algorithmus verwendet.

Der Algorithmus setzt vorraus, das alle Schatten werfende Objekte eine geschlossene Hülle haben.

Zunächst werden die Konturen des Schatten werfenden Objektes (der Uhr) vom Betrachtungsstandpunkt der Lichtquelle aus gefunden -

Wenn die Skalarprodukte der Normalenvektoren zweier benachbarter Polygone "n1" und "n2" mit dem Vektor von einem Punkt auf der Kante zwischen den Polygonen zur Position der Lichtquelle "l" zwei Resultate mit unterschiedlichem Vorzeichen liefern, wird diese Kante als zur Kontur gehörig gezählt.

Die entstehende (geschlossene) Konturlinie wird nun von der Lichtquelle weg extrahiert und vorne und hinten verschlossen, um ein Schattenvolumen zu erzeugen.

Um dieses Volumen sichtbar zu machen, wird die Szene in folgenden Schritten gerendert ("depth fail"-Methode)

(1) Die Szene wird gezeichnet, als wären alle Oberflächen im Schatten.

(2) Schreiben in den Farb-und Tiefenpuffer wird deaktiviert, Schreiben in die Stencil-Maske wird aktiviert

(3) Front-face-culling wird benutzt

(4) Die Stencil-Operation wird auf "increment on depth fail" gesetzt

(5) Das Schattenvolumen wird gezeichnet.

(6) Back-face-culling wird benutzt

(7) Die Stencil-Operation wird auf "decrement on depth fail" gesetzt

(8) Nochmaliges Zeichnen des Schattenvolumens.

(9) Die Szene wird nocheinmal gezeichnet (diesmal mit Licht), die in den Stencilbuffer geschriebenen Werte werden dabei als Maske verwendet

Verwendete Algorithmen aus der Bildverarbeitung (matlab)

Gesichtserkennung mit CV System Toolbox

-bild, video, webcam

-vision.VideoFileReader(), vision.VideoPlayer(), vision.CascadeObjectDetector('UpperBody'), vision.CascadeObjectDetector('Nose'), step(noseDetector,videoFrame)

Optical Flow Analysis

-Beispiel von Matlab Hilfe angepasst

-->http://www.mathworks.de/de/help/vision/examples/tracking-cars-using-optical-flow.html

Bilddifferenz in Grauwertbild (Webcam)

-Bewegung (Änderungen/Differenzen erkennen)

-Bewegung in verschiedenen Bildbereichen

Bilddifferenz im Kantenbild (Webcam)

-Matlab Funktionen imabsdiff(), wiener2()

-gutes Ergebnis, nutzbar!

-prozentualer Bewegungsanteil links, oben und rechts

-Schwellwerte um Bewegung zu indentifitieren

Kantenerkennung in der Zeit (Webcam)

-Sobel Filter über 3 Bilder in Folge, timeFilter()

-auf Kantenbild

-gutes Ergebnis aber nicht nutzbar => verworfen

Segmentierung über Boundary Detection

->zunächst auf ganzem Bild

->Abgewandelt auf Kantenbild

->matlab funktionen edge(), imfill(), strel(), bwperim(), bwtraceboundary(), boundaries()

-->http://www.mathworks.de/de/help/images/examples/detecting-a-cell-using-image-segmentation.html

Finaler Algorithmus

-Bilddifferenz auf Kantenbild

-Segmentierung des Kantenbildes

-abwägen von segmentierter bewegung um genauere entscheidung zu treffen

-GUI zeigt verschiedene Stati der Person und erkannte Bewegungsmuster

Quellcode-Listing Computergrafik

CLOCKVISUALIZATIONBARS.H

```
/// \file src\ClockVisualizationBars.h
///
/// \brief Declares the clock visualization bars class.

#ifndef CLOCK_VISUALIZATION_BARS_H

#define CLOCK_VISUALIZATION_BARS_H

#include "ClockVisualisation.h"

/// \class ClockVisualizationBars
///
/// \brief The class ClockVisualizationBars displays a clock via three bars for hours, minutes
and seconds.
///
///    The visualization displays three bars for each hours, minutes and seconds. The
visualization itself
/// rotates to the camera. The scene also contains a light that orients itself to the mouse
position in 3D.
///
/// \sa ClockVisualisation, Ogre::FrameListener
/// \author Hans Ferchland
/// \date 19.12.2012

class ClockVisualizationBars : public Ogre::FrameListener, public ClockVisualization
{
public:

       /// \fn ClockVisualizationBars::ClockVisualizationBars(Ogre::SceneManager*
sceneManager,
       /// Ogre::Camera* cam, int hourFormat = 1);
       ///
       /// \brief Constructor of the bars visualization.
       ///
       ///    Takes a SceneManager from Ogre to hang nodes into the scenegraph and a camera
that
       /// will be used to orient the clock visualization to it.
       ///    The viualization can handle 12 and 24 hour format.
       ///
       /// \author Hans Ferchland
       /// \date 19.12.2012
       ///
       /// \param [in,out] sceneManager
       /// If non-null, manager for scene. Gets access to the scene graphs main root node.
       /// \param [in,out] cam
       /// If non-null, the camera that will view the scene.
       /// \param hourFormat
       /// (optional) the hour format where 1 is 12h format and 2 is 24h format.
       ///    \sa Ogre::SceneManager, Ogre::Camera

       ClockVisualizationBars(Ogre::SceneManager* sceneManager, Ogre::Camera* cam, int
hourFormat = 1);

       /// \fn bool ClockVisualizationBars::frameRenderingQueued(const Ogre::FrameEvent& evt);
       ///
       /// \brief Frame rendering for updateing from Ogre3D via FrameListener interface.
       ///
       ///    Updates the clock every second. Updates the camera and animation of the clock.
       ///
       /// \author Hans Ferchland
       /// \date 19.12.2012
       ///
       /// \param evt
       /// The frame event from Ogre3D.
       ///
       /// \return true if rendering should continue, false otherwise.
       ///    \sa Ogre::FrameEvent

       bool frameRenderingQueued(const Ogre::FrameEvent& evt);
```

```
protected:

	/// \fn void ClockVisualizationBars::createClock();
	///
	/// \brief Creates the bar clock visualization with all components.
	///
	///    Creates the nodes and hangs them into scenegraph. Creates materials and
	/// geometry and attaches them to the nodes.
	///
	/// \author Hans Ferchland
	/// \date 19.12.2012

	void createClock();

private:
	Ogre::SceneNode* mClockNode;  /*!< The clock node */
	Ogre::SceneNode* mHoursBarsNode;     /*!< The hours bars node */
	Ogre::SceneNode* mMinutesBarsNode;  /*!< The minutes bars node */
	Ogre::SceneNode* mSecondsBarsNode;  /*!< The seconds bars node */
	Ogre::SceneNode* mDebugNode;  /*!< The debug node */

	// references the ogre basic camera
	Ogre::Camera* mCamera;  /*!< The camera */
	Ogre::Light* mLight;  /*!< The light */

};

#endif // !CLOCK_VISUALIZATION_BARS_H
CLOCKVISUALIZATIONBARS.CPP


#include "stdafx.h"

#include "StandupApplication.h"
#include "ClockVisualisation.h"
#include "ClockVisualizationBars.h"

ClockVisualizationBars::ClockVisualizationBars(Ogre::SceneManager* sceneManager, Ogre::Camera*
cam, int hourFormat) :
	ClockVisualization(sceneManager, hourFormat), mCamera(cam) {
	// set max values
	mHours = 12 * mHourFormat;
	mMinutes = 60;
	mSeconds = 60;
	// creates the clock
	createClock();
	mLight = mSceneMgr->createLight("light1");
	mLight->setType(Ogre::Light::LT_POINT);
	mLight->setDiffuseColour(0.8,0.8,0.8);
	//mLight->setPosition
}

bool ClockVisualizationBars::frameRenderingQueued(const Ogre::FrameEvent& evt) {
	// counter for slowed clock update (needs only once per second ;)
	static int second=-1;
	static Ogre::Vector3& dir = Ogre::Vector3();
	static Ogre::Ray& ray = Ogre::Ray();
	// get current time from clock
	const tm& localTime = Clock::getDisplayTime(Clock::getCurrentSecond());
	float dayInterpolationTime = localTime.tm_hour + (localTime.tm_min * 0.01667f);
	// get the current secs, mins and mHours
	mCurrentSeconds = localTime.tm_sec;
	mCurrentMinutes = localTime.tm_min;
	mCurrentHours = localTime.tm_hour % (12 * mHourFormat); // in right time format (12 vs
24)

	if(mCurrentSeconds!=second) {
		second = mCurrentSeconds;

		float hrsScale = (float) mCurrentHours/(float) mHours;
		float minsScale = (float) mCurrentMinutes/(float) mMinutes;
		float secsScale = (float) mCurrentSeconds/(float) mSeconds;

		mHoursBarsNode->setScale(0.1f, 0.1f, hrsScale);
		mMinutesBarsNode->setScale(0.1f, 0.1f, minsScale);
		mSecondsBarsNode->setScale(0.1f, 0.1f, secsScale);

		mHoursBarsNode->setPosition(0,12,(hrsScale * 50)-49);
```

```
			mMinutesBarsNode->setPosition(0,0,(minsScale * 50)-49);
			mSecondsBarsNode->setPosition(0,-12,(secsScale * 50)-49);
		}

		// mouse orientation
		OIS::MouseState state = StandupApplication::getInstance()->getMouse()->getMouseState();
		ray = Ogre::Ray();
		float x = state.X.abs;
		float y = state.Y.abs;
		float relX = x/(float)state.width;//mCamera->getViewport()->getWidth();
		float relY = y/(float)state.height;//mCamera->getViewport()->getHeight();
		relX = (1-relX);
		relY = (1-relY);
		mCamera->getCameraToViewportRay(relX * mCamera->getViewport()->getWidth(),
			relY * mCamera->getViewport()->getHeight(), &ray);
		dir = Ogre::Vector3(ray.getDirection().x, ray.getDirection().y, ray.getDirection().z);
		dir.normalise();
		Ogre::Vector3 left = dir.crossProduct(mCamera->getUp());
		left.normalise();
		//dir *= -1;
		mLight->setPosition(ray.getOrigin());
		mClockNode->setOrientation(
			Ogre::Quaternion(
			dir, mCamera->getUp(), left));
			/*mClockNode->setOrientation(Ogre::Quaternion(Ogre::Radian(Ogre::Math::PI * 0),
			Ogre::Vector3(0,1,1)));*/
		return true;
}

void ClockVisualizationBars::createClock() {
		Ogre::SceneNode* rootNode = mSceneMgr->getRootSceneNode();
		mClockNode = rootNode->createChildSceneNode("MainBarsClockNode");
		//mDebugNode = rootNode->createChildSceneNode("DebugNode");
		mClockNode->setPosition(0,0,0);
		mClockNode->setScale(0.66f, 0.66f, 0.66f);
		/*mClockNode->setOrientation(Ogre::Quaternion(Ogre::Radian(Ogre::Math::PI),
			Ogre::Vector3(0,1,1)));*/
		/*mClockNode->setOrientation(Ogre::Quaternion(Ogre::Radian(Ogre::Math::PI * 0.5f),
		Ogre::Vector3::NEGATIVE_UNIT_Z));*/

		mHoursBarsNode = mClockNode->createChildSceneNode("HoursBarsNode");
		mMinutesBarsNode = mClockNode->createChildSceneNode("MinutesBarsNode");
		mSecondsBarsNode = mClockNode->createChildSceneNode("SecondsBarsNode");

		Ogre::String hoursMaterialName = "Standup/Clock/Hours";
		Ogre::String minutesMaterialName = "Standup/Clock/Minutes";
		Ogre::String secondsMaterialName = "Standup/Clock/Seconds";
		Ogre::String backgroundMaterial = "Standup/Clock/Background";


		Ogre::Entity* tempGeometry;

		// DEBUG
		//tempGeometry = mSceneMgr->createEntity(Ogre::SceneManager::PT_CUBE);

		//mDebugNode->attachObject(tempGeometry);
		//mDebugNode->setScale(0.01,0.3,0.01);
		// Hours
		tempGeometry = mSceneMgr->createEntity(Ogre::SceneManager::PT_CUBE);
		tempGeometry->setMaterialName(hoursMaterialName);

		mHoursBarsNode->attachObject(tempGeometry);
		mHoursBarsNode->setPosition(0, 12, 0);

		// Minutes
		tempGeometry = mSceneMgr->createEntity(Ogre::SceneManager::PT_CUBE);
		tempGeometry->setMaterialName(minutesMaterialName);

		mMinutesBarsNode->attachObject(tempGeometry);
		mMinutesBarsNode->setPosition(0, 0, 0);

		// Seconds
		tempGeometry = mSceneMgr->createEntity(Ogre::SceneManager::PT_CUBE);
		tempGeometry->setMaterialName(secondsMaterialName);

		mSecondsBarsNode->attachObject(tempGeometry);
		mSecondsBarsNode->setPosition(0, -12, 0);
```

```
        //Background
        Ogre::SceneNode* background;
        // Hours
        background = mClockNode->createChildSceneNode("Background_Hours");

        tempGeometry = mSceneMgr->createEntity(Ogre::SceneManager::PT_CUBE);
        tempGeometry->setMaterialName(backgroundMaterial);

        background->attachObject(tempGeometry);
        background->setScale(0.075f, 0.075f, 1.0f);
        background->setPosition(0,12,0);
        // Minutes
        background = mClockNode->createChildSceneNode("Background_Minutes");

        tempGeometry = mSceneMgr->createEntity(Ogre::SceneManager::PT_CUBE);
        tempGeometry->setMaterialName(backgroundMaterial);

        background->attachObject(tempGeometry);
        background->setScale(0.075f, 0.075f, 1.0f);
        background->setPosition(0,0,0);
        // Seconds
        background = mClockNode->createChildSceneNode("Background_Seconds");

        tempGeometry = mSceneMgr->createEntity(Ogre::SceneManager::PT_CUBE);
        tempGeometry->setMaterialName(backgroundMaterial);

        background->attachObject(tempGeometry);
        background->setScale(0.075f, 0.075f, 1.0f);
        background->setPosition(0,-12,0);
        //mHoursBarsNode->setScale(0.1f, 0.1f, 1.0f);
        //mMinutesBarsNode->setScale(0.1f, 0.1f, 1.0f);
        //mSecondsBarsNode->setScale(0.1f, 0.1f, 1.0f);

        //mHoursBarsNode->setPosition(10,12,0);
        //mMinutesBarsNode->setPosition(10,0,0);
        //mSecondsBarsNode->setPosition(10,-12,0);
}
CLOCKVISUALIZATION.H
/////////////////////////////////////////////////////////////////////////////////////////////
//////
/// \file src\ClockVisualisation.h
///
/// \brief Declares the clock visualisation base class.
/////////////////////////////////////////////////////////////////////////////////////////////
//////

#ifndef CLOCK_VISUALISATION_H

#define CLOCK_VISUALISATION_H

/////////////////////////////////////////////////////////////////////////////////////////////
//////
/// \class ClockVisualization
///
/// \brief Clock visualization base class for diffrent visualizations of a clock.
///
/// \author Hans Ferchland
/// \date 19.12.2012
/////////////////////////////////////////////////////////////////////////////////////////////
//////

class ClockVisualization
{
public:

        /////////////////////////////////////////////////////////////////////////////////////////////
/////////////
        /// \fn ClockVisualization::ClockVisualization(Ogre::SceneManager* sceneManager,
        /// int hourFormat = 1)
        ///
        /// \brief Constructor of the visualization.
        ///
        ///     Takes a SceneManager from Ogre to hang nodes into the scenegraph.
        ///     The viualization can handle 12 and 24 hour format.
        ///
        /// \author Hans Ferchland
        /// \date 19.12.2012
        ///
```

```
	/// \param [in] sceneManager
	/// If non-null, manager for scene. Gets access to the scene graphs main root node.
	/// \param hourFormat
	/// (optional) the hour format where 1 is 12h format and 2 is 24h format.
	////////////////////////////////////////////////////////////////////////////////
////////////

	ClockVisualization(Ogre::SceneManager* sceneManager, int hourFormat = 1)
		: mSceneMgr(sceneManager), mHourFormat(hourFormat) {}
protected:
	Ogre::SceneManager* mSceneMgr;  /*!< Manager for the scene */
	// saves the static values for max hours, mins and secs
	int mHours; /*!< The maximum hours */
	int mMinutes;   /*!< The maximum minutes */
	int mSeconds;   /*!< The maximum seconds */
	// saves the current time and gmt offset
	int mCurrentSeconds;  /*!< The current seconds */
	int mCurrentMinutes;  /*!< The current minutes */
	int mCurrentHours;  /*!< The current hours */
	int mHourFormat;      /*!< The hour format */
};

#endif // !CLOCK_VISUALISATION_H

CLOCK.H


/// \file src\Clock.h
///
/// \brief Declares the Clock, AlarmEventHandler, Person and AlarmClock classes.

#ifndef CLOCK_H
#define CLOCK_H

/// \enum AlarmState
///
/// \brief enums for the AlarmClock state machine.

enum AlarmState { ACTIVE, INACTIVE, MOVEMENT, AWAKENING };

/// \enum PersonState
///
/// \brief enums for the Person state machine.

enum PersonState { SLEEPING, SLUMBERING, GOTUP, AWAKE };

/// \class ClockException
///
/// \brief Exception for signalling clock errors. Inherited from std::exception.
///
/// \author Roman Hillebrand
/// \date 19.12.2012

class ClockException : std::exception
{
	/// \fn std::string ClockException::what()
	///
	/// \brief name of the exception.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \return "ClockException"

	std::string what() { return "ClockException"; }
};

/// \class Clock
///
/// \brief wrapper class for the c library header time.h. static class.
///
/// \author Roman Hillebrand
/// \date 19.12.2012

class Clock
{
	public:
```

```
	/// \fn static const time_t Clock::gmtoff();
	///
	/// \brief get the offset from gmt to local time in seconds.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \return time in seconds

	static const time_t gmtoff();

	/// \fn static const time_t Clock::getCurrentSecond();
	///
	/// \brief get the seconds passed since 1/1/1970 (wrapper for c library)
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \return time in seconds

	static const time_t getCurrentSecond();

	/// \fn static const tm& Clock::getDisplayTime(const time_t &second);
	///
	/// \brief fill a tm structure (wrapper for c library)
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \param second
	/// seconds since 1/1/1970
	///
	/// \return input converted to a tm struct

	static const tm& getDisplayTime(const time_t &second);

	// constants for conversion between seconds and hours or days, respectively

	static const time_t HOUR = (60*60); /*!< seconds per hour */
	static const time_t DAY = (HOUR*24); /*!< seconds per day */


};

/// \class AlarmEventHandler
///
/// \brief abstract class, inherited by CameraTest. Handles events triggered by AlarmClock.
///
/// \author Roman Hillebrand
/// \date 19.12.2012

class AlarmEventHandler
{
public:

	/// \fn virtual void AlarmEventHandler::watchOutEvent() = 0;
	///
	/// \brief triggered one hour before alarm should ring, the camera should start its
prerun at this point.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	virtual void watchOutEvent() = 0;

	/// \fn virtual void AlarmEventHandler::alarmEvent() = 0;
	///
	/// \brief triggered at the exact moment when the alarm should ring.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	virtual void alarmEvent() = 0;

	/// \fn virtual void AlarmEventHandler::stopRingingEvent() = 0;
	///
	/// \brief triggered when the should stop ringing.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	virtual void stopRingingEvent() = 0;

	/// \fn virtual void AlarmEventHandler::everythingCompleteEvent() = 0;
	///
	/// \brief triggered when the wake-up cycle is complete and the alarm state machine has
gone back to its default state.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	virtual void everythingCompleteEvent() = 0;
};

/// \class Person
///
/// \brief Represents one persons state for use with the alarm clock.
///
/// \author Roman Hillebrand
/// \date 19.12.2012

class Person
{
private:

	PersonState state; /*!< the state this person is in */

	time_t lastActionTime;  /*!< Time of this persons last action */

	/// \fn void Person::action()
	///
	/// \brief call this method if something qualified as an action.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	void action() { lastActionTime = Clock::getCurrentSecond(); }

public:

	/// \fn Person::Person()
	///
	/// \brief assume person is sleeping when initializing the state machine.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	Person() { state = SLEEPING; lastActionTime=0; }

	/// \fn const PersonState const Person::getCurrentState()
	///
	/// \brief Gets current state.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \return The current state.

	const PersonState const getCurrentState() { return state; }

	/// \fn const time_t const Person::getLastActionTime()
	///
	/// \brief Gets the time of this persons last action.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \return time in seconds.

	const time_t const getLastActionTime() { return lastActionTime; }

	// state machine implementation

	/// \fn void Person::getUpEvent()
	///
	/// \brief called by computer vision part when a corresponding movement has been 
detected.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \exception ClockException
	/// Thrown when a Clock error condition occurs.

	void getUpEvent()   { if (state==AWAKE) throw (new ClockException()); state = GOTUP; 
action(); }

	/// \fn void Person::getDownEvent()
	///
	/// \brief called by computer vision part when a corresponding movement has been 
detected.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \exception ClockException
	/// Thrown when a Clock error condition occurs.

	void getDownEvent() { if (state==AWAKE) throw (new ClockException()); state = 
SLUMBERING; action(); }

	/// \fn void Person::finallyAwakeEvent()
	///
	/// \brief called by computer vision part when a corresponding movement has been 
detected.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \exception ClockException
	/// Thrown when a Clock error condition occurs.

	void finallyAwakeEvent() { if (state!=GOTUP) throw (new ClockException()); state = 
AWAKE; }

};

/// \class AlarmClock
///
/// \brief Represents an alarm clock in general and the (possible) interface to matlab.
///
/// \author Roman Hillebrand
/// \date 19.12.2012

class AlarmClock : public Ogre::FrameListener
{
private:

	/// \property std::vector <Person*> peopleWatched
	///
	/// \brief all the people that should get up.

	std::vector <Person*> peopleWatched;

	time_t timeOfLastUpdate;      /*!< time, in seconds, frameRenderingQueued() has last 
been called */

	time_t alarmTime,snoozeTime,prerunTime;

	AlarmState alarmState;  /*!< current state of the state machine */
	AlarmEventHandler *alarmEventHandler;   /*!< which object to notify of changes */

	/// \fn bool AlarmClock::isCurrent(time_t ct, time_t event)
	///
	/// \brief did this happen in between the last event and now?
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \param ct
	/// The current time in seconds.
	/// \param event
	/// The time when te event triggered in seconds.
```

```
        ///
        /// \return true if current, false if not.

        bool isCurrent(time_t ct, time_t event) { return ct>=event && event>timeOfLastUpdate; }

        ClockException clex;  /*!< Details of the exception */

public:

        /// \fn AlarmClock::AlarmClock();
        ///
        /// \brief Default constructor.
        ///
        /// \author Roman Hillebrand
        /// \date 19.12.2012

        AlarmClock();

        /// \fn void AlarmClock::setAlarmTime( time_t t);
        ///
        /// \brief Sets alarm time.
        ///
        /// \author Roman Hillebrand
        /// \date 19.12.2012
        ///
        /// \param t
        /// new time of alarm in seconds.

        void setAlarmTime( time_t t);

        /// \fn void AlarmClock::setActive ( bool state);
        ///
        /// \brief activate or deactivate the alarm clock (called by GUI).
        ///
        /// \author Roman Hillebrand
        /// \date 19.12.2012
        ///
        /// \param state
        /// true: activate, false: deactivate

        void setActive ( bool state);

        /// \fn void AlarmClock::hookAlarmEventHandler ( AlarmEventHandler *handler );
        ///
        /// \brief supply a handler for change events (called by AlarmTest).
        ///
        /// \author Roman Hillebrand
        /// \date 19.12.2012
        ///
        /// \param handler
        /// pointer to the object which should become the alarm event handler.

        void hookAlarmEventHandler ( AlarmEventHandler *handler );

        /// \fn void AlarmClock::watchPerson ( Person *p );
        ///
        /// \brief add a person to the list of persons to watch. (called by AlarmTest)
        ///
        /// \author Roman Hillebrand
        /// \date 19.12.2012
        ///
        /// \param p
        /// pointer to the Person object which should be added to the list.

        void watchPerson ( Person *p );

        /// \fn bool AlarmClock::frameRenderingQueued(const Ogre::FrameEvent& evt) override;
        ///
        /// \brief update handler.
        ///
        /// \author Roman Hillebrand
        /// \date 19.12.2012
        ///
        /// \param evt
        /// The event.
        ///
        /// \return true if OGRE should continue rendering.
        bool frameRenderingQueued(const Ogre::FrameEvent& evt) override;
};


#endif
CLOCK.CPP


#include "stdafx.h"

#include "Clock.h"

// get the offset from gmt to local time in seconds

const time_t Clock::gmtoff()
{
        // select a value that is high enough
        // so that mktime can produce a valid result
        // ( computed value in secs never < 0 )

        const time_t t=100000;

        // fill a tm structure with the gm time
        tm *li=gmtime(&t);

        // convert back using local time
        int secs=mktime(li);

        // compute difference
        return secs-t;

}

// get the seconds passed since 1/1/1970
// (wrapper for c library)

const time_t Clock::getCurrentSecond() { static time_t t; time(&t); return t; };

// fill a tm structure
// (wrapper for c library)

const tm& Clock::getDisplayTime(const time_t &second) { static tm *lt; lt=localtime(&second);
return *lt; };



AlarmClock::AlarmClock()
{

        alarmState = INACTIVE;

        snoozeTime = 60; // Clock::HOUR;
        prerunTime = 5;  // Clock::HOUR;

        timeOfLastUpdate = 0;
        alarmTime = 0;


}

void AlarmClock::setAlarmTime( time_t t) { if (alarmState!=INACTIVE) throw clex; alarmTime =
t; }
void AlarmClock::setActive ( bool state) { alarmState=state?ACTIVE:INACTIVE; }

// Hady Created for display Alarmtime
//static const time_t getAlarmTime() {return alarmTime;};

void AlarmClock::hookAlarmEventHandler ( AlarmEventHandler *handler ) { alarmEventHandler =
handler; }

void AlarmClock::watchPerson ( Person *p ) { peopleWatched.push_back( p ); }

bool AlarmClock::frameRenderingQueued(const Ogre::FrameEvent& evt)
{
        // helper variable
        bool anyoneSleeping=false;

        // get person count
```

```
	std::vector <Person>::size_type i, npeople;
	npeople = peopleWatched.size();

	// now
	time_t ct = Clock::getCurrentSecond();

	switch(alarmState)
	{
	case ACTIVE:
		if (isCurrent(ct,alarmTime-prerunTime))
		{
			// alarm is activated and prerun time is reached
			// tell camera to begin its prerun and change state

			alarmEventHandler->watchOutEvent();
			alarmState = MOVEMENT;
		}
		break;

	case MOVEMENT:

		// check wether anyone is sleeping

		for(i=0;i<npeople;++i)
		{
			switch(peopleWatched[i]->getCurrentState())
			{
			case SLEEPING:
			case SLUMBERING:
				anyoneSleeping=true;
			}

		}

		if (ct>alarmTime && anyoneSleeping)
		{
			// if someone is sleeping and time of alarm is reached,
			// ring alarm and change state

			alarmEventHandler->alarmEvent();
			alarmState = AWAKENING;
			//actualTimeOfAlarm = ct;

		}

		if (!anyoneSleeping && isCurrent(ct,alarmTime+snoozeTime))
		{
			// noone is sleeping and the snooze time has run out
			// reset state machine to its original state
			// and set alarm to next day

			alarmState = ACTIVE;
			alarmTime += Clock::DAY;
			alarmEventHandler->everythingCompleteEvent();
		}

		break;

	case AWAKENING:
		for (i=0; i< npeople; ++i)
		{
			time_t pt = peopleWatched[i]->getLastActionTime();

			switch(peopleWatched[i]->getCurrentState())
			{
			case SLEEPING:
			case SLUMBERING:
				anyoneSleeping=true;
				break;

			case GOTUP:
				if (isCurrent(pt,alarmTime+snoozeTime))
				{
					// this person can be regarded as totally awake

					peopleWatched[i]->finallyAwakeEvent();
				}
				break;
```

```
                }
            }

            if (anyoneSleeping)
            {

            }
            else
            {
                // noone is sleeping anymore

                alarmEventHandler->stopRingingEvent();
                alarmState = MOVEMENT;
            }

            break;

        }


        // keep track of last update time
        timeOfLastUpdate=ct;

        return true;
}
```

CAMERATEST.H

```
/// \file src\CameraTest.h
///
/// \brief Declares the CameraTest class.

#ifndef CAMERATEST
#define CAMERATEST

/// \enum InputTest
///
/// \brief events that could hypotetically be triggered by the computer vision code

enum InputTest { GETUP, LAYDOWN, ISAWAKE };

/// \class CameraTest
///
/// \brief Camera test. test class for webcam. input of computer vision part is simulated.
///
/// \author Roman Hillebrand
/// \date 19.12.2012

class CameraTest : public AlarmEventHandler
{
private:

        Sound* mSound;  /*!< The sound object */
        AlarmClock* mAlarmClock;      /*!< The alarm clock object */
        Person* mPerson;      /*!< The person object */

        CEGUI::DefaultWindow *mLabel;   /*!< The label object */

public:

        /// \fn void CameraTest::watchOutEvent();
        ///
        /// \brief inherited from AlarmEventHandler.
        ///
        /// \author Roman Hillebrand
        /// \date 19.12.2012

        void watchOutEvent();

        /// \fn void CameraTest::alarmEvent();
        ///
        /// \brief inherited from AlarmEventHandler.
        ///
        /// \author Roman Hillebrand
        /// \date 19.12.2012

        void alarmEvent();
```

```
	/// \fn void CameraTest::stopRingingEvent();
	///
	/// \brief inherited from AlarmEventHandler.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	void stopRingingEvent();

	/// \fn void CameraTest::everythingCompleteEvent();
	///
	/// \brief inherited from AlarmEventHandler.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	void everythingCompleteEvent();

	/// \fn CameraTest::CameraTest ();
	///
	/// \brief creates Sound, AlarmClock and Person objects and registers event handlers.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	CameraTest ();

	/// \fn CameraTest::~CameraTest ();
	///
	/// \brief destroys Sound, AlarmClock and Person objects.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	~CameraTest ();

	/// \fn AlarmClock* CameraTest::getAlarmClock();
	///
	/// \brief get a reference to the AlarmClock member.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \return null if it fails, else the alarm clock.

	AlarmClock* getAlarmClock();

	/// \fn void CameraTest::inputTest(InputTest what);
	///
	/// \brief called by GUI to simulate computer vision triggered events.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \param what
	/// what event to simulate.

	void inputTest(InputTest what);

	/// \fn void CameraTest::setOutputLabel(CEGUI::DefaultWindow* label)
	///
	/// \brief called by GUI to specify a label which should display the corresponding
texts, should an event occur.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \param [in,out] label
	/// If non-null, the label.

	void setOutputLabel(CEGUI::DefaultWindow* label) { mLabel=label; }

};

#endif
CAMERATEST.CPP
```

```
#include "stdafx.h"

#include "CameraTest.h"
#include "StandupApplication.h"

CameraTest::CameraTest ()
{
	mLabel = NULL;
	mSound = NULL;
	mAlarmClock = NULL;
	mPerson = NULL;

	mSound = new Sound();

	mAlarmClock = new AlarmClock();
	mPerson = new Person();

	mAlarmClock->watchPerson(mPerson);
	mAlarmClock->hookAlarmEventHandler(this);

	StandupApplication::getInstance()->getRoot()->addFrameListener(mAlarmClock);
}

CameraTest::~CameraTest ()
{
	if (mSound) delete mSound;
	if (mAlarmClock) delete mAlarmClock;
	if (mPerson) delete mPerson;
}

AlarmClock* CameraTest::getAlarmClock()
{
	return this->mAlarmClock;
}

void CameraTest::watchOutEvent()
{
	mLabel->setText("watchOutEvent() was triggered.");
}

void CameraTest::alarmEvent()
{
	mLabel->setText("alarmEvent() was triggered.");
}

void CameraTest::stopRingingEvent()
{
	mLabel->setText("stopRingingEvent() was triggered.");
}

void CameraTest::everythingCompleteEvent()
{
	mLabel->setText("everythingCompleteEvent() was triggered.");
}

void CameraTest::inputTest(InputTest what)
{
	switch(what)
	{
	case GETUP:

		mPerson->getUpEvent();
		break;

	case LAYDOWN:

		mPerson->getDownEvent();
		break;

	case ISAWAKE:

		mPerson->finallyAwakeEvent();
		break;
	}
}

BASEAPPLICATION.H
```

```
/// \file src\BaseApplication.h
///
/// \brief Declares the base application class.

/*
-----------------------------------------------------------------------------
Filename:    BaseApplication.h
-----------------------------------------------------------------------------

This source file is part of the
   ___                 __    __ _ _    _ 
  /___\__ _ _ __ ___  / / /\ \ (_) | _(_)
 //  // _` | '__/ _ \ \ \/  \/ / | |/ / |
/ \_// (_| | | |  __/  \  /\  /| |   <| |
\___/ \__, |_|  \___|   \/  \/ |_|_|\_\_|
      |___/                              
      Tutorial Framework
      http://www.ogre3d.org/tikiwiki/
-----------------------------------------------------------------------------
*/
#ifndef __BaseApplication_h_

/// \def __BaseApplication_h_();
///
/// \brief A macro that defines base application h.
///
/// \author Hans Ferchland
/// \date 19.12.2012

#define __BaseApplication_h_

#include "stdafx.h"
#include <OgreCamera.h>
#include <OgreEntity.h>
#include <OgreLogManager.h>
#include <OgreRoot.h>
#include <OgreViewport.h>
#include <OgreSceneManager.h>
#include <OgreRenderWindow.h>
#include <OgreConfigFile.h>



#include <SdkTrays.h>
#include <SdkCameraMan.h>

#include "AnimationBuilder.h"
#include "CameraTest.h"
#include "GUI.h"


#define CUBEFACE_SIZE 1024

class ViewManager;

/// \class BaseApplication
///
/// \brief base class for the application. responsible for creating and managing the ogre
specific objects.
///
/// \author Hans Ferchland
/// \date 19.12.2012

class BaseApplication : public Ogre::FrameListener, public Ogre::WindowEventListener, public
OIS::KeyListener, public OIS::MouseListener, OgreBites::SdkTrayListener
{
public:

    /// \fn BaseApplication::BaseApplication(void);
    ///
    /// \brief initializes all the member object pointers to null
    ///
    /// \author Hans Ferchland
    /// \date 19.12.2012

    BaseApplication(void);

    /// \fn virtual BaseApplication::~BaseApplication(void);
    ///
    /// \brief does nothing.
    ///
    /// \author Hans Ferchland
    /// \date 19.12.2012

    virtual ~BaseApplication(void);

    /// \fn virtual void BaseApplication::go(void);
    ///
    /// \brief called by main: setup, main loop, destroy
    ///
    /// \author Hans Ferchland
    /// \date 19.12.2012

    virtual void go(void);
       // get the Ogre Renderer
       virtual CEGUI::OgreRenderer* getOgreCEGUIRenderer();
       // get the Ogre3D root node
       virtual Ogre::Root* getRoot(void);
protected:

    /// \fn virtual bool BaseApplication::setup();
    ///
    /// \brief basic instatiation of ogre.
    ///
    /// \author Hans Ferchland
    /// \date 19.12.2012
    ///
    /// \return true if it succeeds, false if it fails.

    virtual bool setup();

       /// \fn virtual bool BaseApplication::configure(void);
       ///
       /// \brief Configures this BaseApplication.
       ///
       /// \author Hans Ferchland
       /// \date 19.12.2012
       ///
       /// \return true if it succeeds, false if it fails.

       virtual bool configure(void);

    /// \fn virtual void BaseApplication::chooseSceneManager(void);
    ///
    /// \brief Chooses a scene manager.
    ///
    /// \author Hans Ferchland
    /// \date 19.12.2012

    virtual void chooseSceneManager(void);

    /// \fn virtual void BaseApplication::createCamera(void);
    ///
    /// \brief Creates the main camera.
    ///
    /// \author Hans Ferchland
    /// \date 19.12.2012

    virtual void createCamera(void);

    /// \fn virtual void BaseApplication::createFrameListener(void);
    ///
    /// \brief Creates a frame listener.
    ///
    /// \author Hans Ferchland
    /// \date 19.12.2012

    virtual void createFrameListener(void);

    /// \fn virtual void BaseApplication::createScene(void) = 0;
    ///
    /// \brief Creates the scene.
    ///
    /// \author Hans Ferchland
    /// \date 19.12.2012
```

```
virtual void createScene(void) = 0; // Override me!

/// \fn virtual void BaseApplication::destroyScene(void);
///
/// \brief Destroys the scene.
///
/// \author Hans Ferchland
/// \date 19.12.2012

virtual void destroyScene(void);

/// \fn virtual void BaseApplication::createViewports(void) = 0;
///
/// \brief Creates the viewports.
///
/// \author Hans Ferchland
/// \date 19.12.2012

virtual void createViewports(void) = 0;

/// \fn virtual void BaseApplication::setupResources(void);
///
/// \brief Sets up the resources.
///
/// \author Hans Ferchland
/// \date 19.12.2012

virtual void setupResources(void);

/// \fn virtual void BaseApplication::createResourceListener(void);
///
/// \brief Creates resource listener.
///
/// \author Hans Ferchland
/// \date 19.12.2012

virtual void createResourceListener(void);

/// \fn virtual void BaseApplication::loadResources(void);
///
/// \brief Loads the resources.
///
/// \author Hans Ferchland
/// \date 19.12.2012

virtual void loadResources(void);

   /// \fn CEGUI::MouseButton BaseApplication::convertButton(OIS::MouseButtonID buttonID);
   ///
   /// \brief CEGUI convertButton.
   ///
   /// \author Hans Ferchland
   /// \date 19.12.2012
   ///
   /// \param buttonID
   /// Identifier for the button.
   ///
   /// \return The button converted.

   CEGUI::MouseButton convertButton(OIS::MouseButtonID buttonID);

/// \fn virtual bool BaseApplication::frameRenderingQueued(const Ogre::FrameEvent& evt);
///
/// \brief Ogre::FrameListener.
///
/// \author Hans Ferchland
/// \date 19.12.2012
///
/// \param evt
/// The event.
///
/// \return true if it succeeds, false if it fails.

virtual bool frameRenderingQueued(const Ogre::FrameEvent& evt);

/// \fn virtual bool BaseApplication::keyReleased( const OIS::KeyEvent &arg );
///
/// \brief Key released.
```

```
///
/// \author Hans Ferchland
/// \date 19.12.2012
///
/// \param arg
/// The argument.
///
/// \return true if it succeeds, false if it fails.

virtual bool keyReleased( const OIS::KeyEvent &arg );

/// \fn virtual bool BaseApplication::mouseMoved( const OIS::MouseEvent &arg );
///
/// \brief OIS::MouseListener.
///
/// \author Hans Ferchland
/// \date 19.12.2012
///
/// \param arg
/// The argument.
///
/// \return true if it succeeds, false if it fails.

virtual bool mouseMoved( const OIS::MouseEvent &arg );

/// \fn virtual bool BaseApplication::mousePressed( const OIS::MouseEvent &arg,
/// OIS::MouseButtonID id );
///
/// \brief Mouse pressed.
///
/// \author Hans Ferchland
/// \date 19.12.2012
///
/// \param arg
/// The argument.
/// \param id
/// The identifier.
///
/// \return true if it succeeds, false if it fails.

virtual bool mousePressed( const OIS::MouseEvent &arg, OIS::MouseButtonID id );

/// \fn virtual bool BaseApplication::mouseReleased( const OIS::MouseEvent &arg,
/// OIS::MouseButtonID id );
///
/// \brief Mouse released.
///
/// \author Hans Ferchland
/// \date 19.12.2012
///
/// \param arg
/// The argument.
/// \param id
/// The identifier.
///
/// \return true if it succeeds, false if it fails.

virtual bool mouseReleased( const OIS::MouseEvent &arg, OIS::MouseButtonID id );

/// \fn virtual void BaseApplication::windowResized(Ogre::RenderWindow* rw);
///
/// \brief Ogre::WindowEventListener Adjust mouse clipping area.
///
/// \author Hans Ferchland
/// \date 19.12.2012
///
/// \param [in,out] rw
/// If non-null, the rw.

virtual void windowResized(Ogre::RenderWindow* rw);

/// \fn virtual void BaseApplication::windowClosed(Ogre::RenderWindow* rw);
///
/// \brief Unattach OIS before window shutdown (very important under Linux)
///
/// \author Hans Ferchland
/// \date 19.12.2012
///
```

```
    /// \param [in,out] rw
    /// If non-null, the rw.

    virtual void windowClosed(Ogre::RenderWindow* rw);

       /// \fn virtual void BaseApplication::createCEGUI();
       ///
       /// \brief creates the base components of Crazy Eddie GUI.
       ///
       /// \author Hans Ferchland
       /// \date 19.12.2012

       virtual void createCEGUI();

    Ogre::Root *mRoot;  /*!< The root */
    Ogre::Camera* mCamera;  /*!< The camera */
    Ogre::SceneManager* mSceneMgr;  /*!< Manager for scene */
    Ogre::RenderWindow* mWindow;     /*!< The window */
       Ogre::Viewport* mWindowViewport;     /*!< The window viewport */
    Ogre::String mResourcesCfg; /*!< The resources configuration */
    Ogre::String mPluginsCfg;   /*!< The plugins configuration */
       //CEGUI Fields
       CEGUI::OgreRenderer* mOgreCEGUIRenderer;    /*!< The ogre cegui renderer */
       CEGUI::System* mCEGUISystem;  /*!< The cegui system */
    bool mCursorWasVisible; /*!< true to show, false to hide the cursor was */
    bool mShutDown; /*!< true to shut down */

    //OIS Input devices
    OIS::InputManager* mInputManager;   /*!< Manager for input */
    OIS::Mouse*    mMouse;  /*!< The mouse */
    OIS::Keyboard* mKeyboard;   /*!< The keyboard */
       // saves the inital cam pos so you can reset with F3
       Ogre::Vector3* mDefaultCamPosition; /*!< The default camera position */
       // the refernce to the mGUI resposible class
    GUI* mGUI;  /*!< The graphical user interface */
};

#endif // #ifndef __BaseApplication_h_
BASEAPPLICATION.CPP

/*
-----------------------------------------------------------------------------
Filename:    BaseApplication.cpp
-----------------------------------------------------------------------------

This source file is part of the
   ___                 __    __ _ _    _
  /___\__ _ _ __ ___  / / /\ \ (_) | _(_)
 //  // _` | '__/ _ \ \ \/  \/ / | |/ / |
/ \_// (_| | | |  __/  \  /\  /| |   <| |
\___/ \__, |_|  \___|   \/  \/ |_|_|\_\_|
      |___/
      Tutorial Framework
      http://www.ogre3d.org/tikiwiki/
-----------------------------------------------------------------------------
*/
#include "stdafx.h"

#include "BaseApplication.h"
#include "CEGUIOgreRenderer.h"
#include "CEGUIOgreImageCodec.h"
#include "OgreResourceGroupManager.h"
#include "OgrePixelFormat.h"
#include "OgreCompositorManager.h"
#include "OgreParticleAffector.h"

//#include "alarmSound.h"

//-------------------------------------------------------------------------------------
BaseApplication::BaseApplication(void)
    : mRoot(0),
    mCamera(0),
    mSceneMgr(0),
    mWindow(0),
    mResourcesCfg(Ogre::StringUtil::BLANK),
    mPluginsCfg(Ogre::StringUtil::BLANK),
    mCursorWasVisible(false),
    mShutDown(false),
    mInputManager(0),
    mMouse(0),
    mKeyboard(0),
        mDefaultCamPosition(new Ogre::Vector3(0,45,0)),
        mOgreCEGUIRenderer(0),
        mCEGUISystem(0),
        mGUI(0),
        mWindowViewport(0)
{

}
//-------------------------------------------------------------------------------------
BaseApplication::~BaseApplication(void)
{
    //Remove ourself as a Window listener
    Ogre::WindowEventUtilities::removeWindowEventListener(mWindow, this);
    windowClosed(mWindow);
    delete mRoot;

}

//-------------------------------------------------------------------------------------
void BaseApplication::createCEGUI()
{
        // SIMPLE BOOTSTRAP
        //
        // Bootstrap CEGUI::System with an OgreRenderer object that uses the
        // default Ogre rendering window as the default output surface, an Ogre based
        // ResourceProvider, and an Ogre based ImageCodec.
        mOgreCEGUIRenderer =
                &CEGUI::OgreRenderer::bootstrapSystem();
        // Manually control when the gui is rendered
        mOgreCEGUIRenderer->setRenderingEnabled(false);
        //Ogre::Root::getSingleton().addFrameListener(this);
        //  set the so-called default resource groups for each of
        //  CEGUI'S resource managers
        CEGUI::Imageset::setDefaultResourceGroup("Imagesets");
        CEGUI::Font::setDefaultResourceGroup("Fonts");
        CEGUI::Scheme::setDefaultResourceGroup("Schemes");
        CEGUI::WidgetLookManager::setDefaultResourceGroup("LookNFeel");
        CEGUI::WindowManager::setDefaultResourceGroup("Layouts");
        // select the skin
        CEGUI::SchemeManager& mSchemeManager = CEGUI::SchemeManager::getSingleton();
        mSchemeManager.create("OgreTray.scheme");
        mSchemeManager.create("WindowsLook.scheme");

        mCEGUISystem = &CEGUI::System::getSingleton();
        // Set default tooltip
        mCEGUISystem->setDefaultTooltip("OgreTray/Tooltip");
        //  set the default mouse cursor
        mCEGUISystem->setDefaultMouseCursor("WindowsLook", "MouseArrow");
}

Ogre::Root* BaseApplication::getRoot(void){
        return mRoot;
}

//-------------------------------------------------------------------------------------
bool BaseApplication::configure(void)
{
    // Show the configuration dialog and initialise the system
    // You can skip this and use root.restoreConfig() to load configuration
    // settings if you were sure there are valid ones saved in ogre.cfg
    if(mRoot->showConfigDialog())
    {
        // If returned true, user clicked OK so initialise
        // Here we choose to let the system create a default rendering window by passing
'true'
        mWindow = mRoot->initialise(true, "TutorialApplication Render Window");

        return true;
    }
    else
    {
        return false;
    }
}
//-------------------------------------------------------------------------------------
```

```
void BaseApplication::chooseSceneManager(void)
{
    // Get the SceneManager, in this case a generic one
    mSceneMgr = mRoot->createSceneManager(Ogre::ST_GENERIC);
}
//-------------------------------------------------------------------------------------
void BaseApplication::createCamera(void)
{
    // Create the camera
    mCamera = mSceneMgr->createCamera("PlayerCam");

    // Position it at 500 in Z direction
    mCamera->setPosition(*mDefaultCamPosition);
    // Look back along -Z
    mCamera->lookAt(Ogre::Vector3(0,0,0));
    mCamera->setNearClipDistance(5);

    //mCameraMan = new OgreBites::SdkCameraMan(mCamera);   // create a default camera 
controller
}
//-------------------------------------------------------------------------------------
void BaseApplication::createFrameListener(void)
{
    Ogre::LogManager::getSingletonPtr()->logMessage("*** Initializing OIS ***");
    OIS::ParamList pl;
    size_t windowHnd = 0;
    std::ostringstream windowHndStr;

    mWindow->getCustomAttribute("WINDOW", &windowHnd);
    windowHndStr << windowHnd;
    pl.insert(std::make_pair(std::string("WINDOW"), windowHndStr.str()));

    mInputManager = OIS::InputManager::createInputSystem( pl );

    mKeyboard = static_cast<OIS::Keyboard*>(mInputManager->createInputObject( 
OIS::OISKeyboard, true ));
    mMouse = static_cast<OIS::Mouse*>(mInputManager->createInputObject( OIS::OISMouse, true 
));

    mMouse->setEventCallback(this);
    mKeyboard->setEventCallback(this);

    //Set initial mouse clipping size
    windowResized(mWindow);

    //Register as a Window listener
    Ogre::WindowEventUtilities::addWindowEventListener(mWindow, this);

    mRoot->addFrameListener(this);
}
//-------------------------------------------------------------------------------------
void BaseApplication::destroyScene(void)
{
}
//-------------------------------------------------------------------------------------
void BaseApplication::setupResources(void)
{
    // Load resource paths from config file
    Ogre::ConfigFile cf;
    cf.load(mResourcesCfg);

    // Go through all sections & settings in the file
    Ogre::ConfigFile::SectionIterator seci = cf.getSectionIterator();

    Ogre::String secName, typeName, archName;
    while (seci.hasMoreElements())
    {
        secName = seci.peekNextKey();
        Ogre::ConfigFile::SettingsMultiMap *settings = seci.getNext();
        Ogre::ConfigFile::SettingsMultiMap::iterator i;
        for (i = settings->begin(); i != settings->end(); ++i)
        {
            typeName = i->first;
            archName = i->second;
            Ogre::ResourceGroupManager::getSingleton().addResourceLocation(
                archName, typeName, secName);
        }
    }
```

```
}
//-------------------------------------------------------------------------------------
void BaseApplication::createResourceListener(void)
{

}
//-------------------------------------------------------------------------------------
void BaseApplication::loadResources(void)
{
    Ogre::ResourceGroupManager::getSingleton().initialiseAllResourceGroups();
       Ogre::ResourceGroupManager::getSingleton().addResourceLocation("../../media/materials/s
cripts/", "FileSystem");
}
//-------------------------------------------------------------------------------------
void BaseApplication::go(void)
{
#ifdef _DEBUG
    mResourcesCfg = "resources_d.cfg";
    mPluginsCfg = "plugins_d.cfg";
#else
    mResourcesCfg = "resources.cfg";
    mPluginsCfg = "plugins.cfg";
#endif

    if (!setup())
        return;

    mRoot->startRendering();

    // clean up
    destroyScene();
}
//-------------------------------------------------------------------------------------
bool BaseApplication::setup(void)
{
    mRoot = new Ogre::Root(mPluginsCfg);
       // collect and parse resources
    setupResources();

    bool carryOn = configure();
    if (!carryOn) return false;

    chooseSceneManager();
    createCamera();
    createViewports();

    // Set default mipmap level (NB some APIs ignore this)
    Ogre::TextureManager::getSingleton().setDefaultNumMipmaps(5);

    // Create any resource listeners (for loading screens)
    createResourceListener();
    // Load resources
    loadResources();
       // create listener for mouse and keyboard
    createFrameListener();

       //////////////////////////////////////////////////////////////////////////
       //Crazy Eddie GUI Setup
       //////////////////////////////////////////////////////////////////////////
       createCEGUI();
       //////////////////////////////////////////////////////////////////////////
       //Create the Scene
       //////////////////////////////////////////////////////////////////////////
       createScene();

    return true;
};
//-------------------------------------------------------------------------------------
bool BaseApplication::frameRenderingQueued(const Ogre::FrameEvent& evt)
{
    if(mWindow->isClosed())
        return false;

    if(mShutDown)
        return false;

    //Need to capture/update each device
    mKeyboard->capture();
    mMouse->capture();

        ////////////////////////////////////////////////////////////////////////////
        //             CEGUI Update
        ////////////////////////////////////////////////////////////////////////////
        //Need to inject timestamps to CEGUI System.
        CEGUI::System::getSingleton().injectTimePulse(evt.timeSinceLastFrame);
    return true;
}



bool BaseApplication::keyReleased( const OIS::KeyEvent &arg )
{
        // CEGUI key down injection
        CEGUI::System::getSingleton().injectKeyUp(arg.key);
    return true;
}

bool BaseApplication::mouseMoved( const OIS::MouseEvent &arg )
{
        ////////////////////////////////////////////////////////////////////////////
        //             CEGUI Mouse Movement Methods
        ////////////////////////////////////////////////////////////////////////////
        CEGUI::System &sys = CEGUI::System::getSingleton();
        sys.injectMouseMove(arg.state.X.rel, arg.state.Y.rel);
        // Scroll wheel.
        if (arg.state.Z.rel)
               sys.injectMouseWheelChange(arg.state.Z.rel / 120.0f);

    return true;
}

bool BaseApplication::mousePressed( const OIS::MouseEvent &arg, OIS::MouseButtonID id )
{
        // CEGUI mousePressed injection
        CEGUI::System::getSingleton().injectMouseButtonDown(convertButton(id));
    return true;
}

bool BaseApplication::mouseReleased( const OIS::MouseEvent &arg, OIS::MouseButtonID id )
{
        // CEGUI mouseReleased injection
        CEGUI::System::getSingleton().injectMouseButtonUp(convertButton(id));
    return true;
}

//Adjust mouse clipping area
void BaseApplication::windowResized(Ogre::RenderWindow* rw)
{
    unsigned int width, height, depth;
    int left, top;
    rw->getMetrics(width, height, depth, left, top);

    const OIS::MouseState &ms = mMouse->getMouseState();
    ms.width = width;
    ms.height = height;
}

//Unattach OIS before window shutdown (very important under Linux)
void BaseApplication::windowClosed(Ogre::RenderWindow* rw)
{
    //Only close for window that created OIS (the main window in these demos)
    if( rw == mWindow )
    {
        if( mInputManager )
        {
            mInputManager->destroyInputObject( mMouse );
            mInputManager->destroyInputObject( mKeyboard );

            OIS::InputManager::destroyInputSystem(mInputManager);
            mInputManager = 0;
        }
    }
}

CEGUI::OgreRenderer* BaseApplication::getOgreCEGUIRenderer() {
        return mOgreCEGUIRenderer;
```

```
}

//-------------------------------------------------------------------------------------
CEGUI::MouseButton BaseApplication::convertButton(OIS::MouseButtonID buttonID)
{
	switch (buttonID)
	{
	case OIS::MB_Left:
		return CEGUI::LeftButton;

	case OIS::MB_Right:
		return CEGUI::RightButton;

	case OIS::MB_Middle:
		return CEGUI::MiddleButton;

	default:
		return CEGUI::LeftButton;
	}
}
```

ANIMATIONBUILDER.H

```
/// \file src\AnimationBuilder.h
///
/// \brief Declares the animation builder class.

#ifndef __AnimationBuilder_h_

/// \def __AnimationBuilder_h_
///
/// \brief A macro that defines animation builder h.
///
/// \author Hady Khalifa
/// \date 19.12.2012

#define __AnimationBuilder_h_

#include "stdafx.h"
#include "AnimationBuilder.h"

/// \class AnimationBuilder
///
/// \brief Animation builder.
///
/// \author Hady Khalifa
/// \date 19.12.2012

class AnimationBuilder
{
public:

	/// \fn AnimationBuilder::AnimationBuilder(void);
	///
	/// \brief Default constructor.
	///
	/// \author Hady Khalifa
	/// \date 19.12.2012

	AnimationBuilder(void);

	/// \fn AnimationBuilder::~AnimationBuilder(void);
	///
	/// \brief Destructor.
	///
	/// \author Hady Khalifa
	/// \date 19.12.2012

	~AnimationBuilder(void);

	/// \fn void AnimationBuilder::createAnimations( CEGUI::String v1, CEGUI::String v2,
	/// CEGUI::String v3, CEGUI::String r1, CEGUI::String r2, CEGUI::String r3 );
	///
	/// \brief Creates the animations.
	///
	/// \author Hady Khalifa
	/// \date 19.12.2012
	///
	/// \param v1
	/// The first CEGUI::String.
	/// \param v2
	/// The second CEGUI::String.
	/// \param v3
	/// The third CEGUI::String.
	/// \param r1
	/// The first CEGUI::String.
	/// \param r2
	/// The second CEGUI::String.
	/// \param r3
	/// The third CEGUI::String.

	void createAnimations( CEGUI::String v1, CEGUI::String v2, CEGUI::String v3,
CEGUI::String r1, CEGUI::String r2, CEGUI::String r3 );

	/// \fn void AnimationBuilder::yRotation(CEGUI::Animation* anim,CEGUI::String
valueStart,
	/// CEGUI::String value);
	///
	/// \brief Y coordinate rotations.
	///
	/// \author Hady Khalifa
	/// \date 19.12.2012
	///
	/// \param [in,out] anim
	/// If non-null, the animation.
	/// \param valueStart
	/// The value start.
	/// \param value
	/// The value.

	void yRotation(CEGUI::Animation* anim,CEGUI::String valueStart, CEGUI::String value);

	/// \fn void AnimationBuilder::unifiedXPositionInterpolation(CEGUI::Animation* anim,
	/// CEGUI::String pos1, CEGUI::String pos2);
	///
	/// \brief Unified x coordinate position interpolation.
	///
	/// \author Hady Khalifa
	/// \date 19.12.2012
	///
	/// \param [in,out] anim
	/// If non-null, the animation.
	/// \param pos1
	/// The first position.
	/// \param pos2
	/// The second position.

	void unifiedXPositionInterpolation(CEGUI::Animation* anim, CEGUI::String pos1,
CEGUI::String pos2);

	/// \fn CEGUI::Animation* AnimationBuilder::createAnimation(CEGUI::String name, float
duration,
	/// CEGUI::Animation::ReplayMode replayMode);
	///
	/// \brief Creates an animation.
	///
	/// \author Hady Khalifa
	/// \date 19.12.2012
	///
	/// \param name
	/// The name.
	/// \param duration
	/// The duration.
	/// \param replayMode
	/// The replay mode.
	///
	/// \return The new animation.

	CEGUI::Animation* createAnimation(CEGUI::String name, float duration,
CEGUI::Animation::ReplayMode replayMode);

	/// \fn void AnimationBuilder::fadeInAlpha(CEGUI::Animation* anim, CEGUI::String a1,
	/// CEGUI::String a2);
	///
	/// \brief Fade in alpha.
	///
	/// \author Hady Khalifa
	/// \date 19.12.2012
	///
	/// \param [in,out] anim
	/// If non-null, the animation.
	/// \param a1
	/// The first CEGUI::String.
	/// \param a2
	/// The second CEGUI::String.

	void fadeInAlpha(CEGUI::Animation* anim, CEGUI::String a1, CEGUI::String a2);
};

#endif // #ifndef __AnimationBuilder_h_
ANIMATIONBUILDER.CPP

#include "stdafx.h"
#include "AnimationBuilder.h"

AnimationBuilder::AnimationBuilder(void)
{
}


AnimationBuilder::~AnimationBuilder(void)
{
}



//Param startposition of the 3 Windows
void AnimationBuilder::createAnimations( CEGUI::String v1,  CEGUI::String v2,  CEGUI::String
v3, CEGUI::String r1, CEGUI::String r2, CEGUI::String r3 )
{
	/************************************************************************/
	/* moveWindow1Right
	                              */
	/************************************************************************/
	CEGUI::Animation* anim = createAnimation("MoveWindow1Right", 0.6f,
CEGUI::Animation::RM_Once);
	unifiedXPositionInterpolation(anim,v1,v3);
	// yRotation
	yRotation(anim,r1,r3);

	/************************************************************************/
	/* moveWindow2Right
	                              */
	/************************************************************************/
	CEGUI::Animation* anim2 = createAnimation("MoveWindow2Right", 0.6f,
CEGUI::Animation::RM_Once);
	unifiedXPositionInterpolation(anim2,v2,v1);
	// yRotation
	yRotation(anim2,r2,r1);


	/************************************************************************/
	/* moveWindow3left
	                              */
	/************************************************************************/
	CEGUI::Animation* anim3 = createAnimation("MoveWindow3Left", 0.6f,
CEGUI::Animation::RM_Once);
	// now we define affector inside our Testing animation
	unifiedXPositionInterpolation(anim3,v3,v1);
	// yRotation
	yRotation(anim3,r3,r1);


	/************************************************************************/
	/* moveWindow1left
	                              */
	/************************************************************************/
	CEGUI::Animation* anim4 = createAnimation("MoveWindow1Left", 0.6f,
CEGUI::Animation::RM_Once);
	unifiedXPositionInterpolation(anim4,v1,v2);
	// yRotation
	yRotation(anim4,r1,r2);
	/*********************************************************************/
	/* moveWindow1FromRightToStart
	                                             */
	/*********************************************************************/
	CEGUI::Animation* anim5 = createAnimation("moveWindow1FromRightToStart", 0.6f,
CEGUI::Animation::RM_Once);
	// now we define affector inside our Testing animation
	unifiedXPositionInterpolation(anim5,v3,v1);
	yRotation(anim5,r3,r1);


	/*********************************************************************/
	/* moveWindow1FromLeftToStart
	                                      */
	/*********************************************************************/
	CEGUI::Animation* anim6 = createAnimation("moveWindow1FromLeftToStart", 0.6f,
CEGUI::Animation::RM_Once);
	unifiedXPositionInterpolation(anim6,v2,v1);
	yRotation(anim6,r2,r1);


	/*********************************************************************/
	/* moveWindow2FromStarttoLeft
	                                      */
	/*********************************************************************/
	CEGUI::Animation* anim7 = createAnimation("moveWindow2FromLeftToStart", 0.6f,
CEGUI::Animation::RM_Once);
	unifiedXPositionInterpolation(anim7,v1,v2);
	// yRotation
	yRotation(anim7,r1,r2);


	/*********************************************************************/
	/* moveWindow3FromStartTORight
	                                             */
	/*********************************************************************/
	CEGUI::Animation* anim8 =	createAnimation("moveWindow3FromRightToStart", 0.6f,
CEGUI::Animation::RM_Once);
	// now we define affector inside our Testing animation
	unifiedXPositionInterpolation(anim8,v1,v3);
	// yRotation
	yRotation(anim8,r1, r3);


	/*********************************************************************/
	/*                       FadeIn1                                       */
	/*********************************************************************/
	CEGUI::Animation* anim9 =	createAnimation("FadeIn1", 0.6f,
CEGUI::Animation::RM_Once);
	fadeInAlpha(anim9,"0.0","0.3f");


/*********************************************************************/
/*                       FadeOut                                     */
/*********************************************************************/
CEGUI::Animation* anim10 =	createAnimation("FadeOut", 0.6f, CEGUI::Animation::RM_Once);
	fadeInAlpha(anim10,"0.3f","0");
}


/*
 *
 *	createAnimation
 *
 */
CEGUI::Animation* AnimationBuilder::createAnimation(CEGUI::String name, float duration,
CEGUI::Animation::ReplayMode replayMode)
{
	CEGUI::Animation* anim = CEGUI::AnimationManager::getSingleton().createAnimation(name);
	anim->setDuration(duration); // duration in seconds
	anim->setReplayMode(replayMode); // when this animation is started, only play it once,
then stop
	return anim;
}


/*
 *
 *      fadeInAlpha
 *
 */
void AnimationBuilder::fadeInAlpha(CEGUI::Animation* anim, CEGUI::String a1, CEGUI::String a2)
{
       // this affector will again use float interpolator
       CEGUI::Affector* affector = anim->createAffector("Alpha", "float");
       affector->createKeyFrame(0.0f, a1); // at 0.0 seconds, set alpha to 1.0
       affector->createKeyFrame(0.6f, a2, CEGUI::KeyFrame::P_QuadraticDecelerating); // at 0.3
seconds, set alpha to 0.5, now decelerating!
}




/*
 *
 *      unifiedXPositionInterpolation
 *
 */
void AnimationBuilder::unifiedXPositionInterpolation(CEGUI::Animation* anim, CEGUI::String
pos1, CEGUI::String pos2)
{
       // this affector changes YRotation and interpolates keyframes with float interpolator
       CEGUI::Affector* affector = anim->createAffector("UnifiedXPosition", "UDim");
       // at 0.0 seconds, the animation should set YRotation to 0 degrees
       affector->createKeyFrame(0.0f, pos1);
       // at 0.3 seconds, YRotation should be 10.0 degrees and animation should progress
towards this in an accelerating manner
       affector->createKeyFrame(0.6f, pos2, CEGUI::KeyFrame::P_QuadraticAccelerating);
}


/*
 *
 *      yRotation
 *
 */
void AnimationBuilder::yRotation(CEGUI::Animation* anim, CEGUI::String valueStart ,
CEGUI::String valueEnd)
       {

              // this affector changes YRotation and interpolates keyframes with float
interpolator
              CEGUI::Affector* affector = anim->createAffector("YRotation", "float");
              // at 0.0 seconds, the animation should set YRotation to 0 degrees
              affector->createKeyFrame(0.0f, valueStart);
              // at 0.8 seconds, YRotation should be 10.0 degrees and animation should
progress towards this in an accelerating manner
              affector->createKeyFrame(0.6f, valueEnd,
CEGUI::KeyFrame::P_QuadraticAccelerating);
       }
STDAFX.H

/// \file src\stdafx.h
///
/// \brief Declares the stdafx class.

#include <time.h>

#include <sstream>
#include <string>

#include <vector>
#include <exception>

// Ogre headers you need

#include <Ogre.h>
#include <OgreMaterialManager.h>
#include <OgreException.h>
#include <OgreRoot.h>
#include <OgreConfigFile.h>
#include <OgreCamera.h>
#include <OgreViewport.h>
#include <OgreSceneManager.h>
#include <OgreRenderWindow.h>
#include <OgreEntity.h>
```

```
#include <OgreWindowEventUtilities.h>
#include <OgrePrerequisites.h>
#include <OgreStringConverter.h>

// OIS include

#include <OISEvents.h>
#include <OISInputManager.h>
#include <OISKeyboard.h>
#include <OISMouse.h>

// CEGUI include

#include <CEGUI.h>
#include <CEGUIOgreRenderer.h>
#include <CEGUIOgreRenderTarget.h>
#include <CEGUIOgreImageCodec.h>
#include <CEGUISystem.h>
#include <CEGUISlider.h>


// FMOD

#include <fmod.hpp>
#include <fmod_errors.h>


#include "Sound.h"
#include "Clock.h"


// any other header can be included, as usual
#if OGRE_PLATFORM == OGRE_PLATFORM_WIN32

/// \def WIN32_LEAN_AND_MEAN
///
/// \brief A macro that defines window 32 lean and mean.
///
/// \author Hans Ferchland
/// \date 19.12.2012

#define WIN32_LEAN_AND_MEAN
#include "windows.h"
#endif
STDAFX.CPP

// stdafx.cpp
#include "stdafx.h"

// anything you want after that
STANDUPAPPLICATION.H

/// \file src\StandupApplication.h
///
/// This source file is part of the
///       ___                 __    __ _ _    _
///      /___\__ _ _ __ ___  / / /\ \ (_) | _(_)
///     //  // _` | '__/ _ \ \ \/  \/ / | |/ / |
/// / \_// (_| | | |  __/  \  /\  /| |   <| |
/// \___/ \__, |_|  \___|   \/  \/ |_|_|\_\_|
///      |___/
///      Tutorial Framework
///      http://www.ogre3d.org/tikiwiki/
///
///      Modified by Hans Ferchland, Roman Hillebrand and Hady Khalifa
///      during Project for Computergraphic.
///
/// \brief Declares the standup application class.

#ifndef __StandupApplication_h_

#define __StandupApplication_h_

#include "stdafx.h"
#include "BaseApplication.h"

using namespace CEGUI;
```

```
/// \class StandupApplication
///
/// \brief StandupApplication class is the core of Standup.
///
///    This class extends the BaseApplication and implements
///    content and behaviour.
///
/// \author Hans Ferchland
/// \date 20.12.2012
/// \sa BaseApplication

class StandupApplication : public BaseApplication
{
public:
       /// \fn virtual bool BaseApplication::keyPressed( const OIS::KeyEvent &arg );
       ///
       /// \brief OIS::KeyListener.
       ///
       /// \author Hans Ferchland
       /// \date 19.12.2012
       ///
       /// \param arg
       /// The argument.
       ///
       /// \return true if it succeeds, false if it fails.
       virtual bool keyPressed( const OIS::KeyEvent &arg );

       /// \fn static StandupApplication* StandupApplication::getInstance();
       ///
       /// \brief Gets the one and only pointer to the StandupApplication.
       ///
       /// Creates the application if not instatiated already.
       ///    Returns the refernce if it is there.
       ///
       /// \author Hans Ferchland
       /// \date 20.12.2012
       ///
       /// \return the pointer to the one and only StandupApplication.

       static StandupApplication* getInstance();

       /// \fn OIS::Mouse* StandupApplication::getMouse()
       ///
       /// \brief Gets the mouse pointer of the ogre application.
       ///
       /// \author Hans Ferchland
       /// \date 20.12.2012
       ///
       /// \return null if it fails, else the ogre3d mouse.
       ///    \sa Ogre::Mouse

       OIS::Mouse* getMouse() { return mMouse; }

       /// \fn OIS::Keyboard* StandupApplication::getKeyboard()
       ///
       /// \brief Gets the keyboard pointer of the ogre application.
       ///
       /// \author Hans Ferchland
       /// \date 20.12.2012
       ///
       /// \return the pointer to the ogre keyboard
       ///    \sa Ogre::Keyboard

       OIS::Keyboard* getKeyboard() { return mKeyboard; }

       /// \fn void StandupApplication::createCEGUI() override
       ///
       /// \brief Creates the CEGUI components and the applications main gui-content.
       ///
       /// \author Hans Ferchland
       /// \date 20.12.2012
       ///    \sa BaseApplication.createCEGUI(), GUI, CEGUI

       void createCEGUI() override {
              BaseApplication::createCEGUI();
              // CEGUI
              mGUI = new GUI(mCEGUISystem, mRoot);
              mRoot->addFrameListener(mGUI);
```

```
		mGUI->createScene();

	}

	/// \fn Ogre::Root* StandupApplication::getRoot(void)
	///
	/// \brief Gets the root of the ogre application.
	///
	/// \author Hans Ferchland
	/// \date 20.12.2012
	///
	/// \return null if it fails, else the ogre3d root.

	Ogre::Root* getRoot(void) {
		return mRoot;
	}
protected:

	/// \fn bool StandupApplication::configure();
	///
	/// \brief Configures the options window and window title.
	///
	/// \author Hans Ferchland
	/// \date 20.12.2012
	///
	/// \return true if it succeeds, false if it fails.

	bool configure();

	/// \fn virtual void StandupApplication::createViewports(void);
	///
	/// \brief Creates the applications viewport.
	///
	/// \author Hans Ferchland
	/// \date 20.12.2012

	virtual void createViewports(void);

	/// \fn virtual void StandupApplication::createCamera(void);
	///
	/// \brief Creates the camera for the application.
	///
	/// \author Hans Ferchland
	/// \date 20.12.2012

	virtual void createCamera(void);

	/// \fn virtual void StandupApplication::createScene(void);
	///
	/// \brief Creates the initial scene for the application.
	///
	/// \author Hans Ferchland
	/// \date 20.12.2012

	virtual void createScene(void);

private:
	static StandupApplication* instance; /*!< Instance of the one and only instance of the
application */

	/// \fn StandupApplication::StandupApplication(void);
	///
	/// \brief Default Hidden Constructor due singleton pattern.
	///
	/// \author Hans Ferchland
	/// \date 20.12.2012

	StandupApplication(void);

	/// \fn virtual StandupApplication::~StandupApplication(void);
	///
	/// \brief Destructor of the Application is hidden due singleton.
	///
	/// \author Hans Ferchland
	/// \date 20.12.2012

	virtual ~StandupApplication(void);
};
```

```
#endif // #ifndef __StandupApplication_h_
STANDUPAPPLICATION.CPP

/*
-----------------------------------------------------------------------------
Filename:    StandupApplication.cpp
-----------------------------------------------------------------------------

This source file is part of the
   ___                 __    __ _ _    _ 
  /___\__ _ _ __ ___  / / /\ \ (_) | _(_)
 //  // _` | '__/ _ \ \ \/  \/ / | |/ / |
/ \_// (_| | | |  __/  \  /\  /| |   <| |
\___/ \__, |_|  \___|   \/  \/ |_|_|\_\_|
      |___/                              
      Tutorial Framework
      http://www.ogre3d.org/tikiwiki/
-----------------------------------------------------------------------------
*/
#include "stdafx.h"
#include "StandupApplication.h"
#include "ClockVisualizationCircle.h"
#include "ClockVisualizationBars.h"
#include "clock.h"

//-------------------------------------------------------------------------------------
StandupApplication* StandupApplication::instance = 0;

//-------------------------------------------------------------------------------------
StandupApplication::StandupApplication(void) : BaseApplication()
{

}
//-------------------------------------------------------------------------------------
StandupApplication::~StandupApplication(void)
{
}
/*
 *	create Viewports
 */
void StandupApplication::createViewports(void)
{
	// Create one viewport, entire window
	mWindowViewport = mWindow->addViewport(mCamera);
	mWindowViewport->setBackgroundColour(Ogre::ColourValue(0,0,0));
	Ogre::CompositorManager::getSingleton().addCompositor(mWindowViewport, "Bloom");
	Ogre::CompositorManager::getSingleton().setCompositorEnabled(mWindowViewport, "Bloom", 
true);
	Ogre::CompositorManager::getSingleton().addCompositor(mWindowViewport, "HDR");
	Ogre::CompositorManager::getSingleton().setCompositorEnabled(mWindowViewport, "HDR", 
true);
	// Alter the camera aspect ratio to match the viewport
	mCamera->setAspectRatio(
		Ogre::Real(mWindowViewport->getActualWidth()) / Ogre::Real(mWindowViewport-
>getActualHeight()));
}

/*
 *	create Camera
 */
void StandupApplication::createCamera(void)
{
	mCamera = mSceneMgr->createCamera("PlayerCam");
	mCamera->setPosition(*mDefaultCamPosition);
	mCamera->setFixedYawAxis(true, Ogre::Vector3::UNIT_X);
	mCamera->lookAt(Ogre::Vector3(0,0,0));
	mCamera->setNearClipDistance(1);
	mCamera->setFOVy(Ogre::Radian(Ogre::Math::PI * 0.4f));
}

/*
 *	create Scene
 */
void StandupApplication::createScene(void)
{

	// create Ground Entity and Material, Enable Shadows
	Ogre::Plane plane(Ogre::Vector3::UNIT_Y, 0);
	Ogre::MeshManager::getSingleton().createPlane("ground",
Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
		plane, 10000, 10000, 20, 20, true, 1, 5, 5, Ogre::Vector3::UNIT_Z);
	Ogre::Entity* entGround = mSceneMgr->createEntity("GroundEntity", "ground");
	Ogre::SceneNode* groundNode = mSceneMgr->getRootSceneNode()->createChildSceneNode();
	groundNode->attachObject(entGround);
	groundNode->setPosition(-50,-45,0);
	entGround->setMaterialName("Standup/RustySteel");
	mSceneMgr->setShadowTechnique(Ogre::SHADOWTYPE_STENCIL_ADDITIVE);

	// Create a skybox
	mSceneMgr->setSkyBox(true, "Standup/TimedSkyBox");

	// set lights
	mSceneMgr->setAmbientLight(Ogre::ColourValue(0.75, 0.75, 0.75));

	ClockVisualizationCircle* clockVis = new ClockVisualizationCircle(mSceneMgr, mCamera,
2);
	mRoot->addFrameListener(clockVis);
}

bool StandupApplication::configure() {
	// Show the configuration dialog and initialise the system
	// You can skip this and use root.restoreConfig() to load configuration
	// settings if you were sure there are valid ones saved in ogre.cfg
	if(mRoot->showConfigDialog())
	{
		// If returned true, user clicked OK so initialise
		// Here we choose to let the system create a default rendering window by
passing 'true'
		mWindow = mRoot->initialise(true, "Standup - Ogre - Win32");

		return true;
	}
	else
	{
		return false;
	}
}

//-------------------------------------------------------------------------------------
bool StandupApplication::keyPressed( const OIS::KeyEvent &arg )
{
	if(arg.key == OIS::KC_F3)   // reset camera position
	{
		mCamera->setPosition(*mDefaultCamPosition);
	}

	if(arg.key == OIS::KC_F5)   // refresh all textures
	{
		Ogre::TextureManager::getSingleton().reloadAll();
	}
	else if (arg.key == OIS::KC_SYSRQ)   // take a screenshot
	{
		mWindow->writeContentsToTimestampedFile("screenshot", ".jpg");
	}
	else if (arg.key == OIS::KC_ESCAPE)
	{
		mShutDown = true;
	}
	else if (arg.key == OIS::KC_1)
	{
		mGUI->setPersonState(GETUP);
	}
	else if (arg.key == OIS::KC_2)
	{
		mGUI->setPersonState(LAYDOWN);
	}


	//////////////////////////////////////////////////////////////////////////
	//		CEGUI Input Methods
	//////////////////////////////////////////////////////////////////////////
	CEGUI::System &sys = CEGUI::System::getSingleton();
	sys.injectKeyDown(arg.key);
	sys.injectChar(arg.text);
```

```
        return true;
}


StandupApplication* StandupApplication::getInstance(){
        if (instance == NULL)
                return instance = new StandupApplication();
        else
                return instance;
}

//-----------------------------------------------------------------------------------
// MAIN METHOD
//-----------------------------------------------------------------------------------
#if OGRE_PLATFORM == OGRE_PLATFORM_WIN32
#define WIN32_LEAN_AND_MEAN
#include "windows.h"
#endif

#ifdef __cplusplus
extern "C" {
#endif

#if OGRE_PLATFORM == OGRE_PLATFORM_WIN32
    INT WINAPI WinMain( HINSTANCE hInst, HINSTANCE, LPSTR strCmdLine, INT )
#else
    int main(int argc, char *argv[])
#endif
    {

        // Create application object
        StandupApplication* app = StandupApplication::getInstance();

        try {
            app->go();
        } catch( Ogre::Exception& e ) {
#if OGRE_PLATFORM == OGRE_PLATFORM_WIN32
            MessageBox( NULL, e.getFullDescription().c_str(), "An exception has occured!",
MB_OK | MB_ICONERROR | MB_TASKMODAL);
#else
            std::cerr << "An exception has occured: " <<
                e.getFullDescription().c_str() << std::endl;
#endif
        }

        return 0;
    }

#ifdef __cplusplus
}
#endif
SOUND.H


/// \file src\Sound.h
///
/// \brief Declares the Sound class.

#ifndef SOUND_H
#define SOUND_H

/// \enum SoundEffect
///
/// \brief enumarates sound effects that can be played.

enum SoundEffect { PRERUN, ALARM };

/// \class SoundException
///
/// \brief Exception for signalling sound errors.
///
/// \author Roman Hillebrand
/// \date 19.12.2012

class SoundException : std::exception
{
        char *what () { return "SoundException"; }
};
```

```
/// \class Sound
///
/// \brief integrates the FMOD library to play the alarm sound effects.
///
/// \author Roman Hillebrand
/// \date 19.12.2012

class Sound : public Ogre::FrameListener
{
private:

	FMOD::System     *system;   /*!< The system */
	FMOD::Sound      *sound[2]; /*!< The sound[ 2] */

public:

	/// \fn Sound::Sound();
	///
	/// \brief create FMOD and load sound effects.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	Sound();

	/// \fn Sound::~Sound();
	///
	/// \brief destroy everything.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012

	~Sound();

	// update event
	bool frameRenderingQueued(const Ogre::FrameEvent& evt);

	/// \fn void Sound::reloadSoundFile(SoundEffect which, Ogre::String path);
	///
	/// \brief load or reload a sound file.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \param which
	/// Which sound effect to load
	/// \param path
	/// Full pathname of the file.

	void reloadSoundFile(SoundEffect which, Ogre::String path);

	/// \fn void Sound::play(SoundEffect which);
	///
	/// \brief play a sound file.
	///
	/// \author Roman Hillebrand
	/// \date 19.12.2012
	///
	/// \param which
	/// Which sound effect to play

	void play(SoundEffect which);
};

#endif
```

SOUND.CPP

```
#include "stdafx.h"
#include "Sound.h"

Sound::Sound()
{
	unsigned int      version;
	FMOD_RESULT       result;
```

```
        system=NULL;
        sound[0]=NULL;
        sound[1]=NULL;

        // load fmod library

        result = FMOD::System_Create(&system);
        if(result!=FMOD_OK) throw new SoundException();

        // check dll version

        result = system->getVersion(&version);
        if(result!=FMOD_OK) throw new SoundException();

        if (version < FMOD_VERSION)
        {
                //printf("Error!  You are using an old version of FMOD %08x.  This program
requires %08x\n", version, FMOD_VERSION);
                throw new SoundException();
        }

        // initialize

        result = system->init(32, FMOD_INIT_NORMAL, 0);
        if(result!=FMOD_OK) throw new SoundException();

        // load sounds

        reloadSoundFile(PRERUN, "stuka.wav");
        reloadSoundFile(ALARM, "stuka.wav");
}

bool Sound::frameRenderingQueued(const Ogre::FrameEvent& evt) {

        // continue playing sounds if needed
        // (fills buffers etc)

        system->update();
        return true;
}

void Sound::reloadSoundFile(SoundEffect which, Ogre::String path) {

        // if sound file already loaded, release old sound

        if (sound[which]) sound[which]->release();

        // load a sound file

        FMOD_RESULT       result;
        result = system->createSound(path.c_str(), FMOD_HARDWARE, 0, &sound[which]);
        if(result!=FMOD_OK) throw new SoundException();
}

Sound::~Sound()
{
        // destroy everything!

        FMOD_RESULT       result;

        if(sound[0]) result = sound[0]->release();
        if(sound[1]) result = sound[1]->release();
        if(system) result = system->close();
        if(system) result = system->release();
}

void Sound::play(SoundEffect which)
{
        // start playing a sound

        FMOD::Channel *ch;
        system->playSound(FMOD_CHANNEL_FREE,sound[which],false,&ch);
}
```

COLORS.H

```
static unsigned char const interpolationTable[768] =
{
     31,  36,  70,  34,  36,  69,  36,  37,  69,  39,  37,  69,  43,  38,  69,  46,
     37,  69,  49,  38,  69,  52,  38,  69,  56,  38,  68,  60,  38,  68,  63,  38,
     68,  67,  39,  68,  71,  38,  68,  76,  39,  67,  79,  39,  66,  84,  40,  67,
     88,  40,  66,  93,  40,  65,  98,  41,  65, 102,  40,  64, 108,  41,  63, 113,
     42,  63, 118,  42,  63, 122,  41,  61, 128,  41,  60, 134,  42,  59, 140,  42,
     58, 144,  42,  57, 151,  42,  55, 156,  42,  54, 162,  42,  53, 168,  42,  51,
    173,  42,  50, 179,  42,  49, 184,  42,  47, 189,  41,  46, 195,  41,  45, 198,
     40,  43, 203,  41,  42, 208,  40,  40, 211,  39,  38, 216,  39,  37, 220,  38,
     36, 223,  38,  35, 226,  38,  33, 229,  37,  33, 229,  39,  32, 230,  41,  33,
    230,  44,  32, 230,  46,  33, 230,  50,  33, 230,  52,  33, 230,  56,  33, 231,
     58,  33, 231,  61,  32, 231,  65,  33, 231,  69,  33, 232,  73,  32, 232,  76,
     33, 233,  80,  33, 234,  86,  34, 234,  89,  34, 235,  93,  33, 235,  97,  34,
    236, 101,  34, 236, 105,  34, 237, 108,  33, 238, 114,  34, 238, 117,  33, 239,
    121,  33, 240, 126,  33, 240, 130,  33, 241, 134,  32, 241, 138,  31, 243, 142,
     33, 243, 146,  32, 243, 149,  31, 245, 154,  31, 246, 158,  30, 246, 162,  30,
    246, 164,  29, 248, 167,  29, 247, 169,  28, 248, 172,  26, 249, 176,  27, 249,
    177,  26, 249, 179,  26, 250, 183,  25, 251, 185,  25, 251, 187,  23, 252, 191,
     23, 252, 192,  22, 253, 194,  21, 253, 197,  20, 254, 199,  19, 254, 201,  19,
    255, 203,  17, 255, 205,  15, 255, 207,  14, 255, 209,  12, 255, 211,  11, 255,
    213,   8, 255, 214,   7, 255, 216,   7, 255, 217,   6, 255, 219,   5, 255, 219,
      4, 255, 221,   2, 255, 223,   1, 255, 224,   0, 255, 225,   0, 254, 226,   0,
    253, 226,   0, 253, 227,   0, 253, 228,   0, 252, 228,   0, 251, 229,   0, 250,
    229,   0, 249, 229,   0, 249, 230,   0, 248, 231,   1, 247, 231,   2, 246, 230,
      3, 246, 231,   3, 245, 230,   5, 244, 231,   6, 243, 230,  14, 242, 230,  22,
    241, 229,  29, 239, 228,  33, 237, 228,  39, 236, 227,  45, 235, 227,  49, 233,
    226,  55, 231, 224,  58, 230, 224,  65, 227, 222,  67, 225, 221,  74, 223, 219,
     77, 221, 218,  81, 220, 217,  85, 217, 215,  90, 214, 213,  93, 212, 211,  98,
    209, 210, 102, 207, 207, 105, 204, 205, 109, 201, 203, 113, 198, 201, 116, 196,
    199, 119, 193, 197, 123, 190, 194, 125, 187, 192, 128, 184, 190, 132, 181, 187,
    135, 179, 185, 136, 176, 183, 139, 173, 180, 142, 170, 177, 145, 167, 175, 147,
    164, 172, 148, 162, 170, 151, 159, 168, 153, 156, 165, 154, 152, 162, 155, 151,
    161, 158, 148, 158, 158, 146, 156, 160, 142, 152, 161, 139, 150, 161, 138, 148,
    163, 134, 145, 164, 132, 143, 164, 130, 141, 165, 126, 138, 166, 124, 136, 167,
    122, 133, 167, 119, 130, 167, 117, 128, 168, 114, 126, 167, 112, 123, 167, 110,
    122, 168, 108, 119, 168, 106, 116, 168, 104, 114, 168, 102, 112, 168,  99, 110,
    167,  99, 109, 168,  97, 106, 168,  94, 104, 167,  92, 102, 167,  91, 101, 167,
     90,  99, 167,  88,  97, 166,  86,  95, 166,  84,  94, 165,  83,  92, 165,  81,
     91, 165,  81,  90, 164,  80,  89, 164,  79,  88, 164,  77,  86, 162,  74,  84,
    160,  73,  82, 159,  72,  80, 157,  69,  79, 155,  68,  78, 154,  67,  76, 153,
     66,  74, 151,  64,  73, 149,  63,  71, 148,  60,  70, 146,  60,  68, 144,  58,
     66, 142,  58,  66, 140,  57,  65, 138,  56,  63, 135,  55,  62, 134,  54,  61,
    132,  52,  60, 130,  51,  58, 127,  50,  57, 125,  49,  56, 123,  49,  55, 121,
     49,  55, 119,  48,  54, 117,  47,  53, 115,  46,  52, 113,  45,  51, 111,  45,
     51, 109,  44,  49, 106,  43,  49, 104,  43,  48, 103,  42,  48, 101,  41,  47,
     99,  40,  46,  96,  39,  45,  94,  38,  44,  92,  39,  44,  91,  39,  43,  89,
     38,  43,  88,  38,  43,  86,  36,  42,  85,  37,  42,  84,  36,  41,  81,  35,
     40,  80,  35,  40,  79,  34,  39,  77,  33,  39,  76,  33,  38,  75,  32,  38,
     74,  32,  38,  73,  31,  37,  72,  31,  36,  71,  31,  37,  71,  31,  37,  70
  };
```

CLOCKVISUALIZATIONCIRCLE.H

```
/// \file src\ClockVisualizationCircle.h
///
/// \brief Declares the clock visualization circle class.

#ifndef CLOCK_VISUALIZATION_CIRCLE_H

#define CLOCK_VISUALIZATION_CIRCLE_H

#include "ClockVisualisation.h"

/// \class ClockVisualizationCircle
///
/// \brief The class ClockVisualizationCircle displays a clock via three cirlces of spheres or
cubes for hours, minutes and seconds.
///
///    The visualization displays three cirlces for each hours, minutes and seconds. The
camera rotates around the clock.
/// The scene also contains a light that will light the scene according to the daytime.
///
/// \author Hans Ferchland
/// \date 19.12.2012
```

```
class ClockVisualizationCircle : public Ogre::FrameListener, public ClockVisualization
{
public:
	// --------------------------------------------------------------------------------
-
	//		Methods
	// --------------------------------------------------------------------------------
-

	/// \fn ClockVisualizationCircle::ClockVisualizationCircle(Ogre::SceneManager*
sceneManager,
	/// Ogre::Camera* cam, int hourFormat = 1);
	///
	/// \brief Contructor of the circle visualization.
	///
	///    Takes a SceneManager from Ogre to hang nodes into the scenegraph and a camera
that
	/// will rotate around the clock.
	///    The viualization can handle 12 and 24 hour format.
	///
	/// \author Hans Ferchland
	/// \date 19.12.2012
	///
	/// \param [in] sceneManager
	/// If non-null, manager for scene.
	/// \param [in] cam
	/// If non-null, the camera that will view the scene and rotate around the clock.
	/// \param hourFormat
	/// (optional) the hour format.
	///    \sa Ogre::SceneManager, Ogre::Camera

	ClockVisualizationCircle(Ogre::SceneManager* sceneManager,
		Ogre::Camera* cam, int hourFormat = 1);

	/// \fn bool ClockVisualizationCircle::frameRenderingQueued(const Ogre::FrameEvent&
evt);
	///
	/// \brief Frame rendering for updateing from Ogre3D via FrameListener interface.
	///
	///    Updates the clock every second. Updates the camera and animation of the clock.
	///
	/// \author Hans Ferchland
	/// \date 19.12.2012
	///
	/// \param evt
	/// The frame event from Ogre3D.
	///
	/// \return true if rendering should continue, false otherwise.
	///    \sa Oger::FrameEvent

	bool frameRenderingQueued(const Ogre::FrameEvent& evt);

private:

	/// \fn Ogre::ColourValue ClockVisualizationCircle::interpolateColors(float time);
	///
	/// \brief Interpolates the colors for the daytime-light.
	///
	///    Uses a color-ramp implemented in colours.h and interpolates between two values
for each update.
	///
	/// \author Hans Ferchland
	/// \date 19.12.2012
	///
	/// \param time
	/// The current time from 0 to 24.
	///
	/// \return The new color value of the daytime-light.

	Ogre::ColourValue interpolateColors(float time);

	/// \fn void ClockVisualizationCircle::createClock();
	///
	/// \brief Creates the circle clock visualization with all components.
	///
	///    Creates the nodes and hangs them into scenegraph. Creates materials and
	/// geometry and attaches them to the nodes.
	///
	/// \author Hans Ferchland
	/// \date 19.12.2012

	void createClock();

	// ---------------------------------------------------------------------------------
-
	//		Variables
	// ---------------------------------------------------------------------------------
-.
	Ogre::SceneNode* mClockNode;  /*!< The clock node */
	Ogre::SceneNode* mMiddleSphereNode; /*!< The middle sphere node */
	// Nodes for Symbols
	Ogre::SceneNode* mHoursNode;  /*!< The hours node */
	Ogre::SceneNode* mMinutesNode;  /*!< The minutes node */
	Ogre::SceneNode* mSecondsNode;  /*!< The seconds node */
	// Light node
	Ogre::Light* mDaytimeLight; /*!< The daytime light */
	Ogre::Vector3* mLightPosition;  /*!< The light position */
	// time update
	float mAnimationTime;   /*!< Time of the animation */
	// references the ogre basic camera
	Ogre::Camera* mCamera;  /*!< The camera */
	Ogre::Vector3 mCameraPosition;  /*!< The camera position */

	/// \property std::vector <Ogre::Entity*> mVectorHourGeom
	///
	/// \brief holds all nodes for the hours.
	///
	/// \return The vector with the hour geometrys.

	std::vector <Ogre::Entity*> mVectorHourGeom;

	/// \property std::vector <Ogre::Entity*> mVectorMinuteGeom
	///
	/// \brief holds all nodes for the minutes.
	///
	/// \return The vector with the minute geometrys.

	std::vector <Ogre::Entity*> mVectorMinuteGeom;

	/// \property std::vector <Ogre::Entity*> mVectorSecondGeom
	///
	/// \brief holds all nodes for the seconds.
	///
	/// \return The vector with the second geometrys.

	std::vector <Ogre::Entity*> mVectorSecondGeom;
};

#endif
CLOCKVISUALIZATIONCIRCLE.CPP


#include "stdafx.h"

#include "StandupApplication.h"
#include "ClockVisualisation.h"
#include "ClockVisualizationCircle.h"

ClockVisualizationCircle::ClockVisualizationCircle(Ogre::SceneManager* sceneManager,
	Ogre::Camera* cam, int hourFormat) : ClockVisualization(sceneManager, hourFormat),
	mCamera(cam) {
	// set max values
	mHours = 12 * mHourFormat;
	mMinutes = 60;
	mSeconds = 60;
	// create the clock components
	createClock();
	// get the initial time
	const tm& localTime = Clock::getDisplayTime(Clock::getCurrentSecond());
	mCurrentSeconds = localTime.tm_sec;
	mCurrentMinutes = localTime.tm_min;
	mCurrentHours = localTime.tm_hour % (12 * mHourFormat);
	mAnimationTime = 0;
	// create ambient light for daytime depented lighting
	mDaytimeLight = mSceneMgr->createLight("ClockAmbientLight");
	mDaytimeLight->setType(Ogre::Light::LT_POINT);
```

```
        mDaytimeLight->setCastShadows(true);
        mDaytimeLight->setPosition(*(mLightPosition = new Ogre::Vector3(20,100,0)));
        //mDaytimeLight->setAttenuation(1000, 0.f, 1.0f, 1.f);
        // set initial cam distance and alginment in front of mMinutes plane
        mCameraPosition = Ogre::Vector3(0,15,-66);
}

//-------------------------------------------------------------------------------------
//Updates the clock continously for each frame.
bool ClockVisualizationCircle::frameRenderingQueued(const Ogre::FrameEvent& evt) {
        ///////////////////////////////////////////////////////////////////////////
        //              DEBUG
        ///////////////////////////////////////////////////////////////////////////
        //static float dayInterpolationTime=0.0f;
        //mAnimationTime += (evt.timeSinceLastEvent);
        //dayInterpolationTime+=evt.timeSinceLastEvent;
        //if(dayInterpolationTime>24.0f) dayInterpolationTime-=24.0f;
        ///////////////////////////////////////////////////////////////////////////

        // counter for slowed clock update (needs only once per second ;)
        static int second=-1;
        // step forwad in time for animation
        mAnimationTime += (evt.timeSinceLastEvent);
        // get current time from clock
        const tm& localTime = Clock::getDisplayTime(Clock::getCurrentSecond());
        float dayInterpolationTime = localTime.tm_hour + (localTime.tm_min * 0.01667f);
        // get the current secs, mins and mHours
        mCurrentSeconds = localTime.tm_sec;
        mCurrentMinutes = localTime.tm_min;
        mCurrentHours = localTime.tm_hour % (12 * mHourFormat); // in right time format (12 vs
24)

        // Apply Time visualization every second
        if(mCurrentSeconds!=second)
        {
                second = mCurrentSeconds;
                // Seconds
                for (int i = 0; i < mSeconds; i++) {
                        Ogre::Entity* node = mVectorSecondGeom[(i+mSeconds/4) % mSeconds];
                        // set material and shadows for the second that really is
                        node->setMaterialName(i <
mCurrentSeconds?"Standup/Clock/Cubemap_Seconds":"Standup/Clock/Cubemap_Trans");
                        node->setCastShadows(i < mCurrentSeconds);
                }

                // Minutes
                for (int i = 0; i < mMinutes; i++) {
                        Ogre::Entity* node = mVectorMinuteGeom[(i+mMinutes/4) % mMinutes];
                        // set material and shadows for the minute that really is
                        node->setMaterialName(i <
mCurrentMinutes?"Standup/Clock/Cubemap_Minutes":"Standup/Clock/Cubemap_Trans");
                        node->setCastShadows(i < mCurrentMinutes);
                }
                // Hours
                for (int i = 0; i < mHours; i++) {
                        Ogre::Entity* node = mVectorHourGeom[(i+mHours/4) % mHours];
                        // set material and shadows for the minute that really is
                        node->setMaterialName(i < mCurrentHours ? "Standup/Clock/Cubemap_Hours"
: "Standup/Clock/Cubemap_Trans");
                        node->setCastShadows(i < mCurrentHours);
                }
        }
        // Light Color interpolate
        Ogre::ColourValue newColor =
interpolateColors(dayInterpolationTime);//localTime.tm_hour + (localTime.tm_min * 0.01667f));
        mDaytimeLight->setDiffuseColour(newColor);
        mDaytimeLight->setSpecularColour(newColor);
        // light position update animation
        //
        float speed = (mAnimationTime * 0.2f);
        float x = 10 * Ogre::Math::Cos(Ogre::Math::PI * 2 * speed);
        float z = 10 * Ogre::Math::Sin(Ogre::Math::PI * 2 * speed);
        float y;
        mLightPosition->x = x;
        mLightPosition->z = z;
        mDaytimeLight->setPosition(*(mLightPosition));

        // Animate the clock via rotation
        // Rotation of hours, minutes and seconds
        Ogre::Quaternion q1,q2,q3;
        float hoursTime = mAnimationTime * (1.0f/3600.0f);
        mHoursNode->setOrientation(q1=Ogre::Quaternion(Ogre::Radian(Ogre::Math::PI * hoursTime
* 2.0f), Ogre::Vector3::NEGATIVE_UNIT_Y));
        float minutesTime = mAnimationTime * (1.0f/60.0f);
        mMinutesNode->setOrientation(q2=Ogre::Quaternion(Ogre::Radian(Ogre::Math::PI *
minutesTime * 2.0f), Ogre::Vector3::UNIT_Y));
        float secondsTime = mAnimationTime * 0.2f;
        mSecondsNode->setOrientation(q3=Ogre::Quaternion(Ogre::Radian(Ogre::Math::PI *
secondsTime * 2.0f), Ogre::Vector3::UNIT_Z));
        // camera rotation
        Ogre::Matrix3 m1;
        Ogre::Matrix3 m2;

        Ogre::Matrix3 rm;
        rm.FromAngleAxis(Ogre::Vector3::NEGATIVE_UNIT_Z, Ogre::Radian(Ogre::Math::PI*0.5f));
        Ogre::Matrix3 rm2;
        rm2.FromAngleAxis(Ogre::Vector3::UNIT_Y, Ogre::Radian(Ogre::Math::PI*0.5f));

        q1.ToRotationMatrix(m1);
        q2.ToRotationMatrix(m2);
        Ogre::Matrix4 m4 = Ogre::Matrix4( (m1*m2*rm2).Inverse());
        m4.setTrans(mCameraPosition);
        mCamera->setCustomViewMatrix(true, m4);
        return true;
}

//-------------------------------------------------------------------------------------
//              Methods
//-------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------
// interpolates a colour value by given Daytime from 0f - 24f
Ogre::ColourValue ClockVisualizationCircle::interpolateColors(float time) {

        // colours.h contains unsigned char interpolationTable[768]
        // holding 256 rgb triplets
        //
        #include "colours.h"

        float scaledTime = time*(256.0f/24.0f);
        int tableindex=(int)scaledTime;
        float t = scaledTime-(float)tableindex;
        // linear interpolation between two neighbouring table entries
        return Ogre::ColourValue(((float)interpolationTable[tableindex*3+0]*(1-
t)+(float)interpolationTable[(tableindex*3+3)%768]*t)*(1.0f/256.0f),

        ((float)interpolationTable[tableindex*3+1]*(1-
t)+(float)interpolationTable[(tableindex*3+4)%768]*t)*(1.0f/256.0f),

        ((float)interpolationTable[tableindex*3+2]*(1-
t)+(float)interpolationTable[(tableindex*3+5)%768]*t)*(1.0f/256.0f), 1);
}
//-------------------------------------------------------------------------------------
// Creates the clocks base-components initially
void ClockVisualizationCircle::createClock() {
        Ogre::SceneNode* rootNode = mSceneMgr->getRootSceneNode();

        mClockNode = rootNode->createChildSceneNode("MainCircleClockNode");
        mClockNode->setPosition(0,0,0);
        mMiddleSphereNode = mClockNode->createChildSceneNode("ClockRotationNode");
        //mClockNode->setOrientation(Ogre::Quaternion(Ogre::Radian(Ogre::Math::PI * 0.5f),
Ogre::Vector3::NEGATIVE_UNIT_Z));
        mHoursNode = mClockNode->createChildSceneNode("HoursNode");
        mMinutesNode = mHoursNode->createChildSceneNode("MinutesNode");
        mSecondsNode = mMinutesNode->createChildSceneNode("SecondsNode");

        Ogre::Entity* sphere = mSceneMgr->createEntity(Ogre::SceneManager::PT_SPHERE);
        sphere->setMaterialName("Standup/Clock/Cubemap_Hours");
        sphere->setCastShadows(false);
        mMiddleSphereNode->attachObject(sphere);
        mMiddleSphereNode->setScale(0.2,0.2,0.2);

        Ogre::SceneNode* tempNode;
        Ogre::Entity* tempGeometry;
        Ogre::String hoursMaterialName = "Standup/Clock/Cubemap_Hours";
        Ogre::String minutesMaterialName = "Standup/Clock/Cubemap_Minutes";
        Ogre::String secondsMaterialName = "Standup/Clock/Cubemap_Seconds";
        float scaleHours = 0.05f;
        float scaleMinutes = 0.02f;
        float scaleSeconds = 0.005f;
        float x;
        float y;
        float theta;
        float r_hours = 25;
        float r_minutes = 20;
        float r_seconds = 15;
        char numstr[21]; // enough to hold all numbers up to 64-bits

        for (int i  = 0; i < mHours; i++) {
                theta = i/(float)mHours * Ogre::Math::PI * 2;
                x = r_hours * Ogre::Math::Cos(theta);
                y = r_hours * Ogre::Math::Sin(theta);
                sprintf(numstr, "%dHourNode", i);

                tempGeometry = mSceneMgr->createEntity(Ogre::SceneManager::PT_SPHERE);
                tempGeometry->setMaterialName(hoursMaterialName);
                tempGeometry->setCastShadows(true);

                tempNode = mHoursNode->createChildSceneNode(numstr, Ogre::Vector3(0, y, x),
                        Ogre::Quaternion(Ogre::Radian(theta), Ogre::Vector3::NEGATIVE_UNIT_X));

                float rscale = scaleHours;
                if((i%(3 * mHourFormat))!=0) rscale*=0.66f;
                tempNode->setScale(rscale, rscale, rscale);
                tempNode->attachObject(tempGeometry);
                //tempNode->setVisible(false);
                mVectorHourGeom.push_back(tempGeometry);
        }

        for (int i  = 0; i < mMinutes; i++) {
                theta = i/(float)mMinutes * Ogre::Math::PI * 2;
                x = r_minutes * Ogre::Math::Cos(theta);
                y = r_minutes * Ogre::Math::Sin(theta);
                sprintf(numstr, "%dMinuteNode", i);

                tempGeometry = mSceneMgr->createEntity(Ogre::SceneManager::PT_SPHERE);
                tempGeometry->setMaterialName(minutesMaterialName);
                tempGeometry->setCastShadows(true);

                tempNode = mMinutesNode->createChildSceneNode(numstr, Ogre::Vector3(0, y, x),
                        Ogre::Quaternion(Ogre::Radian(theta), Ogre::Vector3::NEGATIVE_UNIT_X));

                float rscale = scaleMinutes;
                if((i%5)!=0) rscale*=0.66f;
                tempNode->setScale(rscale, rscale, rscale);
                tempNode->attachObject(tempGeometry);
                mVectorMinuteGeom.push_back(tempGeometry);
        }

        for (int i  = 0; i < mSeconds; i++) {
                theta = i/(float)mSeconds * Ogre::Math::PI * 2.0f;
                x = r_seconds * Ogre::Math::Cos(theta);
                y = r_seconds * Ogre::Math::Sin(theta);
                sprintf(numstr, "%dSecondNode", i);

                tempGeometry = mSceneMgr->createEntity(Ogre::SceneManager::PT_CUBE);
                tempGeometry->setMaterialName(secondsMaterialName);
                tempGeometry->setCastShadows(true);

                tempNode = mSecondsNode->createChildSceneNode(numstr, Ogre::Vector3(0, y, x),
                        Ogre::Quaternion(Ogre::Radian(theta), Ogre::Vector3::NEGATIVE_UNIT_X));

                float rscale = scaleSeconds;
                if((i%5)!=0) rscale*=0.66f;
                tempNode->setScale(rscale, rscale, rscale*5.0f);
                tempNode->attachObject(tempGeometry);
                mVectorSecondGeom.push_back(tempGeometry);
        }
        //Ogre::SceneNode* zeroNode = mClockNode->createChildSceneNode("ZeroNode",
Ogre::Vector3(0, 20, 0));
        //zeroNode->scale(0.05f, 0.05f, 0.05f);
        //zeroNode->attachObject(sphere);
}
```

Quellcode-Listing Bildverarbeitung

```
Standup_GUI.m
===========================================================================

function varargout = Standup_GUI(varargin)
% STANDUP_GUI MATLAB code for Standup_GUI.fig
%      STANDUP_GUI, by itself, creates a new STANDUP_GUI or raises the
existing
%      singleton*.
%
%      H = STANDUP_GUI returns the handle to a new STANDUP_GUI or the
handle to
%      the existing singleton*.
%
%      STANDUP_GUI('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in STANDUP_GUI.M with the given input
arguments.
%
%      STANDUP_GUI('Property','Value',...) creates a new STANDUP_GUI or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Standup_GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Standup_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Standup_GUI

% Last Modified by Hans Ferchland v2.5 20-Dec-2012 17:03:13

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Standup_GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @Standup_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Standup_GUI is made visible.
function Standup_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Standup_GUI (see VARARGIN)

% Choose default command line output for Standup_GUI
handles.output = hObject;
% possible states of the person (1, 2, 3, 4, 5)
handles.personState = struct('state', {'sleeping', 'up', 'down', 'in',
'out'});
% current state index
handles.currentStateNo = 1;
% current state of the person
handles.currentState = handles.personState(handles.currentStateNo);
% create video input handle from plug and play hardware (must support
% YUY2_320x240)
handles.vid = videoinput('winvideo', 1, 'YUY2_320x240');
handles.stateUpdateFlag = true;
handles.timerCounter = 0;
handles.timerMax = 25;
% Update handles structure
guidata(hObject, handles);
webcamStream(handles.vid, handles, hObject);
% UIWAIT makes Standup_GUI wait for user response (see UIRESUME)
% uiwait(handles.standup);

% --- Outputs from this function are returned to the command line.
function varargout = Standup_GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function webcamStream(vid, handles, hObject)
%% Set video input object properties for this application.
% Set Framegrab intervall to every two frames
set(vid,'TriggerRepeat',Inf);
vid.FrameGrabInterval = 2;

%% Set value of a video source object property.
% set source for video
vid_src = getselectedsource(vid);
% set a tag for the webcam stream
set(vid_src,'Tag','motion detection setup');
% set color space to rgb
set(vid,'ReturnedColorSpace','rgb');

%% Start acquiring frames.
start(vid)

%% Declare variables
% wiener filter neighbourhood
wienerFilter = [7 7];
% segmentatiio parameter for threshold adjustment
fudgeFactor = .5;
% top, left and right boundary
data = getdata(vid,1);
% find out the initial width and height
first_gray = rgb2gray(data(:,:,:,1));
[height, width] = size(first_gray);
```

```
height = uint32(height);
width = uint32(width);
% calculate the side space and top space
top = uint32(0.33 * height);
left = uint32(0.2 * width);
right = uint32(0.8 * width);
% holds the values for motion percentage for the last 5 frames
topAverage = [0 0 0 0 0];
leftAverage = [0 0 0 0 0];
rightAverage = [0 0 0 0 0];
% current value in calculation for the average motion value
avgCounter = 1;
avgValues = 5;
% handle for the GUI frame
axes(handles.cameraStream);

%% Calculate difference image and display it.
while( isvalid(vid) ) %vid.FramesAcquired<=frameCount+2) % Stop after 100
frames
    %% Get data from the last two frames
    data = getdata(vid,2);
    % increase the framecounter
    frameCounter = vid.FramesAcquired;

    %% get images from data
    first_gray = rgb2gray(data(:,:,:,1));
    second_gray = rgb2gray(data(:,:,:,2));
    % smooth images with wiener filter
    first_img_wiener =  wiener2(first_gray, wienerFilter);
    second_img_wiener = wiener2(second_gray, wienerFilter);

    %% calculate the difference
    % adjust the histogram and calc the difference
    diff_im_wiener = adjustImage( imabsdiff(first_img_wiener,
second_img_wiener) );

    %% Boundary Segmentation
    segout = boundaryDetect( diff_im_wiener, fudgeFactor );

    %% get the top, left and right part of the image
    % parts for the segmentation components
    top_im_seg = segout(1:top,left:right);
    left_im_seg = segout(top:height,1:left);
    right_im_seg = segout(top:height,right:width);
    % parts for the motion components
    top_im_diff = diff_im_wiener(1:top,left:right);
    left_im_diff = diff_im_wiener(top:height,1:left);
    right_im_diff = diff_im_wiener(top:height,right:width);

    %% Analyse the motion and segmentation in the picture
    topMotion = analyseMotion(top_im_diff, top_im_seg, 64);
    leftMotion = analyseMotion(left_im_diff, left_im_seg, 64);
    rightMotion = analyseMotion(right_im_diff, right_im_seg, 64);
    % get the current values and write them into average array
    topAverage(avgCounter) = topMotion;
    leftAverage(avgCounter) = leftMotion;
    rightAverage(avgCounter) = rightMotion;
    % calculate the average from the last five values
    topAvg = sum(topAverage)/avgValues;
    leftAvg = sum(leftAverage)/avgValues;
    rightAvg = sum(rightAverage)/avgValues;

    %% Print results for the frame in the GUI
    set(handles.leftValueAvg,'String',topAvg);
    set(handles.topValueAvg,'String',leftAvg);
    set(handles.rightValueAvg,'String',rightAvg);

    set(handles.leftValueCur,'String',leftMotion);
    set(handles.topValueCur,'String',topMotion);
    set(handles.rightValueCur,'String',rightMotion);

    set(handles.frameCount,'String',frameCounter);

    %% Respond via GUI according to the average values
    handles = response(topAvg, leftAvg, rightAvg, handles);
    %% Counter for average of the values of motion
    if (avgCounter < avgValues)
        avgCounter = avgCounter + 1;
    else
        avgCounter = 1;
    end

    if (handles.timerCounter < handles.timerMax)
        handles.timerCounter = handles.timerCounter + 1;
    else
        handles.timerCounter = 0;
        handles.stateUpdateFlag = true;
    end

    %% Show plots for each part analyzed
    % normal image and wiener image
    subplot(2,1,1), imshowpair(first_gray, first_img_wiener, 'montage');
axis off;
    % segmentation and image difference
    subplot(2,1,2), imshowpair(segout, diff_im_wiener, 'montage'); axis
off;

    %% Update handles structure
    guidata(hObject, handles);
end
%% stop video
stop(vid)

% --- Executes when user attempts to close standup.
function standup_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to standup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% delete the video handle at closing
delete(handles.vid);
% Hint: delete(hObject) closes the figure
delete(hObject);


% --- Executes when standup is resized.
function standup_ResizeFcn(hObject, eventdata, handles)
% hObject    handle to standup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

===========================================================================
adjustImage.m
===========================================================================

function imgCorrect = adjustImage( img )
%adjustImage correct the image
```

```
%   streches the histogram of the image (historgram linearizing) and appys
%   a gamma correction by gamma=1.1
    % find max gray-value
    % convert matrix to vector and get maximum
    maxValue = max(img(:));
    % find min gray-value
    % get a vector with minimums of each column. get min
    minValue = min(min(img));

    if (maxValue == minValue)
        imgCorrect = img;
    else
        % correct the image
        imgCorrect = imadjust( img, [double(minValue)/255
double(maxValue)/255], [0 1], 1.1 );
    end
end

===========================================================================
analyseMotion.m
===========================================================================

function motionDegree = analyseMotion( motionPic, segPic, sensivity )
% analyseMotion analyses the motion in a given picture based on grey-value
threshold
%   checks for gray values above sensitity and weightens them with the
segmentation results.
%   this value is divided by the total number of pixels in the
%   picture.
    % get the size of the image part
    [height, width] = size(motionPic);
    % finds all pixels greater than the sensivity
    motionTreshold = motionPic>sensivity;
    % weightens the motion with the segmentation
    weightedMotion = motionTreshold .* segPic;
    % summs all up
    sumWeighted = sum(weightedMotion(:));
    % get the percentage of weighted motion in the image frame.
    motionDegree = sumWeighted/(height*width);
    % scale up for better values
    motionDegree = motionDegree*10.0;

end

===========================================================================
boundaryDetect.m
===========================================================================

function [ BWfinal ] = boundaryDetect( image, fudgeFactor )
% BOUNDARYDETECT detects the boundary and segments the image according to
% the fudge factor.
% gets the bw threshold and get the edges via sobel filter. Dilates them
% and fills the gaps. Finally smoothes the image via dimonds.

    % Detect Entire Cell
    [~, threshold] = edge(image, 'sobel');
    BWs = edge(image,'sobel', threshold * fudgeFactor);
    % Dilate the Image
    se90 = strel('line', 3, 90);
    se0 = strel('line', 3, 0);
    BWsdil = imdilate(BWs, [se90 se0]);
    %Fill Interior Gaps
    BWdfill = imfill(BWsdil, 'holes');
    % Smoothen the Object
    seD = strel('diamond',1);
    BWfinal = imerode(BWdfill,seD);
    BWfinal = imerode(BWfinal,seD);

end

===========================================================================
boundaryDetect.m
===========================================================================

function handles = response( topMotion, leftMotion, rightMotion, handles )
%RESPONSE analyses the motion values into a GUI response.
%   Checks for left, right and top values above a defined value and returns
%   GUI changes.

    % Left motion found?
    if (leftMotion > 0.15)
        % Change GUI response label
        set(handles.response,'String','Left In/Out');
        % selects the next state of the person
        handles = selectState(handles, true, false);
    % Right motion found?
    elseif (rightMotion > 0.15)
        % Change GUI response label
        set(handles.response,'String','Right In/Out');
        % selects the next state of the person
        handles = selectState(handles, true, false);
    % Top motion found?
    elseif (topMotion > 0.1)
        % Change GUI response label
        set(handles.response,'String','Top In/Out');
        % selects the next state of the person
        handles = selectState(handles, false, true);
    else
        % Change GUI response label
        set(handles.response,'String','Nothing');
    end
end

===========================================================================
selectState.m
===========================================================================

function [ handles ] = selectState( handles, horiz, vertic )
%SELECTSTATE selects the next state of the person.
%   Changes the state according to the last one and the registered
%   movements.
    if (handles.stateUpdateFlag == false)
        return;
    end
    %% Change the current state
    % current state is sleeping
    if (handles.currentStateNo == 1)
        % movement registered this frame is horizontal or vertical
        if (horiz)
            handles = setState(2, handles);
        elseif (vertic)
            handles = setState(2, handles);
        end
    % current state is up
    elseif (handles.currentStateNo == 2)
        % movement registered this frame is horizontal or vertical
        if (horiz)
            handles = setState(5, handles);
        elseif (vertic)
            handles = setState(3, handles);
        end
    % current state is down
    elseif (handles.currentStateNo == 3)
        % movement registered this frame is horizontal or vertical
        if (horiz)
        elseif (vertic)
            handles = setState(2, handles);
        end
    % current state is in
    elseif (handles.currentStateNo == 4)
        % movement registered this frame is horizontal or vertical
        if (horiz)
            handles = setState(5, handles);
        elseif (vertic)
            handles = setState(3, handles);
        end
    % current state is out
    elseif (handles.currentStateNo == 5)
        % movement registered this frame is horizontal or vertical
        if (horiz)
            handles = setState(4, handles);
        elseif (vertic)
        end
    else

    end
    %% Change GUI state label
    set(handles.stateLabel,'String',char(handles.currentState.state));
    handles.stateUpdateFlag = false;
end

function handles = setState(stateNumber, handles)
%SETSTATE sets the state for the person/camera
    handles.currentStateNo = stateNumber;
    handles.currentState.state = handles.personState(stateNumber).state;
    handles.timerCounter = 0;
    switch (stateNumber)
        case 1 % 'sleeping'
            handles.timerMax = 10;
        case 2 % 'up'
            handles.timerMax = 30;
        case 3 % 'down'
            handles.timerMax = 30;
        case 4 % 'in'
            handles.timerMax = 20;
        case 5 % 'out'
            handles.timerMax = 20;
    end
end
```