UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Weiterentwicklung von MoleARlert

Studienarbeit

im Studiengang Computervisualistik

vorgelegt von
Tim Oppermann

Betreuer: Prof. Stefan Müller
AG Computergrafik

Koblenz, im Oktober 2009

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

. . . . . .

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Kapitel 1

Einführung

1.1 Was ist MoleARlert?

1.1.1 Projektpraktikum

„MoleARlert“ entstand im Rahmen eines Projektpraktikums der AG Computergrafik, unter Leitung Herrn Prof. Müllers und Herrn Dipl.-Inf. Stefan Rilling, im Wintersemester 2008/2009. Das System wurde von insgesamt zwölf Studierenden der Universität Koblenz-Landau entwickelt.

1.1.2 Kurzbeschreibung

MoleARlert ist ein Spiel, das „Lemmings“-Idee (siehe 2.1.1) aufgreift, Spielfiguren durch Modifikation ihrer Umgebung in ein Ziel zu lenken. Im Gegensatz zu dem klassischen Spiel bewegen sich die virtuellen Figuren, hier Maulwürfe, jedoch in einem real existierendem Ort, hier einem „Mikadoplatz“, dem zentralen Platz auf dem Campus der Universität Koblenz.

Das Ziel des Spiels ist es sie zu Heißluftballons zu führen. Mit Hilfe von Markern lässt sich die Laufrichtung der Maulwürfe beeinflussen. Durch Gesten, die die Spieler durch ihren Körper vollführen, erlernen die Maulwürfe kurzfristig spezielle Fähigkeiten wie das Graben durch Wände und spontan das Bauen von Brücken über Wasserhindernisse hinweg.

1.1.3 Das Team

Das Team bestand, neben den schon erwähnten Betreuern, aus diesen, alphabetisch aufgelisteten, zwölf Studierenden der Computervisualistik:

- Sandy Engelhardt
- Annabell Langs
- Gerrit Lochmann

- Vera Müllenbach
- Tim Oppermann
- Dominik Ospelt
- Elena Root
- Irini Schmidt
- Nicole Thorn
- Tobias Tropper
- Hans-Christian Wollert
- Elisa Ziegler

1.2 Einleitung: Warum diese Arbeit?

1.2.1 Hintergrund

Zu Beginn des Projekts „MoleARlert“ waren die Ziele und Ambitionen sehr hoch gesteckt, daher gab es im Laufe der Entwicklung viele Hürden und Rückschläge.

Am Ende des Projektpraktikum stand ein Projekt, das viele Hindernisse überwunden und Neuland gewonnen hatte. Jedoch litt das System an allerlei Kinderkrankheiten mit den verschiedensten Symptomen, wie regelmäßigen Abstürzen, welche wiederholt in den bestimmten Situationen aber auch augenscheinlich zufällig auftraten. Hinzu kamen eine schlechte Performanz, fehlerhafte Markererkennung und Übertragungsfehler, die in ihrer Summe dazu führten, dass die mühevoll erarbeiteten Fortschritte und Erkenntnisse sich nicht in der Praxis, also in einem funktionierenden Spiel, zeigen konnten.

1.2.2 Das Ziel

Die Idee das Spiel so weit weiter zu entwickeln, dass ein fertiges, funktionierendes Produkt entsteht, dass sich der Öffentlichkeit zeigen kann, lag auf der Hand. Hierzu waren viele Änderungen nötig, die prinzipiell nicht über normale Fehlersuche und -korrektur hinausgehen. Allerdings waren auch viele Punkte offen, die eine intensive Weiterentwicklung nötig machten. Der vielleicht entscheidendste Aspekt war hierbei die Verarbeitung des Kamerabildes (siehe 4.2).

1.2.3 Diese Studienarbeit

Inhalt dieser Studienarbeit ist neben der Beschreibung des Systems vor allem die Veränderungen die an ihm vorgenommen wurden unter besonderer Berücksichtigung der Neu- und Weiterentwicklungen die dazu führten die Reife des Systems zu verbessern.

1.3 Augmented Reality

„Augmented Reality“[1] bezeichnet das anreichern der Realität um virtuelle Bestandteile. Dies geschieht möglichst in Echtzeit mit einem Computersystem, das ein virtuelles, dreidimensionales Bild mit einem realen Bild mischt. Das reale Bild kann, aber muss nicht, von einer Kamera aufgenommen werden.

1.3.1 Anwendungen von Augmented Reality

Augmented Reality findet in vielen Bereichen Anwendung. Hier zu zählen unter anderem:

- Simulationen für Flugzeuge, Fahrzeuge oder anderer technischer Konstruktionen
- Navigation, beispielsweise in Head-Up-Displays in Autos oder Flugzeugen
- Konstruktion und Wartung von technischen Anlagen
- Medizin
- Spiele, so wie MoleARlert oder ARQuake [arq].

Im Allgemeinen finden sich für AR überall dort sinnvolle Anwendungsbereiche, wo zusätzliche komplexe und interaktive Informationen in der Realität gewünscht sind.

Außerdem ergeben sich neue Anwendungsfelder durch neue Technologien und Geräte. So ist es vorstellbar, dass beispielsweise die bisherige Desktop-Metapher für Benutzungsoberflächen für Rechner durch eine Mischung aus tatsächlichen und virtuellen Objekten ersetzt wird.

1.3.2 Möglichkeiten der Bildsynthese

Bildsynthese bezeichnet in der AR die Kombination des realen mit dem virtuellen Bild. Hierzu gibt es verschiedene Technologien.

[1] kurz: AR

Mit Durchsicht

Zum einen kann über einen transparenten Bildschirm das Bild mit der Umgebung gemischt werden.

Häufig wird dabei je ein Display vor den Augen des Betrachters platziert. Ein Computersystem muss nun die Position und Richtung (oder kurz: Pose) des Benutzers im Raum bestimmen, um das Computerbild in der richtigen Perspektive vor die Augen des Betrachters projektieren zu können. Da bei solchen Systemen auch die Position der Augen zur Brille hin berücksichtigt werden muss, ist hier eine aufwendige Kalibrierung nötig.

Nachteile dieser Konstruktion ist, dass die Sicht auf einen Benutzer beschränkt ist und sie leicht verrutschen kann. Außerdem kann eine sichtbare Verzögerung zwischen realen und virtuellen Bild entstehen, da es selbst und die Kamerapose ja erst berechnet werden müssen, während das reale Bild immer sofort zur Verfügung steht.

Auf einem Monitor

Möchte man die Bildsynthese auf einem Bildschirm angezeigt bekommen, so muss das Bild vorher durch mindestens eine Kamera erfasst werden. Das aufgenommene Echtzeitbild wird dann um die zusätzlichen Informationen erweitert und auf einem Monitor ausgegeben.

Dieses Verfahren wurde bei MoleARlert eingesetzt.

Weitere Varianten

Holographische oder dreidimensionale Bildschirme könnten in der Zukunft eine interessante Rolle bei AR-Anwendungen spielen, da sie bei ausgereifter Technik die dreidimensionale Betrachtung für mehrere Benutzer gleichzeitig ohne mehr Hardware ermöglichen würden und gleichzeitig die Realität mit der virtuellen Welt synchronisieren könnten.

Es gibt jetzt schon Projekte, die auf aktuellen mobilen Geräten wie Handys oder PDAs[2] aufbauen, um zusammen mit GPS[3] und Bilderkennung zusätzliche Informationen zum aktuell betrachteten Objekt zu liefern.

[2]Personal Digital Assistant
[3]Global Positioning System

Kapitel 2

Konzeption der Spiels und die verwendeten Ressourcen

In diesem Kapitel werden die wichtigsten Ideen, die das Konzept des Spiels ausmachen erörtert und teilweise auch solche die nicht vollständig umgesetzt wurden.

2.1 Spielidee

Die Idee des Projektes war es, ein Spiel zu entwickeln, dass ein allgemein bekanntes Konzept auf in die Augmented-Realtity-Welt adaptiert.

2.1.1 Inspiration „Lemmings“

Die Inspiration ging dabei von dem Spieleklassiker „Lemmings“ (Bildschirmfoto siehe Abbildung 2.1) von der Firma Psygnosis[1] [roc] aus dem Jahr 1991 aus. In „Lemmings“ hat der Spieler die Aufgabe virtuelle Lemminge durch eine zweidimensionale Welt in der Seitenansicht zu einem Ziel zu führen. Die Herausforderung besteht darin die Hindernisse auf dem Weg dort hin durch den geschickten Einsatz von Spezialfähigkeiten zu überwinden.

Der Spieler steuert also aktiv die passiven, stupiden Lemminge. Dieser Umstand findet sich auch bei MoleARlert wieder.

2.1.2 „MoleARlert“

Der Name „MoleARlert“

MoleARlert ist ein Kunstwort aus den beiden englischen Wörtern Mole für Maulwurf, Alert für Alarm und der Abkürzung AR die für Augmented Reality, also angereicherte Realität steht.

[1] Heute: Rockstar North

Abbildung 2.1: Bildschirmfoto aus dem Spiel Lemmings.

2.1.3 Maulwürfe

In MoleARlert werden Maulwürfe statt Lemmingen eingesetzt um Marken rechtliche Bedenken auszuräumen. Die Blindheit der Maulwürfe passt auch gut zu ihrem starren, nicht vorausschauenden Verhalten in der Spielwelt von MoleARlert.

2.1.4 Die Welt

Anders als in Lemmings ist die Welt in MoleARlert dreidimensional und nur teilweise virtuell, da das Spiel gleichzeitig im Computer wie auch auf einem realen Platz existiert. Das reale Spielfeld wird durch eine Kamera als Videobild erfasst. Die virtuellen Spielaspekte werden mit einer 3D-Grafik-Engine so auf das Bild gerechnet, dass die beiden Welten für den Betrachter verschmelzen sollen.

Mehrere Spieler

Ein weiterer Unterschied besteht in der Mehrspielerfähigkeit von MoleARlert: Das Spiel ist sogar darauf ausgelegt, dass mehrere Spieler zusammen versuchen ein Ziel zu erreichen. Dazu müssen sie miteinander kommunizieren und sich selbst organisieren. In der Praxisanwendung zeigte sich

durch diese Komponente die vielleicht herrausragenste Eigenschaft des Spiels (siehe 7.3).

Bildsynthese

Ziel der Entwicklung war auch ein Erreichen einer möglichst hohen Immersion und Involviertheit der Benutzer in das Spielgeschehen, dass jedoch nur eingeschränkt erreicht werden konnte. Um dies zu erreichen sollte beispielsweise Personen im Videobild erkannt und durch grobe Umrisse ausgeschnitten werden. Hier sind weitere Ansätze in Zukunft denkbar.

2.2 Spielfeld

Das reale Spielfeld wurde auf dem „Mikadoplatz“ auf dem Campus der Universität Koblenz aufgebaut. Der Platz besteht aus 110cm x 110cm großen Platten. Mit Hilfe von Absperrband wurde auf dem Platz ein 12 mal 12 Kacheln großes Feld markiert. Das virtuelle Spielfeld hat entsprechend viele Kacheln und wird durch Kamerakalibrierung über das reale Feld gelegt. Allerdings ist das virtuelle Feld ein wenig idealisiert, da die geringen, aber sichtbaren Unebenheiten des Platzes nicht berücksichtigt werden.

2.3 Interaktion

Die Interaktion des Benutzers findet nun in der realen Welt statt und hat dabei Auswirkungen auf die gemischte Ansicht im Computer. Der selbe Umstand besteht streng genommen zwar auch, wenn der Benutzer mittels eines herkömmlichen Steuergeräts wie einer Maus oder in der fortgeschrittenen, modernen Variante, mit einer Wii-Fernbedienung mit dem Computer interagiert. Jedoch wird der Benutzer dabei in der Regel nicht Teil des Spiels selber. So ist es in MoleARlert ja der Fall, dass die Interaktion des Benutzers auf dem realen Platz zur Folge hat, dass er zwangsläufig dabei auch von der Kamera erfasst wird und somit direkt im Spielgeschehen integriert ist. Außerdem wird die größte mögliche Natürlichkeit nur dann erreicht, wenn gar keine Eingabeeinheit vorhanden ist, die speziell bedient werden muss. Das Spiel bietet so eine geringe Einstiegshürde und wird für jeden in der geeigneten körperlichen Verfassung spiel- und begreifbar.

Die Interaktion mit den Maulwürfen findet einerseits durch farbige Schilder mit einem aufgedruckten Richtungspfeil, sogenannten Markern, und durch das ausführen von zwei Gesten statt.

Die Marker dienen dazu die Laufrichtung der Maulwürfe zu beeinflussen und stellen meistens die wichtigste Interaktionskomponente im tatsächlichen Spielgeschehen dar, wobei dies natürlich enorm durch das Leveldesign beeinflusst wird. Sie haben eine Größe von 90 x 90 cm und pas-

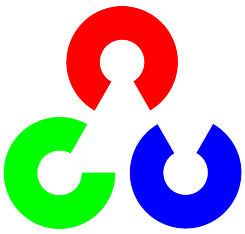
sen so mit einem guten Abstand zum Rand einer jeweiligen Kachel auf die je ca. 110 cm breiten und hohen Kacheln des Mikadoplatzes. Die auf den Markern aufgedruckten Richtungspfeile zeigen an in welche Richtung der Marker zeigt. Diese dienen ausschließlich dem Spieler als Orientierung. Die Bilderkennung nutzt die grüne Ecke um die Richtung heraus zu finden.

Neben der Richtungsmanipulation mit Markern können Spieler auch zwei verschiedene simple Gesten ausführen. Die eine besteht einfach im geraden Stehen auf einem Feld und die andere im Hinhocken. Das Stehen erzeugt eine Brücke um Schluchten zwischen zwei hohen Ebenen oder Wasser zu überbrücken. Das Hocken sorgt dafür, dass der nächste Maulwurf der das Feld erreicht zum Gräber und so mit durch die nächste Wand gräbt bis er wieder auf einem herkömmlichen Feld angekommen ist.

2.4 Bibliotheken

MoleARlert verwendet eine Reihe von externen Bibliotheken, die alle frei erhältlich sind.

2.4.1 OpenCV



OpenCV [ocv] ist eine quelloffene Programmbibliothek und wurde von Intel vor allem für die Echtzeitbildverarbeitung entwickelt. Sie bietet schon ab Werk viele für das Projekt nützliche Funktionen wie radiale Entzerrung, Kamerakalibrierung und den direkten Zugriff auf die Kamera an.

Die verwendete Version ist 1.1 pre 1, also eine Vorveröffenlichungsversion, die eingesetzt wurde um eine höhere Kamerakompatibilität zu erreichen.

2.4.2 OGRE - Object-Oriented Graphics Rendering Engine

OGRE [ogr] ist eine quelloffene, plattformunabhänginge 3D-Grafik-Engine. Der Aufbau ist strikt objektorientiert gehalten. Eine Vielzahl von Plug-Ins ermöglichen eine einfache Erweiterbarkeit der Engine. Ogre ist auch für Multiprozessoren optimiert und verwaltet dazu den Speicher für sämtliche Objekte selbst. Auch interne Manager für die Verwaltung von bestimmten Ressourcen werden zur Verfügung gestellt um den Umgang mit elementaren Objekten, wie beispielsweise Texturen und Modellen, zu vereinfachen.

OGRE kann sowohl mit DirectX als auch mit OpenGL betrieben werden. Außerdem ist Unterstützung für die Shader-Technologien Cg, HLSL

und GLSL fest integriert. Diese können in Materialskripts die zur Laufzeit geladen einfach definiert werden. Zur effizienten Vertex-Erzeugung können außerdem Frame-Buffer-Objekte und weitere Buffermethoden eingesetzt werden.

In MoleARlert wurde die Version 1.6.1 verwendet.

2.4.3 RakNet

Bei RakNet [jen] handelt es sich um eine plattformübergreifende, quelloffene Programmbiblitohek, die in C++ geschrieben ist. Das Besondere bei ihr ist, dass sie eine verlässliche Datenübertragung über UDP ermöglicht. UDP bietet im Gegensatz zu TCP keine Paketsicherung aber ist wesentlich performanter als herkömmliches TCP. Bei UDP ist normalerweise nicht sicher gestellt in welcher Reihenfolge oder ob überhaupt Pakete ankommen. RakNet übernimmt hier auf UDP die Sicherung dieser Aufgaben sehr effizient.

Die in MoleARlert verwendete Version ist 3.261.

2.4.4 FMOD

FMOD [fmd] ist eine kommerzielle, plattformübergreifende Programmbibliothek zum Abspielen von Tönen. Sie bietet zahlreiche Möglichkeiten Töne auch mit dreidimensionaler Raumanordnung abzuspielen.

In MoleARlert wurde sie vor allem wegen der leichten Einbindbarkeit und dem geringen Res

In MoleARlert wird die Version 4.22.07 verwendet.

2.4.5 tinyXML

tinyXML [xml] ist eine simple, in C++ geschriebene Bibliothek zum einlesen und verarbeiten von XML-Dateien. Das Hauptaugenmerk liegt dabei auf der einfachen Integration in bestehende Projekte.

Da in MoleARlert die Komplexität der verwendeten XML-Datenstrukturen relativ gering blieb war tinyXML eine gute und zuverlässige Wahl.

MoleARlert verwendet die Version 2.5.3 von tinyXML.

2.4.6 Plattform

Das System ist komplett auf Windows-Basis entwickelt. Die Verwendung von DirectX 9 erschwert die Portierung. Die Plattformeinschränkung wur-

de durch die Verwendung der Kameras nötig. Künftige Entwicklungen könnten durch den Einsatz von FireWire-Kameras eine Portierbarkeit beispielsweise nach Linux ermöglichen.

Für ein fest installiertes System ist eine Cross-Plattform-Lösung weniger relevant, daher wurden Anstrengungen in diese Richtung zu Gunsten anderer Aufgaben in der Entwicklung nicht unternommen.

MoleARlert läuft und wurde entwickelt auf Windows XP 32-bit mit ServicePack 3 und DirectX 9.0c. Diese Versionen werden auch unbedingt benötigt, damit die Web-Kameras korrekt angesteuert werden können.

Kapitel 3

Aufbau und Installation der Hardware

Das System benutzt drei Kameras um das Spielfeld zu erfassen. Zwei der Kameras erfassen das Spielfeld vom Geländer des D- und eine vom Dach des G-Gebäudes aus.

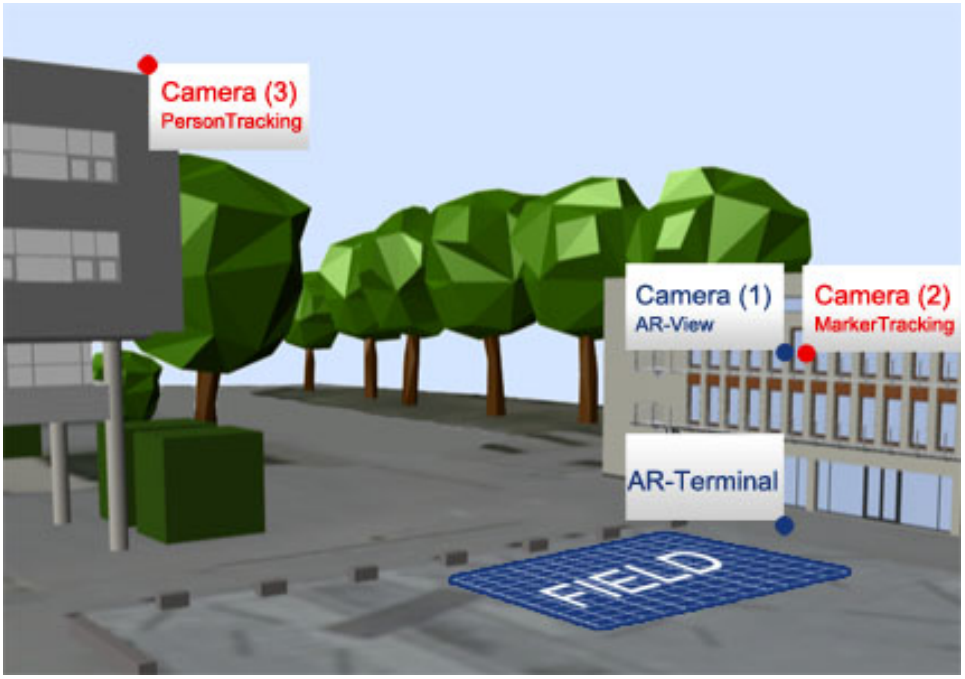


Abbildung 3.1: Anordnung der Kameras auf dem Campus. Quelle: [**?**]

Jede dieser Kameras überträgt ihr Bild per USB zu einem Computer der das Bild dann weiter verarbeitet. Zwei der Computer verwenden das Kamerabild um daraus Informationen über Marker, Gesten und Personen zu

finden und senden ihre jeweiligen Ergebnisse gegeben falls über eine Netzwerkverbindung zu einem zentralen Computer. Dieser verwendet sein Kamerabild um es mit dem virtuell erzeugten Bild zu mischen, so dass das AR-Bild dabei herauskommt. Die diesem Computer über das Netzwerk von den anderen beiden Computer zur Verfügung gestellten Informationen verwendet er um die virtuelle Welt nach programmierten Regeln so zu verändern, dass es den Eindruck macht, als wenn die realen Objekte einen direkten Einfluss auf die virtuelle Welt haben.

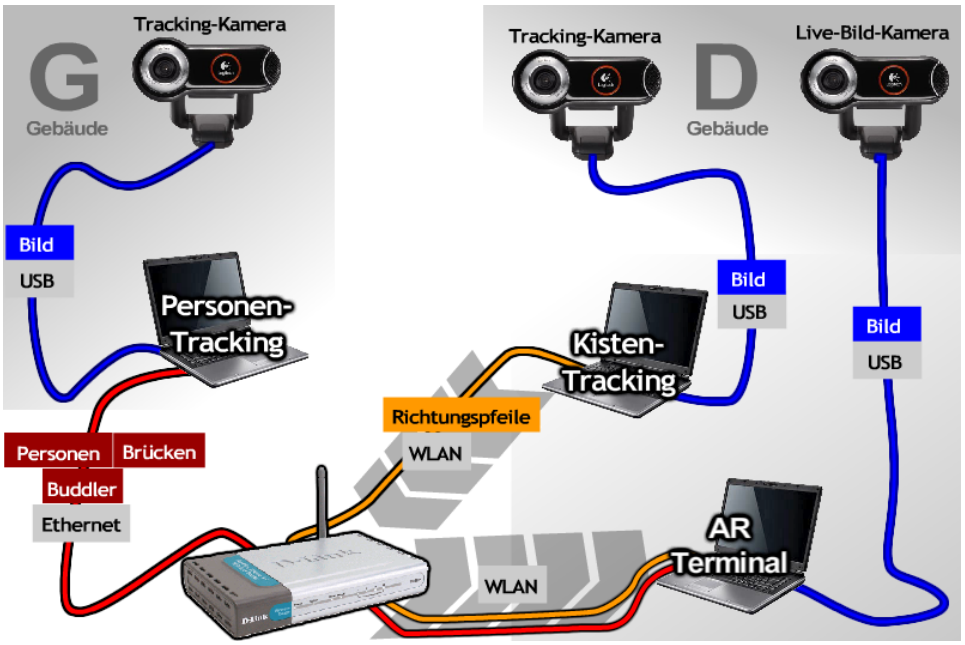


Abbildung 3.2: Schematischer Aufbau der Hardware mit Übertragungstechniken

3.1 Hardware

Die Hardware besteht aus ca. zwei Jahre alten handelsüblichen Rechnern mit dedizierter 3D-Beschleunigung und bis zu 2GB RAM. Die Rechenleistung ist für die AR-Sicht ausreichend.

Für die Bilderkennung werden handelsübliche Web-Kameras im Preissegment um ca. 60 Euro eingesetzt, die jeweils über USB angeschlossen werden.

3.1.1 Logitech Quickcam 9000 Pro

Die Quickcam von Logitech schafft laut Hersteller eine Auflösung von 1600 zu 1200 Pixeln bei 30 Bildern pro Sekunde. Dies wird allerdings nur mit der

mitgelieferten Software und nicht im Zugriff über OpenCV erreicht.

3.1.2 Microsoft VX 7000

Bei der Microsoft VX 7000 handelt es sich auch um eine Web-Kamera mit einem 1600 zu 1200 Pixel auflösenden CMOS Sensor. Mehr Informationen sind leider vom Hersteller nicht zu bekommen.

3.2 Kistentracking

Das Tracken der Kisten geschieht aus quasi der selben Perspektive in der sich der AR-Befindet. Dies ist aber eine rein praktische Lösung um den Aufbauaufwand zu verringern. Theoretisch wäre eine andere Perspektive, insbesondere eine steilere, durchaus möglich und wünschenswert gewesen. Die Kistenerfassung wird auf einem eigenen Rechner ausgeführt. Die Ergebnisse werden dann über das Universitätsnetzwerk an den AR-Viewer übermittelt.

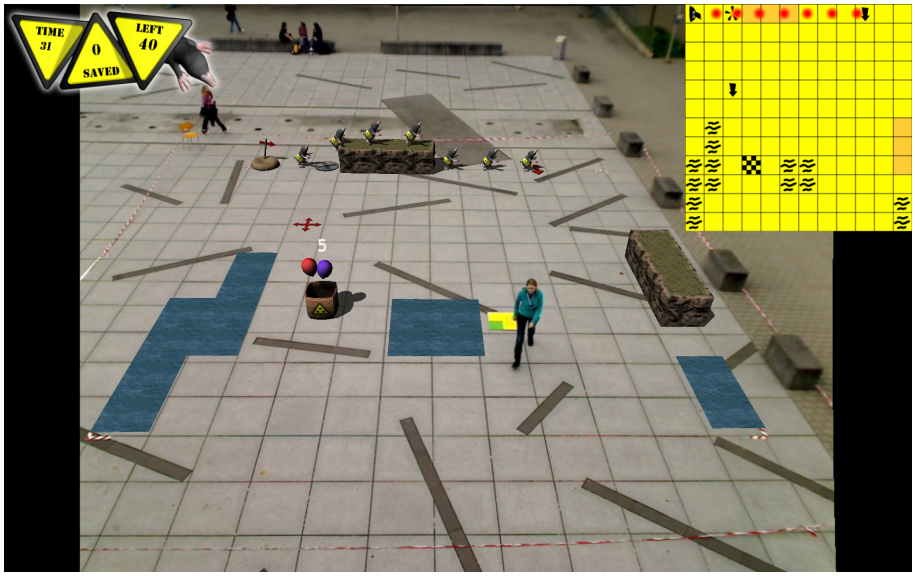


Abbildung 3.3: Sicht vom D-Gebäude im laufenden Spiel.

3.3 Tracking mit Gesten

Für das Tracking mit Gesten wurde eine spezielle Perspektive und zwar die vom G-Gebäude aus verwendet, da es aus der steileren Ansicht leichter ist Überdeckungen mit anderen Personen oder Gegenständen zu erkennen.

Außerdem ist die Kamerapostion noch flach genug um den Höhenunterschied zwischen der gehockten Stellung und der aufrechten unterscheiden zu können.

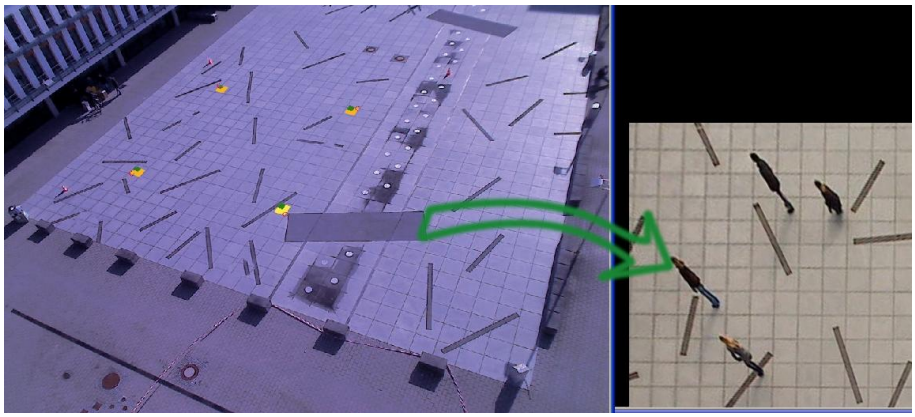


Abbildung 3.4: Sicht vom G-Gebäude mit perspektivischer Transformation des Eingabebildes.

Um dies zu realisieren bestanden besondere Herausforderungen an den Hardwareaufbau, da eine Verbindung mit einem direkten Kabel zum AR-View nicht als praktikable Lösung erschien. Das System wurde als so entwickelt, dass das Tracking, das vom G-Gebäude aus statt findet auf einem separaten Rechner läuft, der lokal auf die Kamera zugreift. Dieser Rechner wird dann über das Netzwerk der Universität mit dem AR-Viewer verbunden.

Kapitel 4

Weiterentwickelte Komponenten

In diesem Kapitel werden Komponenten vorgestellt, die im Rahmen meiner Arbeit weiterentwickelt wurden. Das primäre Ziel lag stets in der Steigerung der Stabilität, gefolgt von Performanz und einfacher Wartbarkeit.

4.1 Netzwerk

Die Netzwerkkomponente musste weiterentwickelt werden, da der Später beschriebene MiddleServer 5.3 (siehe Seite 26) eine enorm schnelle Datenverarbeitung nötig machte.

4.1.1 Zustand und Anforderungen

Die Paketverarbeitung unterschied nicht zwischen Steuerungs- und Datenpaketen. So konnte eine Überfüllung des Puffers entstehen, da die Datenpakete nicht richtig ohne Steuerbefehle verarbeitet werden konnten.

4.1.2 Weiterentwicklung

Die Weiterentwicklung der Komponente löste dieses Problem, in dem im Empfangspuffer erst nach Steuerpaketen gesucht wurde. Dabei wurden Datenpakete wieder an das Ende des Puffers eingereiht. Erst wenn keine Steuerpakete mehr im Puffer sind wird dann in der neuen Lösung die Verarbeitung der Datenpakete begonnen.

Dieses Vorgehen löste das bisherige Problem vollständig.

4.2 Web-Kamera-Zugriff

Der Zugriff auf die Webcam gestaltete sich im Laufe der Entwicklung als eins der schwierigsten Probleme, da die Priorität und die Anforderungen unausweichlich hoch sind. Schließlich ist eine nennenswerte Immersion ohne diese Komponente schlicht nicht möglich. Das Problem wird durch die notwendige radiale Entzerrung zusätzlich erschwert, da pro Frame eine zusätzlich Rechenlast anfällt. Die Anforderungen an die Zeiteffizienz steigt so mit weiter.

4.2.1 Anforderungen

Die Anforderungen lagen hoch:

- Unterstützung von USB-Kameras mit hoher Auflösung
- Effiziente radiale Entzerrung des hochauflösenden Videobildes
- Möglichst leichte Interoperabilität mit OGRE

4.2.2 Ausgangsstadium

Zu Beginn der Weiterentwicklung von MoleARlert wurde ein bereits für OGRE von Tuan Kuranes entwickeltes Plugin verwendet.[xml]

Bildwiederholfrequenz

Die Bildwiederholrate bei einer Videobildauflösung von 640 zu 480 Pixeln lag auf dem Referenzsystem1 bei ca. 10. Bei der gewünschten maximalen Auflösung von 960 zu 720 Pixeln wurden durchschnittlich etwa 5 Wiederholungen erreicht. Geht man nun davon aus, dass 30 Bilder pro Sekunde das Bild für die menschliche Wahrnehmung erst vollkommen flüssig erscheinen lassen war dieser Zustand leider höchstens als unbefriedigend einzuschätzen.

Parallelität

Der 3D-Bildaufbau wurde synchron zur Bildausgabe des Algorithmus der Web-Kamera ausgeben. Daraus folgte eine verschlechtere Benutzererfahrung mit dem System. Die Verlangsamung der Web-Kamera war extrem: Ohne, wurden Bildwiederholraten im dreistelligen Bereich problemlos und dauerhaft auf dem Ausgabesystem erreicht.

4.2.3 Weiterentwicklung

Im Rahmen der Weiterentwicklung wurde das bisherige Web-Kamera-Plugin komplett aus dem System entfernt und durch eine integrierte Lösung ersetzt die nur eine simple Klasse verwendet. Allerdings waren auch kleine Änderungen in anderen Programmteilen notwendig, da nun die Steuerung der Komponente nicht mehr über das Plugin-Management von OGRE möglich war, sondern manuell erfolgen musste. Dabei handelt es sich aber im Wesentlichen nur darum, dass vor jedem Bild das von OGRE erstellt werden soll überprüft wird ob ein neues Kamerabild zur Verfügung steht um dies dann ggf. in den Texturspeicher zu laden.

Asynchrone Textkuraktualisierung

Um den reibungslosen Zugriff auf den Speicher sicher zu stellen werden zwei Mutex verwendet. Der eine überprüft aus der Perspektive des Bildaufbaus heraus, ob in den Speicher gerade geschrieben wird während der andere das Verändern des Speichers während dem Auslesen verhindert. Beide zusammen stellen einen jederzeit exklusiven Zugriff sicher. Außerdem wird durch den Semaphor auf Seite des Web-Kamera-Moduls sicher gestellt, dass nicht von zwei Thread-Instanzen gleichzeitig her in den Speicher geschrieben wird. Mit der richtigen Methode ist der Zugriff auf den Speicher sehr einfach. Man erhält einen ganz herkömmlichen Zeiger. Einzig die Begrenzung des Speichers lässt sich nicht direkt abfragen. Der Zugriff mit `HBL_DISCARD` bietet zum normalen Texturzugriff einen enormen Geschwindigkeitsvorteil. Bei einem herkömmlichen Zugriff, wird eine Kopie es Textur in den Hauptspeicher des Rechners kopiert um ihn beim Auflösen der Sperre wieder auf die Karte zu kopieren. Die Grafikkarte muss dann so lange die Sperre besteht auf ihre Aufhebung warten um mit dem Bildaufbau fortzufahren. Da mit `HBL_DISCARD` das Warten auf die Textur wegfallen kann.

Die Größe des Texturspeichers ist stets in Zweierpotenzen begrenzt, wobei diese verschieden groß dimensioniert werden können.

Defintion der Textur

Bei der Defintion der Textur ist wichtig, dass sie mit der richtigen Option, nämlich `TU_DYNAMIC_WRITE_ONLY_DISCARDABLE`, erstellt wurde und keine MipMaps verwendet werden, da die dynamische Erstellung der MipMaps zusätzliche Rechenzeit in Anspruch nehmen würde, die bei einer statischen Ansicht schlicht überflüssig wäre.

Die `TU_DYNAMIC_WRITE_ONLY_DISCARDABLE`-Option geht mit dem Lock mittels `HBL_DISCARD` für den Texturzugriff einher. Verwendet man an einer der Stellen eine andere Option wird der spezielle Zugriffsmodus auf den

Texturspeicher nicht mehr erreicht mit dem Ergebnis, dass der Zugriff wesentlich länger dauert.

Ent

```
mTexture = TextureManager::getSingletonPtr()->
 createManual(
  "WebcamTexture", /* Eindeutiger Name der Textur zur
     Materialbindung */
  ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
  TEX_TYPE_2D, // Texturtyp. Hier: Einfache 2D-Textur
  mTextureWidth, // Breite in Pixel
  mTextureHeight, // Höhe in Pixel
  0, // Anzahl der MipMaps
  PF_BYTE_RGBA,// lässt sich nicht tatsächlich ändern
  TU_DYNAMIC_WRITE_ONLY_DISCARDABLE);// !wichtig
```

Listing 4.1: Initialisierung der Textur: Wichtig ist die Verwendung von `TU_DYNAMIC_WRITE_ONLY_DISCARDABLE`

Anbindung der Textur an das Material

Nach dem die Textur, wie exemplarisch in Listing 4.1 gezeigt, erstellt worden ist muss es mit einem Material an eine Oberfläche gebunden werden. Dies geschieht zur Laufzeit, wie beispielsweise in Listing 4.2. Das Material bleibt nach der Defintion ebenfalls konstant.

```
/* Textur an Material binden: */
mMaterial->getTechnique(0)->getPass(0)->
 createTextureUnitState("WebcamTexture");
 /* Eventuelle Standarttransformationen (wie z.B.
    Spiegelung) anwenden */
mMaterial->getTechnique(0)->getPass(0)->
 getTextureUnitState(0)->
  setTextureTransform( OGRE::Matrix4::IDENTITY );
/* Schatten aus da die Plane für das Videobild eher
   schräg im Raum steht und die Schatten so nicht
   passen würden. */
mMaterial->setReceiveShadows(false);
```

Listing 4.2: Anbinden an das Material. Diese Bindung bleibt konstant

```
unsigned char* textureDataPtr = (unsigned char*)
   mTexture->getBuffer()->lock(OGRE::HardwareBuffer::
   HBL_DISCARD);
```

Listing 4.3: Zugriff auf den von OGRE verwalteten Texturspeicher

```
char* videoImagePtr = static_cast<char*>(mTemporaryImage
    ->imageData);
```

Listing 4.4: Zugriff auf den von OpenCV verwalteten Bildspeicher

4.2.4 Einbindung und Verarbeitung des Videobildes

Das Videobild wird mit OpenCV eingebunden.

Herstellen der Verbindung

```
mDevicePointer = cvCreateCameraCapture(deviceId);
cvSetCaptureProperty(mDevicePointer,
 CV_CAP_PROP_FRAME_WIDTH,
 mCaptureWidth);
cvSetCaptureProperty(mDevicePointer,
 CV_CAP_PROP_FRAME_HEIGHT,
 mCaptureHeight);
```

Listing 4.5: Zugriff und Konfiguration der Kamera

Die Verbindung zur Kamera wird, wie in Listing 4.5 aufgeführt, dabei mit der Funktion `cvCreateCameraCapture` hergestellt. Sie bekommt eine deviceId zugeteilt, wobei es sich um einen Integer-Wert handelt. Dieser Wert richtet sich nach dem System und wird von Windows beginnend mit Null aufsteigend für jedes angeschlossene DirectShow-Gerät vergeben.

Die Standartausgabegröße des Videobildes beträgt 640 zu 480 Pixel. Dies muss entsprechend der eingestellten Auflösung geändert werden.
`int cvSetCaptureProperty( CvCapture* capture, int property_id , double value )` ermöglicht das im Allgemeinen das Ändern von Kameraparametern und somit auch wie in Listing 4.5 das Ändern das Aufnahmegröße.

Abgreifen des Bildes

```
mImagePointer = NULL;
cvGrabFrame( mDevicePointer );
mImagePointer = cvRetrieveFrame( mDevicePointer );
```

Listing 4.6: Zugriff auf ein einzelnes Bild

Um ein Bild von der Web-Kamera zu bekommen kann man die beiden OpenCV-Befehle `cvGrabFrame` und `cvRetrieveFrame`, wie in Listing 4.6, benutzten. Alternativ ließe sich auch `cvQueryFrame` verwenden.

Das Bild, dass von `cvRetrieveFrame` zurückgegeben wird darf nicht gelöscht oder verändert werden, da sonst ein Speicherkonflikt auftritt.

Radiale Entzerrung

Dies wird in 4.2.6 und in den Listings 4.9 und 4.10 ausführlich beschrieben.

Texturschreiben

Das Schreiben der Textur aus dem Hauptspeicher in den Texturspeicher der Grafikkarte geschieht am schnellsten mit der Funktion `memcpy`, wie in Listing 4.7.[1]

```
memcpy(textureDataPtr,videoImagePtr,mTextureHeight*
    mTextureWidth*4);
```

Listing 4.7: Kopieren des Videobildes in den Texturspeicher

4.2.5 Die Web-Kamera-Klasse

Die gesamte Verwaltung der Web-Kamera übernimmt die Klasse `WebcamInterface` . Zur einfachen Erweiterbarkeit ist sie in einen Namensraum namens `WebcamModule` eingeschlossen.

Verwaltete Daten

Sie speichert das Web-Kamera-Bild in Rohfassung, entzerrt und im RGBA-Format, vorbereitet für die Texturverwaltung OGREs. Außerdem verwaltet sie den Zeiger für den Kamerazugriff und bietet eine Methode, `reloadMaterial` , um die Textur neu zu befüllen. Die Textur und das Material, das OGRE verwendet um das Web-Kamera-Bild darzustellen werden ebenfalls von `WebcamInterface` verwaltet.

Konstruktion

Bei der Konstruktion der Klasse wird der Zugriff zur Web-Kamera hergestellt, die Textur und das Material erstellt und die nachher verwendeten Bilder initialisiert.

4.2.6 Weitere Ansätze zur Geschwindigkeitsoptimierung

Reduktion des nötigen Speicherdurchsatzes

Partielle Bild-Aktualisierung

Die Überlegung ging von der Feststellung aus, dass sich der obere Teil des Videobildes eher selten ändert, wenn auf dem Mikadoplatz nichts los ist. Daher würde es reichen diesen Bildbereich seltener als das restliche Spielfeld neu zu zeichnen.

In der aktuellen Implementierung werden die ersten 80 Zeilen des Video-Bildes nur in jedem zweiten Frame aktualisiert. Der Effizienzgewinn ist

[1] Voraussetzung ist hier, dass der Speicher wie in „Asynchrone Textkuraktualisierung“ 4.2.3 adressiert wird.

zwar nur gering, aber je nach Kamera-Position sind auch noch mehr Pixel auszuschließen. Außerdem könnte die Akutalisierung noch seltener sein.

Vorteil dieses Verfahrens ist der geringe Implementierungsaufwand. Der Geschwindigkeitsvorteil ist jedoch eher niedrig, wenn nur eine Textur verwendet wird. Ein andere Ansatz wäre, das Bild in eine Textur der Größe 1024 zu 512 und in zwei mit je 1024 zu 128 Pixeln zu unterteilen und die Texturen jeweils mit Teilbildern des Videobildes zu füllen. So könnten die 128 Pixel hohen Texturen den oberen und den unteren Rand bilden und sich gegeben falls seltener aktualisieren. Der Speicherdurchsatz um das gesamte Bild zu aktualisieren wäre so auf jeden Fall geringer, da sonst eine 1024 zu 1024 Pixel große Textur verwendet wird. Voraussetzung hier ist natürlich, dass das Spielfeld auch in die 1024 zu 512 Pixel große Textur passt. Anderenfalls wäre eine höhere Granulierung des Bildes erforderlich, so dass ein 128 Pixel hohe Textur in zwei je 64 Pixel hohe unterteilt wird. raussetzung hier ist natürlich, dass das Spielfeld auch in die 1024 zu 512 Pixel große Textur passt. Anderenfalls wäre eine höhere Granulierung des Bildes erforderlich, so dass ein 128 Pixel hohe Textur in zwei je 64 Pixel hohe unterteilt wird.

Unter dem gegeben Zeitdruck war eine Optimierung in dieser Richtung jedoch nicht möglich. Weitere Arbeiten könnten untersuchen wie groß hier der tatsächliche Geschwindigkeitsvorteil ist.

Verbesserung der Datenverarbeitung

Neben der Reduktion der Daten, die übertragen werden müssen, bietet die Verbesserung der Datenverarbeitung eine zusätzliche Möglichkeit die Bildwiederholrate des Videobildes zu steigern.

Vorberechnung der Radialen Entzerrung

Im Rahmen von Testläufen stellte sich heraus, dass die bisherige Implementierung der Radialen Entzerrung die Bildwiederholfrequenz von 15 Bildern pro Sekunde auf 10 Bilder pro Sekunde senkte. Das Problem bestand darin, dass sie für jeden Pixel und für jedes Bild neu berechnet wurde. Da die Rechnung für jeden Pixel zwei polynomiale Rechnungen beinhaltet und diese sowieso so lange wie sich die Koeffizienten dieser Funktion nicht ändern gleich bleiben,[2] brachte die Vorberechnung der Funktion einen erheblichen Geschwindigkeitsschub, der so groß war, dass die Radiale Entzerrung kaum noch ins Gewicht fiel war.

Vorteilhaft ist bei der Vorberechnung auch, dass X und Y voneinander unabhängig sind, da man so nur zwei eindimensionale Speicherblöcke benötigt und der Aufwand der Vorberechnung außerdem noch niedrig

[2] Die Koeffizienten können sich in MoleARlert zur Laufzeit nicht ändern.

ist, auch wenn dies nicht unbedingt wichtig gewesen wäre. Die Methode zur Vorberechnung ist, genau wie die Methode ohne Vorberechnung, auch schon in OpenCV fest integriert. Leider ist die Dokumentation, wie oft in OpenCV, eher dürftig.

```
cvUndistort2(
   mImagePointer, // Zeiger auf das Web-Kamera-Bild
   mUndistoredImagePointer, // Zeiger auf das Zielbild
   &intrinsic_param, // Kameraparameter (typ CvMat)
   &dist_coeffs);
```

Listing 4.8: Radiale Entzerrung vor der Weiterentwicklung

```
mUndistortionMapX =  // Erstellen des Speichers für X
 CV::cvCreateImage(
  CV::cvSize(
  mCaptureWidth,
  mCaptureHeight),
  IPL_DEPTH_32F, // 32bit Fließkomma
  1);
mUndistortionMapY = // Erstellen des Speichers für Y
 CV::cvCreateImage(
  CV::cvSize(
   mCaptureWidth,
   mCaptureHeight),
   IPL_DEPTH_32F, // 32bit Fließkomma
   1);
// Vorberechnung der Entzerrung
cvInitUndistortMap(
   &intrinsic_param, // Eingabe der intrinsischen
      Parameter
   &dist_coeffs,   // Eingabe der Größenkoeffizienten
   mUndistortionMapX,
   mUndistortionMapY
);
```

Listing 4.9: Radiale Entzerrung nach der Weiterentwicklung

```
cvRemap(
   mImagePointer,
   mUndistoredImagePointer,
   mUndistortionMapX,
   mUndistortionMapY);
```

Listing 4.10: Anwendung der Radialen Entzerrung auf das Videobild mit vorberechneten Parametern

Parallele Bildverarbeitung

Eine weitere Möglichkeit zur Steigerung der Performanz ist die Parallelisierung der Verarbeitungskette. Da vor dem Grabben des neuen Bildes möglicherweise gewartet werden muss bis das alte komplett verarbeitet ist um einen Speicherkonflikt zu vermeiden, entsteht möglicherweise für einen Prozess eine Wartezeit, obwohl dieser in der Zwischenzeit bereits ein neues Bild von der Kamera holen, dies entzerren und in das RGBA-Format konvertieren könnte.

Im Rahmen der Entwicklung wurde dieser Ansatz kurz verfolgt, aber auf Grund der wahrscheinlich nicht sehr hohen Steigerung der Perfomanz zu Gunsten anderer Entwicklungen vorerst ruhen gelassen.

Intrinsische Kameraentzerrung mit hardwareseitigen Pixelshadern

Die Grafikkarte könnte mit Fragment-Shadern die Entzerrung des Bildes hoch parallel und sehr effizient ausführen. Im Rahmen der Entwicklung wurde dieser Ansatz nicht verfolgt.

Kapitel 5

Zusätzlich entwickelte Komponenten

Zur Verbesserung der Wartbarkeit und zur Fehlerbehebung wurden verschiedene neue Komponenten im Rahmen dieser Arbeit entwickelt.

5.1 Calibration-File-Reader

Der Calibration-File-Reader liest die extrinsischen Kamerakalibrierungsdaten ein, die durch ein anderes Programm erstellt werden.

5.1.1 Anforderungen

Um das Spielfeld aus der selben Perspektive zu berechnen in der auch die Kamera den Mikadoplatz sieht, existiert ein Programm, dass durch die Bestimmung von vier Eckpunkten die extrinischen Kameraparameter berechnet. Die berechneten Daten, bestehend aus einem dreidimensionalen Translations- und einem vierdimensionalen Rotatationsvektor, müssen in den AR-Viewer übertragen werden.

5.1.2 Zustand

Vor der Entwicklung dieser Klasse geschah die Übertragung manuell über Arrays im Quelltext.

5.1.3 Funktionalität

Die Klasse CalibrationFileReader stellt einfache Methoden zur Verfügung um aus einer Textdatei die Werte auszulesen. Dabei handelt es sich um 32bit-Gleitkommawerte. Die Klasse kann außerdem die Werte auch in die Datei schreiben.

5.2 Konfigurations-Klasse

5.2.1 Zustand und Anforderungen

Die Konfiguration der Laufzeit-Instanzen erfolgte stets über Defines zur Compile-Zeit was die entsprechenden Nachteile brachte. Allerdings war auf unserer Entwicklungsebene sowieso nicht viel frei zu definieren. Die sonst interessanten Konfigurationsparameter wie Ausgabeauflösung, Farbtiefe, Antialiasing, Texturen, Materialien und so weiter übernahm ohnehin schon OGRE.

Während der Weiterentwicklung der Systembausteine, die für die Web-Kamera relevant sind, stellte sich jedoch raus, dass eine dynamische Konfiguration der Aufnahmeauflösung sehr wünschenswert wäre.

Aus Software-struktureller Sicht ist es auch sehr wünschenswert über eine globale statische Klasse auf die Konfiguration bedenkenlos zugreifen zu können ohne dabei auf eine bestimmte Einbindungsreihenfolge oder auf untereinander bestehende Abhängigkeiten achten zu müssen.

5.2.2 Konfigurationsdatei

```
<configuration>
 <Webcam.width>960</Webcam.width>
 <Webcam.height>720</Webcam.height>
 <network.port>1405</network.port>
 <log.file>myLog.log<log.file/>
 <game.molesExplodeOnXkeyDown>0</game.
    molesExplodeOnXkeyDown>
</configuration>
```

Listing 5.1: Konfigurationsdatei wie sie auch zur Anwendung kam

Um eine übersichtliche, menschenlesbare Konfigurationsdatei zu verwenden wurde auf XML zurückgegriffen. Das Format der Datei ist sehr simpel gehalten: Die Konfiguration wird durch einen `<configuration>`-Tag eingeklammert. In ihr selbst ist es dann absolut frei definierbar, welche Tags mit welchen Werten verwendet werden. Wichtig ist nur, dass im Quelltext auch die selben Bezeichner für die jeweiligen Werte verwendet werde. Listing 5.1 zeigt die Konfigurationsdatei, die im Spiel verwendet wird. Dabei wurde versucht die Semantik der Werte durch einen Bezug zum Programmteil deutlich zu machen in dem jeder Wert mit einem entsprechendem Bezeichner gefolgt von einem Punkt beginnt.

5.2.3 Implementierung

Die Konfigurationskomponente besteht nur aus einer Klasse. Diese Klasse kann eine Konfigurationsdatei einlesen und verarbeiten. Anschließend

stellt sie über zwei Methoden die eingelesenen Werte zur Verfügung. Die eine Methode gibt den Wert als String, die andere als Integer zurück.

Die Implementierung ist sehr einfach gehalten und wurde auch im Laufe der Endproduktion durchgeführt. Daher wurde hier auf Effizienz weitestgehend verzichtet. Der mögliche Spielraum wäre ohnehin wahrscheinlich sehr klein gewesen, da Zugriffe auf die Methoden der Klasse äußerst selten sind.

5.2.4 Verwendete Bibliotheken

Wie auch zum Speichern und Laden der Highscore wird die Softwarebibliothek tinyXML (siehe 2.4.5) verwendet.

5.3 MiddleServer

Der „MiddleServer“ ist eine im Rahmen dieser Arbeit komplett neu entwickelte Anwendung, die zwischen Tracking und AR-View ins System eingebaut wird.

Entwicklungsbeginn war eine Woche vor dem CV-Tag. So wurden Dokumentation und Optimierung eher sekundär bis überhaupt nicht berücksichtigt, da die eindeutig wichtigste Eigenschaft hier die Zuverlässigkeit war.

5.3.1 Anforderungen

In Praxistests des Gesamtsystems hatten sich eklatante Schwächen gezeigt, wenn mehr als ein Marker gleichzeitig benutzt wurde. Jedoch ging das Debugging über die Fehlerdiagnose in diesem Fall nicht hinaus. Wir konnten nur feststellen, dass das was das Trackig erkennt nicht das ist was das Tracking sendet.

5.3.2 Vorgehensweise

Datenübertragung

Unter dem gegebenen Zeitdruck und unter Berücksichtigung der Tatsache, dass das Tracking-Team nicht über die benötigte Zeit verfügte um den Fehler zu beheben erschien es und als die sinnvollste Vorgehensweise ein Programm zu schreiben, dass sich in die Mitte der Kommunikation zwischen Tracking und AR-View stellt. Auf der Seite des Tracking musste nun nur geändert werden, dass sämtliche erkannten Objekte direkt eine Meldung über das Netzwerk auslösen und zwar in jedem einzelnen Bild in der das Objekt erkannt wurde. Zusätzlich musste noch hinzugefügt werden, dass der Anfang und das Ende jedes Bildes durch eine jeweilige Meldung kenntlich

gemacht wird um den Bildbezug für die Objekte zu erhalten. Da die Bildwiederholfrequenz der Bilderkennung als äußerst variabel angenommen werden kann, wäre eine Trennung über Zeitstempel wahrscheinlich nicht sehr zuverlässig. Alternativ wäre noch eine Übermittlung einer eindeutigen Frame-ID möglich gewesen. Allerdings wurde das Konzept auf softwaretechnische Einfachheit hin entwickelt und der Optimierungsgedanke zu Gunsten der Zuverlässigkeit nach hinten gestellt.

Steuerungssigenale

Die Steuersignale für Bildende und -anfang wurden durch improvisierte Lösungen erreicht um Entwicklungszeit zu sparen: Für den Beginn eines Bildes wird ein Richtungspfeil übertragen dessen X- und Y-Wert jeweils „13“ beträgt. Für das Ende eines Bildes geschieht das selbe mit „14“. Diese Lösung ist selbstverständlich nicht schön und bietet Raum für weitere Entwicklungen.

5.3.3 Arbeitsweise der Komponente

Verbindungsaufbau

Die MiddleServer-Applikation stellt einen UDP-Anschluss zur Verfügung zu dem sich ein oder mehrere Bilderkennungsprogramme verbinden können. Um die Informationen weiter zu leiten baut die Anwendung eine ausgehende Verbindung zum AR-View auf. Für die interne Verwaltung werden Frame-Objekte angelegt. Diese speichern alle Objekte die jeweilig zwischen Begin- und End-Signal-Frame empfangen wurden.

Objektabgleich

Nach Ablauf jedes Frames überprüft ein Frame-Manager welche Objekte wie oft vorhanden sind. Durch den Abgleich mit einer gespeicherten Menge der bekannten Objekte, steht fest, ob das Objekt dem Spiel bekannt ist oder nicht. Unter Einbeziehung der Information ob die Häufigkeit sich entscheidend geändert hat, wird gegebenenfalls ein Erstellungs- bzw. Löschungsbefehl an das AR-Terminal gesendet. Zum Speicherabgleich besitzt der FrameManager eine Referenz-Frame die auch von Typ Frame ist. Über eine Methode, `addObjectUnique`, die eine eindeutige Identifikationsnummer für jedes eingehende Objekt zurück gibt und das übergebene Objekt gegebenenfalls speichert, wird bestimmt wie oft das gegebene Objekt schon vorhanden ist.

Es wird dabei zu keiner Zeit ein informelles Signal zum Bilderkennungsprogramm gesendet.

5.3.4 Rahmenwerk

Der MiddleServer verwendet die selben Netzwerk-Komponenten wie die Bilderkennung und die AR-Ansicht auch. Unterschiedlich jedoch ist, dass die Datenstrukturen zur Datenübertragung zusätzliche Methoden zum Identitätsvergleich haben. Zu den Richtungspfeilen ist hier anzumerken, dass auch die Richtung und nicht nur ihre Position zur Identität beiträgt. So kann es durchaus passieren, dass im Prinzip unlogische Daten den AR-Viewer erreichen. Jedoch sollte ein gutes Programm auch unsinnige Daten richtig verarbeiten können und so lange der MiddleServer richtig arbeitet, muss er sich um die Sinnhaftigkeit der Daten auch nicht kümmern.

5.3.5 Anwendung in der Praxis und Ausblick

In der Praxis funktionierte das System den Anforderungen entsprechend. Es traten jedoch gelegentlich, aber erst nach längerer Zeit, Fehler auf. Eine genaue Ursache hier zu konnte jedoch im Rahmen der Testläufe nicht ermittelt werden, da die Tests unter Laborbedingungen diese Probleme nicht reproduzieren konnten. Weitere Entwicklungen in Richtung Stabilität sollten auf jeden Fall prüfen ob stets die richtige Bildöffungs und -Schließungsinformation übertragen wird und wenn nicht, welche Komponente der Verursacher ist. Wichtig ist hier eine ausführliche Protokollierung.

Die Integration

in die Hauptanwendung wäre ebenfalls eine denkbare Variante um die Fehleranfälligkeit zu verringern, falls sich herausstellen sollte, dass die Übertragung über einen Socket an dieser Stelle zu Verlusten neigt. Ebenfalls würde die Wartbarkeit vereinfacht werden.

Kapitel 6

MiniMoleARlert - Maulwürfe auf den Schreibtisch gebracht

Während der Entwicklung von MoleARlert gab es immer wieder die Idee das System in eine kleine, mobile Variante zu erstellen.

6.1 Ziel

Ziel der Entwicklung wäre eine Variante des Spiels, die es ermöglicht das System im kleinen Rahmen zu testen und zu präsentieren. Eine Live-Demo auf Messen, Ausstellungen oder bei Einführungsveranstaltungen für den Studiengang Computervisualistik wären hier denkbar.

6.1.1 Wenig Hardware

Sehr wünschenswert wäre hierzu, dass das System mit möglichst wenig und möglichst unspektakulärer Hardware auskommt. Konkret heißt das, dass zum Beispiel nur eine einzige Web-Kamera genügt um das Erkennen der Objekte und die Darstellung des Videobildes zu realisieren. Somit bräuchte man einerseits weniger Hardware und weniger Aufwand zum Kalibrieren. Zudem ist es wesentlich wahrscheinlicher, dass man eine und nicht zwei oder mehr Web-Kameras besitzt: Die Verteilbarkeit der Software wäre somit denkbar und MoleARlert wäre aus der Beschränkung des aufwendigen Aufbaus befreit.

6.1.2 Wenig Aufbau und viel Robustheit

Ein aufwendiger Aufbau muss ebenfalls vermieden werden. Störanfälligkeit durch wechselnde Lichtverhältnisse müssen ebenfalls berücksichtigt werden.

6.2 Testen

6.2.1 Bilderkennungskomponente

Mit einem solchen System ist das Testen und Entwickeln der Bilderkennungskomponente wesentlich einfacher, da kein aufwendiger Aufbau der Hardware nötig ist. Das System könnte man in einer fest installierten Umgebung aufbauen. Diese Umgebung könnte sehr einfach ausfallen, da nur eine feste Position für die Kamera, ein abgestecktes Spielfeld und ein Computer benötigt ist.

6.2.2 Wetter und Licht

Ein weiterer Vorteil ist, dass ein Aufbau in einem geschlossenen Raum möglich ist. Somit könnte man unabhängig vom Wetter testen. Das Wetter bzw. wechselnde Lichtverhältnisse ließen sich durch das Verwendenden verschiedener Beleuchtungen beliebig simulieren.

Bisher wurden, zum Testen während der Entwicklung der Bilderkennungskomponente, aufgezeichnete Videos als Referenzmaterial verwendet. Allerdings ist die Spanne der Szenarios, die dabei erfasst wird sehr gering. So ist eine Entwicklung einer wirklich dynamischen Lösung schwierig.

6.2.3 Umsetzung

6.2.4 Versuchsaufbau

In Abbildung 6.1 befindet die Kamera am linken Bildrand. Die gelben Zettel markieren die Feldbegrenzung zusammen mit der Tischkante. Zur Beleuchtung werden zwei Halogen-Schreibtischlampen verwendet. Die zweite Halogen-Lampe ist nicht im Bild und befindet sich über der Kamera.

6.2.5 Ausgangssituation

Die Verwendung von nur einer statt zwei Web-Kameras stellte sich als sehr schwierig heraus, da der Quelltext der Bilderkennung überhaupt nicht darauf ausgelegt war jemals wieder verwendet zu werden.

Dies drückte sich darin aus, dass der Programmablauf komplett in der main-Methode statt fand. Zwar wurden dann Objekte zur Datenspeicherung verwendet, aber sinnvoll war der Einsatz dieser nicht. Der Quelltext war auch fast ohne Kommentare, die erklären würden warum nun etwas gemacht oder eben nicht gemacht wird und was da überhaupt geschieht. Die Kommentare waren eher in die Richtung elementare Operationen zu beschreiben wie beispielsweise bei Berechnung eines Minimums stand dort, dass ein Minimum berechnet wird, allerdings nicht warum dies überhaupt notwendig wäre und was man nachher damit macht.

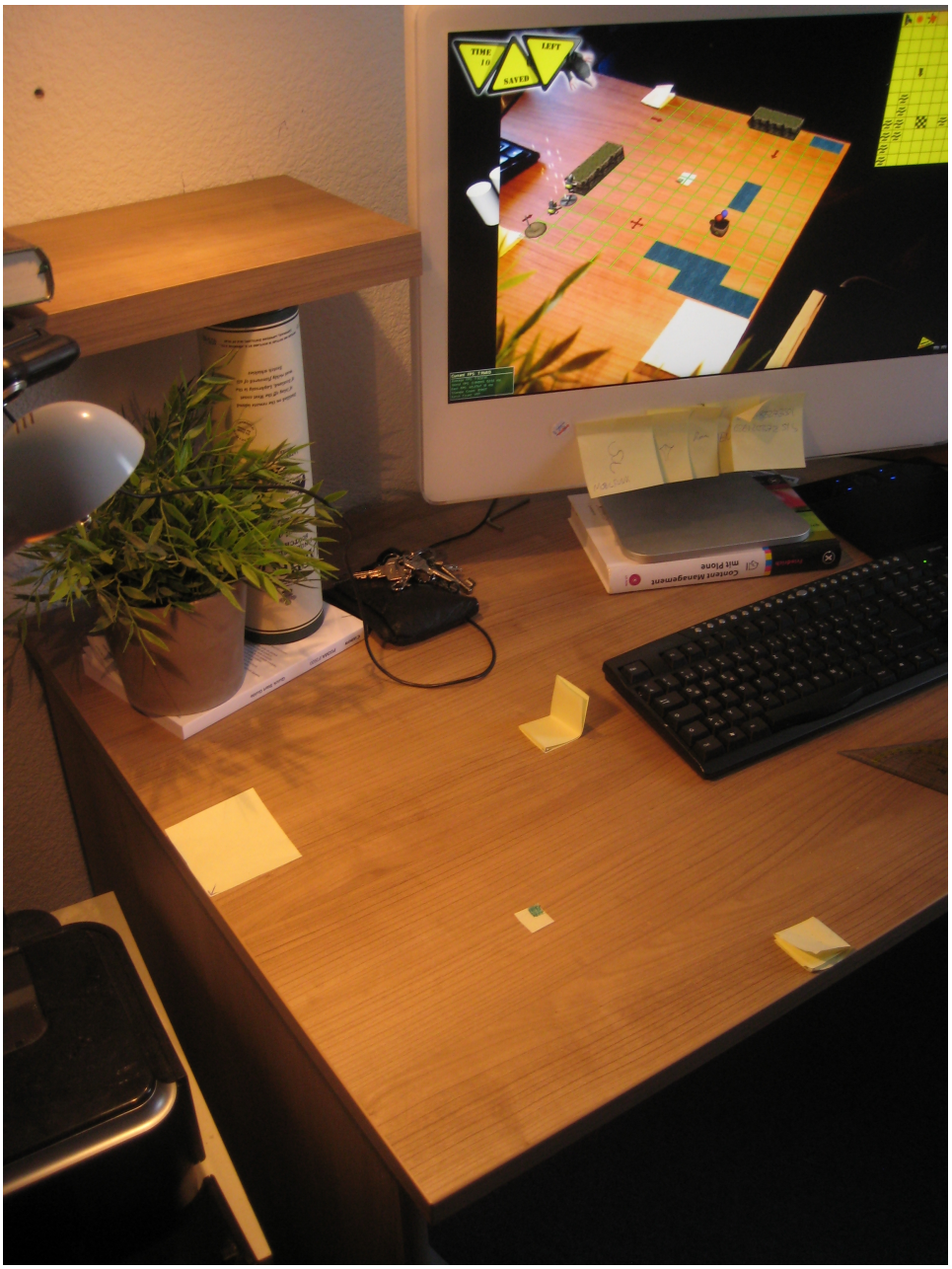


Abbildung 6.1: Aufbau der lokalen Variante auf einem Schreibtisch mit AR-Ansicht im Hintergrund.

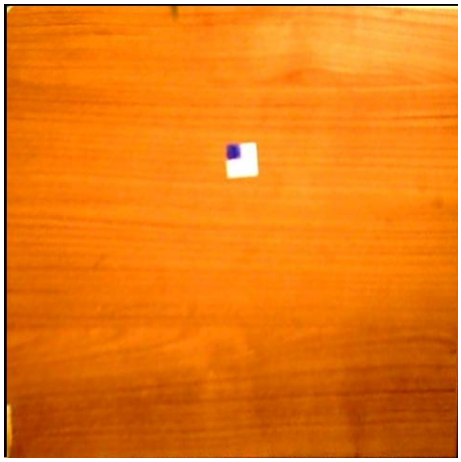


Abbildung 6.2: Bild radial und perspektivisch entzerrt; Beleuchtet mit zwei Strahler

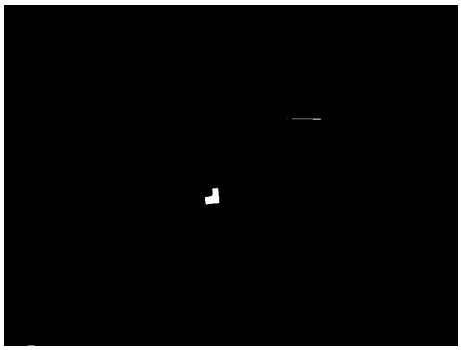


Abbildung 6.3: Bild radial und perspektivisch entzerrt, im Blau-Kanal nach hellen Regionen gesucht (weiss), anschließend dilatiert und erodiert.

Fehlende Abgeschlossenheit

Die Wiederverwertbarkeit wurde auch dadurch eingeschränkt, dass konsequent darauf verzichtet wurde irgendwelche Namensräume oder Klassenmodelle zur Kapselung des Codes zu verwenden. Schön wäre es gewesen, wenn durch eine einfache Instantiierung einer Bilderkennungs-Super-Klasse die anspruchsvollste Arbeit getan gewesen wäre.

Fehlende Konfiguration

Leider war die Konfiguration auch durch wenig aussagekräftige und unkommentierte `DEFINE`s gelöst.

6.3 Erfolgreiche Änderungen

6.3.1 Bilderkennung

Einkapselung

Die Integration der Bilderkennung in den AR-Viewer wurde durch das Einkapseln in einen eigenen Namensraum der Bilderkennung ermöglicht. Dabei kam es zu einem häufig auftreten Konflikt: Wenn OGRE und OpenCV zusammen verwendet werden entsteht ein Compile-Fehler, dass die Überladene Funktion `assert` nicht eindeutig zugeordnet werden kann. Dies liegt daran, dass es nötig ist OpenCV in einen eigenen Namensraum zu kapseln um andere Konflikte zu vermeiden, somit gibt es dann eine Methode `assert` und eine `CV::assert`. Der Compiler konnte mit dieser Art der Überladung leider nichts anfangen. In der übrigen Programmierung trat dieser Fehler nicht auf, da nie auf das Konstrukt **`using namespace`** `CV` zurückgegriffen, sondern immer `CV::` zur Qualifizierung genutzt wurde. Eine entsprechende Bearbeitung des Quelltextes erschien jedoch als enorm aufwendig. Das Problem konnte schließlich dadurch gelöst werden, dass die Standard-Datei „assert.h“ um die Compiler-Direktive **`#pragma`** `once` ergänzt wurde. Diese Datei wurde sowohl von OGRE als auch von OpenCV inkludiert, was auch nicht weiter verwunderlich ist. Da die OpenCV-Inkludierungen allerdings in einem Namensraum stattfanden, erhielten so auch die eingebundenen Funktionen aus „assert.h“ den Namensraum.

Einschließung in eine Klasse

Die Bilderkennung wurde in eine Klasse namens `TrackingInterface` eingebunden. Der vorhandene Quelltext wurde in einen Teil zur Initialisierung, einen zur Dekonstruktion und in einen sich bildweise wiederholenden Teil aufgeteilt. Außerdem wurde bei allen eingebundenen Klassen sicher gestellt, dass sie vollständig initialisiert sind, wenn man eine Instanz

ihrer erzeugt, was leider nicht durchgängig der Fall war. So waren beispielsweise die Zielkoordinaten für die perspektivische Transformation in der entsprechenden Klasse (`Field`) standardmäßig nicht initialisiert.

Trackig über Kontraste

Die bisherige Vorgehensweise, die Marker über Farben zu erkennen, gestaltete sich bei wechselnden Lichtfarben und einem holzfarbenden Tisch mit vielen Farbfacetten als schwierig und gegenüber einer schwarz-weiß-Erkennung als zu aufwendig. Das Farbenfinden wurde daher auch komplett aus dem Programm gestrichen.

Entfernung des Personentrackings

Da keine Personen auf der Fläche zu erwarten sind, wurde auch die Personenerkennung komplett entfernt. Ein Maskierung von überlagernden Objekten, insbesondere von Händen, wäre mit dem aktuellen Projekt nicht möglich gewesen, da nur nach Mittelpunkten von Personen und nicht nach überlagernden Figuren gesucht wird.

Teilen des Bildes

Um das von der Kamera eingefangene Bild sowohl für die Bilderkennung als auch für den Videohintergrund zu verwenden wurde in der WebcamInterface-Klasse eine Methode `getPicture` hinzugefügt, die eine Kopie des zuletzt eingefangenen Bildes in einen ihr übergebenen, durch einen Zeiger markierten Speicherbereich schreibt.

Sobald ein neues Bild von der Kamera bereit steht, wird die Bilderkennungsschnittstelle darüber benachrichtigt und führt den Quelltext aus, der bildweise nach obiger Aufteilung vorgesehen ist.

Signalfilterung

Die Filteralgorithmen, die im Rahmen der Erstellung des MiddleServers (siehe 5.3) eingeführt wurden, wurden rudimentär in den Listener für die Bilderkennung integriert. Die Frame-Klasse, die die Konfiguration jedes Bildes speichert konnte vollständig übernommen werden und wurde nur dahin gehend optimiert, dass Konstrukte die nichts mit Richtungsmarkern zu tun hatten, entfernt wurden. Die gleiche Reduktion fand auch mit den aus dem Frame-Manager stammenden Methoden und Datenstrukturen statt.

Der Grad der Signalfilterung ist passiv einstellbar: Es ist konfigurierbar wie groß der Puffer für die empfangen Daten, also das „Gedächtnis“ des Programms, ist. Weiterhin ist einstellbar, wie oft in diesem Puffer ein Objekt

vorkommen muss, damit es als erkannt gilt. Schließlich ist noch einstellbar aber welcher Anzahl von Vorkommen ein Objekt als nicht mehr existent gilt, wenn es schon existiert. Dieser Wert muss kleiner sein als der erste, da sonst ständig gelöscht und erstellt werden würde. Jedoch ist es so möglich, dass nach einer kürzeren Dauer der Marker wieder verschwindet als er erstellt wurde.

6.4 Nichterreichtes

Die Markererkennung funktionierte leider nicht so zuverlässig wie erhofft. Die bisherige Erkennung über Farben war in der gegeben Beleuchtungssituation wenig optimal und auch augenscheinlich relativ rechenintensiv. Hier wurde eine Markererkennung über Umrisse verwendet, die noch der ein oder anderen Optimierung bedarf. Insgesamt scheinen Lösungen, die über Farb- und Helligkeitskontraste operieren insgesamt sinnvoller zu sein, als ausschließlich Farbwerte zu analysieren. Dies ist auch der Beleuchtungssituation geschuldet, die bei einer Spotlight-Beleuchtung nicht gleichmäßig ist. Die unterschiedliche Helligkeit ist bei der Verwendung von nur einer Lichtquelle erstaunlich stark im Kamerabild erkennbar.

Kapitel 7

Fazit und Ausblick

7.1 Fazit

Am Ende der Entwicklung steht ein AR-Spiel, das den Praxistest beim diesjährigen CV-Tag gemeistert hat. Damit ist das Ziel der Entwicklung erreicht worden.

Dennoch ist ein System natürlich nie vollständig ausgereift und lässt immer noch Raum für neue Ideen.

7.2 Ausblick

Auf der ISMAR[1] 2009, einer Ausstellung und Symposium für Gemischte und Angereicherte Realität wird dieses Projekt mit einem Plakat präsentiert werden. Die Veranstaltung findet vom 19. bis zum 22. Oktober in Orlando, Florida in den USA statt.

Ein weiterer Aufbau des Spiels oder eine Weiterentwicklung ist im Moment eher unwahrscheinlich.

Die Erfahrungen und Technologien jedoch, die bei der Entwicklung des Spiels gesammelt wurden, können nützlicher Nährboden für viele weitere Projekte sein und liefern hoffentlich die Grundlage und die Inspiration für eine Menge wissenschaftlicher Erfolge.

7.3 CV-Tag

Der CV-Tag ist eine jährliche Veranstaltung des Instituts für Computervisualistik um der Öffentlichkeit, Firmen Studieninteressierten, Studierenden, Ehemaligen und allen weiteren Interessierten einen Einblick in die Möglichkeiten der Computervisualistik in Koblenz zu geben.

[1] International Symposium on Mixed and Augmented Reality

Neben vielen anderen Projekten war MoleARlert auch vertreten und konnte von den Besuchern selbst ausprobiert werden. Die größtenteils positive Resonanz führte dazu, dass das Projekt in der Publikumswertung den ersten Preis gewann (siehe Abbildung 7.1).

Abbildung 7.1: Der Publikumspreis vom CV-Tag 2009 mit Urkunde.

Literaturverzeichnis

[arq] http://wearables.unisa.edu.au/arquake/, 20. September 2009

[roc] http://www.rockstarnorth.com/, 20. September 2009

[ocv] http://opencv.willowgarage.com/wiki/, 20. September 2009

[jen] http://www.jenkinssoftware.com/, 20. September 2009

[fmd] http://www.fmod.org/, 20. September 2009

[xml] http://www.grinninglizard.com/tinyxml/, 20. September 2009

[ogr] http://www.ogre3d.org/, 20. September 2009

[ENG09] Engelhardt, S., Langs, A., Lochmann, G., Schmidt, I., Müller, S.: MoleARlert - An Augmented Reality Game Based On Lemmings. Universität Koblenz-Landau, 2009