Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Sandro Wolf

Evaluierung und Entwicklung von Spielkonzepten auf dem Microsoft-Surface

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Sandro Wolf

Evaluierung und Entwicklung von Spielkonzepten auf dem Microsoft-Surface

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Olaf Zukunft
Zweitgutachter : Prof. Dr. Stefan Sarstedt

Abgegeben am 4. Dezember 2009

Sandro Wolf

Thema der Bachelorarbeit

Evaluierung und Entwicklung von Spielkonzepten auf dem Microsoft-Surface

Stichworte

Microsoft Surface, Spielkonzepte, Spielentwicklung, NUI, Interaktion

Kurzzusammenfassung

Im Rahmen dieser Arbeit wurden die technischen und konzeptionellen Grundlagen des Microsoft Surface analysiert sowie dessen Eignung für Spielanwendungen überprüft. Weiterhin wurden die besonderen Anforderungen an die Konzeption und Entwicklung von Spielen untersucht und exemplarisch ein Spielkonzept entwickelt und prototypisch implementiert.

Ende des Textes

Sandro Wolf

Title of the paper

Evaluation and development of game concepts for the Microsoft Surface

Keywords

Microsoft Surface, game concepts, game development, NUI, interaction

Abstract

In this paper, the technical and conceptual specification of the Microsoft Surface was analysed and reviewed for it's suitability for gaming. Furthermore the specific requirements regarding conception and development of games were examined. Finally a game concept was designed and realized prototypically.

End of text

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

CLI	Command-Line-Interface; text-basierte Benutzerschnittstelle
CPU	Central-Processing-Unit; (Haupt-)Prozessor eines Computers
DI	Direct Illumination; Methode in der Sensorik für Touch-Technologie
GC	Garbage-Collector, Mechanismus bei OO zur beseitigung ungenutzer Objektinstanzen
GPU	Graphical-Processing-Unit; Grafikprozessor der Grafikkarte
GUI	Grafische Benutzerschnittstelle, oft Fenster-basierte Systeme
MS	Microsoft
NUI	Natural User Interface, s. Kapitel Grundlagen
OO	Objekt-Orientierung, objektorientiert
RAM	Random-Access Memory, i.A. Arbeitsspeicher des Rechners
SDK	Software-Development-Kit
UCD	„User-Centered-Design“
UI	„User-Interface“, Benutzerschnittstelle
UX	„User-Interaction“, Benutzerinteraktion
XNA	„XNA is Not an Acronym“, Framework zur Spielentwicklung von Microsoft

1 Einleitung

Die folgenden Abschnitte erklären Ziele, Anlass und Inhalt dieser Arbeit sowie das Vorgehen zur Bearbeitung der gegebenen Thematik.

1.1 Motivation und Ziele der Arbeit

Insbesondere seit Ende der neunziger Jahre haben sich vermehrt Projekte im Bereich Surface-Computing/TableTops entwickelt, die mit ihren Ansätzen und Technologien bisweilen neue Lösungsmöglichkeiten und Produkte hervorbrachten wie z. B. das Microsoft Surface (im Weiteren: „Surface“). Dieses ist auch aufgrund seines Potenzials, in absehbarer Zeit für den Consumer-Markt verfügbar zu sein, ein interessanter und darüber hinaus bereits zugänglicher[1] Untersuchungsgegenstand.

Das Surface bietet durch die Plattform-spezifische Nutzerschnittstelle neue Möglichkeiten der Interaktion, die insbesondere auch für die Spielentwicklung genutzt werden können. Dabei ist das Surface schon als Tisch von seiner Form und Funktion her, aber auch grundlegend konzeptionell ausgelegt für den Mehrbenutzer-Betrieb, was weitere interessante Spielkonzepte zulässt. Außerdem gelten für die Spielentwicklung ähnliche Anforderungen und Ziele, wie sie an Surface-spezifische Anwendungen gestellt werden (analoge Attribute sind z. B. „enjoying“, ästhetisch, intuitiv, immersiv).

Primäres Ziel der Arbeit ist, die Möglichkeiten und Grenzen der speziellen Surface-Plattform sowie ihre spezifischen Eigenschaften insbesondere im Hinblick auf die Spielentwicklung und -konzeption herauszuarbeiten. Eine Recherche über bereits verfügbare Spiele für das Surface oder Surface-ähnliche Systeme soll den aktuellen Stand von Markt und Forschung wiedergeben. Je nach Resultat könnten u. a. Vergleiche zum Surface gezogen und dadurch dessen besondere Eignung für gewisse Spielarten oder -konzepte ermittelt werden.

Sekundäres Ziel ist ferner, herauszuarbeiten, was bei der Entwicklung von Spielen auf der speziellen Surface-Plattform Besonderes zu beachten ist. Dabei sollen die verschiedenen Aspekte der (programmierenden/technischen) Spielentwicklung für das Surface beschrieben und die Aufgaben und Möglichkeiten des Spiele-Entwicklers herausgestellt werden, die bei der Implementation eines Spiels bestehen.

Da die Spielentwicklung von der Idee über Konzept und Design, der Produktion von zu testenden Prototypen bis hin zum marktreifen Produkt bekanntermaßen eine sehr aufwändige Unternehmung ist, kann im Rahmen dieser Arbeit nicht auf alle Aspekte diesbezüglich eingegangen werden. Weiterhin erfordert es gerade für die umfangreichen kreativen Tätigkeiten vielerlei Talente aus verschiedensten Disziplinen, die in ihrer Gesamtheit kaum von einer einzelnen Person mitgebracht werden.

Abschließend soll jedoch auf Basis der gewonnenen Erfahrungen und unter Nutzung der vorhandenen Ressourcen die Erstellung und Beschreibung verschiedener Surface-spezifischer Spiele-Konzepte bzw. -Szenarien sowie die prototypische Implementation eines ausgewählten Spielbeispiels erfolgen.

1 An HAW Surface-Workstation und Entwicklungsumgebung vorhanden

1.2 Aufbau der Arbeit

Zentraler Untersuchungsgegenstand dieser Arbeit ist das Microsoft Surface, daher soll eingangs in Kapitel 2 die Zielplattform, beschrieben werden, deren Benutzerschnittstelle in die Kategorie „Natural User Interface“ einzuordnen ist. Um die dabei für die Spielentwicklung relevanten technischen Möglichkeiten und Grenzen des Systems bereits im Vorfeld abstecken zu können, sollen sowohl die Surface Hardware als auch seine Software-Umgebung untersucht werden (Abschnitte 2.2.1 und 2.2.2). In Abschnitt 2.2.3 wird weiter auf die Interaktionsfähigkeiten und -möglichkeiten eingegangen, die das Surface auszeichnen. Hier werden auch die Prinzipien, Regeln und Tipps von Microsoft erläutert, welche bei der Gestaltung und Umsetzung von Anwendungen für das Surface zu beachten sind. Und da auch stets der „Blick über den Tellerrand“, bzw. in diesem Fall eine ausgiebigen Literatur- und Marktrecherche, hilft zu erkennen, werden in Abschnitt 2.2.4 auch dem Surface ähnliche Systeme vorgestellt und verglichen.

Neben der Plattform selbst ist für die Evaluierung und Entwicklung von Spielkonzepten auf dem Surface der Begriff des Spiels bzw. Spielkonzepts von zentraler Bedeutung. In Abschnitt 2.3 werden im Wesentlichen seine Merkmale, und der Entwicklungsprozess dargestellt und in Abschnitt 2.4 für das Surface bereits existierende Anwendungen bzw. Spiele aus spielkonzeptioneller Sicht vorgestellt.

In Kapitel 3 folgt eine erste Zusammenfassung dieses Grundlagenteils. Hierauf aufbauend werden in Kapitel 4 eigene Spielgrobkonzepte für das Surface beispielhaft vorgestellt, die eigentliche Umsetzung und Implementation erfolgt anhand des ausgewählten Konzept-Beispiels „Curling“.

In Kapitel 5 werden die grundlegenden fachlichen und konzeptionellen Aspekte der prototypisch umzusetzenden Spielsoftware dargelegt und in Kapitel 6 die technische Realisierung der wesentlichen zuvor beschriebenen Aspekte.

In Kapitel 7 werden abschließend ein Gesamtfazit über die gesammelten Erkenntnisse sowie ein Ausblick gegeben.

2 Grundlagen

In den folgenden Abschnitten wird die grundlegende Terminologie beschrieben, die zur Bearbeitung der Thematik relevant ist. Weiterhin ist dieser Teil der Arbeit das Resultat aus den Literaturrecherchen und so Argumentationsgrundlage für das weitere Vorgehen im praxisbezogenen Teil der Arbeit.

Im Folgenden wird zunächst die Zielplattform beschrieben, deren Benutzerschnittstelle in die Kategorie „Natural User Interface“ einzuordnen ist [ADLR:SPTP].

2.1 Natural User Interface (NUI)

„Natural User Interface“ (NUI) beschreibt einen andauernden Paradigmenwechsel in der Mensch-Maschine-Interaktion, bei dem verstärkt auf traditionelle menschliche Fertigkeiten wie

- Berührung (Touch)
- Bewegung und Gesten (Motion)
- Sicht (Vision)
- Sprache/Akustik (Speech)
- (Hand-)Schrift

fokussiert wird [NUIGroup].

Gegenüber den klassischen grafischen Benutzerschnittstellen (GUI) und der systemischen Bedienung mit Tastatur und Maus soll ein NUI dem Anwender „organischer“ (natürlich/intuitiv) erscheinen, um den Umgang mit Maschinen und Computern zu vereinfachen. Die Bearbeitung einer Aufgabe oder Problemstellung soll durch direkte, einfache und natürliche Bedienung des Systems in den Vordergrund gerückt werden. Kreativität, Produktivität, Spaß und Freude sollen auf diese Weise erhöht werden und dem Benutzer eine positive Erfahrung vermitteln.

Dabei dient die Kommunikation von Mensch zu Mensch als Vorbild. Immer leistungsfähigere Computer und KI ermöglichen die Weiterentwicklung der Mensch-Maschine-Kommunikation und geben ihr eine neue Qualität.

Technologie, d. h. die Realisierung von Systemen, sollte dabei möglichst „transparent“ sein und *unsichtbar* aus dem Hintergrund heraus wirken. Komplexität soll verborgen werden, um im ubiquitären Computerzeitalter nicht nur die Benutzung von vielen verschiedenartigen Systemen zu ermöglichen, sondern dies auch einfach zu gestalten und die notwendige Benutzerakzeptanz zu erzielen.

Abbildung 1 zeigt die typischen Eigenschaften der Bedienung von NUIs [ADLR:SPTP], die sich darin auszeichnet,

- **dass die Bedienung aufgefordert/ungehemmt erfolgt**: Eine durchdachte Gestaltung und ein ästhetisches Design der Systeme bewirken eine niedrige Hemmschwelle, bzw. erhöhen ihre Akzeptanz bei der Benutzung. Dies betrifft die Entwicklung sowohl von Hardware als auch Software.
- **dass die Bedienung intuitiv/unvermittelt/direkt erfolgt**: Hierfür sorgt die Ausrichtung des Systems nahe an unserer Wirklichkeit; Abbildungen sollten realitätsbezogen und die

Benutzung gewohnt/“aus dem Alltag gelernt“ sein. Der Benutzer „weiß“, was er wie tun kann und muss es möglichst nicht (lange) erlernen.

- **dass die Interaktion schnell erfolgt**: Wenig Eingabe erreicht komplexe Ziele; insgesamt konzentrierte Ausgabe wesentlicher Informationen und Optionen
- **dass die Bedienung kontextbezogen erfolgt**: Bedienung und Feedback des Systems sind abhängig vom Zustand der Umgebung, des Benutzers, der Anwendung und deren aktueller Kontext, zu einem gegebenen Zeitpunkt

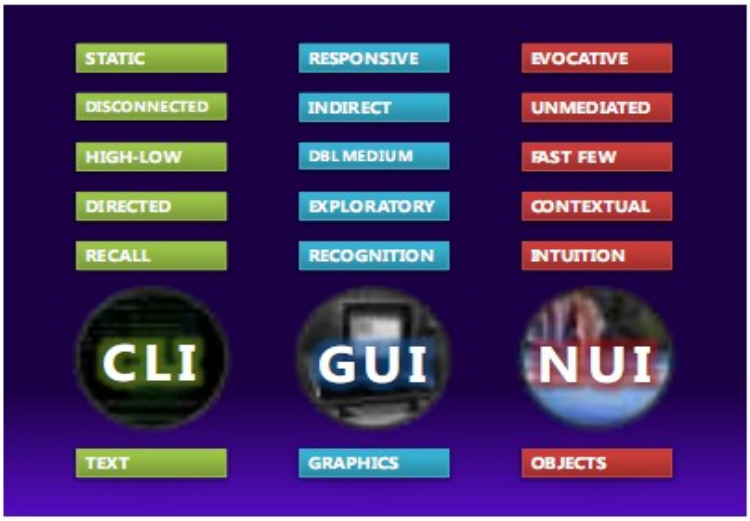

Abbildung 1: Von der CLI zur NUI [ADLR:SPTP]

Um diese Eigenschaften zu erreichen, wird versucht, unsere Gewohnheiten aus der physikalischen/realen Welt besser in der virtuellen/digitalen Welt abzubilden [Saffer2008]. Je nach Anwendung interagiert man durch den Körper oder mittels anderer physikalischer Objekte mit dem System. Dies kann durch Berührung oder Gesten, also durch Finger-, Hand- und Armbewegungen geschehen, aber auch durch Sprache, Mimik oder visuelles Fokussieren. Unabhängig von der Anwendung ist es stets das Ziel, einfache, dem Menschen nahe und bekannte, gut benutzbare und bequeme Schnittstellen und hohe Usability zu bieten.

Abhängig vom Einsatz und der Funktionsweise eines Systems muss es mit diversen Sensoren ausgestattet sein, die Eingaben aus der Umwelt aufnehmen und zur Auswertung weitergeben. Speziell für die Gesten-basierte Steuerung kommen Sensoren zum Einsatz für die Messung von:

- **Druck**: Oftmals mechanisch gemessen, z. B. durch Prüfung ob etwas gedrückt wird oder darauf steht
- **Licht**: Erkennung von Lichtquellen i. Allg. durch Photo-Dioden, Kameras
- **Nähe/Lage** (Proximity): Existenz/Anwesenheit eines Objekts im Raum erfasst durch diverse Sensorik wie Infrarot oder Sonar (akustisch)
- **Akustik**: Erfassung von Geräuschen, Tönen, Sprache, Musik durch Mikrofone
- **Bewegung**: Erfassung von Positionsveränderungen und Geschwindigkeit durch Lage- und Beschleunigungssensoren, Radar
- **Position und Ausrichtung, Neigung** („Orientation“, „Tilt“): Erfassung durch GPS, Funk,

Kameras

Praktische Beispiele für NUI-orientierte Anwendungen heute sind beispielsweise [Saffer:2008, NUIGroup:2009]:

- „**Apple iPhone**“ [APPLE:IPHONE]: Apples Smartphone ausgestattet mit sensitivem Touch-Screen und Kamera, mit Unterstützung für Multi-Touch und Gesten, Sprachsteuerung, Beschleunigungs- und Lagesensoren, Funk, GPS und anderen Ortungssystemen sowie mit digitalem Kompass.
- „**Nintendo Wii**“ [NINTENDO:WII]: Nintendos aktuelle Spielkonsole ausgestattet mit diversen „anwendungsspezifischen“ Controllern u. a. mit Funk, Infrarot, dediziertem Audiokanal, Vibration und Sensoren für Lage, Beschleunigung und Druck.
- „**Microsoft Natal**“ [MS:NATAL:2009]: „Controller-freie“ Steuerung durch Körperbewegungen, Gesten, Sprache und Gesichtserkennung für Microsofts X-Box Spielkonsole. Es existieren erste Demoversionen der Hardware, deren Sensorik nur aus Kameras und Mikrofon besteht und freies Bewegen im Raum ermöglicht.

Eine gesonderte Stellung von NUIs nehmen die so genannten „Surface Computing“-Systeme ein [Saffer:2008]. Sie zeichnen sich im Wesentlichen dadurch aus, dass der Interaktionsbereich für den Anwender gleichzeitig die Ausgabefläche darstellt. Prinzipiell existiert hier eine Unterscheidung zwischen vertikalen und horizontalen Interaktionsflächen; die Übergänge hier sind jedoch fließend, gerade im Hinblick auf den zunehmend ubiquitären Einsatzes solcher Flächen in bzw. auf verschiedensten Gegenständen und (mobilen) Geräten, wie auf Böden und Decken, Teppichen und Bildern, im Haushalt auf Kühlschränken oder sogar auf Kleidung und Verpackungen [ADLR:SPTP].

Primär vertikale interaktive Oberflächen finden bisher zumeist auf „digitalen“ Wänden Anwendung. Ein Produkt von Microsoft, welches sich derzeit noch in der Entwicklung befindet, ist die TouchWall [MS:TOUCHWALL]. Kommerziell verfügbare Systeme in diesem Bereich existieren bisher vornehmlich in Form exklusiver, speziell auf die Anwendung und den Kunden angepasster Lösungen, wie die Produkte „Multi-Touch Wall“ und „Multi-Touch Workstation“ von Jeff Han [PerceptivePixel] oder „iBar“ und „iTable“ der Firma Mindstorm [Mindstorm] mit interaktiven Flächen für Wände und Theken.

Abbildung 2: iBar von Mindstorm Interactive Surface Solutions [Mindstorm:iBar]

Ein horizontales Surface-Computing-System ist das Microsoft Surface. Für Geräte dieser Art findet man auch immer wieder den Begriff „TableTop“-Systeme vor.

2.2 Microsoft Surface

Als „revolutionary surface computing platform“ beschreibt Microsoft die Entwicklung des interaktiven Tisches mit grafischer Ausgabe durch Projektion vom Tisch-Inneren auf eine Acrylglasscheibe, der Tisch-Oberfläche [MS:SDS]. Diese Oberfläche dient zugleich der Abnahme von Kontakt-Informationen (Touch) durch eine eingebaute Sensorik. Die Bedienung erfolgt also vornehmlich direkt mit Händen und Fingern. Dieser Aufbau bringt die Funktionalität eines Touch-Screens auf die Tischoberfläche, ohne weiterer externer Hilfsmittel zu bedürfen oder die grundsätzliche Form und Funktion eines Tisches zu beeinträchtigen. Die Sensorik erlaubt mehrere gleichzeitige Berührungen, was die Plattform „Multi-Touch“-fähig macht und die eigentliche Kernfunktion darstellt: digitale Inhalte direkt greifbar und durch Gesten manipulierbar zu machen. Darüber hinaus erfasst das System so genannte zweidimensionale „Tags“, mit denen reale Objekte ausgestattet werden können, um ihre Anwesenheit auf der Oberfläche und Positionierung/Lage vom System bestimmbar zu machen. Durch Multi-Touch kann das Surface grundsätzlich auch von mehreren Benutzern gleichzeitig bedient werden (Multi-User).

In Abwesenheit von Maus und Tastatur soll das Surface einen neuen Zugang zu digitalen Inhalten ermöglichen [MS:SUEG]. Anders als ein herkömmlicher PC, der hauptsächlich als „Single-User-System“ ausgelegt ist, soll das Surface das Miteinander und die Kommunikation durch Multi-User-Nutzung fördern. Wichtiger Einsatzschwerpunkt für das Medium ist daher die „Kollaboration“, also die Zusammenarbeit von Personen am Tisch, um gemeinsam ein Problem zu lösen, zu diskutieren oder anderweitig kreativ, bzw. produktiv zu werden.

Dabei sind neue Bedienungskonzepte gefordert, die die natürliche „User-Experience“ vermitteln [MS:SUEG]. Die Handhabung des Geräts und seiner Anwendungen soll nach Microsoft dem Benutzer, ganz im Sinne des NUI, natürlich, einfach, direkt und intuitiv erscheinen und zudem Spaß machen. Die systemimmanenten Interaktionsmöglichkeiten, wie Multi-Touch- und Multi-User-Funktion sowie eine „360°-Ansicht“, sollen dies entsprechend erfahrbar machen. Die Nutzung dieser Komponenten u. a. manifestiert Microsoft in Form von Regeln und Richtlinien für die Anwendungsentwicklung in ihren speziell für das Surface formulierten „User Experience and Interaction Guidelines“, auf die später detaillierter eingegangen wird (s. Abschnitt 2.2.3.2). Diese Richtlinien sind insofern maßgebend, als dass sie von Microsoft als zu erfüllende Qualitätsmerkmale in der Entwicklung von Anwendungen für das Surface aufgestellt werden, und deren Einhaltung man mit dem Erwerb der Entwickler-Lizenz zu gewährleisten hat.

2001 startete Microsoft die Entwicklung des Surface [MS:SH2007], das in seiner aktuellen Version seit Mai 2007 in kleiner Auflage für ausgewählte Microsoft-Partner in den USA und in Kanada verfügbar ist. In der Öffentlichkeit anzutreffen ist es dort bereits in einigen Kasinos, Hotellobbys, Restaurants, Bars und AT&T Läden mit speziell für Einsatz-Zweck, bzw. -Ort und Umgebung entwickelten Anwendungen in den Bereichen Entertainment, Infotainment und eCommerce.

Seit März 2009 stehen erstmals in einigen europäischen Ländern kommerzielle Exemplare für ausgewählte Partner-Firmen bereit [MS:Cebitnews:2009]. Bestreben des exklusiven Partner-Programms ist es, für die noch relativ teure Plattform hochwertige und speziell abgestimmte Softwarelösungen für einen breiteren Markt zu generieren, indem Entwickler und Anbieter von Softwarelösungen früh in die Produktentwicklung einbezogen werden. Microsoft selbst sieht die Technologie bis 2010/11 als reif und günstig genug, um für den Consumer-Markt verfügbar zu sein.

Für zukünftige Versionen des Surface plant Microsoft bereits eine stark verbesserte Hardware-

Ausstattung bezüglich Prozessor, RAM und GPU [TechEd:2009]. Weiterhin ist die Nutzung neuer Sensorik geplant, die auch Hand-Gesten oberhalb der Touch-Oberfläche erkennen soll. Eine zusätzliche Projektionsebene soll die Möglichkeiten im Bereich Augmented-Reality ausbauen.

Abbildung 3: Microsoft Surface in der aktuellen Fassung [MS:SURFACE]

2.2.1 Surface Hardware

Da in der Regel für eine spezifische und damit einheitliche (Hardware-)Zielplattform entwickelt wird, sind deren konkreten technischen Spezifikation relevant, um die technischen Möglichkeiten und Grenzen des Systems zu kennen.

Gerade für die Spielentwicklung ist die Zielplattform nicht nur aus Vermarktungsgründen (Verbreitung, Zielgruppe, etc.) interessant [Adams:2006]. Vielmehr lassen sich mit einem einheitlichen Zielsystem Anwendungen einfacher testen, hinsichtlich der Performance und User-Experience besser abstimmen und die besonderen Funktionen nutzen.

Die Eigenschaften der Surface-Hardware lassen sich wie folgt beschreiben [MS:SDS]:

- **Physikalische Dimensionen:** LxBxH, 108 x 69 x 54 cm, Gewicht ca. 90 kg. Ein massiver Tisch mit den groben Abmessungen eines Couchtisches.
- Ein eingebauter 30“ XGA DLP® **Projektor** dient der Darstellung des Bildes. Die maximale Auflösung beträgt 1024 x 768 Pixel, entspricht also einer 4:3 Auflösung auf einer Bilddiagonalen von ca. 76cm.

  Kritisch angemerkt wird, dass die interaktive Fläche zu klein und niedrig auflösend sei für kollaborative Anwendungen [MS:TechReport:2009] vs. [Ryall:2004].
  Der Projektor liefert ein für normal beleuchtete Räume angenehm helles Licht, gute Kontraste und leuchtende Farben. Neben Helligkeit und Kontrast ist der Außeneinsatz des Tisches jedoch problematisch, insofern Infrarot-Anteile im Umgebungslicht die Berührungssensorik stören können.

- **CPU:** 2.13-GHz Intel® CoreTM 2 Duo Prozessor: Ein auf Intel x86 basierender Prozessor wie er bereits 2007 in handelsüblichen PCs verbaut wurde mit zwei Rechenkernen (Dual-Core). Dies ermöglicht die Verarbeitung von 2 Prozessen bzw. Threads parallel.

- **Arbeitsspeicher:** 2 GB RAM DDR. 1GB ist die von Microsoft empfohlene Größe des Arbeitsspeichers für den Betrieb von Windows Vista Business Edition [MS:VistaSR]. Die Belegung des Arbeitsspeichers ist abhängig von allen laufenden Prozessen und ihrem jeweiligen Datenaufkommen. Da in der Praxis zunächst keine Anwendung auf dem Surface geschlossen werden, sondern bloß in den Hintergrund treten, kann der verfügbare Arbeitsspeicher schnell knapp werden.

  Generell sollte die Nutzung virtuellen Speichers (Swapping) vermieden werden. Daher ist dafür Sorge zu tragen, dass zu jeder Zeit ausreichend physikalischer Speicher von der Anwendung allozierbar ist.

- **Grafikkarte** ATI X1650 [ATI1650]: Eine Grafikkarte der Radeon X1-Serie, die bereits seit 2006 auf dem Markt ist und zu der Zeit für den Mainstream-Einsatz ausgelegt war. Der Grafikchip (GPU) ist ein ATI RV535 in 80nm Fertigungstechnik, welcher 157 Millionen Transistoren fasst und mit 500MHz getaktet ist. Er soll eine theoretische Füllrate von zwei Giga-Texel pro Sekunde (GT/s) erreichen.

  Die Karte besitzt 256 MB Speicher mit 128-Bit Speicherbus-Anbindung und ist mit 400MHz getaktet, womit eine theoretische Bandbreite von 12GB/s zwischen Grafikkartenspeicher und GPU erreicht werden soll.

  Zum Vergleich: Die GPU einer aktuellen ATI-Mainstream-Karte aus der Radeon-4000er Serie wird in 55nm gefertigt und fasst ca. 500 Millionen Transistoren [ATI4670]. Die theoretische Füllrate liegt bei 24GT/s (gegenüber zwei!) und die Datenübertragungsrate zum Speicher beträgt 32GB/s (gegenüber 12). Performance- oder Highend-Grafikkarten, wie sie von anspruchsvollen Spielern oder professionellen Grafikanwendungen gefordert werden, verdoppeln bzw. vervierfachen die theoretischen Leistungsdaten der Mainstream-Reihe nochmals [ATI4870].

  Besonders interessant für die Grafikprogrammierung im Bereich Spiele- und multimedialer Anwendungen ist die Hardware-Unterstützung für visuelle Effekte durch Shader-Programmierung. Die Grafikkarte des Surface unterstützt dabei Microsoft® DirectX® 9.0 und Shader Modell 3.0 mit fünf programmierbaren Vertex- und 12 Pixel-Shadern.

  Aktuelle Grafikkarten (seit der Radeon 2000er Serie, eine Generation nach der X1) weisen bzgl. der Shader-Strukturen einen gänzlich neuen Aufbau auf, die so genannte Unified-Shader-Architektur [ATI2000]. Im Wesentlichen werden die Shader-Einheiten nicht mehr nach Aufgaben unterschieden, sondern es steht ein Pool/Cluster von einheitlichen parallelen Recheneinheiten zur Verfügung, auf die die Rechenlasten nach Bedarf dynamisch verteilt werden [Rubin:2008]. ATI bzw. AMD nennt dies „Stream-Computing“ [AMD:Stream]. Das Surface bietet mit der aktuellen Grafikkartenausstattung diese Funktionen noch nicht, was sich aber mit zukünftigen Versionen sicher ändern wird.

  Zumindest werden die Funktionen des Shader-Modell 3.0 durch die Hardware unterstützt und können somit gut verwendet werden. Der Einsatz von großen 3D-Modellen mit vielen Elementen kann sehr rechenintensiv werden und unter Einsatz von verschiedenen Effekten schwer berechenbaren Einfluss auf die Performance des Gesamtsystems haben. In der Praxis sind die gängigen Methoden Benchmarking, Profiling und Optimierungen

durch spezielle Debugger [MS:PIX], um die nötige Performance zu erreichen oder zu verbessern.

- **Berührungssensorik**: Fünf Kameras bilden das Herzstück der Berührungssensorik für die Tisch-Oberfläche. Die Funktionsweise ist optisch durch eine Technik, die sich „Direct-Illumination“ nennt, realisiert und vollkommen im Tisch integriert. Das System erkennt und verarbeitet bis zu 52 gleichzeitige Kontakte, bzw. Objekte. Dabei muss auch eine tatsächliche Berührung mit der Oberfläche stattfinden, d.h. über der Oberfläche schwebende Objekte werden nicht wahrgenommen.

  Es existiert durch den Aufbau keine technische Möglichkeit, Berührungen bestimmten Benutzern zuzuordnen, was eine wichtige Einschränkung bei der Konzeption und Entwicklung von Mehrbenutzer-Anwendungen darstellt.

  Die Abtastrate der Sensor-Hardware beträgt 60Hz, d.h. es steht jede 16,7ms ein (Infrarot-)Bild der Kontaktflächen zur Analyse der Bewegungen zur Verfügung. Die Analyse, Auswertung und das Tracking von Kontakten übernimmt ein Software-Prozess, der auf dem Surface ständig im Hintergrund arbeitet. Dieser muss sozusagen den letzten erkannten Zustand mit dem neuen Bild vergleichen und z. B. neue Kontakte und schnelle Bewegungen unterscheiden, was bei vielen gleichzeitigen Berührungen umso aufwändiger ist. Diese Zustände gibt er per Inter-Prozess-Kommunikation an die aktive Surface-Anwendung weiter. Das bedeutet auch, dass hierzu ständig ein Teil der CPU-Ressourcen verbraucht werden, praktische Tests zeigten eine bis zu 25%-ige Auslastung eines CPU-Kerns, je nach Anzahl der Kontakte).

  Der Aufbau birgt natürlich gewisse Latenzzeiten von der tatsächlichen Berührung bis zur Erkennung und Kenntnisnahme in einer Surface-Anwendung, was zum Teil auch von der aktuellen CPU-Auslastung abhängig ist.

  Weiterhin benötigt jeder Kontakt eine Mindestfläche, um als solcher erkannt zu werden (ca. 1,2mm²). Die Auflösung des optischen Systems entspricht in etwa der Bildschirmauflösung, womit theoretisch Pixel-genaues Anwählen möglich wäre, praktisch jedoch die Präzision für Berührungen durch Finger nicht hoch genug ist.

  Tracking (Abtastrate), Latenzen, Auflösung und Genauigkeit sind letztlich alles Dinge, die nicht am Surface-Simulator getestet werden können.

- **Audio**: Der Tisch bietet die Möglichkeit der Stereo-Ausgabe über zwei eingebaute „flat panel“ Lautsprecher. Weiterhin sind zwei Kopfhörerausgänge am Tisch verfügbar. Es gibt keine Audio-Eingänge und auch kein Mikrofon am Surface.

- **Netzwerk**: IEEE802.11b, IEEE802.11g, Bluetooth 2.0, Gigabit Ethernet. Es besteht die Möglichkeit der Vernetzung des Tisches mit einem LAN, WAN oder Wireless LAN sowie über Bluetooth-Funkverbindungen mit nahen Endgeräten, z. B. Smart-Phones, PDAs oder anderen Hardware-Controllern wie Gamepads und dergleichen.

- **S-VGA Anschluss**: Es existiert ein externer, analoger VGA Monitor-Anschluss für Entwicklungs- und Administrationszwecke. Im Prinzip wird darauf der primäre Windows-Desktop ausgegeben, während der interne Projektor den sekundären Windows-Desktop benutzt.

- **6 USB 2.0 Ports**: Die USB-Anschlüsse werden ebenfalls zu Entwicklungszwecken zum Anschluss von Maus und Tastatur benötigt. Denkbar wäre auch hier der Anschluss von

externen Controllern, wie Gamepads, von Soundkarten und zusätzlicher Sensorik wie Kameras, Mikrofonen, etc., jedoch empfiehlt Microsoft dies nur bedingt [MS:SUEG].

2.2.2 Surface Software-Umgebung

Das Betriebssystem der Surface-Plattform in der aktuell vorliegenden Version bzw. im aktuellen Auslieferungszustand ist MS-Windows-Vista Business Edition, 32Bit SP1 und .NET Framework Version 3.5 [MS:SST2008].

Da es sich beim Surface um ein Standard Windows Vista und Intel Dual-Core basiertes PC-System handelt, kann die ganze Bandbreite an Funktionen des .NET-Frameworks bei der Entwicklung genutzt werden. Dazu gesellen sich Klassen und Komponenten, welche speziell für die Surface-Plattform als Teil des Surface-SDK (Software-Development-Kit, vgl. Abschnitt 2.2.2.5) enthalten sind, z. B. in Form von UI-Komponenten oder für den Zugriff auf und die Verarbeitung von Berührungsinformationen.

Das Surface-SDK enthält weiterhin einige Anwendungs- und Codebeispiele, Projektvorlagen und einen „Surface-Simulator“. Dieser ermöglicht, zumindest rudimentär die Anwendung unabhängig von der Surface-Hardware auf einem Entwicklungsrechner zu testen. Das Surface-SDK liegt zum aktuellen Zeitpunkt in der Version SP1 vor.

Mit dem Surface ausgeliefert werden auch einige Beispielanwendungen wie „Piano“ und „ScatterPuzzle“, für die teilweise auch der Sourcecode verfügbar ist.

Einige Beispielanwendungen (Sample Applications) sind nur über die Surface-Community verfügbar [MS:SurfaceCommunity]. So z. B. auch die „Mobile Connect Sample Application“ [MS:MobileConnect], welche den Austausch von Daten wie Bildern, MP3-Dateien, Klingeltönen, Adressen, etc. zwischen dem Surface und Mobiltelefonen über Bluetooth veranschaulicht. Es existieren dazu Clients für Windows-basierte Smartphones und auch Java-ME. Das „Microsoft Surface Games Pack“ [MS:SGP] enthält weitere Spielanwendungen für das Surface, auf die später detaillierter eingegangen wird (vgl. Abschnitt 2.4).

2.2.2.1 Entwicklungssoftware

Grundsätzlich entsprechen die Grundanforderungen an die Entwicklung von Anwendungen für das Surface denen, die sich bei der Anwendungsentwicklung für Microsoft Windows Vista-basierte PC-Systeme stellen, denn im Prinzip handelt es sich im Inneren um einen Standard-Intel-basierten Windows-Vista PC, wie er 2007 modern war.

Die von Microsoft präferierte Entwicklungsumgebung für Surface-Plattformen ist Microsoft Visual Studio [MS:SST2008], welches in der Express-Edition kostenlos erhältlich ist. Wie die vollständige Entwicklungsumgebung mit dem Surface-SDK aufzusetzen ist und welche anderen Hard- und Softwareanforderungen bestehen, entnimmt man der dem SDK mitgelieferten Dokumentation.

Für die Anwendungsentwicklung, bzw. zum Installieren, Testen und Debuggen auf dem Surface selbst, müssen ein Monitor sowie Tastatur/Maus angeschlossen werden. Durch Visual-Studio kann dann direkt auf der Plattform entwickelt, kompiliert und debugged werden. In der Visual-Studio Umgebung stehen Projektvorlagen für Anwendungen auf dem Surface auf Basis von Microsoft Windows Presentation Foundation (WPF) und Microsoft XNA bereit. Diese Vorlagen ähneln vom Aufbau her stark den Projekt-Vorlagen für Standard-Windows-Anwendungen. Darüber hinaus binden sie allerdings einige Bibliotheken des Surface-SDK ein und überschreiben Routinen zur Initialisierung und Behandlung spezieller Modi der Anwendung (Fullscreen, Modal und

Pausierung), z. B. wenn diese in den Hintergrund gerät.

2.2.2.2 Entwicklungswerkzeuge

Microsoft und andere Spiele-Entwickler [MS:XNA:Creators, Gamasutra, Reed:2009, u.a.] empfehlen den Einsatz von (externen) Werkzeugen (Tools) z. B. für die Erzeugung, Bearbeitung oder Konvertierung eigener Inhalte. Da eine erschöpfende Auflistung empfohlener Lösungen sehr umfangreich wäre, folgt eine Liste ausgewählter Werkzeuge, die zur Aufbereitung von Inhalten für WPF- und XNA-Anwendungen oder zur Unterstützung bei der Entwicklung von (Spiele-)Anwendungen für das Surface genutzt werden können.

- Das Audiotool „XACT“ (Cross-Platform Audio Creation Tool) ist in der XNA-Entwicklungsumgebung als eigenständige Anwendung enthalten. Es dient dem Anlegen und Verwalten von Audio-Projekt-Dateien, die über die Content-Pipeline in XNA-Projekte importiert werden können. Die Projektdateien stellen im Wesentlichen eine Sound-Datenbank dar, wobei Samples zusätzlich mit verschiedenen vorgefertigten Effektinformationen belegt werden können.
- DirectX SDK [MS:DirectX]:
  - „PIX“ [MS:PIX]: Dient Debugging und Profiling-Zwecken von Direct3D-Operationen
  - Diverse Konvertierungs- und Betrachtungs-Werkzeuge stehen für die Aufbereitung von Texturen zur Verfügung[2]
  - „Sample Conversion Tool“: dient der Konvertierung von 3D-Modell- und Szeneninformationen von AutoDesk FBX-Format basierten Dateien in ein DirectX kompatibles Format[3].
- Der „Surface-Simulator“ ist im Surface-SDK enthalten und dient dem Testen von Surface-Anwendungen auf Windows-PCs. Insbesondere die (Multi-)Touch-Eingabe und Erkennung von „ID-Tags“ kann damit simuliert und so getestet werden. Dazu können auch Eingaben aufgezeichnet und wiederholt abgespielt werden.
- Microsoft® Expression® Studio 2 und Microsoft® Expression Blend™ 2 sind kommerzielle Werkzeuge von Microsoft. Ähnlich zu Adobes Flash dienen sie vornehmlich Designern, um Grafiken zu entwerfen, Oberflächen zu gestalten oder Animationen zu erstellen. Sie integrieren sich in Visual-Studio und unterstützen WPF.
- Blender [Blender]: Hierbei handelt es sich um ein freies Werkzeug zur Modellierung von 3D-Modellen und zur Format-Konvertierung

Prinzipiell kommen aufgrund der vorliegenden Hard- und Softwareausstattung des Surface neben Visual-Studio auch andere Entwicklungswerkzeuge, -sprachen und -umgebungen in Frage; in dem Bereich der Spielentwicklung auch integrierte Lösungen wie z. B. Unity3D [Unity3D] oder DarkGAME-Studio [DarkGAME] und eine ganze Reihe anderer kommerzieller Produkte. Jedoch haben alle diese Produkte das Problem, nicht direkt auf die spezifisch über das Surface-SDK implementierten Funktionen zugreifen zu können. Zur Lösung könnten für entsprechende Bereiche Schnittstellen und Kommunikationsprotokolle konzeptioniert und Inter-Prozess-Kommunikation betrieben werden.

2 URL: http://msdn.microsoft.com/en-us/library/bb219724(VS.85).aspx [Online 29.10.2009]
3 URL: http://msdn.microsoft.com/en-us/library/dd607389(VS.85).aspx [Online 29.10.2009]

2.2.2.3 Windows Presentation Foundation (WPF)

„Windows Presentation Foundation“ (WPF) [MS:WPF] ist Teil des .NET-Frameworks von Microsoft. Es stellt eine Ergänzung zu Microsofts „Windows-Forms“ dar, was hauptsächlich auf die Entwicklung von Benutzerschnittstellen formular-basierter Anwendungen ausgelegt ist. WPF hingegen soll die Entwicklung multimedialer Anwendungen fördern mit Unterstützung für Video und Audio, Animation, usw. Mit der Version 3.5 der WPF ist auch die Unterstützung von 3D-Ausgaben über Direct 3D ähnlich möglich wie in XNA. Weiterhin ermöglicht die Unterstützung von XAML [MS:XAML] durch Visual Studio und weitere Werkzeuge von Microsoft, die schnelle und teilweise auch visuelle Entwicklung von Anwendungen. Das Surface-SDK bietet für die Entwicklung mit WPF einige interaktive Komponenten an, die speziell für die Eingabe durch Berührung ausgelegt sind und sich vollkommen in die .NET/WPF Umgebung integrieren (vgl. Abschnitt 2.2.2.5).

Für die Spielentwicklung ist es durchaus eine Überlegung wert, WPF einzusetzen, zumal es für multimediale, „rich-user-experience“-Anwendungen ausgelegt ist und der Funktionsumfang mit kommenden .NET Versionen weiter ausgebaut wird. [WPF3.5] empfiehlt jedoch für die professionelle Spielentwicklung WPF lediglich für die Entwicklung einfacher Spiele oder von Prototypen zur Veranschaulichung von Konzepten oder Szenarien.

2.2.2.4 Microsoft XNA

XNA [MS:XNA:DEV] ist ein .NET basiertes Framework von Microsoft, welches bei der plattform-unabhängigen Spielentwicklung auf windowsbasierten Systemen unterstützt. Es wird vornehmlich für die Entwicklung von Spielen unter Windows XP und Vista, aber auch für Microsoft „XBox 360“ und Microsoft „Zune“-Player eingesetzt. Im Wesentlichen ist der Umfang des Frameworks in 4 Schichten einzuteilen, wie die folgende Abbildung 4 veranschaulicht.

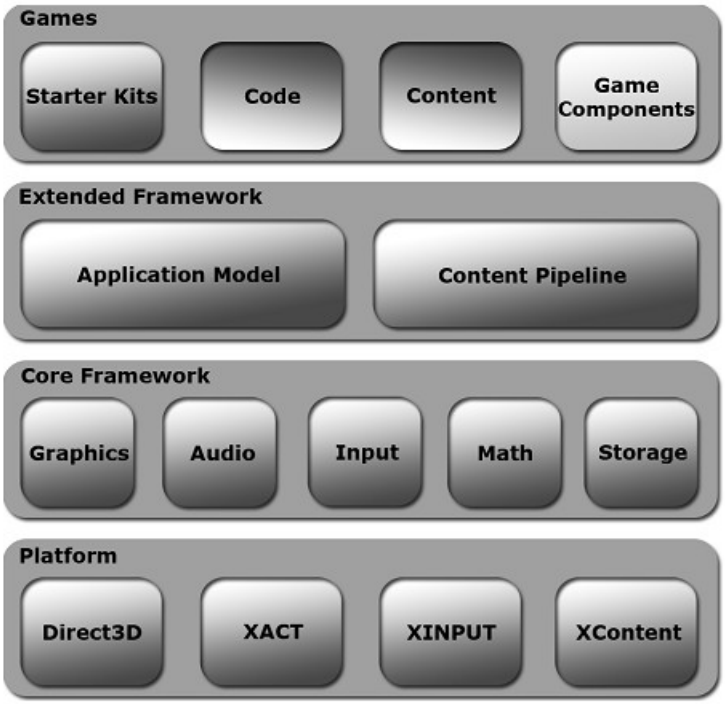

Abbildung 4: Die vier Schichten von XNA [XNA:Schichten]

- „**Platform**“ stellt die unterste Schicht und damit das Fundament des Frameworks dar, das die Plattformunabhängigkeit über native Anwendungsschnittstellen zu DirectX realisiert. Es dient der grafischen Ausgabe (Direct3D), der Audioausgabe (XACT), der Kommunikation mit der Peripherie wie z. B. mit Gamepads oder der Maus (XINPUT) und der Verarbeitung von Ressourcen und Spielinhalten, wie Texturen, Audiodateien und dergleichen (XContent).

- „**Core Framework**“ bietet dem Spiel-Programmierer Klassen und Schnittstellen zur einfachen Nutzung der Kernfunktionen von XNA. Neben den unter „Platform“ bereits genannten Ein- und Ausgabefunktionalitäten unterstützt die „Math“-Bibliothek bei komplexeren mathematischen Operationen, z. B. Matrixmultiplikation. „Storage“ ermöglicht einheitlichen Lese- und Schreibzugriff auf Daten auf Zielplattform-spezifischen Speichermedien. „Graphics“ bietet hier eine vereinfachte Schnittstelle zur Nutzung von Direct3D, die dabei Zugriff auf wichtige Strukturen und Funktionen z. B. aus der Shader-Programmierung erlaubt. So ist z. B. die Nutzung von Vertex- und Pixel-Shader-Routinen durch Microsoft High-Level-Shader-Language (HLSL) in XNA integriert.

- „**Extended Framework**“ umfasst grundlegende Mechanismen und Strukturen, die einen gewissen „Best-Practice“-Standard gewährleisten sollen, um die Spielentwicklung zu vereinheitlichen, aber auch zu vereinfachen, bzw. zu beschleunigen. Ein einheitliches „Application Model“ soll dabei den Ablauf der Anwendungen auf verschiedenen Plattformen mit einer einheitlichen Software-Architektur ermöglichen. Teil dieser Konstruktion ist z. B. die Initialisierung und einheitliche Bereitstellung von Hardware-Ressourcen auf unterschiedlichen Plattformen oder die so genannte „Game-Loop“, in der die Aktualisierung der Spiel-Logik und -Daten je nach Eingaben und Umgebung in einem festen Intervall stattfindet. Das „Application Model“ sieht bei diesen Vorgängen die Einbindung von „Game-Components“ (s. u.) über entsprechende Schnittstellen vor und bietet einen zentralen Zugriffspunkt für so genannte „Game-Services“. Dieser Aufbau erlaubt die lose Kopplung zwischen Komponenten und Ressourcen durch eine einheitliche Schnittstelle.

  Die „Content-Pipeline“ stellt außerdem einen einheitlichen Mechanismus zur Benutzung von Mediendaten zur Verfügung und sorgt für die Zielplattform-spezifische Aufbereitung der Formate.

- „**Games**“ ist schließlich der Bereich, in dem die eigentliche Spielentwicklung stattfindet.

  Unter „Starter Kits“ sind dabei XNA-Beispielprojekte hinterlegt wie z. B. ein Autorennspiel oder ein Jump-and-Run-Spiel, die von Microsoft frei zur Verfügung gestellt werden. Sie können zum Experimentieren oder als Grundlage für eigene Entwicklungen dienen.

  Die Subbereiche „Code“, „Content“ und „Game Components“ umfassen die eigentliche Implementation eigener Spiele, also Quelltexte und Inhalte wie Grafiken, Musik und dergleichen. „Game Components“ spielen hier eine besondere Rolle, da sie als wieder verwendbare, leicht integrierbare und in sich abgeschlossene Einheiten konzipiert sind. Sie sollen sich wiederholende Aufgaben lösen und so die Entwicklung beschleunigen. Außerdem sieht das Framework Austauschbarkeit und Integrierbarkeit durch das „Application Model“ und entsprechende Schnittstellen vor.

  Es existiert eine aktive Online-Community im Bereich XNA-Entwicklung betrieben von Microsoft [MS:XNA:Creators], die vielfältiges Material rund um die Entwicklung von Spielen mit XNA anbietet wie Dokumentationen, Tutorials und insbesondere auch Spiele teilweise

mit Sourcecode. Hier finden sich auch „Game-Components“ für verschiedene Aufgaben, die für eigene Entwicklungen genutzt werden dürfen. Gleichzeitig bietet die Online-Community auch Raum zur Veröffentlichung eigener Komponenten, Tutorials oder Spiele.

Die Entwicklung mit XNA kann über die kostenlos erhältliche Microsoft Visual-C# 2008 Express Edition, über XNA Game Studio oder andere Visual-Studio 2008 C# Produkte von Microsoft erfolgen. Bei der Installation des Surface-SDK werden bereits installierte Visual Studio 2008-basierte Produkte für die Nutzung der SDK-Bibliotheken und Projektvorlagen automatisch konfiguriert.

Für „XBox 360“-Spiele existiert über „Microsoft Live 360-Marketplace“ auch ein Distributionskanal für die kommerzielle Vermarktung eigener Entwicklungen. Für Windows-PC- bzw. Surface-Spiele gibt es einen solchen Kanal derzeit nicht.

2.2.2.5 Surface-SDK

Das Surface-SDK ist die Basis, um eigene Anwendungen für das Surface zu entwickeln. Die Klassen des Surface-SDK sind allesamt im .NET Namensraum „Microsoft.Surface“ untergebracht. Die Grundlage stellen die Klassen im Namensraum „Microsoft.Surface.Core“, welche sich ausschließlich der Verarbeitung von Eingabe und Berührungsinformationen widmen, was das Surface auszeichnet. Darunter fällt auch die Erkennung von physikalischen Objekten auf der Oberfläche durch „Tags“ und die Eingabe über das Surface „on-Screen“ Keyboard.

Für WPF-basierte Projekte bringt das Surface-SDK weitere User-Interface-Elemente und Controls wie Buttons, Menüs und Einträge, Listen und dergleichen im Namensraum „Microsoft.Surface.Presentation“ mit. Eine wichtige visuelle Komponente stellt die „ScatterView“, bzw. die „ScatterItem“-Control dar. Hierbei handelt es sich um eine Konstruktion aus interaktiver Fläche und Container, die auch komplexere Berührungsinformationen verarbeiten und grundlegende Gesten in Form von Ereignissen (Events) bereitstellen können. Zudem implementieren sie ein physikalisches Standardverhalten bei der Interaktion, welches zudem konfigurierbar ist. Die mitgelieferten Anwendungsbeispiele wie der Photo-Browser oder das ScatterPuzzle-Spiel zeigen den Einsatz dieser Komponenten. Seit dem Service Pack 1 des Surface SDK 1.0 sind zumindest die „ScatterView“-Komponenten auch für XNA verfügbar, ansonsten bleibt dort Zugriff auf den „Core“-Namensraum beschränkt.

Alle im Surface-SDK enthaltenen visuellen und interaktiven Elemente sind lediglich für die zweidimensionale Darstellung und Verarbeitung von Daten geeignet.

2.2.3 Interaktion mit dem Surface

Der folgende Abschnitt soll eine Übersicht über die Interaktionsfähigkeiten und -möglichkeiten des Surface sowie über seine diesbezüglichen Beschränkungen geben. Zunächst wird auf die Regeln („Application Certification Rules“) und Tipps von Microsoft eingegangen, welche im Rahmen der „User Experience Guidelines“ [MS:SUEG] bei der Gestaltung und Umsetzung von Anwendungen für das Surface zu beachten sind.

2.2.3.1 Interaktions-Gestaltungsprinzipien

Eine erhöhte „User-Experience“ durch eine natürliche, einfache und intuitive Bedienungsweise steht, ganz im Sinne der NUI-Prinzipien, an oberster Stelle im Konzept der Surface-Plattform [MS:SUEG]. Dies wirkt sich auf das Design von Anwendungen und deren Benutzeroberflächen aus, weshalb Microsoft die folgenden 8 Interaktions-Gestaltungsprinzipien (Interaction Design Principles) formuliert:

- „**Seamless**“: Der Übergang von der realen/physikalischen zur virtuellen Welt sollte möglichst „nahtlos“ sein, so dass der Benutzer mental und emotional in die Anwendung „eintauchen“ (i.. S. v. „immersion“) und frei von Hemmungen neuen Erfahrungen begegnen kann. Dies wird erreicht, indem Objekte und Verhalten aus der realen Welt nachgeahmt und um Möglichkeiten der virtuellen Welt erweitert werden. Weiterhin sollte das Surface bei jeder Art von Berührung eine Antwort durch visuelles Feedback oder eine entsprechende (Verhaltens-)Anleitung geben und möglicherweise sogar korrigierend auf unklare Eingaben reagieren. Dem Benutzer soll das System lebendig und intelligent erscheinen und schließlich eine möglichst fehlerfreie Bedienung ermöglichen.

- „**Social**“: Das Surface ist konzeptionell für den Mehrbenutzer-Betrieb ausgelegt, den auch die Anwendungen unterstützen sollen. Um die Kommunikation zwischen Benutzern zu fördern, sollte die Anwendung neben ihrer einfachen Bedienung möglichst unaufdringlich und dezent erscheinen. Die Benutzung des Surface sollte nicht ablenken, damit der Fokus auf dem Wesentlichen bleibt und sich mehrere Benutzer vielmehr aufeinander konzentrieren können. Im Rahmen der Anwendungsentwicklung sollte ferner beachtet werden, wie mehrere Benutzer gleichzeitig Aufgaben bearbeiten oder virtuelle Objekte manipulieren, was unter Umständen kooperativ geschehen kann. Hierbei ist auch zu berücksichtigen, wie Benutzer um den Tisch angeordnet sind, bzw. wie sie sich die Interaktionsfläche teilen und wie die Anwendung zu ihnen ausgerichtet ist.

- „**Spatial**“: Die Ausgabe und insbesondere Bedienung des Surface durch Touch findet auf der planen Oberfläche statt; Eingaben werden also nur zweidimensional entgegengenommen. Anwendungen jedoch sollen dem Benutzer „Räumlichkeit“ vermitteln und dargestellte Objekte Volumen und Tiefe besitzen, so dass sie greifbar erscheinen. Dies kann relativ einfach und performant durch visuelle Effekte der 2.5-dimensionalen Darstellungstechnik realisiert werden, bei der gegenüber der bloßen 2D-Darstellung zusätzlich die Tiefenachse (Z-Achse) zur Unterscheidung von Ebenen vorliegt. Damit können virtuelle Objekte in den Vorder-/Hintergrund rücken oder sich überlagern und z. B. der Schattenwurf von sich überlagernden Elementen dargestellt werden. Auch müssen nicht immer alle Elemente sichtbar sein, sondern es können auch nur Ausschnitte einer virtuellen Umgebung visualisiert werden, die durch den/die Benutzer navigierbar sind. Auch wenn es sich nicht für jede Anwendung eignen dürfte, ist es grundsätzlich möglich,

eine Abbildung einer 3-dimensionalen Umgebung aus beliebigen Positionen und mit verschiedenen Perspektiven zu projizieren. Das ist zwar vergleichsweise rechenintensiver, wird aber durch Hard- und Software unterstützt. Wichtig zu beachten ist hierbei, dass der Benutzer möglichst intuitiv und einfach durch diese Welt navigieren und sich orientieren kann. Idealerweise sollte er eine gedankliche Karte der virtuellen Umgebung entwickeln (Beispiel: "Globe-Application") und wissen, wie er zu einem bestimmen Punkt/ zu einer bestimmten Position gelangt.

- „**Super-realism**“: Indem physikalisches Verhalten von Objekten aus der realen Welt auf virtuelle Objekte abgebildet wird, entstehen natürliche und bekannte Interaktionsmöglichkeiten für den Benutzer im Umgang mit diesen Objekten. Diese Möglichkeiten sollen mit den Mitteln der Virtualität erweitert und dem Benutzer auf intuitive Weise zugänglich gemacht werden. Z. B. können bei der Microsoft Photo-Browser-Applikation einerseits Bilder durch Berührung beliebig bewegt, gedreht, etc. werden. Zusätzlich können sie durch Gesten skaliert werden, was mit realen Fotos und „bloßen Händen“ nicht möglich wäre. Fähigkeiten wie diese sollen die Erfahrung für den Benutzer besser und attraktiver gestalten als es die reale Welt zulässt.

- „**Contextual Environments**“: Surface-Anwendungen sollten kontextbezogen auf den internen und externen Zustand der Umgebung sowie auf Benutzereingaben reagieren. Diese Reaktion soll für den Benutzer klar als Ergebnis seines Verhaltens erkennbar sein, damit er erlernen und antizipieren kann, wie sich sein Handeln auswirkt und er die Kontrolle behält/erlangt.

- „**Scaffolding**“: Dem Benutzer sollte immer nur eine kleine, aber sinnvolle, Auswahl an Optionen für den (Entscheidungs-)Moment zur Verfügung stehen, um nicht verwirrend oder überfordernd zu wirken. Dabei soll der Benutzer unterstützt werden, die Lösung eines Problems oder einer Fragestellung aktiv zu erkunden und zu entdecken. „Scaffolding“ beschreibt die Methode, durch gezielte Fragestellung und Hinweise komplexe Problemstellungen für den Benutzer in Teilprobleme und -fragen zu zerlegen, welche für ihn einfacher lösbar sind.

- „**Performance Aesthetics**“: Surface-Anwendungen sollen performant und ästhetisch sein, um dem Benutzer ein gutes Gefühl zu geben und ihn zu ermutigen, das System, seine Regeln und Kultur zu erkunden. „Performance“, also die Leistung, bzw. das Laufzeitverhalten des Systems, spielt dabei eine grundlegende Rolle, denn es sollen zur Darstellung nur hochwertige Grafiken, Animationen und andere multimediale Inhalte benutzt werden. Dabei soll das System auf jegliche Eingabe möglichst sofort Reaktion zeigen und zumindest direktes visuelles Feedback geben. Das Haken oder Stocken der Anwendung, z. B. beim Abspielen von Animation oder Sounds ist stets zu vermeiden, und eine flüssige Bedienung sollte immer möglich sein.

- „**Direct Manipulation**“: Die Steuerung einer Surface-Anwendung geschieht i. d. R. über Berührung der Oberfläche mit einem Finger oder mehreren Fingern zugleich, mit der ganzen Hand oder durch physikalische Objekte. Inhalte sind auf diese Weise direkt und unvermittelt manipulierbar und ändern ihren Zustand oder ihre Eigenschaften in Abhängigkeit von der jeweiligen Aktion. Dem Benutzer wird das Ergebnis seiner Aktion (ein veränderter Zustand/Inhalt digitaler Objekte) dabei direkt zurückgemeldet.

2.2.3.2 Interaktions-Richtlinien

Basierend auf den beschriebenen Interaktions-Gestaltungsprinzipien beinhalten die Interaktions-Richtlinien („Interaction Guidelines“) von Microsoft [MS:SUEG] etwas konkreter formulierte Regeln zur Gestaltung der Benutzer-Interaktion von Anwendungen für das Surface. Die Interaktions-Richtlinien sind in die Kategorien „must“, „should“ and „could“ eingeteilt. Die „must“-Regeln beschreiben die Mindestanforderungen zur Erfüllung dieser Prinzipien und müssen für eine Zertifizierung einer Anwendung durch Microsoft eingehalten werden (**„Application Certification Rules“**):

1. „Make Virtual Objects behave like Physical Objects“: Objekte sollten generell physikalische Eigenschaften und Verhalten wie Masse, Reibung, Beschleunigung, Viskosität, u. ä. aufzeigen. Dabei sollen Statuswechsel virtueller Objekte flüssig, idealerweise animiert, sein. Nichts sollte abrupt erscheinen oder verschwinden.
2. „Provide a 360-Degree User Interface“: Die Anwendung und deren Inhalte sollten die Steuerung von allen Seiten des Surface erlauben. Insbesondere auch neue Inhalte sollten dem Benutzerstandort, bzw. bereits vorhandenen Elementen entsprechend ausgerichtet sein.
3. „Support Using 2D Planar Space“: Die Anwendung soll bezüglich Layout, Feedback, Verhalten und Bildern/Icons für Touch optimiert sein. Dazu sollten virtuelle Objekte, die auf Touch reagieren, nicht kleiner als 15mm in jede Richtung sein und bei Minimalgröße einen Mindestabstand von 5mm nicht unterschreiten. Weiterhin sollten interaktive Elemente so angeordnet werden, dass relevante Inhalte nicht von physikalischen Objekten oder von dem Arm oder der Hand des Benutzers verdeckt werden.
4. „Adhere to Principles of 3D Space Utilization (the Z-Axis)“: Bewegliche virtuelle Objekte sollten die Z-Achse zumindest für das Feedback von Tiefe und Fokus nutzen (2.5D), sofern die Z-Achse nicht anderweitig genutzt wird (3D).
5. „Create Immediate responses to all user input that will receive a response“: Das Feedback bei der Bedienung soll stets direkt und visuell geschehen. Diese und die nächsten 3 „must“-Regeln (5-8) stammen aus der Richtlinie zu „Make Experiences Natural and Better than Real“.
6. „Enable single-finger drag and flick movements on movable content“: Diese Basis-Gesten (siehe auch Abschnitt 2.2.3.4) sollten bei jeder Anwendung implementiert werden.
7. „Enable inertia on objects and content that users can move about the screen“: Massenträgheit als physikalisches Verhalten soll virtuellen Objekten implementiert werden. Hierbei können die Klassen des so genannten „Inertia-Processor“ unterstützen, die Teil des Surface-SDK sind.
8. „Do not use time-based gestures (such as press-and-hold) on content“: Solche Gesten bedeuten eine Verzögerung (Delay) in der Bedienung, was insbesondere professionelle/schnelle Benutzer behindert und einer flüssigen und natürlichen Bedienung entgegenwirkt.
9. „Properly Integrate with Surface-Shell“: Ein Video oder eine Slide-Show (mehrere Bilder) sollen als Vorschau im „Launcher“ angezeigt werden. Dabei soll schon konkret gezeigt werden, was die Anwendung genau darstellt und gegebenenfalls wie sie zu bedienen ist.
10. Ergänzend zu Regel 9 soll ein beschreibender Text in der Vorschau des „Launcher“

bereitgestellt werden.

11. Nicht mehr aufgeführt (entfallen)
12. Nicht mehr aufgeführt (entfallen)
13. „Use Progressive Disclosure to Reveal Functionality“: Dem Benutzer sollen nicht immer alle Inhalte und Funktionen zur gleichen Zeit angeboten werden, ihm sollten eher wenige, aber dafür relevante Optionen im aktuellen Kontext gegeben werden. Dazu können dem Benutzer bekannte Interaktions-Metaphern und -Paradigmen aus der realen Welt bei der Gestaltung virtueller Objekte helfen. Z. B. lässt sich eine Visitenkarte oder ein Musikalbum umdrehen, um weitere Informationen zu erhalten. Falls ungesehene Inhalte existieren, sollte der Benutzer möglichst (visuell) angeleitet werden, wie er auf diese Inhalte zugreifen kann, z. B. durch Anzeige einer animierten „Flip-Geste“.
14. „Ensure a Sense of Success“: Ergänzend zu Regel 5, soll dem Benutzer stets der Zustand der Anwendung klar sein und wie er fortfahren kann. So sollten z. B. nicht bedienbare oder deaktivierte Elemente als solche gekennzeichnet sein, bzw. bedienbare Inhalte sich deutlich visuell abheben.
15. „Make the Content the Interface“: Generell sollten Anwendungen wenige (Steuer-)Elemente anzeigen, die nicht auch „Inhalte“ sind. Dem Benutzer soll immer das Gefühl gegeben werden, möglichst direkt mit den (virtuellen) Inhalten zu interagieren. Deswegen ist auf die Verwendung von „Buttons“, „Scrollbars“ und anderen Steuerelementen, wie sie von traditionellen GUIs her bekannt sind, zu verzichten. Stattdessen sind die Inhalte durch Gesten direkt manipulierbar zu gestalten.
16. „Use Manipulation Gestures, Not System Gestures“: Während „System Gestures“ keinen Bezug zu aktuell angezeigten Inhalten haben, beziehen sich „Manipulation Gestures“ direkt auf virtuelle Inhalte und ändern z. B. deren Positionen oder Zustände. Dabei sollte möglichst auf die Standardgesten und ihre Funktionsweisen zurückgegriffen und konsistent eingesetzt werden.

2.2.3.3 Touch/Multi-Touch

Die Steuerung des Surface durch Berührung der Oberfläche stellt die wichtigste Eingabeform für den Benutzer dar, wie es auch aus einem aktuellen „Tech-Report“ von Microsoft [MS:TechReport:2009] über die Nutzung interaktiver horizontaler Flächen hervorgeht. Insbesondere die Funktionalität des Multi-Touch, also die Erkennung mehrerer gleichzeitiger Berührungen, ermöglicht prinzipiell auch den „Mehrbenutzer-Betrieb“, der naturgemäß wichtigsten Eigenschaft kollaborativer Anwendungen. Hardware-bedingt verfügt das Surface jedoch nicht über die Möglichkeit, Berührungen bestimmten Benutzern zuzuordnen, was auch bei der Gestaltung von (Multi-User-)Anwendungen, bzw. ihrer Benutzeroberflächen, zu berücksichtigen ist.

Das Surface erfasst und verwaltet bis zu 52 gleichzeitige Kontakte, bzw. Objekte auf der Oberfläche [MS:SUEG]. Dabei werden zusammenhängende Kontaktflächen zu einem so genannten „Blob“ zusammengefasst. Ein Kontakt, bzw. „Blob“, wird durch eine elliptische Fläche beschrieben, wobei Mittelpunkt, Rotation, Höhe und Breite der Ellipse den Ausprägungen der Kontaktfläche entsprechen. Entsteht also ein „Blob“ durch mehrere zusammenhängende Kontaktflächen, etwa durch Auflegen der ganzen Hand, liegt für die Form der Fläche nur noch *eine* Beschreibung ihrer maximalen Breite, Höhe und Ausrichtung vor. Für jeden „frischen“ Kontakt wird vom System eine eindeutige Identität (GUID) vergeben, womit Kontakte bei Bewegung verfolgt

werden können (Tracking). Das ermöglicht dem System das Erkennen von Kontakt- und Bewegungsereignissen sowie die Berechnung von Geschwindigkeits- und Beschleunigungswerten.

Softwaretechnische Unterstützung zur Verarbeitung der Kontakt-Informationen erhält man durch das Surface-SDK in Form der „Contact“-Klassen. Sie bieten Zugriff auf die oben genannten Informationen und werden auch von anderen visuellen Komponenten wie „ScatterView“ und „ScatterItem“ benutzt.

2.2.3.4 Gesten

Zu der Steuerung von und dem intuitiven Umgang mit virtuellen Objekten sieht Microsoft den Einsatz von Manipulationsgesten vor [MS:SUEG]. Eine Geste setzt sich aus der Kombination gewisser Kontakt- und Bewegungsinformationen zusammen, die, im zeitlichen Ablauf zu einer entsprechenden „Aktion“ zusammengefasst, direkt den Zustand des Systems oder Objekts verändert. Standardmäßig implementiert Microsoft durch die „ScatterView“-Komponenten Gesten zur Manipulation der Position, Rotation und Größe von Objekten. Gesten können, auch durch den Einsatz mehrere Finger oder Hände, durch folgende Bewegungsabläufe erfolgen:

- „Tap“: Berühren und Loslassen; entspricht der Auswahl/Aktivierung eines Elements („Maus-Klick“)
- „Slide or push/Drag“: Berühren eines Elements und Bewegung über die Oberfläche; verschiebt das Element oder einen (Bild-)Ausschnitt entsprechend
- „Flick“: Wie „slide“; plötzliches Loslassen in der Bewegung lässt das Element weiter gleiten in Richtung der Bewegung beim Loslassen (Massenträgheit)
- „Touch-and-turn“: Wie „slide“, jedoch kreist der Finger um die ursprüngliche Position zum Zeitpunkt der ersten Berührung, was die Rotation des Elements um die Position des Fingers bewirkt (Massenträgheit)
- „Pin turn“: Rotation des Elements, indem zunächst ein Finger auf dem Element die Position der Drehachse fixiert und ein zweiter Finger durch Bewegung auf dem Element die Drehung bewirkt
- „Twist“: Durch Berührung eines Elements und Drehung mit zwei Fingern einer Hand wie an einem Knopf, wird eine Rotation des Elements bewirkt
- „Spin“: Wie „twist“, die Rotationsbewegung wird jedoch, ähnlich zu “flick“, durch schnelle abrupte Loslass-Bewegung weitergeführt wie bei einem Kreisel
- „Stretch“ oder „Spread“: Vergrößerung eines Elements oder Ausschnitts durch Auseinanderbewegen zweier Finger einer oder beider Hände
- „Shrink“ oder „Squeeze“: Verkleinerung eines Elements durch Zusammenführen zweier Finger oder Hände

Es bleibt anzumerken, dass standardmäßig nur „ScatterView“-basierte Komponenten Unterstützung für diese Manipulationsgesten erlauben. Weiterhin sind nicht alle diese Gesten direkt/unmittelbar verfügbar, teilweise müssen in der Ereignis- und Zustandsverwaltung, bzw. im Grundverhalten der ScatterView-basierten Komponenten Anpassungen vorgenommen werden, um den gewünschten Effekt zu erreichen.

Prinzipiell existieren noch viele weitere Möglichkeiten, durch zweidimensionale Gesten digitale Inhalte zu manipulieren [Saffer:2008, Wobbrock:2009]. Im Wesentlichen beruhen diese aber auf den genannten Grundbewegungen, bzw. deren Kombination. Wichtig bei der Entwicklung eigener Gesten und Verhalten ist die Berücksichtigung der jeweiligen Intention der durchzuführenden Aktion, denn die Bedienung soll stets einfach und intuitiv sein. Zu komplexe Gesten, z. B. durch den zwingenden Einsatz beider Hände, ungeeignetes Timing oder die Fehler-intolerante Verarbeitung von Eingabedaten können den Benutzer schnell irritieren und frustrieren. Schließlich unterliegt die Erkennung komplexerer Gesten der Analyse von Bewegungsmustern, welche beim Surface in Form von Vektordaten zu den Kontaktinformationen zur Verfügung stehen. Die Unterscheidung von Gesten, die im Bewegungsablauf ähnlich sind, wird erschwert durch fehlende Informationen über die Zuordnung von Benutzern und Händen, bzw. Händen und Fingern.

Die beschriebenen Gesten können auch zur Navigation in 3D-Welten angewendet werden, wie die "Globe-Application" auf dem Surface oder das Projekt "The DabR" [Edelman:2009] zeigen. Beide Beispiele zeigen die Steuerung einer virtuellen Kamera, welche sich durch zweidimensionale Gesten im 3D-Raum positionieren, drehen, kippen, rollen und zoomen lässt. Zur freien Positionierung und Änderung der Darstellungsperspektive im virtuellen Raum werden zur Unterscheidung der Kamera-Funktionen teilweise zweihändig auszuführende Gesten benötigt. Weiterhin ist die Projektion des 3D-Raums oftmals nur zu einer Position am Tisch sinnvoll orientiert und eine Änderung der Perspektive wirkt sich i. Allg. auf alle Betrachter aus, weshalb sich diese Art der Interaktion vornehmlich für einen oder wenige gleichzeitige Benutzer eignet.

Den Einsatz kooperativer Gesten und deren Gestaltung beschreibt [Morris:2006]. Darunter sind Gesten zu verstehen, die erst von mehreren Benutzern in Kombination durchgeführt eine Aktion im System auslösen. Da in der genannten Arbeit die zugrunde liegende Plattform die Zuordnung von Kontakten zu Benutzern erlaubt, was essentiell zur Anwendung beschriebener Techniken ist, findet die Thematik hier keine weitere Beachtung.

Ein weiteres interessantes Anwendungsbeispiel zweihändiger Gesten beschreibt, wie die Präzision von Touch für die Auswahl von Text oder sogar Pixeln erhöht werden kann [Benko:2006]. Der Ansatz beruht im Wesentlichen darauf, dass eine Hand die Selektion/Markierung vornimmt, während die andere Hand den ausgewählten Bildbereich in Größe/Auflösung frei skalieren kann.

2.2.3.5 Objekterkennung

Generell erkennt das Surface Objekte auf der Oberfläche durch einen Kontakt bzw. „Blob“ [MS:SUEG]. Physikalische Objekte können jedoch mit einem so genannten „Tag“ („Marker, Markierung“) ausgestattet werden, den die visuelle Sensorik des Surface erkennen und lesen kann. Wie für „Blobs“ werden Position und Ausrichtung eines „Tags“ bei der Platzierung auf der Oberfläche durch die „Contacts“-Komponenten übermittelt. Zusätzlich wird eine dem jeweiligen „Tag“ inhärente Kennung geliefert, wodurch verschiedene „Tags“, bzw. Objekte unterscheidbar sind. Es existieren so genannte „Byte-Tags“ und „ID-Tags“. „Byte-Tags“ sind kleiner in den Abmessungen und können nur, wie der Name schon vermuten lässt, 256 verschiedene Kennungen kodieren. Sie sind damit nur geeignet, um Objekte einer vergleichsweise groben Gruppe erkennbar zu machen, wie z. B. Marke- oder Typ eines Mobiltelefons. „ID-Tags“ können mehr Informationen aufnehmen (8 Byte) und so z. B. auf eine ganz gewisse Instanz eines Objektes oder Gerätes referenzieren.

Den Interaktions-Richtlinien entsprechend empfiehlt Microsoft, möglichst viele Objekte zur Interaktion mit dem Surface mit „Tags“ auszustatten. Ein „Tag“-spezifisches Feedback der

Anwendung im aktuellen Kontext soll den Benutzer zum Experimentieren und Erforschen einladen. Es bestehen weiter keine zwingend zu erfüllenden Richtlinien im Umgang mit „Tags“ oder Objekten seitens Microsoft.

2.2.3.6 Visuelle Ausgabe

Feedback des Systems erhält der Benutzer vornehmlich über die visuelle Anzeige auf der Tischoberfläche. Ergänzend zu ihren „Interaction Design Principles“ gibt Microsoft explizit für die visuelle Gestaltung von Surface-Anwendungen weitere Prinzipien vor („Visual Design Principles“) [MS:SUEG]:

- „**Consistency**“: Visuelle Elemente sollen innerhalb der Anwendung in der Darstellung (Form, Farbe, Textur und andere Gestaltungselemente) konsistent sein, so dass der Benutzer sich besser orientieren kann.
- „**Flexibility**“: Visuelle Inhalte sollten frei verschiebbar, drehbar und skalierbar sein, was analog zur Interaktion die so genannte „360°-View“ unterstützen soll. Visuelle Eigenschaften sollten auf andere Elemente übertragbar sein, damit dem Benutzer ein konsistentes und harmonisches Bild vermittelt wird.
- „**Premium Quality**“: Visuelle Inhalte sollen detailliert und hochwertig gestaltet sein und dem Benutzer zu jeder Zeit logisch, vorhersehbar und ästhetisch Zustand und Kontext vermitteln.
- „**Understatement**“: Die visuelle Darstellung sollte mehr zurückhaltend als auffällig sein. Das Design selbst sollte nicht im Vordergrund stehen, sondern eher dezent sein und „im Hintergrund unterstützend wirken“.
- „**Minimalism**“: Die Anzahl visueller Elemente sollte generell gering gehalten werden, da sie zumeist interaktive Inhalte darstellen und diese den Benutzer nicht überfordern sollen. Auf unnötige, passive Designelemente wie Verzierungen oder Ornamente sollte verzichtet werden („less is more“), die Gestaltung der Anwendung sollte einfach und elegant anmuten.
- „**Welcoming**“: Die Anwendung sollte einladend erscheinen, was durch das visuelle Design (schöne Formen, warme Farben, etc.) stark bestimmt wird. Weitere zu erfüllende Attribute, bei denen die visuelle Gestaltung Einfluss nimmt, sind: „approachable“, „discoverable“, „forgiving“, „exploratory“.

Weiterhin gibt Microsoft in Form optionaler Richtlinien („Visual Design Guidelines“) Tipps, z. B. im Umgang mit Farben, Grafiken, Texturen, Perspektive, Dimensionen und dergleichen. Im Gegensatz zu den *Interaktions*-Richtlinien sind hier keine Regeln zwingend zu erfüllen, aber auch hier geben die „should“ und „could“ Hinweise wertvolle Hilfestellung für die visuelle Gestaltung der Anwendung, sind aber zu umfangreich, um an dieser Stelle weiter erörtert zu werden. Teil der visuellen Ausgabe ist auch die Darstellung von Text, bzw. Sprache, wovon ein weiterer Abschnitt der „User Experience Guidelines“ [MS:SUEG] handelt, den „Language and Text Principles“. Die vier Grundprinzipien lauten hier:

- „Casual, Clear and Personal Tone“: Der „Tonfall“ von Texten soll locker, klar, freundlich und persönlich sein. Generell sind kurze und prägnante Fragen oder Anweisungen zu formulieren, die dem Benutzer Anleitung geben und ihn dabei unterstützen, schnell

korrekte Entscheidungen zu treffen.

- „Using Text judiciously“: Der Einsatz von Text sollte generell minimal gehalten werden. Die Verwendung sollte möglichst ausschließlich bei kritischen Benutzeraktionen (z. B. beim Beenden der Anwendung) erfolgen oder wenn die Art der Information keine andere (visuelle) Übermittlung zulässt.
- „Text as Graphics“: Text ist ein grafisches Element wie ein Bild, eine Textur und dergleichen. Daher gelten bei der Gestaltung auch die Prinzipien der „Visual Design Principles“. Zusätzlich muss bei Text für eine gute Lesbarkeit gesorgt werden, was durch sorgfältige Auswahl von Schriftart, -größe und -farbe geschieht. Dabei werfen insbesondere die „360°-View“, bzw. Rotation oder Skalierung von Textelementen zusätzliche Schwierigkeiten auf.
- „Text as Audio“: Die Ausgabe von Text in Form von Ton/Sprache darf generell eingesetzt werden, aber auch hierbei sollten die ersten beiden Prinzipien Beachtung finden. Insbesondere sollte der Einsatz von Ton in Form von Sprache wohl überlegt sein, da im Allgemeinen nicht bekannt ist, wie laut das Surface und die Umgebung aktuell sind, wie gut der Benutzer hört, die Sprache versteht, usw.

Auch hierzu folgen innerhalb der („Language and Text Guidelines“) entsprechende Richtlinien, die nicht zwingend zu erfüllen sind, aber wertvolle Tipps für die Gestaltung der Ausgabe von Text geben können.

2.2.3.7 Audio

Der Einsatz von Ton und Musik wird ergänzend zur Optimierung der User-Experience unbedingt empfohlen [MS:SUEG]. Hierbei gilt es, eine unaufdringliche Geräuschkulisse mit qualitativ hochwertigem Material zu erzeugen. Der Einsatz von Geräuschen und deren Lautstärke auf gewisse Aktionen/Ereignisse hin, sollte daher gut überlegt sein. Wie bei der visuellen Ausgabe sollte sich das Surface auch hier eher zurückhaltend zeigen („understatement“).

Das Surface unterstützt die Ausgabe von Stereo-Signalen über zwei eingebaute Lautsprecher [MS:SDS]. Es stehen weiterhin zwei parallel geschaltete Stereo-Kopfhörer-Ausgänge bereit. Damit ist ohne Erweiterung der Sound-Hardware, z. B. durch USB-Soundkarten, kein Surround-Sound oder Benutzer-individuelles Audio-Feedback möglich, wie es [Morris:2004] in „Individual Audio Channels with Single Display Groupware“ beschreibt. Ebenso ist die Aufnahme von, bzw. Eingabe durch Ton und Sprache aufgrund der fehlenden Mikrofone bzw. Line-In Eingänge hardware-bedingt nicht ohne Erweiterungen möglich.

2.2.3.8 Konnektivität

Das Surface bietet verschiedene Möglichkeiten mit anderen Geräten zu kommunizieren. Neben einem Ethernet-Anschluss zur Einbindung des Surface in ein LAN (oder WAN) unterstützt es auch die Funkstandards IEEE802.11b und IEEE802.11g zur Einbindung in ein WLAN. Damit steht der Vernetzung z. B. zur tischübergreifenden Kommunikation oder mit dem Internet prinzipiell nichts im Wege. Software-Unterstützung zur Kommunikation auf Basis von IP (TCP/UDP) erhält man durch die .NET System.Net-Komponenten. XNA enthält außerdem Komponenten zur Nutzung von Microsoft „Games-For-Windows Live“ als zentralem Benutzer- und Spiele-Dienst, was allerdings (bisher) insbesondere für Microsoft „Xbox“ und Windows-PC-Spiele genutzt wird.

Zur Kommunikation mit mobilen Endgeräten wie PDAs, Smart-Phones und anderen funkbasierten Geräten steht außerdem der Bluetooth 2.0-Standard bereit. Die Demoanwendung „Surface Mobile Connect“ [MS:MobileConnect] zeigt, wie eine Kopplung zwischen dem Surface und mobilen Endgeräten bewerkstelligt werden kann, die mit einer Kamera ausgestattet sind. Diese nimmt dabei vom Surface ausgegebene Farbwechsel auf, in denen ein Kommunikationsschlüssel kodiert ist, welcher per Bluetooth bestätigt wird. Die Demoanwendung zeigt den Austausch von Dateien wie Fotos, MP3s oder Adressdaten zwischen Surface und anderen Endgeräten. Interessant ist die Nutzung mobiler Endgeräte aber auch als erweiterte Ein- und Ausgabemöglichkeit und zur Verarbeitung „privater“ Informationen [FalseProphets:2002, Morris:2004, ...]. Zumal dies im Mehrbenutzer-Betrieb auf der jedem zugänglichen Ausgabefläche des Surface im Allgemeinen, d.h. ohne weitere Maßnahmen, nicht geschehen kann.

Grundsätzlich erlauben Bluetooth und USB auch die Anbindung externer Controller, wie z. B. von Maus, Gamepad, Nintendos Wii Remote-Control, etc. Dies empfiehlt Microsoft jedoch nicht und verbietet es sogar [MS:SUEG], wenn dadurch das Interaktionskonzept grundsätzlich gestört oder beeinträchtigt bzw. nicht adäquat erweitert würde. Software-technische Unterstützung zur Kommunikation mit Maus, Keyboard, Joysticks und Gamepads erhält man durch XNA „X-INPUT“.

2.2.4 Dem Surface ähnliche Systeme

Um die Merkmale des Surface besser bewerten zu können, folgt eine Übersicht über Konkurrenzprodukte, also dem Surface ähnliche Systeme anderer Hersteller. Die Auswahl der Produkte ist das Ergebnis einer ausgiebigen Literatur- und Marktrecherche und geschah nach den Kriterien: Ähnliche grundlegende Merkmale und Funktionen (Touch/Multitouch), ähnliche Bauweise, Idee/Anwendung, Erfahrungen und Verfügbarkeit.

- „**Mitsubishi DiamondTouch**“ [DiamondTouch:2001]: Hierbei handelt es sich um ein interaktives Multitouch-Tabletop-System mit Frontprojektion. D.h. im Wesentlichen besteht die Hardware aus der berührungsempfindlichen Oberfläche, welche auf einem Tisch platziert bzw. darin eingelassen werden kann. Diese Fläche arbeitet mit kapazitiver Kopplung, bei der verschiedene elektrische Signale je nach Ort der Berührung durch den Benutzer an seinen spezifischen Empfänger geleitet werden. Das hat den großen Vorteil, dass Berührungen direkt einzelnen Benutzern zugeordnet werden können, was das Surface nicht bietet. Dazu muss jeder Nutzer aber auch Kontakt mit einem Empfänger haben. Ein weiteres Merkmal der kapazitiven Kopplung ist die Debris-Toleranz, d. h. Kontakte, z. B. durch Gegenstände auf dem Tisch, ohne Kopplung mit einem Empfänger werden ignoriert. Die Empfänger-Hardware für bis zu 4 simultane Benutzer wird standardmäßig mitgeliefert. Die berührungsempfindliche Oberfläche gibt es in Ausführungen mit 32“ und 42“ Diagonalen. Die Projektion des Bildes darauf erfolgt, wie bereits erwähnt, durch Front-Projektion, d.h. ein Projektor wird oberhalb des Tisches aufgehängt und strahlt nach unten. Das hat den Nachteil, dass die Nutzer mit ihren Händen und Armen Schatten werfen und dabei an diesen Stellen die Projektion ausbleibt. Neben einem Projektor wird auch ein Windows-basierter PC zur Verarbeitung der Informationen und zur Darstellung des Bildes benötigt. Unterstützung der Touch-Funktionalitäten für die Anwendungsentwicklung bietet das mitgelieferte DiamondTouch-SDK. DiamondTouch wird von der Firma Circle Twelve Inc. [CircleTwelve] kommerziell vertrieben, Preise sind dort zu erfragen.

Abbildung 5: Mitsubishi DiamondTouch [DiamondTouch]

- „**Philips Entertaible**“ [Entertaible:2006]: Das Entertaible ist ein 30“ LCD-Schirm, der auf einen beliebigen Tisch gelegt werden kann. Über eingebaute Infrarot-Sensorik ist der Schirm Multi-Touch-fähig und insbesondere für die Erkennung von physikalischen

Würfelergebnissen und Positionen von Spielfiguren ausgelegt. Philips beschreibt das System selbst als „electronic game which combines the features of computer gaming, such as dynamic playing fields and gaming levels, with the social interaction and tangible playing pieces, such as pawns and dies, of traditional board games.". Neben einzelnen Prototypen und Demospielen wurde insbesondere das Spiel „WeatherGods“ [Weathergods:2007] realisiert und analysiert. Leider sind seit der Vorstellung des Geräts keine Neuigkeiten mehr veröffentlicht worden. Offenbar wurde bisher kein kommerzieller Vertrieb, bzw. keine größere Produktion angestrebt, warum blieb trotz Recherchen unklar.

Abbildung 6: Philips Entertaible [ENTERTAIBLE]

- „**Perceptive Pixel Multi-Touch-Wall**“ und „**Multi-Touch-Workstation**“ [PerceptivePixel]: Die „Multi-Touch-Wall“ ist ein vertikales für bis zu 4 kooperative Nutzer ausgelegtes System; die „Multi-Touch-Workstation“ in Form eines Architekten-Zeichen-Tisches ist dagegen nach Angaben des Unternehmens primär für den Single-User und produktives Arbeiten ausgelegt. Jeff Han präsentierte eine erste Version des Tisches bei der „TED-Talks“ Konferenz im Februar 2006 [Han:2006]. Neben der selbst entwickelten Hardware bietet die Firma auch individuell angefertigte Anwendungen und ein SDK zur eigenen Entwicklung an.

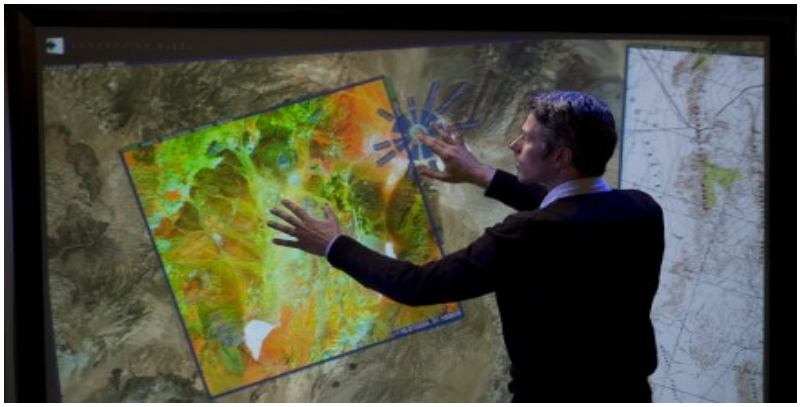

Abbildung 7: Perceptive Pixel Multi-Touch-Wall [PerceptivePixel]

- „**Reactable**“ [REACTABLE]: Ein runder, ca. 90cm hoher Tisch, mit Multi-Touch- und Objekterkennungsfunktionalitäten. Wie beim Surface ist die komplette Technik, also Darstellung, Sensorik und Recheneinheit, in dem Gerät integriert. Der Reactable wurde speziell als Synthesizer-Anwendung entwickelt, d.h. für die interaktive Erzeugung von

Tönen und Musik. Dazu werden reale, mit „ID-Tags“ ausgestattete Objekte auf der Tischoberfläche platziert, die funktional z. B. Taktgeneratoren, Samples oder Filter-Effekte repräsentieren und durch ihre Platzierung abgespielt, bzw. aktiviert werden. Entsprechend ihrer Funktion, Position und ihres Rotationswinkels auf dem Tisch beeinflussen sich die Effekte der einzelnen Objekte, was zu einer Vermischung im ausgegeben Audiosignal führt. Weiterhin sind die Parameter der verschiedenen Effekte und Funktionen durch Touch und Gesten manipulierbar. Der Tisch ist von allen Seiten gleichermaßen durch mehrere Benutzer gleichzeitig bedienbar. Der Reactable wurde an der Universtät Pompeu Fabra (Barcelona) entwickelt und wird mittlerweile auch kommerziell von der Firma Reactable Systems vermarktet, ein 2008 gegründetes Start-Up des Entwicklerteams.

Abbildung 8: ReacTable in Aktion [REACTABLE]

- „**PQ Labs iTable / Multi-Touch G²**“ [PQLabs]: Ähnlich dem DiamondTouch ist die Multi-Touch G² Hardware eine berührungssensitive Oberfläche, die allerdings transparent ist und auf handelsüblichen LCD- oder Plasma-Bildschirmen angebracht werden kann. Diese Hardware gibt es für 32“, 42“, 50“ und 52“ Bildschirmdiagonale und bewegt sich preislich je nach Ausführung bei 2500$ bis 5000$. Mitgeliefert wird auch ein SDK zur Anwendungsentwicklung für windows-basierte Systeme und diverse Programmiersprachen und Basistechnologien. Der iTable von PQ Labs ist eine gegenüber dem Surface leicht gewichtige Konstruktion eines LCD-Bildschirms auf einem fahrbaren Gestell mit integrierter Multi-Touch G² -Technologie. Höhe und Neigungswinkel des Bildschirms sind einstellbar, so dass das System sowohl horizontal als auch vertikal eingesetzt werden kann. Zum aktuellen Zeitpunkt wird der iTable nicht kommerziell vertrieben.

Abbildung 9: iTable von PQ Labs [PQLabs:iTable]

- „**Mixed Reality Table – Mr. T.**“ [TZI:MrT:2009]: „Mr. T.“ wurde an an der Universität Bremen zur Erforschung neuer Bedienkonzepte und Mehrbenutzer-Anwendungen im Bereich ubiquitärer Dienste, interaktiver Möbel und intelligenter Wohnräume entwickelt. Der technische Aufbau und die Funktionen des Tisches sind sehr ähnlich zum Surface, allerdings bietet „Mr. T.“ eine größere Darstellungsfläche (130cm Schirm-Diagonale) und eine Full-HD Auflösung, d.h. 1920x1080 Bildpunkte. Eine weitere innovative Erweiterung für die Zuordnung von Händen (Touchs) und Nutzern ist das so genannte „Seitenlinienorgan“, welches im Wesentlichen ein „Nähe“-Sensorsystem (Proximity) realisiert. Für die Vermarktung des Tisches suchen die Entwickler noch Partner in der Industrie.

Abbildung 10: Mixed Reality Table „Mr T“ des TZI, Uni Bremen [TZI:MrT:2009]

Die folgende Tabelle 1 zeigt die wesentlichen Merkmale der vorgestellten Systeme im Überblick. Wie sich gut erkennen lässt, existieren unterschiedliche Bauweisen, Display- und Sensor-Techniken in den betrachteten Systemen, aber alle weisen ein ähnliches Basissystem (Hardware-/Software, Konnektivität, Infrarot-Technologie betreffend) sowie Multi-Touch-Funktionalitäten auf. Daher sind auch Gesten und speziell dafür ausgelegte UI-Elemente zumindest prinzipiell bei allen Geräten vorgesehen und verfügbar.

Alle Systeme haben weiterhin gemein, dass die Verbreitung/Auflage eher gering ist, bzw. die Preise noch zu hoch sind für einen breiten Konsumenten-Markt, z. B. im Home-Entertainment-Bereich. Leider bleibt unklar, warum Philips „Entertaible“ vom Markt „verschwand“, bzw. dort nie richtig erschienen ist, zumal das System von den Fähigkeiten dem Surface sehr ähnlich ist und es speziell für Spiele-Anwendungen konzipiert wurde.

Prinzipielle Unterschiede gibt es bei der Größe der Bilddiagonalen und der dargestellten Auflösung, was aber auch von Bauform und Interaktionskonzept abhängig ist. Auffällig ist hier das Surface mit einer vergleichsweise geringen Auflösung und Bilddiagonalen, insbesondere gegenüber den nicht-spezialisierten Systemen.

Die Erkennung von Objekten, bzw. „Taged“-Items unterstützen neben dem Surface nur die spezialisierten Geräte „Entertaible“ und „Reactable“.

Die Benutzererkennung, bzw. die Zuordnung von Kontakten zu Nutzern bleibt insbesondere dem „Mitsubishi DiamondTouch“, und rudimentär auch „Mr. T.“ vorbehalten. Damit ist bei diesen Geräten die Anzahl gleichzeitiger Benutzer und deren Anordnung um den Tisch jedoch begrenzt.

System/ Eigenschaft	Microsoft Surface	Mitsubishi Diamond-Touch	Philips Entertaible	Multi-Touch-Wall	Reactable	PQ Labs iTable	Mr. T.
Bauform	Massiver Couchtisch	Touch-Sensor Auflage für Tisch	Auf-/ Einlage für Tisch/Möbel	Vertikal/ Wand	Stehtisch	Mobiles Schwenk-gestell	Stehtisch
Darstellung	Rück-projektion, im Tisch integriert	Front-projektion, extern (von oben)	LCD-Monitor	Rück-projektion	Rück-projektion, integriert	LCD- oder Plasma Monitor	Rückpro-jektion, integriert
Bilddiagonale	30“ (1024x768)	32“ und 42“	30“	94“ (Full-HD)	30“ rund	32“-52“ (Full-HD)	50“ (Full-HD)
Touch-Sensorik	Infrarot integriert (DI)	Kapazitative Fläche und Empfänger	Infrarot integriert	Infrarot	Infrarot integriert (DI)	Infrarot-sensorik im Rahmen	Infrarot integriert
Multi-Touch	Ja (52)	Ja (40)	Ja (>45)	Ja (>40)	Ja (>40)	Ja	Ja
Multi-User	Ja	Ja, bis zu 4	Ja	Ja, bis zu 4	Ja	Ja	Ja
Touch-/ Benutzer-Zuordnung	Nein	Ja, durch 4 Empfänger	Nein	Nein	Nein	Nein	Ja, durch Seiten-linien-organ
Objekt-erkennung	Ja, „Blobs“ und „Tags“ (52)	Nein, „Debris Tolerant“	Ja, Spielfiguren und Würfel mit „Tags“	Nein	Ja, durch „Tags“ (40)	Nein	Nein
Hardware-umgebung	Integrierter PC	Externer PC über USB	Integrierter PC	Externer PC	Integrierter PC; spezielle Audio-Hardware	Externer PC über USB	Integrierte r PC
Software-umgebung	Windows-Vista, .NET, Surface-SDK	Treiber und SDK für Windows 2000 / XP, Linux; C, C++, Java, ActiveX, Flash	Windows-XP basiert, API/SDK unklar	Windows-basiert; SDK	Windows, eigenes SDK und Framework: reacTIVision, TUIO	Windows Treiber/ SDK/ Apps für XP, Vista, 7; C++, C#, Flash	Windows-basiert
Konnektivität	LAN/ WLan/ WAN/ Bluetooth	Abhängig, PC-Standards	Entertaible-Vernetzung durchs Internet	PC-Standards	PC-Standards	Abhängig, PC-Standards	Abhängig, PC-Standards
Speziali-sierung/ bes. Konzept	-	-	Digitale TableTop Spiele	-	Audio-Synthesizer	-	-
Marktreife/ Verfügbarkeit	Noch exklusiv für Partner; breitere Verfügbarkeit 2010/11 geplant	Kommerziell zu erwerben bei Circle Twelve	War 2006/07 für kurze Zeit verfügbar	Kommer-ziell zu erwerben bei Perceptive Pixel	Kommer-ziell zu erwerben bei Reactable Systems	Kommer-ziell zu erwerben bei PQ-Labs	Prototyp-Zustand
Preis (Stand 2009)	Ca. 11000€ oder-13000€ (Workstation)	Ab 4000€ abhäng. v. Ausstattung PC u. Projektions-system	-	Ab 8000€	Basis-version ab 6000€	2500-5000 nur Aufsatz	-

Tabelle 1: Übersicht der Merkmale Surface-ähnlicher Systeme

2.3 Spiele

In den folgenden Abschnitten sollen die Grundlagen über Spiele und Spielkonzepte hinsichtlich ihrer kennzeichnenden Merkmale und Komponenten, ihres Aufbaus, des Gestaltungsprozesses und der wesentlichen technischen Aspekte vermittelt werden. Mit dem Begriff „Spiele“ sind im Rahmen dieser Arbeit insbesondere „Computer- oder Video-Spiele“ gemeint soweit es aus dem jeweiligen Kontext nicht anderweitig hervorgeht.

Nach [Adams:2006] ist ein Spiel wie folgt definiert:

„A game is a type of **play** activity, conducted in the context of a **pretended reality**, in which the patricipant(s) try to achieve at least one arbitrary, nontrivial **goal** by acting in accordance with **rules**“

Im Wesentlichen besteht ein Spiel also aus den vier Elementen:

- „**play**“: Das interaktive Verhalten der Spieler, deren Handeln den Verlauf eines Spiels beeinflusst.
- „**pretending**“: Eine Spielwelt oder Umgebung, in der sich der Spieler mental bewegen und eintauchen kann („immersion“).
- „**rules**“: Beschriebene oder unbeschriebene Regeln, die die Spielwelt und Interaktionsmöglichkeiten der Spieler abbilden.
- „**goal**“: Ein oder mehrere Ziele, das/die es innerhalb eines Spiels zu erreichen gilt. Streng genommen sind Ziele Teil der „rules“, welche (dann) explizit beschrieben und überprüfbar sein müssen.

Gegenüber traditionellen, analogen Spielen können Computer-Spiele die Spielwelt digital erweitern. Sie können eine interaktive Geschichte erzählen oder Spieler Dinge tun lassen, die in analogen Spielen nicht möglich wären. Triviale Aufgaben, wie z. B. das Austeilen virtueller Spielkarten, können auch von einem Computer übernommen werden. Weiterhin können Computer-Spiele für die Einhaltung von Spielregeln sorgen und dabei Regeln implementieren, die für die Spieler zunächst unbekannt sind und im Spielablauf erst (explorativ) erlernt werden müssen. Nicht zuletzt können Computer-Spiele die „Geschwindigkeit“ vorgeben, in der gespielt wird, was ein nicht unerheblicher Aspekt bei der Gestaltung und Entwicklung von Computer-Spielen ist. Ein möglicher Nachteil von Computerspielen gegenüber traditionellen Spielen ist, dass der Spieler Spielregeln, die ihm nicht gefallen, i.d.R. nicht verändern kann.

In erster Linie dienen Spiele der Unterhaltung und dem Spaß des Spielers, wobei dies sehr weit gefasst und tatsächlich von vielen Faktoren abhängig ist. Die wichtigsten Elemente, die den Unterhaltungswert von Spielen ausmachen, sind nach Adams insbesondere [Adams:2006]:

- „**Gameplay**“: Ergibt sich aus den besonderen Herausforderungen („Challenges“), die sich dem Spieler im Spielverlauf stellen und aus den Aktionen („Actions“), die ein Spieler in einer Spielwelt zur Bewältigung dieser Herausforderungen durchführen kann (vgl. dazu auch Abschnitt 2.3.4.2).
- „**Aesthetics**“ und „**Harmony**“: Die Gestaltung der Spielwelt, aber auch der Bedienelemente, sollte dem Spieler angenehm, verständlich und konsistent erscheinen.
- „**Storytelling**“: Ein Spiel kann eine interaktive Geschichte erzählen, die innerhalb der

Spielwelt stattfindet und so Dramatik und einen Spannungsbogen erzeugt.

- „**Risks and Rewards**“: Grundsätzlich sollte der Spieler für seine Aktionen und für bestandene Herausforderungen durch Punkte, Auszeichnungen u. Ä. belohnt werden. Besteht der Spieler eine Herausforderung besonders gut oder geht er bei der Lösung ein höheres Risiko ein, sollte er dafür auch zusätzlich belohnt werden.

- „**Novelty**“: Neuartige Konzepte in Visualisierung und/oder Bedienung von Spielen bewirken häufig nur ein kurz andauerndes Unterhaltungselement. Ein Mittel, um die Neugier der Spieler aufrecht zu erhalten ist, mit voranschreitendem Spielverlauf neue Elemente und Aktionen verfügbar zu machen.

- „**Learning**“: Der Begriff des Lernens bezieht sich vornehmlich auf das Entdecken der Spielwelt und ihrer Regeln. Das ist häufig kein einfacher Prozess, der immer und schon an für sich (Spiel-)Spaß bereitet, weshalb das zu Erlernende stets in einem unterhaltsamen Kontext stehen und vom Spieler beherrschbar bzw. zu bewältigen sein sollte.

- „**Creative and Expressive Play**“: Der Spieler möchte i. Allg. innerhalb der Spielwelt kreativ sein und sich ausdrücken/wieder finden können, die Spielwelt vielleicht sogar mitgestalten. Dies geschieht häufig durch Aktionen, die Anpassungen an Spielwelt, Regeln oder Avataren ermöglichen, aber nicht zwingend das Gameplay beeinflussen müssen (z. B. das Wechseln der Farbe eines virtuellen Spieler-Fahrzeugs, der Kleidung, u. Ä.).

- „**Immersion**“: Das mentale Eintauchen oder Versinken des Spielers in ein Spiel birgt für viele Spieler einen besonderen Reiz und kann auf verschiedene Arten geschehen. „Tactical immersion“ geschieht häufig bei schnell ablaufenden Actionspielen, in denen bisweilen innerhalb von Sekundenbruchteilen Entscheidungen getroffen und kleine, ähnlich wiederkehrende Aufgaben vom Spieler gelöst werden müssen (Beispiel „Tetris“). „Strategic immersion“ beschreibt die volle Konzentration des Spielers auf das Beobachten, Kalkulieren und Planen zur Optimierung von Spielzügen (Beispiel „Schach“). „Narrative immersion“ entsteht, wenn der Spieler sich als Teil einer erzählten Geschichte sieht und sich mit der Spielwelt und Charakteren identifizieren kann; es wird durch gutes „Storytelling“ erreicht.

- „**Socializing**“: Traditionelle Spiele sind zumeist für mehrere Spieler ausgelegt, wobei gleiche Ziele Spieler kooperieren oder konkurrieren lassen. Auch typische Computer- und Konsolen-Spiele unterstützen dies sowohl lokal durch Gruppenspiel-Modi und mit Hilfe mehrerer Controller (Eingabegeräte) als auch verteilt (nah und fern) durch Vernetzung. Zwischenmenschliche Interaktion und Wettbewerb können ein Spiel spannender und abwechslungsreicher machen.

2.3.1 Spiel-Entwicklungsprozess

Die Gestaltung und Umsetzung eines Spiels kann sehr komplex und aufwändig sein und viele Fähigkeiten und „Talente“ erfordern [Adams:2006]. Die Aufgaben dabei umfassen sehr viele konzeptionelle und kreative Tätigkeiten und werden je nach Größe und Umfang eines Spiels oft von mehreren Personen übernommen, die unterschiedliche Rollen einnehmen. Typische Rollen und Aufgaben bei der Spielgestaltung sind insbesondere:

- „**Game Designer**“: Er arbeitet die grundlegende Idee und das Spielkonzept aus, definiert Regeln und die Spielwelt sowie die Interaktionsmöglichkeiten des Spielers.

- „**User Interface Designer**“: Er entwickelt zusammen mit dem „Game Designer“ die Steuerung des Spiels, die über Eingabegeräte (Controller), grafische Bedienelemente oder interaktive Dialoge erfolgt.
- „**Level Designer**“: Er gestaltet die einzelnen Spielabschnitte, die ein Spieler während eines Spiels erfahren kann. Oft erfolgt auch direkt die visuelle oder sogar funktionale Umsetzung.
- „**Audio Designer**“: Er erzeugt Musik und Ton zur Implementation in das Spielgeschehen.
- „**Writer**“: Er erzeugt Texte wie Story oder Dialoge, erfindet Charaktere und arbeitet diese aus.
- „**Producer**“: Er produziert Materialien wie Texturen, 3D-Modelle, Icons, Animationen und Ähnliches, welche im Spiel Verwendung finden.
- „**Developer**“: Er setzt Spielmechanik und Regeln oft schon früh im Gestaltungsprozess in Form von Prototypen um. Der Entwickler implementiert letztlich die Funktionsweisen und die Abbildung der Spielwelt sowie des "Gameplay".
- „**Tester**“: Er prüft die Qualität des Spiels. Das Testen sollte schon früh in der Gestaltungsphase anhand von Prototypen beginnen und über den gesamten Zeitraum die Entwicklung begleiten.

Je nach Projektgröße bedarf es gegebenenfalls noch spezieller übergeordneter Rollen wie „Project Manager“, „Lead Designer“, „Art Director“ oder „Audio Director“, die besondere Verantwortungen tragen, Aufgaben verteilen bzw. abstimmen und den Entwicklungsfortschritt überwachen.

Am Anfang der Entwicklung eines Spiels steht eine Idee, welche sich zunächst in 2-3 Sätzen formulieren lassen sollte [Adams:2006]. Zentraler Aspekt dieser Formulierung sollte der Spieler sein und dabei beschreiben, welche Rolle er beim Spielen einnimmt.

Für den weiteren Gestaltungsprozess beschreibt [Adams:2006] drei Phasen („Design Stages“):

1. In der Konzeptphase („**Concept Stage**“) werden die Spielidee sowie verschiedene grundlegende Merkmale der „Spielwelt“ und des vorrangig stattfindenden „Gameplay“ festgelegt. Dabei wird insbesondere auch definiert, welche primären Herausforderungen an den Spieler gestellt werden und wie er sie in der Spielwelt bewältigen kann („Actions“). Weiterhin erfolgt die Einstufung des Spiels hinsichtlich Genre, Zielgruppe und -Plattform (vgl. Abschnitt 2.3.2). Daraus sollte ein grobes Konzeptpapier („High-level Concept“) hervorgehen, welches die Basis für die weitere Entwicklung und nächste Gestaltungsphase bildet.
2. Die weitere Ausarbeitung und Gestaltung („**Elaboration Stage**“) beginnt nach der Fixierung des zuvor entwickelten Konzeptpapiers. In dieser Phase werden insbesondere
    - die Spielwelt, Geschichte (Story) und Charaktere kreiert
    - die Spielregeln ausformuliert
    - das umfassende Gameplay definiert
    - User-Interface und Benutzer-Interaktion gestaltet
    - u.U. weitere Spielmodi mit eigenem User-Interface und/oder Gameplay erzeugt

- sowie die unterschiedlichen Spiellevel gestaltet.

Alle diese Tätigkeiten und wichtige Entscheidungen, die dabei getroffen werden, sollten ständig überprüft werden, um die gewünschte Qualität zu erreichen und "Player-centric-Design" zu unterstützen Daher ist diese Phase als iterativer Prozess zu verstehen, bei dem ständiges Testen von Prototypen den Gestaltungsprozess begleitet und das Endergebnis optimiert, bevor es zur vollen Produktionsreife kommt. Somit ist auch die bereits frühe Einbeziehung von Entwicklern in dieser Phase ratsam und u. U. sogar notwendig, bspw. zur Implementation von Prototypen zur Steuerung und Darstellung, zur Umsetzung Plattform-spezifischer Funktionen, der Bereitstellung von Tools, etc. Im Laufe dieser Phase, wenn die grundlegenden Entscheidungen getroffen sind, beginnt schließlich die Implementation und Produktion des eigentlichen Spiels und seiner Inhalte.

3. Die Abstimmungsphase („**Tuning Stage**“) ist die letzte Phase der Spielgestaltung, bei der keine neuen Funktionen zum Spiel mehr hinzukommen (sollten), aber noch Feinjustierungen an Spielmechanik oder Level-Gestaltung vorgenommen werden können. Zu diesem Zeitpunkt ist das Spiel so gut wie fertig gestellt und bekommt nur noch den letzten "Schliff", um Marktreife zu erreichen.

2.3.2 Spielkonzept

Das Konzept eines Spiels wird nach Adams [Adams:2006] in der ersten Phase („Concept-Stage“) des Gestaltungsprozesses erarbeitet. Es bildet die Grundlage für die nächste Spielgestaltungsphase und weitere Spielentwicklung und sollte zur Findung von Entscheidungen immer herangezogen werden. Ein Konzeptpapier sollte dabei die folgenden Punkte in Abhängigkeit von ihrer Relevanz für die Spielidee mehr oder weniger ausführlich beinhalten:

- ...eine Kurzbeschreibung der **Spielidee** selbst. Die Grundidee oder das Spielprinzip sollten in wenigen Sätzen verständlich formuliert werden können.
- Die **Rolle des Spielers** im Spiel sollte beschrieben werden sowie seine primären Aufgaben und besondere Herausforderungen (rudimentäres „**Gameplay**“, vgl. Abschnitt 2.3.4.2).
- Es sollte eine Zusammenfassung gegeben werden über den groben **Spielablauf** vom Anfang bis zum Ende, über Ideen für Level oder Missionen sowie ein Abriss über die Geschichte, wenn eine erzählt wird.
- Das oder die **Genre**, in das bzw. die das Spiel einzustufen ist (vgl. Abschnitt 2.3.5) sollte definiert werden. Die Wahl des Genres kann die konsistente Zielgruppen-orientierte Entwicklung, z. B. durch Genre-spezifische Usability-Tests, unterstützen [Pinelle:2008].
- Die **Zielgruppe**, für die das Spiel auszulegen ist, ist zu benennen. Schließlich kann die Spielkonzeption für besondere Personengruppen, z. B. bei Spielen für Menschen mit Behinderung, starke Auswirkungen auf die Interaktionsgestaltung, die Inhalte der Spielwelt und deren Darstellung haben. Neben der Differenzierung nach Altersgruppen (Kinder, Senioren) oder dem Geschlecht (Männer/Frauen, Jungen/Mädchen) ist die für ein adäquates Gameplay elementare Unterscheidung, die zwischen „Core“- und „Casual“-Spielern. Der „Core“-Spielertyp mag große Herausforderungen, kann sich über lange Zeit damit befassen und versucht dabei, ein möglichst optimales Ergebnis zu erzielen. Der „Casual“-Spielertyp bevorzugt kurzweilige, einfach zu erlernenden Spiele mit schnellem

Fortschritt und Erfolg.

- Die **Wettbewerbs-Modi**. sollten beschrieben sein; ob es sich um ein Ein-, Zwei- oder Mehrspielerspiel handelt, ob es kooperativ oder kompetitiv ausgeführt wird. Bei Mehrspieler-Szenarien ist zu definieren, wie die direkte oder indirekte Kommunikation zwischen den Spielern ermöglicht/gewährleistet wird. Bspw. werden typische PC-Spiele häufig vernetzt (entfernt oder lokal) von mehreren Spielern gespielt, wobei jeder Spieler eine eigene Ein- und Ausgabeeinheit besitzt. Lokales gleichzeitiges Spielen findet vornehmlich auf Konsolen statt, wobei jeder Spieler ein eigenes Eingabegerät besitzt, die Ausgabeeinheit aber gemeinsam benutzt wird. Lokales Gruppenspiel tritt traditionell bei PC und Konsolen-Spielen dann auf, wenn zu einem Zeitpunkt nur ein Spieler (stellvertretend für seine Gruppe) Aktionen ausüben kann.

- Die **Spezifizierung der Zielplattform** und deren besonderen Gegebenheiten im Hinblick auf die Interaktion mit dem Spieler ist vorzunehmen. Die Wahl der Plattform hat schließlich neben den technischen Abhängigkeiten gegebener Ein- und Ausgabegeräte auch Auswirkungen auf Vermarktung und Zielgruppe aufgrund der jeweiligen Verfügbarkeit, Verbreitung und des Einsatzortes.

- Dementsprechend gilt es auch, Strategien zu Lizenzierung, Vermarktung und Vertrieb auszuarbeiten.

Im Anschluss an diese konzeptionelle Phase findet die weitere Ausarbeitung und Definition der Spielwelt und ihrer Regeln statt. Elementare Entscheidungen zu Interaktion, Spielwelt und Gameplay sollten anhand von Prototypen diskutiert und getestet werden. Im Fokus steht dabei ausnahmslos der Spieler, den es zu erreichen gilt, sein Spielerlebnis und die Erfüllung seiner Erwartungen.

2.3.3 Spielwelt

Die grundlegenden Dimensionen der Spielwelt als „Ort“ des (Spiel-)Geschehens sind [Adams:2006]:

- **Realismus**: Diese Dimension beschreibt den Grad an Realitätsnähe der verschiedenen Aspekte/Elemente eines Spiels. Dies betrifft nicht nur die grafische Darstellung der Spielwelt und -elemente, sondern auch deren Detailgrad und Verhalten sowie die Interaktionsmöglichkeiten des Spielers. So sind z. B. Fahrzeug- oder Flugsimulationen oftmals repräsentativ für die realitätsnahe grafische Darstellung und Umsetzung des physikalischen Verhaltens des jeweiligen Vehikels; aber auch hier werden immer nur gewisse, nämlich spielrelevante, Aspekte der realen Welt aufgenommen (simuliert).. Selten wird der Spieler hierbei z. B. in ein Gebäude fahren, tanken oder Reifen wechseln müssen. Auch Ego-Shooter sind oft realitätsnah, wenn es um die grafische Darstellung der Spielwelt und Abbildung der physikalischen Auswirkungen verschiedener Schusswaffen und Geschosse geht. Dennoch kann der Spieler oftmals „tonnenweise“ Waffen und Munition mit sich tragen. Wie abstrakt oder realitätsnah die einzelnen Aspekte der Spielwelt abgebildet werden, sollte in erster Linie unter Berücksichtigung und zu Gunsten des Gameplay entschieden werden. Weiterhin haben solche Entscheidungen oftmals nicht unerhebliche Auswirkung auf den für die Spielgestaltung und -umsetzung benötigten Aufwand.

- **Physikalische Dimension**: beschreibt den Raum zur Darstellung der Spielwelt und seine

Grenzen. Ist die Spielwelt in irgendeiner Weise an die Realität angelehnt, ist außerdem der möglichst realitätsgetreue Maßstab der verschiedenen Elemente relevant.

- „**Environmental Dimension**“: Beschreibt all das, was in der Spielwelt enthalten ist und stattfindet, und wie dies abgebildet wird. Dies betrifft sowohl die physikalische Umgebung, in der das Spiel stattfindet („wo“), als auch den Detailgrad der Darstellung und Stil („wie“). Die Ausarbeitung der Spielumgebung trägt wesentlich zur Atmosphäre des Spielgeschehens bei und unterscheidet häufig gute von sehr guten Spielen. Hierbei ist auch der jeweilige kulturelle Kontext nicht zu vernachlässigen, den ein Spiel dem Spieler vermitteln möchte. Schließlich soll der Spieler in die Spielwelt eintauchen und in eine gewisse Rolle schlüpfen dürfen, z. B. in die eines Rennfahrers, Hotelmanagers, Raumschiffskommandanten, Gärtners, usw.

- **Zeitliche Dimension**: Beschreibt, was für eine Rolle Zeit in der Spielwelt spielt, wie schnell sie abläuft und ob der Spieler Einfluss darauf hat. Bei Runden-basierten Spielen ist Zeit meistens kein relevanter Faktor. Bei Echtzeitspielen vergeht die virtuelle Zeit im Spiel häufig schneller als die reale oder ist variabel, vielleicht sogar durch den Spieler selbst wählbar. Oft ist ein zeitliches Limit eine Herausforderung als Teil des Gameplay, manchmal wird die Zeit aber auch nur als gestaltendes Element der Spielwelt benutzt, z. B. zur Darstellung eines Tages- und Nachtrhythmus, und hat keine besonderen oder nur geringe Auswirkungen auf das Gameplay.

- **Emotionale Dimension**: Beschreibt, inwiefern das Spielgeschehen die Gefühle des Spielers beeinflussen soll. Neben der vermittelten Atmosphäre und den an den Spieler gestellten Herausforderungen, die oftmals bestimmte Gefühle erzeugen können, können die Gestaltung von Story und Charakteren sowie soziale Interaktion zu einem interessanteren oder aufregenderen Spiel führen.

- **Ethische Dimension**: Behandelt die ethischen und moralischen Aspekte eines Spiels und ist stark abhängig von dessen Einsatz und der zu erreichenden Zielgruppe (Alter, Geschlecht, Kultur). Kritische Themen wie Gewalt oder Pornographie sind prinzipiell schon aufgrund des Jugendschutzes mit Vorsicht zu handhaben, menschen- oder tierverachtende Inhalte sowie Inhalte, die gegen geltendes Recht verstoßen, haben in Spielen selbstverständlich nichts verloren.

2.3.4 Konzeptionelle Struktur und Komponenten eines Spiels

Abbildung 11 zeigt einen Entwurf der Spielen im Allgemeinen zugrunde liegenden Struktur bzw. ihrer wesentlichen Komponenten [Adams:2006], die im Rahmen der Spielentwicklung besondere Berücksichtigung finden müssen. Sie sollen den folgenden Abschnitten näher erläutert werden.

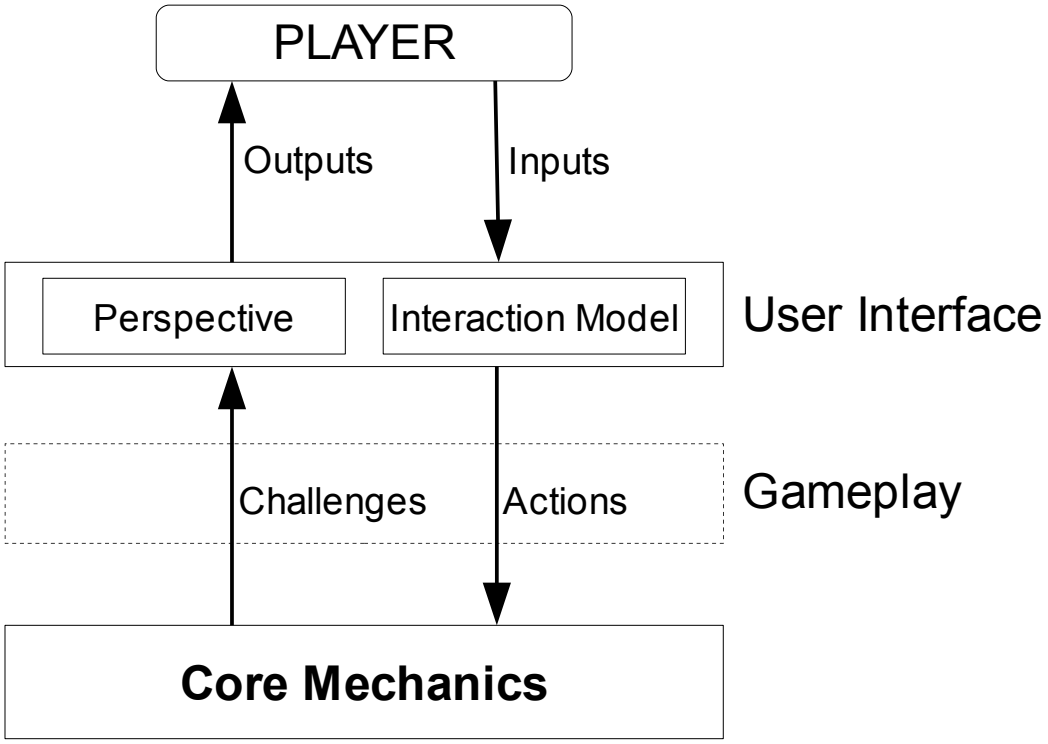

Abbildung 11: Konzeptionelle Struktur eines Spiels [Adams:2006]

2.3.4.1 „User Interface“

Die Aufgabe der „User Interface"-Schicht ist es, wie bei anderen Softwareanwendungen auch, die Interaktion des Benutzers mit der Anwendung und dem System zu ermöglichen. Spielekonzeptionell wird die Benutzerschnittstelle in "Perspective" und "Interaction Model" unterteilt [Adams:2006].

"**Perspective**" beschreibt dabei die Darstellungsperspektive, d.h. aus welcher Sicht die Spielwelt bzw. -umgebung für den Spieler projiziert wird. Typische Perspektiven zur visuellen Darstellung der Spielwelt sind im Besonderen:

- ...die zweidimensionale Perspektive (**2D**) mit der Ansicht von einer Seite, zumeist von oben („Top-Down“, aus der „Luftperspektive“ (s. Adams „aerial perspective). Aufgrund der relativ geringen Anforderungen an die Hardware und deren Rechenleistung ist diese Darstellungsperspektive vornehmlich bei älteren Spielen, Browser-Spielen (Flash) oder auch auf mobilen Endgeräten mit wenig Rechenleistung vorzufinden. Die Spielwelt kann dabei als Ganzes per „Single-Screen“ oder ausschnittsweise dargestellt werden, was häufig über so genanntes „Side-Scrolling“ oder „Top-Scrolling“, also horizontale oder vertikale Verschiebung des Ausschnitts, realisiert ist. Die 2D-Perspektive ist typisch für TableTop- und abstraktere Spiele wie PacMan, Tetris, etc.
- Bei der 2.5-dimensionalen Darstellungsperspektive (**2.5D**) wird die Spielwelt aus einem festen Winkel, meistens von schräg oben (isometrisch) gezeigt (Luftperspektive). Objekte werden also nicht perspektivisch verzerrt sondern immer gleich groß und von derselben Seite/Ansicht gezeichnet. Der Vorteil gegenüber 2D ist die bessere Vermittlung von Räumlichkeit bei vergleichsweise niedrigen Anforderungen an die Rechenleistung gegenüber 3D. In der technischen Realisierung wird häufig eine zusätzliche Z-Achse zur

Sortierung der Bildelemente in fixe Ebenen herangezogen. Visuelle Effekte wie Transparenz und Schatten verstärken den Eindruck von Raum und Tiefe weiter.

- Die dreidimensionale Darstellung (**3D**) der Spielwelt wird immer häufiger von Spielen genutzt und wird aufgrund der benötigten Rechenleistung von modernen Grafikkarten hardwaremäßig unterstützt. Hierzu muss ein Modell als Beschreibung der 3D-Welt und der darin enthaltenen Elemente vorliegen, welche - wie von einer virtuellen Kamera aus variabler Position und/oder variablem Sichtwinkel aufgenommen - projiziert werden („Rendering"-Vorgang). Auch visuelle Effekte wie Schatten und Lichtreflexionen können so „realistischer" berechnet und abgebildet werden. Je nach Ausrichtung und Steuerung der Kamera unterscheidet man üblicherweise die folgenden Perspektiven:

  - „**1st-Person**": Die Ansicht erfolgt im Gegensatz zur Luftperspektive durch die Augen eines Avataren, den der Spieler durch die Spielwelt bewegt. Der Spieler hat den Eindruck, sich in der Welt zu befinden. Typische Perspektive für so genannte „Ego-Shooter"-Spiele.

  - „**3rd-Person**": Ähnlich wie die „1st-Person"-Perspektive und im Unterschied zur Luftperspektive bewegt sich die Kamera mit einem Avataren und dessen Blickrichtung. Allerdings sieht hierbei der Spieler seinen Avataren zumeist von schräg oben oder hinten und kann gegebenenfalls Sichtwinkel und Zoom beeinflussen. Diese Darstellung ist typisch z. B. für Renn- oder Rollenspiele mit Schwerpunkt auf der Präsentation des Avataren.

  Spiele mit 1st-Person- und 3rd-Person-Perspektive eignen sich prinzipiell eher für *einen* (handelnden) Spieler pro Ein- und Ausgabegerät und werden in Mehrspieler-Setups vornehmlich vernetzt mit PCs oder Konsolen (auch „Split-Screen") gespielt.

  - „**Free-Roaming Camera**": Der Benutzer kann die virtuelle Kamera frei bewegen und so die Ansicht auf die Spielwelt bestimmen. Dies ist die favorisierte Realisierung für Ansichten großer Welten aus der „Luftperspektive" und typisch für (party-based) Rollen- oder Strategiespiele. Die Steuerung der Kamera muss dabei dem Spieler zunächst vermittelt werden.

  - „**Context-Sensitive Camera**": Eine intelligente Kamera verfolgt automatisch den Ort des aktuellen Geschehens in der Spielwelt und nimmt Szenen aus günstiger Entfernung und Richtung auf, schwenkt, etc. Das Verhalten der Kamera zum jeweiligen Kontext (Ort, Zeit, Aktion, Zustände) muss definiert und implementiert werden. Typische Einsatzgebiete sind z. B. Adventure-Games, aber auch Sport- und Vehikel-Simulationen.

  Die Beschreibung eines 3D-Modells der Spielwelt ist i. d. R. sehr aufwändig. Neben den 3D-Beschreibungen aller Elemente der Welt müssen Texturen erzeugt und zugewiesen sowie Materialbeschaffenheit von Oberflächen und Lichtquellen im Raum bestimmt werden (Lichtreflektion). Auch Sound und Tonquellen sind in modernen Spielen in der 3D-Welt platziert und werden in Abhängigkeit von Kontext, Perspektive und Position des Spielers wiedergegeben.

Das „**Interaction Model**" als weiterer Bestandteil des User Interface beschreibt die Beziehung zwischen den Eingaben des Benutzers und den daraus resultierenden Aktionen, die sich auf

Spielablauf und Spielwelt auswirken. Wie die konkrete Eingabe und Steuerung durch den Benutzer vorgenommen werden kann, ist stark abhängig von der Plattform und den damit zur Verfügung stehenden Eingabegeräten. Grundsätzlich eignen sich für bestimmte Interaktionsmodelle gewisse Darstellungsperspektiven besser/schlechter als andere. Typische Interaktionsmodelle sind i. Allg.:

- „**Avatar-based**“: Der Spieler steuert aus Sicht einer virtuellen Spielfigur und interagiert durch sie direkt mit der Spielwelt. Charakteristisch für dieses Modell ist, dass der Spieler nur Dinge in seiner näheren Umgebung beeinflussen kann und sich deshalb „physikalisch“ innerhalb der Spielwelt fortbewegen muss (Beispiel: Ego-Shooter, Autorennen). Diese Art der Navigation wird häufig durch „1st-Person“ oder „3rd-Person“ Perspektive realisiert.

- „**Multipresent**“ oder „**Omnipresent**“: Der Spieler kann gleichzeitig Aktionen auf verschiedene Elemente der Spielwelt ausüben und ist damit nicht ortsgebunden (Beispiel: Schach; typisch auch für viele andere traditionelle Brettspiele). Um dem Spieler eine möglichst große Übersicht über das Bestehen und die Anzahl möglicher Aktionen zu geben, ist i. d. R. eine “Luftperspektive“ zur Darstellung empfohlen, insbesondere 2D- (Top-Down) sowie 2.5D- Visualisierungen und „Single-Screen“ werden häufig verwendet, wenn die Darstellungsfläche es zulässt.

- „**Party-based**“: Der oder die Spieler steuern eine Gruppe von Spielfiguren, die in der Regel innerhalb der Spielwelt nahe zusammen bleiben (Beispiel: Rollenspiel). Um dem Spieler genug Übersicht zu bieten, wird ähnlich zum „multipresenten“ Modell eine „Luftperspektive“ zur Darstellung empfohlen. Die Navigation des Spielers durch die Spielwelt besteht üblicherweise au der Auswahl einer Spielfigur und dem Geben von Kommandos („Point and click“).

- „**Contestant-model**“: Beim „Kandidatenmodell“ beschränkt sich die „Navigation“/ Aktion des Spielers darauf, eine einfache Auswahl z. B. einer Antwort auf eine gestellte Frage zu treffen (Beispiel Quizspiele). Da das (räumliche) Ausmaß der Spielwelt hierbei zumeist überschaubar ist, ist eine aufwändige Navigation i.d.R. nicht notwendig. Die Darstellung erfolgt vornehmlich in Form einfacher Eingabeelemente in zweidimensionaler Perspektive und per „Single-Screen“.

- „**Desktop-model**“: Dieses Modell beschreibt die Interaktion wie an einem Arbeitsplatz oder Schreibtisch. Im Vordergrund stehen für den Spieler insbesondere Effizienz und Produktivität beim Spielen, daher erinnert eine solche typische Benutzeroberfläche eher an eine echte Desktop-Anwendung mit meist hohem Informationsgehalt. (Beispiel: Fußballmanager). Die Darstellungsperspektive beschränkt sich meist auf die zweidimensionale Abbildung von Listen, Statistiken, Charts, Icons, etc. per „Single-Screen“. Die Navigation des Spielers geschieht vornehmlich durch hierarchische Listen und Datenansichten, die im Detailgrad variieren und häufig quer verknüpft sind.

Weitere Aspekte der Gestaltung des User Interface/der Benutzerschnittstelle sind die Aufteilung des Bildschirms, Positionierung visueller Elemente, die Kreation von Dialog- und anderen visuellen Elementen wie Status- und Textanzeigen, Farben, Menüs und Knöpfen ("Buttons"), Schriftarten. Neben den visuellen Elementen stellen gegebenenfalls auch Ton und Musik Feedbackkanäle für den Benutzer dar, welche es zu gestalten gilt. Bei der Wahl von Perspektive und Interaktionsmodell müssen weiterhin die für die Plattform spezifischen Ein- und Ausgabemöglichkeiten, sowie deren Beschränkungen berücksichtigt werden.

2.3.4.2 „Gameplay“

„Gameplay“ beschreibt das, was der Spieler in der Spielwelt eigentlich tun soll, und wie er dies tun kann. Das „Gameplay“ soll den Spieler unterhalten, Spaß machen und Grund für ihn sein, dass Spiel zu spielen. Daher sind seine Konzeption und Umsetzung elementar. „Gameplay“ unterteilt sich in die Aspekte „Challenges“ und „Actions“ [Adams:2006]:

- „**Challenges**“ sind die Herausforderungen, die dem Spieler gestellt werden und die es für ihn zu bewältigen gilt. Die Herausforderungen können dabei ganz verschiedener Natur sein und sind dies häufig auch innerhalb eines Spiels zu unterschiedlichen Zeitpunkten. Eine Übersicht spieltypischer Herausforderungen und ihrer Klassifizierung zeigt Tabelle 2.

  Eine „Herausforderung“ muss nicht unbedingt direkt in der Spielmechanik („Core-Mechanics“) implementiert sein, sondern sie kann auch als Teil der freien Regelauslegung oder aus dem persönlichen Anspruchs des Spielers heraus entstehen. Weiterhin müssen nicht alle Herausforderungen dem Spieler direkt, d. h. vorab mitgeteilt werden. Vielmehr kann er die Regeln der Spielwelt durch spielen selbst erkunden, dabei schauen, wie sich die Spielwelt auf bestimmte Aktionen hin verhält/verändert und kann daraus schließlich lernen und sein Spiel ggf. verbessern. Um das Interesse des Spielers aufrecht zu erhalten, sollten ihm fortschreitend neue Herausforderungen gestellt werden, die zunehmend anspruchsvoller werden. Dies wird häufig durch eine hierarchische Anordnung von Herausforderungen in Form von Missionen, Levels und Spielrunden oder -abschnitten realisiert. In Mehrspieler-Situationen ist die Schwierigkeit von Herausforderungen oft abhängig von der Stärke des Mitspielers. Um den „individuellen“ Anforderungsgrad dennoch aufrecht zu erhalten wird in manchen Fällen der bessere Spieler z. B. mit einem Handicap belegt.

- „**Actions**“ sind die Aktionen, die ein Benutzer ausüben kann, um Einfluss auf die Spielwelt und den Spielablauf zu nehmen. Die jeweiligen Aktionen und wie sie konkret durchzuführen sind sollten dem Spieler bekannt oder für diesen leicht erlernbar sein. Wesentlich hierfür ist die (plattformabhängige) Gestaltung des User-Interface (insbesondere des Interaktionsmodells). Typische PC- und Konsolen-Spiele werden vornehmlich über Maus und Keyboard oder andere Controller gesteuert mit digitalen oder analogen, ein- oder zweidimensionalen Eingabemöglichkeiten, die im jeweiligen Kontext auf Aktionen im Spiel abgebildet werden. Grundlegende Aktionen dienen i. d. R. der Navigation durch die Spielwelt, der Auswahl von Spielelementen, Einheiten, etc. sowie der Auswahl von Kommandos. Jede Aktion, die ein Spieler im Laufe des Spiels durchführen kann, muss sowohl in den „Core-Mechanics“ (vgl. Abschnitt 2.3.4.3) als auch im Interaktionsmodell implementiert sein.

Challenge Type	Classic Example
Physical Coordination Challenges	
Speed and reaction time	*Tetris*
Accuracy or precision (steering, shooting)	*Need for Speed*
Timing and rhythm	*Dance Dance Revolution*
Learning combination moves	*Street Fighter*
Formal Logic Challenges	
Deduction and decoding	*Mastermind*
Pattern Recognition Challenges	
Static patterns	*Heaven and Earth*, choosing an optimal layout for cards
Patterns of movement and change	*Sonic the Hedgehog*, behavior patterns of enemies
Time Pressure	
Beating the clock	*Frogger*
Achieving something before someone else	Indycar *Racing*
Memory and Knowledge Challenges	
Trivia	*You Don't Know Jack*
Recollection of objects or patterns	*Concentration*
Exploration Challenges	
Identifying spatial relationships	*Descent*, navigating in three dimensions
Finding keys (unlocking any space)	*Ultima*
Finding hidden passages	*Doom*
Mazes and illogical spaces	*Zork*
Conflict	
Strategy, tactics, and logistics	*Warcraft*, commanding armies
Survival	*Pac-Man*, avoiding being caught
Reduction of enemy forces	*Half-Life*, killing aliens
Defending vulnerable items or units	*ICO*, looking after a little girl who can't fight
Stealth	*Thief: The Dark Project*, avoiding being seen
Economic Challenges	
Accumulating resources or points (growth)	*Civilization*
Establishing efficient production systems	*The Settlers*
Achieving balance or stability in a system	*SimEarth*
Caring for living things	*Creatures*
Conceptual Reasoning Challenges	
Sifting clues from red herrings	*Law and Order*, solving crimes
Detecting hidden meanings	*Planescape: Torment*, understanding characters' motivations from vague hints
Understanding social relationships	*Façade*, reconciling a quarreling couple
Lateral thinking	*The Incredible Machine*, building a machine from limited parts
Creation/ Construction Challenges	
Aesthetic success (beauty or elegance)	*The Sims*, assembling a photo album
Construction with a functional goal	*Mind Rover*, designing a fighting robot

Tabelle 2: Typische "Challenges" des "Gameplay" [Adams:2006]

Im Kontext des Gameplay wird öfter auch von „Gameplay Mode“ gesprochen. Der Gameplay-Modus beschreibt dabei unterschiedliche „Challenges“ und „Actions“ zu verschiedenen Zeiten im Spiel, wobei häufig auch ein jeweils eigens gestaltetes User-Interface zur Verfügung stehen kann. Beispiele dafür sind Sport- oder Rollenspiele, die z. B. in Taktik- und Action-Modus unterteilt sind.

2.3.4.3 „Core Mechanics“

Das Herz eines Spiels bilden die „Core Mechanics“ (Kernmechanismen). Hier findet die Implementation der genauen Regeln für den Spielablauf und die Verwaltung des Zustands der Spielwelt in Form mathematischer Abbildungen und Algorithmen statt [Adams:2006]. Die Kernmechanismen erzeugen die Herausforderungen für den Spieler und überprüfen Sieg- oder Verlust-Bedingungen, bzw. ob und wann diese eintreten. Weiterhin definieren sie konkret, wie sich Aktionen des Spielers im Spielgeschehen äußern und auf die Spielwelt auswirken.

Umfang und Aufwand der Implementation der Kernmechanismen hängen stark von der Gestaltung der Spielwelt und der darin enthaltenen Elemente ab.

Befinden sind innerhalb der Spielwelt rechnergesteuerte Objekte oder Charaktere, kommen häufig Routinen aus dem Bereich der künstlichen Intelligenz (KI) zum Einsatz, welche auch innerhalb der „Core Mechanics“ implementiert sind [Adams:2006]. KI in Spielen wird dabei oftmals als schwache („weak“) KI bezeichnet, gegenüber starker KI, bei der vornehmlich menschliche Intelligenz nachgeahmt/simuliert wird [Bourg:2004]. Typische Anwendungsfälle für KI in Spielen allgemein sind [Bourg:2004]:

- Verfolgen, Fliehen und Ausweichen von Hindernissen („Chasing“, „Evading“, „Obstacle-Avoidance“); dabei auch oft die Abbildung von Bewegungsmustern
- Schwarmverhalten („Flocking“) (z. B. „Potential Function Based Movement“)
- Das Finden von Wegpfaden („Pathfinding“) durch die Spielwelt zu einem bestimmten Ziel (z. B. A*, IDA*, Fringe-Search)
- Die Erkennung von Bewegungsmustern („Pattern Recognition“) durch Eingabegeräte (auch Gesten)
- Agententechnologie; autonome (mobile) Einheiten mit eigenem Wissensstand über die Umwelt, mit einer Strategie und häufig auch Kommunikationsfähigkeiten
- Regelbasierte KI zur unschärferen Findung von Entscheidung bei Abhängigkeit von mehreren Zuständen, z. B. „Finite State Machines“ oder „Fuzzy Logic“
- Lernende oder trainierende Computergegner, die im Spielverlauf oder über größere Zeiträume besser werden (z. B. durch Neuronale Netze oder Genetische Algorithmen)

Für größere Spielwelten mit vielen rechnergesteuerten Elementen und unterschiedlichen Verhaltensweisen, wie z. B. bei typischen Rollenspielen, bietet es sich an, zumindest eine rudimentäre „Scripting Engine“ zu entwickeln, die die Elemente und ihr Verhalten in vereinfachter Form individuell anpassbar macht. Die Spezialisierung von Charakteren kann damit an anderer Stelle (nicht in den Core-Mechanics) in Form von „Scripts“ vorgenommen werden und muss nicht mehr unbedingt von einem Entwickler getan werden.

Für stark repräsentative Spielwelten, also an die Realität angelehnte Umgebungen, wie bei Simulationen, kommen Gesetze der Natur und Physik (abstrahiert) zur Anwendung. Typische

Anwendungsfälle [Bourg:2001] finden sich insbesondere bei der Simulation von

- Sportarten, insbesondere Ball- und Schlagspielen
- Fahrzeug- und Rennsimulationen im Wasser, zu Lande und in der Luft (Autos, Motorräder, Flugzeuge, Schiffe, Segelbote, etc.)
- Waffen, Raketen und Projektilen und deren Wurf-, Flug- und Schussbahn
- Flüssigkeiten, Explosionen, Schwärmen, etc.
- Gesetzen der Mechanik und Kinematik zu Massenträgheit („Inertia“), Kräfte, Masse und Gewicht, Reibung, Energie und Arbeit sowie zu Moment, Impuls, bei Kollisionen von Körpern und dergleichen mehr.

Bei aufwändigen Simulation mit vielen Einflussfaktoren und Objekten, die physikalische Verhaltensweisen zeigen, kommen häufig so genannte „Physik-Engines“ zum Einsatz, die die Gesetzmäßigkeiten anhand der jeweiligen physikalischen Attribute der Objekte abbilden.

2.3.5 Genre eines Spiels

Das Genre ist eine gängige Klassifikation bei Spielen und beschreibt im Wesentlichen, in welche Gruppe die vorrangigen Herausforderungen des Spiels einzustufen sind [Adams:2006]. Erfolgt eine Mischung verschiedenartiger Herausforderungen innerhalb eines Spiel, bedient es mehrere Genres gleichzeitig. Gameplay und Genre sind voneinander abhängig, und haben letztlich auch Auswirkung auf das User Interface. Klassische Genre für Spiele sind:

- „**Action-Spiele**“: Fordern physische Leistung und haben oft Elemente von „Puzzles“ (Geduldspiele, Rätsel), Rennen und anderen „Konflikten“, meist zwischen wenigen Parteien. Häufig sind auch einfache ökonomische Herausforderungen gestellt, wie das Einsammeln bestimmter Dinge.
- **Strategiespiele**: Beinhalten vornehmlich strategische, taktische und manchmal auch logistische Herausforderungen. Um das Spiel interessanter zu gestalten, werden oft auch ökonomische oder kundschaftende (explorative) Herausforderungen gestellt.
- **Rollenspiele**: Fordern häufig taktische, logistische und „explorative“ Leistungen und weisen Elemente von „Puzzles“ auf. Manchmal spielen auch ökonomische Herausforderungen eine Rolle.
- **Sport- und Vehikel-Simulationen**: Erfordern meistens physische und taktische Leistungen, aber so gut wie nie ökonomische, konzeptionelle oder „explorative“.
- **Aufbau- und Managementspiele**: Beanspruchen vornehmlich ökonomische und konzeptionelle Leistungen, gelegentlich sind auch Konfliktbewältigung oder kundschaftende Tätigkeiten gefordert.
- **„Adventure-Games“**: Fordern hauptsächlich das Kundschaften sowie das Lösen von „Puzzles“. Manchmal sind auch konzeptionelle Leistung gefragt, sehr selten physische.
- **„Puzzle-“ und Logik-Spiele**: Stellen fast exklusiv logische oder konzeptionelle Herausforderungen und weisen gelegentlich Elemente aus Action-Spielen auf wie „Zeitdruck“.

2.4 Existierende Spiele für das Surface

In dem folgenden Abschnitt sollen eine Übersicht über bereits vorhandene Spielanwendungen für das Surface gegeben und aus einem spielkonzeptionellen Blickwinkel heraus ihre kennzeichnenden Merkmale beschrieben werden.

Innerhalb der „Surface-Community“ ist das „Microsoft Surface Games Pack“ [MS:SGP] erhältlich, welches die folgenden 4 Spiele-Anwendungen enthält:

- „**Schach (Chess)**“ und „**Dame (Checkers)**“ sind, wie die Namen schon sagen, die Umsetzungen der bekannten Brettspiele für zwei Spieler. Hier kann wahlweise gegen einen zweiten menschlichen Gegner oder gegen einen Computergegner gespielt werden, es ist aber auch möglich, zwei Computergegner verschiedener Schwierigkeitsgrade gegeneinander spielen zu lassen.

  Auf der Anzeige wird vornehmlich das Spielbrett von oben visualisiert, welches durch bekannte Gesten (Stretch, Shrink, Spin) in der Größe und Ausrichtung frei einstellbar ist (360°-View). Auf dem Spielbrett sind die virtuellen Spielsteine, die durch Push&Slide-Geste bewegt werden können, angeordnet. Bei Auswahl eines Spielsteins (Push) werden die regulär möglichen Zielfelder für den Zug visuell hervorgehoben. Eine Besonderheit bei beiden Spielen ist, dass die Kontrolle über das Spiel-Regelwerk ausgeschaltet werden kann und dadurch auch die auf dem Brett befindlichen Spielfiguren frei beweglich werden. Sie zeigen dann physikalisches Verhalten bei Bewegung, wie Gewicht, Reibung und können mit mit anderen Spielelementen oder Gegenständen, wie Gläsern, die auf dem Tisch platziert sind, kollidieren. In diesem Spielmodus ist es auch möglich eine „Flick“-Geste auszuführen, die ein „Schleudern“ der Spielsteine bewirkt. Dieses Verhalten zeigen im regulären Spielmodus auch eroberte Spielsteine, die, neben dem Spielbrett liegend, noch zum „Herumspielen“ während der Partie verfügbar bleiben. Diese Funktionen machen die Bemühungen um die natürliche und spielerische Erfahrung, die der Nutzer im Umgang mit dem Surface haben soll, deutlich.

Abbildung 12: Microsoft Surface-Schach [MS:SGP]

- „**Tiles**“ ist ein Multiplayer-Spiel für bis zu 7 gleichzeitige Spieler. Ziel ist es, verschiedenfarbige Spielsteine („Tiles“) vom eigenen Bereich des Spielfelds weg zu befördern. Es müssen dazu mindestens 3 "Tiles" gleicher Farbe in einer vertikalen oder horizontalen Reihe angeordnet werden, damit sie sich auflösen. So entfernte "Tiles" erscheinen in den Spielbereichen der anderen Spieler, was den kompetitiven Charakter

des Spiels erhöht. Der Spieler, dessen Spielbereich vollständig mit "Tiles" gefüllt ist, verliert, bzw. scheidet aus. Die "Tiles" können durch Touch, bzw. Multi-Touch im eigenen Spielbereich verschoben werden. Das runde Spielfeld ist dazu in Abhängigkeit von der Zahl der teilnehmenden Spieler in bis zu 7 Segmente eingeteilt (360°-View). Zusätzlich sind die Spieler-Segmente um den Spielfeld-Mittelpunkt drehbar, was zusätzliche Freiheiten für die Anordnung der Spieler mit sich bringt. Eine weitere Besonderheit ist, dass neue Spieler jederzeit einem laufenden Spiel beitreten können.

Abbildung 13: Microsoft „Tiles“ [MS:Tiles]

- „**Ribbons**“ ist kein wirkliches Spiel, es ist vielmehr eine unterhaltende Anwendung, an der jederzeit mehrere Benutzer gleichzeitig teilhaben können. Dabei „malen“ die Benutzer mit den Fingern oder einem Pinsel, also durch Touch und Multi-Touch, so genannte „Ribbons“. Ein solches „Ribbon“ entsteht in der Form der Bewegung des Kontakts, sozusagen in einem Zug. Sobald das „Ribbon“ gezeichnet ist, bewegt es sich wiederholend in dem Muster und der Geschwindigkeit der (Eingabe-)Zeichnung über den Bildschirm. Des Weiteren können platzierte Objekte oder andere Kontakte „Ribbons“ anziehen oder abstoßen. Es können theoretisch beliebig viele „Ribbons“ an beliebigen Stellen des Tisches gleichzeitig erzeugt/gezeichnet werden. Die einfache Anwendung soll auf unaufdringliche Weise zur Interaktion mit dem Surface einladen.

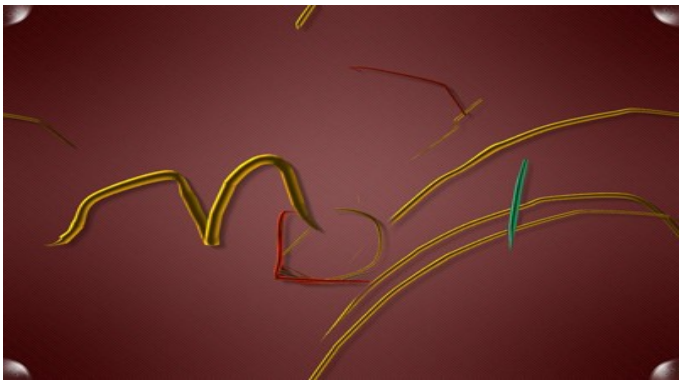

Abbildung 14: Microsoft „Ribbons“ [MS:Ribbons]

Innerhalb des Surface-SDK befinden sich zwei Beispielanwendungen, die durch WPF realisiert sind und deren Quellcode frei zugänglich ist:

- „**Paddle Ball**“ [MS:PaddleBall] ist ein Pong-Klon für bis zu vier simultane Spieler. Punkte bekommt der Spieler, der zuletzt den virtuellen Ball mit dem ebenso virtuellen Schläger

berührt hat, bevor der Ball ins Aus geht. Die Schläger werden durch einfache Gesten (Push&Slide) gesteuert und können jeweils auf einer Achse verschoben werden. Der Spieler, der zuerst die festgelegte Höchstpunktzahl erreicht, gewinnt. Das Anwendungsbeispiel soll den Umgang mit den grundlegenden Software-Komponenten „SurfaceUserControl“ des Surface-SDK veranschaulichen, und wie Kontaktinformationen und Ereignisse empfangen und weiterverarbeitet werden.

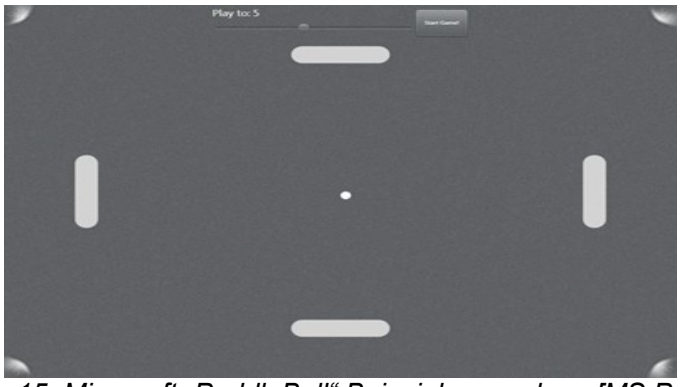

Abbildung 15: Microsoft „PaddleBall“ Beispielanwendung [MS:PaddleBall]

- „**ScatterPuzzle**“ [MS:ScatterPuzzle] ist ein kleines Puzzle-Spiel, wie man es grundsätzlich aus dem echten Leben kennt, virtuell dargestellt. Nach Auswahl eines Motivs, was ein Bild aber auch ein Video sein kann, sind von dem Benutzer die zufällig angeordneten Puzzleteile durch Drehen und Schieben korrekt zusammenzuführen. Der Schwierigkeitsgrad ist einstellbar und äußert sich in der Änderung von Größe und Anzahl der Puzzleteile. Im Wesentlichen veranschaulicht dieses Beispiel die Verwendung der ScatterView- und SurfaceListBox-Komponenten des Surface-SDK und zeigt die Multi-Touch- und gestenbasierte Steuerung, die diese Komponenten bereits implementieren.

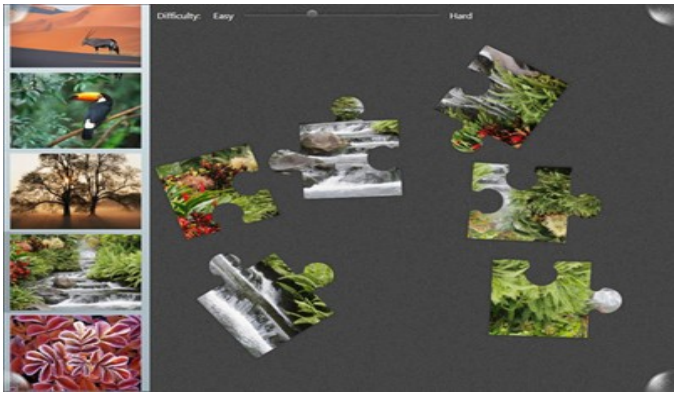

Abbildung 16: Microsoft „ScatterPuzzle“ Beispielanwendung [MS:ScatterPuzzle]

Diversen Foren- und Video-Beiträgen [MS:SurfaceCommunity] zufolge existieren einige nicht veröffentlichte Spiele von Microsoft für das Surface:

- „**FireFly**“ [MS:FireFly] heißt Microsofts „erstes“ Spiel auf dem Surface und wurde bereits im Februar 2008 vorgestellt. Es wurde vornehmlich für interne Tests des Surface-SDK und von XNA verwendet. Ziel des Spiels ist es, möglichst viele virtuelle Glühwürmchen („Fireflies“), die auf dem Bildschirm umher schweben, in einen virtuellen Behälter zu manövrieren. Das geschieht über Multi-Touch, wobei mehrere Finger einer oder beider

Hände eingesetzt werden können. Die Glühwürmchen fliehen vor den Kontaktpunkten und weichen aus, so dass man durch geschickte Anordnung der Finger die Flugrichtung lenken kann. Bis zu vier Spieler sammeln gleichzeitig gegeneinander Glühwürmchen unterschiedlicher Farben in ihren eigenen Behälter. Der Spieler, der zuerst die vorgegebene Anzahl von Glühwürmchen aller Farben gesammelt hat, gewinnt die Runde. Die begrenzte Anzahl an Glühwürmchen macht das Spiel kompetitiver, darüber hinaus werden Glühwürmchen zerstört/entfernt, wenn sie direkt berührt werden. So wäre eine mögliche Spiel-Strategie, Glühwürmchen in den Farben, die der Gegenspieler noch benötigt, zu zerstören. Das kooperative Spiel in Teams (Spieler arbeiten bspw. gemeinsam in einen Behälter o. ä.) ist prinzipiell möglich, obliegt jedoch der Regelprüfung durch die Spieler, da diese aufgrund fehlender Kontakt-Benutzer-Zuordnung i. d. R. nicht durch die Plattform gewährleistet wird.

Abbildung 17: Microsoft „FireFly“ [MS:FireFly]

- „**Ghosts**“ ist ein Logik-/Puzzle-Spiel, bei dem die durch Touch bedienbaren Spielelemente („Ghosts“) so angeordnet werden müssen, dass sich zwischen den Ghosts befindliche Verbindungslinien nicht mehr überschneiden. Dafür steht eine begrenzte Zahl an Zügen (Bewegungsoperationen) zur Verfügung. Leider ist über das Spiel nicht mehr bekannt.

Abbildung 18: Microsoft „Ghosts“ [MS:Ghosts]

Weiterhin existieren Entwicklungen im Spiele- und Entertainment-Bereich von Microsoft-Partnern wie Razorfish oder Harrah's Entertainment:

- „**Surface Table-Toss**“ [TableToss] von „Razorfish Emerging Experiences“ wurde im Februar 2009 vorgestellt. Es ist ein Spiel, bei dem echte „Bean-Bags“ (Jonglierbälle), aus

gewisser Distanz geworfen, möglichst in der Mitte der Tischoberfläche platziert werden müssen. Auf der Oberfläche werden dazu eine Zielscheibe mit konzentrischen Ringen ausgegeben und die erzielten Punkte abhängig vom Wurfergebnis angezeigt. Bei einem Wurf wird gemessen, wo das „Bean-Bag“ letztlich liegen bleibt. Bei mehreren Spielern wird abwechselnd geworfen, die Reihenfolge wird durch Anzeige der Farbe des jeweiligen „Bean-Bags“ visualisiert. Diese sind mit „ID-Tags“ ausgestattet, so dass das System sie unterscheiden, der richtigen (Farb-)Gruppe zuordnen und ihre Lage/Orientierung bestimmen kann.

Abbildung 19: „Surface Table-Toss“ von Razorfish [TableToss]

- „**DaVinci Physics Illustrator**“ [DaVinci] von „Razorfish Emerging Experiences“ ist kein wirkliches Spiel im eigentlichen Sinne, da es kein definiertes Spielziel gibt. Es ist vielmehr die interaktive Abbildung einer physikalischen Simulation in einer zweidimensionalen Umgebung, bei der der Benutzer virtuelle Objekte mit den Fingern zeichnen und manipulieren kann. Diese Objekte haben durch Simulation innerhalb der virtuellen Umgebung physikalische Eigenschaften wie z. B. Masse und können kollidieren oder sich gegenseitig anziehen und abstoßen. Die Anwendung ist dafür vorgesehen, in Lehr-/Unterrichtssituationen reales physikalisches Verhalten wie z. B. die Newtonschen Gesetze zu veranschaulichen.

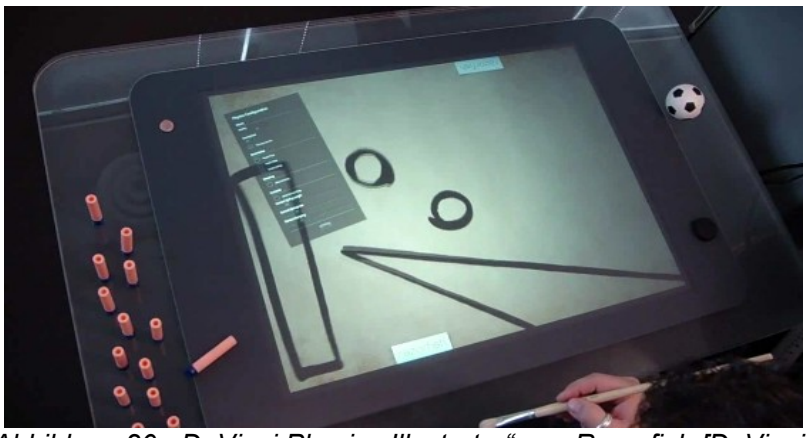

Abbildung 20: „DaVinci Physics Illustrator“ von Razorfish [DaVinci]

- „**Surface DJ**“ [Vectorform] dient dazu, ähnlich dem Reactable, spielerisch und auf einfache Art live elektronische Musik zu erzeugen. Im Gegensatz zu Reactable verzichtet „Surface DJ“ auf den Einsatz physikalischer Objekte zur Steuerung und setzt auf virtuelle Objekte,

die durch Direct-Touch auf der Oberfläche platziert oder manipuliert werden können. Diese Objekte repräsentieren z. B. Audio-Samples und Loops oder Filtereffekte die man in Echtzeit belegen und verändern kann. Auf dem Microsoft Surface kann die Anwendung von mehreren Benutzern gleichzeitig gesteuert werden. Sie ist ebenfalls für Apples iPhone im iTunes-Appstore erhältlich.

Abbildung 21: „Surface DJ 1.0“ von Vectorform [Vectorform]

- Harrah's Entertainment [Harrahs] hat als Partner von Microsoft im Juni 2008 eine Reihe angepasster Surface-Applikationen für den Entertainmentbereich, speziell für Bars und Kasinos, realisiert und installiert. Neben Tischübergreifenden Flirt-, Chat- und Cocktail-Mix-Spielen bietet Harrah's Entertainment einige Mini-Spiele. Darunter eine einfache Bowling-Simulation ("High Roller"), einen Mehrspieler-Flipper ("Dissed Pinball") und ein musikalisches Gedächtnisspiel im Stil von Simon ("Last Call"), bei dem die Spieler kooperativ spielen. Highscore-Listen sollen den Wettkampf unter den Spielern fördern.
- Das „**Surface Academy 2009 Toolkit**“ [SurfaceAcademy:2009] ist ein OpenSource-Projekt unter der „Microsoft Public License (Ms-PL)“. Es beinhaltet unter anderem visuelle Komponenten zur Implementation von Kartenspielen, darunter insbesondere die Spielkarten eines anglo-amerikanischen Blatts (52 Karten). Eine Besonderheit der Spielkarten sind die speziell implementierten Gesten, z. B. um Spielkarten umzudrehen oder bei verdeckten Karten nur eine Ecke davon hinter vorgehaltener Hand aufzurollen/umzuklappen. Weiterhin enthält es einige Komponenten zur Behandlung von Stapeln oder Sets von Karten und ein „Game-Board“, auf dem die Interaktion mit den Karten stattfindet. Enthalten ist außerdem auch ein „Card Game Starter Kit“, das als Basis für eigene Kartenspiel-Anwendungen dienen kann, sowie eine prototypische Implementation des Kartenspiels „Blackjack“. Die Realisierung der Komponenten erfolgte vollständig mit Microsoft WPF und Surface-SDK.

In der weiter unten aufgeführten Tabelle 3 sind die spielkonzeptionellen Merkmale der genannten Spiele, die bereits für das Surface entwickelt wurden, noch einmal in einer Übersicht dargestellt. Eine Zusammenfassung der wesentlichen Merkmale erfolgt im nächsten Kapitel (vgl. Abschnitt 3.2).

Spiel/Aspekt des Spiel-konzepts	Schach/ Chess	Tiles	Ribbons	Paddle Ball	Scatter Puzzle	FireFly	Table-Toss
Zielgruppe	Nicht weiter spezifiziert, i. d. R. für die meisten Altersgruppen geeignet. Anspruch bedient eher Casual-Spieler-Typen.						
Genre	Strategie-/ Logik-Spiel; TableTop	Puzzle-Spiel, Actionspiel	Actionspiel?	Actionspiel	Puzzle-Spiel; TableTop	Action-/ Sammel-Spiel	Actionspiel
Multi-Player/-User	0-2 Spieler/ PC-Gegner, abwechselnd	Bis zu 7 simultane Spieler	beliebig	Bis zu 4 simultane Spieler	Ab einem Spieler, simultan	Bis zu 4 Spieler, simultan	1-2 Spieler oder Teams, abwechselnd
Wettbewerbs-modus	Kompetitiv	Kompetitiv	Ohne Wettbewerb	Kompetitiv	Einzelspieler oder simultanes Gruppenspiel	Kompetitiv	Kompetitiv
Darstellungs-perspektive	2.5D Top-Down-Perspektive; 360°-View; Spielfeld drehbar	Abstrakte 2D Darstellung, Single-Screen; Rundes Spielfeld, in 7 drehbare Segmente unterteilt (360°-View)	Abstrakte 2.5D Darstellung, Single-Screen	Abstrakte 2D Darstellung, Single-Screen	2.5D Top-Down Perspektive, Single-Screen, Puzzle und Teile drehbar (360°-View)	2.5/3D Top-Down-Perspektive, Single-Screen	2D Top-Down-Perspektive, Single-Screen
Interaktions-modell	Multi-/ Omnipresent	Multi-/ Omnipresent	Multi-/ Omnipresent	Multi-/ Omnipresent	Multi-/ Omnipresent	Multi-/ Omnipresent	Multipresent/ Kandidaten-modell
Challenges/ Gameplay	Die bei Schach/Dame üblichen	Logik, Geschwindig-keit; schneller Kombinatio-nen bilden als die Gegner	Kreatives Spiel	Geschick, Reaktions-vermögen, Geschwindig-keit; mehr Bälle treffen als die Gegner	Puzzle lösen, ohne Zeitdruck	Geschwindig keit und Geschick; schneller Sammeln als die Gegner	Präzision, mehr Punkte als der Gegner erreichen
Actions/ Gameplay	Virtuelle Spielsteine per Push &Slide-Geste steuerbar	Virtuelle Spielsteine werden per (Multi-)Touch im eigenen Spielfeld-segment verschoben	Zeichnen von Linien und Mustern durch Multi-Touch;	Steuerung der Schläger auf einer Achse über Push &Slide-Geste	Touch, bzw. Multi-Touch; verschiedene Gesten, um Puzzelteile zu bewegen und zu drehen	Steuerung der Flug-richtung der „Fireflies“, die Kontakt-punkten aus-weichen, per Multi-Touch	Durch gezieltes Platzieren des mit ID-Tag ausge-statteten Bean-Bags auf der Oberfläche
Besonderheiten	Regelwerk ausschaltbar, Spielsteine zeigen dann physikalisches Verhalten (inertia)	Beitritt in ein laufendes Spiel jederzeit möglich	„Mal- und Zeichenspiel “ ohne konkretes Ziel. Ribbons weichen Kontakten auf der Oberfläche aus	Demonstration einer WPF-Implementation	Videos sind „puzzlebar“	MS-interner technischer Prototyp für Tests des Surface-SDK unter XNA	Reale, mit ID-Tags ausgestattete „Spielsteine“

Tabelle 3: Übersicht der Spiele für das Surface

3 Zusammenfassung und Zwischenfazit

Im folgenden Abschnitt erfolgt eine Zusammenfassung des vorangegangenen Kapitels, in welchem die konzeptionellen und technischen Besonderheiten des Surface (in Abgrenzung zu anderen ähnlichen Systemen) beschrieben und analysiert, sowie die wesentlichen Aspekte der Konzeption von Spielen vorgestellt wurden. Die folgenden Abschnitte bilden damit die verdichtete Ausgangsbasis für die Entwicklung und Umsetzung einer eigenen Spielidee für das bzw. auf dem Surface.

3.1 Markt

Aktuell ist die Surface-Hardware noch sehr teuer in der Anschaffung und die Verfügbarkeit exklusiv, da vornehmlich für Entwickler, bzw. Microsoft Partner bestimmt. Aber die Bemühungen, in Zukunft auch einen breiten kommerziellen Markt zu bedienen, sind erkennbar und die Weiterentwicklung und Verbreitung günstigerer Anzeige- und Sensor-Technologien absehbar. Generell nimmt insbesondere die Touch-Technologie für Desktop-PCs und in letzter Zeit vermehrt auch für Laptop-Bildschirme Einzug auf dem breiteren Konsumenten-Markt. Sowohl in dem neuen Betriebssystem von Apple als auch in Windows 7 von Microsoft sind bereits grundsätzliche Touch-Funktionalitäten ähnlich dem Surface implementiert. Wenige Konkurrenzprodukte „kleinerer“ Anbieter im Bereich der (horizontalen) Surface-Computing-Systeme sind mit teilweise günstigerer Basistechnologie bereits verfügbar; das Gerät „Mitsubishi DiamondTouch“ sogar seit 2001, wobei der Gesamtaufbau dieses Systems heute immer noch ähnlich teuer wie das Microsoft Surface ist.

3.2 Spielkonzepte für das Surface

Bisher sind nur wenige Spielanwendungen für das Surface öffentlich verfügbar, vermutlich auch aufgrund seiner bisher noch geringen Verbreitung. Dementsprechend gering ist daher auch der Umfang bisher gesammelter und dokumentierter Erfahrungen speziell von Surface-Nutzern, bzw. -Spielern. In der Literatur beschriebene Projekte zu anderen dem Surface ähnlichen und bereits seit längerer Zeit verfügbaren Plattformen wie bspw. „DiamondTouch“ deuten jedoch an, welches Potenzial, welche Attraktivität und Ausbaufähigkeit derartigen Plattformen insbesondere mit Blick auf den Spiele- und Entertainment-Bereich zugesprochen werden kann.

Dabei ähneln sich die Spielentwicklung im Allgemeinen sowie die Entwicklung von Surface-Anwendungen im Spezifischen insofern sich konzeptionelle Anforderungen und Entwicklungsziele durchaus entsprechen. So sollen Spiele- und Surface-Anwendung prinzipiell der Unterhaltung dienen. Typische Attribute, die sowohl Spiele- als auch Surface-Erlebnisse beschreiben, sind „fun“, „engaging“, „immersive“, „emotional“, „social“, „cultural“, „aestethic“, „magic“ = „Entertainment“.

Die Spielidee selbst und die Rolle des Spielers sind nach Adams bei der Spielentwicklung zunächst zweitrangig; wichtiger für den Unterhaltungswert ist die Umsetzung des „Gameplay“, für das hier insbesondere die Gestaltung von User-Interface und Benutzerinteraktion maßgeblich ist. Es sollte daher im Rahmen der Surface-spezifischen Spielkonzeption und -implementation besondere Aufmerksamkeit gewidmet werden. Zumal sie aufgrund der Touch- und Multi-Touch-Funktionalitäten sowie des zentralen Ein- und Ausgabe-Elements des Surface, *die* Besonderheit der Zielplattform darstellt. Die Interaktionsrichtlinien von Microsoft, die bei der Entwicklung einzuhalten sind, unterstützen dabei grundsätzlich mehr, als dass sie einschränken und geben bisweilen wertvolle Tipps.

Unter spielkonzeptionellem Gesichtspunkt besteht das UI im Wesentlichen aus dem

Interaktionsmodell und der Darstellungsperspektive, aus der die Spielwelt dem Spieler präsentiert wird. Neben den Touch- und Multi-Touch-Funktionalitäten zur direkten Bedienung interaktiver Elemente wirkt sich insbesondere der Mehrbenutzer-Betrieb auf die Gestaltung des UI aus.

Generell ist das Surface für Multi-User-Spiele geeignet und ausgelegt, soziale Komponenten und Inter-Spieler-Kommunikation sind erwünscht und werden auch durch die nebeneinander oder gegenüber (face-to-face) angeordneten Positionen der Spieler begünstigt. Dabei teilen sich die Spieler die Ein- und Ausgabeeinheit und interagieren direkt und simultan mit den abgebildeten Elementen und Objekten.

Hieraus ergeben sich besondere Anforderungen und Problematiken, die es im Rahmen der Spielentwicklung zu berücksichtigen gilt, wie die 360°-View und -Interaktion, die Aufteilung des Sicht- und Interaktionsbereichs, die Darstellung „privater“ Informationen, die fehlende Kontakt-/ Benutzerzuordnung, die gegenüber der Maus- unpräzise Touch-Eingabe und dergleichen mehr.

Bereits entwickelte Spiele für das Surface zeigen naturgemäß Potenzial insbesondere im Bereich der „TableTop“-Games, also Brett-, Karten- oder auch Rollen-Spiele, die auch traditionell an einem Tisch in einer Runde von mehreren Personen gespielt werden. Bei dieser Spielgattung ist zumeist das „omnipresent“- bzw. „party-based“-Interaktionsmodell vorzufinden. Die hier bei Surface-Anwendungen vornehmlich angewandte zwei- oder 2.5-dimensionale Darstellungsperspektive mit Sicht „von oben“ (top-down) und Darstellung der Spielwelt im Ganzen (Single-Screen) bietet prinzipiell den gleichberechtigten/symmetrischen visuellen Zugang für alle Spieler.

Ein Nachteil dieser zweidimensionalen Perspektive ist, dass Objekte nur von einer Seite/Ansicht gezeigt werden können, was den Darstellungsraum sowie die Darstellungsmöglichkeiten stark einschränkt. 3D-Darstellung und auch 2.5-dimensionale Darstellungsperspektiven (z. B. isometrisch) heben diesen Nachteil auf, können aber aufgrund verschiedener Blickwinkel von rund um den Tisch aus unlogisch und verwirrend erscheinen. Daher findet eine 3D-Darstellung und -Navigation, z. B. aus der „First-Person“- oder „Third-Person“-Perspektive, bisher vor allem bei Single-Player-Spielen Einsatz.

Für die konzeptionelle Ausrichtung eines Spiels ist generell neben dem Einsatz- bzw. Standort der Plattform die Definition der gewünschten Zielgruppe wichtig. Typische Unterscheidungsmerkmale der Spieler, die sich auf Konzept und „Gameplay“ auswirken, finden sich in Altersgruppe (Kinder, Senioren), Geschlecht, oder anderen Merkmalen wieder. Die wichtigste Unterscheidung von Spielertypen besteht jedoch in der Differenzierung zwischen „Casual-“ und „(Hard)-Core-Gamern“. Spiele für das Surface können prinzipiell für beide dieser Spielertypen ausgelegt sein. Dabei bedient das Konzept der „Casual-Games“ jedoch eher die Anforderungen an eine Surface-Anwendung: Mit Attributen wie schnelle Erlernbarkeit, andauernder Fortschritt, einfache nicht-triviale Aufgaben und kurzweiliges Spiel steht im Wesentlichen die Unterhaltung im Vordergrund. Die Spieler sollten möglichst nicht unterfordert, aber keinesfalls überfordert werden.

Die Entscheidung für ein bestimmtes (oder mehrere) Spielgenre ist prinzipiell unabhängig von der Zielplattform und wird vielmehr durch „Actions“ und „Challenges“ (=„Gameplay“) beschrieben, die für ein Genre typisch sind. Eine Einordnung in ein bestimmtes Genre ist jedoch insofern hilfreich, da abhängig vom „Gameplay“ typische Interaktionsmodelle, Perspektiven und Testverfahren (z. B. zur Usability) Anwendung finden können und so das „Player-Centric-Design“ unterstützt wird.

Gegenüber runden-basierten Spielen haben Spiele in Echtzeit i. Allg. den Vorteil, dass mehrere Spieler simultan interagieren können, was auch für Surface-Anwendungen generell erwünscht ist. Eine Schwierigkeit bei Echtzeit-Konzepten liegt in der Abstimmung der Spielgeschwindigkeit („pace“), die Spieler mit unterschiedlichen Stärken und Schwächen nicht unter- oder überfordern

soll.

Prinzipiell ist der Wettbewerbsmodus unter den Spielern, also kompetitiv, kooperativ oder Gruppenspiel, nicht relevant für die Auswahl der Zielplattform. Für Surface-Anwendungen ist jedoch explizit die Interaktion zwischen Benutzern erwünscht, frühe Klarheit über den Wettbewerbsmodus in der Konzeptphase kann im Endprodukt die Kommunikation zwischen Spielern fördern, wenn sie nicht sogar Teil der Grundidee des Spiels ist. Freiheiten für die Spieler bei der Auslegung von Spielregeln kann ein Instrument sein, Kommunikation und soziales Verhalten zu unterstützen/fördern.

Das Surface bietet mit der Erkennung realer Objekte und „ID-Tag“-Funktionalitäten interessante Einsatzmöglichkeiten, um die Spielwelt nicht nur rein digital abzubilden und zu erweitern. Vielmehr kann z. B. der Einsatz von realen Würfeln zur Ermittlung von Wurfergebnissen herangezogen werden, können Spielfiguren als Avatare Spieler repräsentieren oder mit Gegenständen Spielparcours „abgesteckt“ werden.

3.3 Spielentwicklung für das Surface

3.3.1 Hardware-technische Aspekte

Die aktuelle Ausstattung des Surface (CPU, GPU, RAM) ist momentan noch „schwach“ für rechenintensive (Echtzeit-)Anwendungen, wie Spiele sie sein können. Dies gilt insbesondere im Vergleich zu aktuellen Spieltiteln für PCs und die dafür geforderte/empfohlene Hardware. Allerdings kann sich die Hardwareausstattung mit kommenden Versionen des Surface schnell und wesentlich ändern. Grundsätzlich ist die Hardware an sich nicht ausschlaggebend für die Güte eines Spiels, bzw. den Spaß, den der Nutzer damit hat, sondern vielmehr das Spieldesign und die Umsetzung des „Gameplay“. „Performance“ und „Aesthetics“ fordern schließlich das Ausreizen, aber nicht Überlasten, der verfügbaren Ressourcen.

Die Dual-Core CPU erlaubt die Verarbeitung von zwei Prozessen/Threads parallel, was auch von der Software-Umgebung, bzw. bei der Programmierung unterstützt wird. Dies kann Performance-Gewinne bringen, wenn man unabhängige Aufgaben für einzelne Threads findet, die parallel bearbeitet werden können.

Die Darstellung von 3D-Welten und -Effekten wird von der GPU durch (wenige) Pixel- und Vertex-Shader-Einheiten unterstützt. Rechenintensive Effekte wie Schatten-/Licht-Berechnungen, Anti-Aliasing, etc. sollten nur eingeschränkt genutzt werden. Dies hängt allerdings auch stark von der Größe der darzustellenden Welt, der Anzahl der darin enthaltenen Objekte, bzw. deren Detailgrad (Anzahl der Vertexe), der tatsächlich verwendeten Hardware und anderen Faktoren ab, die in ihrer Gesamtheit aufgrund der Komplexität schwer abzuschätzen und im Einzelfall zu definieren sind. Hier sind bei der Entwicklung auch Tests und die Optimierung durch Benchmarks auf der Zielplattform empfohlen.

3.3.2 Software-technische Aspekte

Spiele auf dem Surface sind als weiche Echtzeitanwendung einzustufen, da z. B. auch bei runden-basierten Spielen das System ständig lebendig erscheinen und auf Benutzeranforderungen möglichst immer und sofort reagieren sollte. Oftmals bedingt dies den Einsatz von Multi-Threading zur parallelen Behandlung von Ereignissen und ihren Auswirkungen sowie anderen Aufgaben. In diesem Zusammenhang wird von „weicher“ Echtzeitanwendung gesprochen, da zwar vermieden

werden soll, Zeitschranken zu überschreiten (also „Ruckeln“, „Haken“ in der Ansicht und/oder Bedienung), dies aber keinen fatalen System-Fehler darstellt. Es gelten daher die grundlegenden Regeln für die Entwicklung von Echtzeitanwendung, die im Wesentlichenim Wesentlichen besagen, dass die Laufzeiten und Speicheranforderungen von Programmmodulen bekannt und definiert sein müssen.

Microsoft Windows Vista ist prinzipiell geeignet für Spiele, wie es viele PC-Titel zeigen. Windows Vista bietet Unterstützung zur Behandlung diverser Medienformate und beherrscht den Microsoft DirectX 10 Standard, welcher allerdings die mittlerweile veraltete Grafikkarte des Surface nicht direkt (per Hardware) unterstützt. Als Multi-Threading-Betriebssystem und mit der Unterstützung von Mehr-Kern-Prozessoren bietet es eine gute Basis zur Entwicklung weicher Echtzeitanwendungen.

Microsoft XNA verspricht die schnelle Entwicklung von Spielanwendungen. Es werden dazu diverse Hilfsmittel wie das Komponenten-Modell, eine Gameloop oder ein vereinfachter DirectX Zugriff sowie dessen Programmierung direkt zur Verfügung gestellt. Der Vorteil der Plattform-unabhängigen Implementation mit XNA schwindet geringfügig aufgrund der speziellen Interaktionsmöglichkeiten des Surface und erschwert eine Kompatibilität in der Steuerung der Anwendung. So ist „X-Input“ für das Surface praktisch nicht nutzbar, erlaubt aber die Implementation alternativer Steuerungen. Die Programmierung in XNA ist grundsätzlich auf die Sprache C# beschränkt, welche ständig erweitert wird und alle Möglichkeiten moderner objektorientierter Sprachen bietet, allerdings auch deren Nachteile mit sich bringt, wie z. B. Overhead durch automatische Objektverwaltung und Laufzeitprüfungen. Diesen Nachteilen kann man zum Teil durch gutes Softwaredesign zumindest entgegenwirken (Object-Pools, Caching, etc.).

XNA bietet direkt keine Unterstützung für Physik und KI, aber es beinhaltet einige Basiskomponenten, die zur Realisierung solcher Spielmechanismen hilfreich sein können, wie Vektor- und Matrixberechnungen oder ein Zufallszahlengenerator. Weiterhin unterstützt die XNA-Community bei der Entwicklung mit Softwarekomponenten und Tutorials, die typische Aspekte, Schwerpunkte und Problemfelder bei der Spieleprogrammierung ansprechen.

WPF eignet sich eher für die Programmierung einfacher Spiele wie das „ScatterPuzzle“, was sich durch den Einsatz der Surface-SDK-Komponenten relativ einfach realisieren lässt. Die Verwendung von XAML unter WPF ermöglicht die schnelle Entwicklung nicht zuletzt durch die Einbindung anderer (visueller) Entwicklungswerkzeuge in den Entwicklungsprozess.

Zumindest für die prototypische Umsetzung von Spielkonzepten eignen sich sowohl WPF als auch XNA. Die Wahl hängt dabei auch stark vom Einsatzgebiet und den darzustellenden Funktionen ab, z. B. ist WPF prädestiniert für die Darstellung typischer zweidimensionaler GUIs und multimedialer Inhalte, wie Audio und Video, mit einer breiten Unterstützung für diverse Formate. Grundlegend ist WPF ereignisbasiert, unter XNA hat man dagegen das Konstrukt der „GameLoop“, die für einen zeitgesteuerten Ablauf (des Hauptthreads) sorgt und im Laufe derer z. B. Eingabedaten ständig abgefragt werden („polling“).

Es existieren auch andere (spezialisiertere) Spielentwicklungsumgebungen für Windows/PC, die für das Surface ebenfalls anwendbar/nutzbar wären. Vorteile gegenüber XNA ist z. B. bei Unity3D die vollständig visuelle Entwicklungsumgebung, welche direkt 3D-(Editor)-Funktionalitäten, physikalische Eigenschaften und vordefinierte Verhaltensweisen für Spielelemente bereit stellt. „Third-Party“-Produkten fehlt allerdings der direkte Zugriff auf die durch das Surface-SDK für WPF und XNA bereitgestellten Komponenten zur Ermittlung von Kontaktinformationen.

Nativere Programmierung der Plattform ist auch durch C++ und DirectX möglich, was theoretisch Vorteile im Laufzeitverhalten durch direktere Zugriffe und detailliertere Optimierungsmöglichkeiten auf Code-Ebene bringen kann. Aber auch hierfür steht das Surface-SDK nicht direkt zur Verfügung. Weiterhin ist ohne Nutzung eines Frameworks oder einer Spiele-API sehr viel Basisarbeit bei der Umsetzung zu leisten, z. B. zur Initialisierung des Systems oder von DirectX, bei der Handhabung von Netzwerk- oder Dateisystemzugriffen, der Verarbeitung multimedialer Daten, etc.

Die besonderen Qualitätsmerkmale, die bei der Entwicklung von Spielanwendungen für das Surface zu beachten sind, werden insbesondere durch UI und UX gefordert: Hohe Usability, eine besondere User-Experience und stabiles Laufzeitverhalten. Der in Abschnitt 2.3.1 beschriebene Prozess zur Spielentwicklung unterstützt dies durch User-/Player-Centric-Design, die Erstellung von Prototypen, begleitende Tests und Optimierungsphasen schon während der Gestaltung.

In der folgenden Tabelle 4 werden alle wesentlichen Merkmale nochmals komprimiert wiedergegeben und gegenübergestellt, die grundsätzlich für die Konzeption und Entwicklung von Spielen für das Surface relevant sind.

Spielkonzept	
Zielgruppe	- Anforderungen und Konzept entsprechen eher „Casual“-Spielertypen - Einfache direkte Steuerung erlaubt prinzipiell auch Benutzung durch Kinder
Wettbewerbsmodus	- Kompetitiv, kooperativ, Team- oder Gruppenspiel möglich - Lokales Multi-User und Gruppen-Spiel simultan möglich - Soziale Aspekte und Interaktion zwischen Spielern sollen im Vordergrund stehen
Genre	- Prinzipiell unabhängig, traditionelle TableTop-Spiele sind in der Regel Logik- und Puzzle-, Taktik- und Strategie- oder Rollenspiele - Surface kann Spielwelt um Elemente des Actionspiel-Genre erweitern
Gameplay (Aktionen)	- Interaktion direkt mit digitalen Inhalten durch Touch, Multi-Touch, Gesten - Klare und wenige Funktionen/Optionen - Interaktion durch reale Objekte, z. B. Avatare für TableTop-Systeme „üblich“ - Abhängig von Zielgruppe und Einsatzzweck; nach Prinzipien von Casual-Games, kurzweilige, leicht zu erlernende Spiele
Darstellungsperspektive	- 2.5D wird zumindest empfohlen, d. h. 3D-Eindruck durch Effekte - häufig Top-Down Ansicht, andere Luftperspektiven prinzipiell anwendbar - Freie 3D-Projektion, insbesondere unter Multi-User durch Steuerung und Perspektive erschwert/eingeschränkt realisierbar (360°-View)
Interaktionsmodell	- „Multipresent“- oder „Party-based“-Modell präferiert eingesetzt und von der Perspektive unterstützt (Typisch für TableTop Spiele) - Mehrere Spieler nutzen eine Eingabeeinheit u. U. simultan - ...und rund um den Tisch (360°-Interaktion)
Spielwelt-Dimensionen	
Physical	- Eindruck von Tiefe und Raum soll vermittelt werden
Environmental	- Visuelle Spielelemente symbolisch und intuitiv gestalten - Elemente der Spielwelt sind an die Realität angelehnt und virtuell verbessert - Die Spielwelt sollte lebendig/organisch erscheinen
Realism	- Super-realism, Vorbild für Verhalten ist reale Welt - Hochwertige und ästhetische, aber zurückhaltende, Anmutung und Darstellung - Realistische physikalische Verhaltensweisen von virtuellen Objekten
Surface Interaktion	
Touch/ Multi-Touch	- Ist auf die Aktionen im Spiel abzubilden

	- Digitale Inhalte sind direkt manipulierbar zu gestalten - 360°-Interaktion
Multi-User	- Soziale Aspekte und Kommunikation sollen gefördert werden - 360°-View und -Interaktion - Aufteilung (Ausrichtung, Symmetrie, Erreichbarkeit) der Ein- und Ausgabefläche - Problem der Sichtbarkeit „privater“ Daten - Zuordnung von Kontakten zu Benutzern vom System nicht vorgesehen
Objekterkennung	- reale Objekte machen das Spiel „anfassbar“ - interessant auch für den Einsatz von Spielfiguren als Avataren (Benutzerzuordnung), Würfeln, etc.
Konnektivität	- Vernetzung durch LAN und WAN möglich - Einsatz weiterer Controller möglich, aber nicht empfohlen - Kommunikation mit Smartphones u. Ä. Endgeräten möglich
Richtlinien	- Abbildung von realem Verhalten (Super-Realism) - 360°-User Interface - Nutzung von 2.5D und 3D Raum und Darstellung - Optimierung des User-Interface für Touch-Interaktion - Direktes visuelles Feedback auf Interaktionen - Konsistenz, z. B. bei Darstellung oder Wiederverwendung von Gesten - Minimalistische, zurückhaltende Anmutung - Vermittlung von Fortschritt und Erfolg
Surface-Hardware	
CPU + RAM	- Hardwareressourcen relevant für die Rechenleistung und Daten(vor)haltung - Dual-Core CPU und OS unterstützen Multi-Threading 2 paralleler Prozesse - Stand der Hardwareausstattung von 2007
GPU	- Gibt vornehmlich die Leistung der 3D-Visualisierung vor - Unterstützt Direct3D und Shader-Modell 3.0 - Wenige Pixel- und Vertex-Shader im Vergleich zu modernen GPUs - Stand der Hardwareausstattung von 2007
Projektion	- 30“ bei fixen 1024x768 Pixeln Darstellungsfläche fest vorgegeben
Surface-Softwareumgebung	
OS	- Microsoft Windows Vista Business 32Bit - Multi-Threading-fähiges Betriebssystem - Als multimediales Betriebssystem und mit DirectX gut für Spiele geeignet
Surface-SDK	- Liefert Kontakt-Informationen für WPF- und XNA-Anwendungen - Unterstützt mit visuellen Komponenten und vorgefertigten Gesten die Entwicklung von WPF-basierten Anwendungen
WPF	- .NET-Technologie zur Entwicklung multimedialer Anwendungsoberflächen
XNA	- Framework zur Entwicklung von Spielen

Tabelle 4: Übersicht relevanter Merkmale für Spielkonzepte und -entwicklung auf dem Surface

3.4 Nächste Schritte

Nachdem die Merkmale der Spielentwicklung für das Surface aus konzeptioneller und technischer Sicht bekannt sind, soll das erlangte Wissen Anwendung finden. Dies soll in Form eines Spiels geschehen, das von der Idee über Konzept und Design ausgearbeitet und letztlich prototypisch implementiert wird.

Hierzu werden zunächst in Kapitel 4 ausgewählte Spielideen für die Umsetzung auf dem Surface vorgestellt und grob skizziert. Dabei sollen die jeweils spieleigenen Besonderheiten bezüglich des Surface identifiziert und erläutert sowie Vorschläge für die weitere Surface-spezifische Ausgestaltung gemacht werden.

Alle vorgestellten Spielideen sind prinzipiell für mehrere Spieler ausgelegt. Weiterhin wurde darauf geachtet, sowohl kooperative als auch kompetitive Spiele oder Mischformen aufzugreifen. Zur Auswahl sollten weiterhin mindestens ein simultanes und ein abwechselnd zu spielendes Spiel gehören.

Nachdem die Entscheidung zur Umsetzung für eine viel versprechende Spielidee gefallen ist, erfolgt in Kapitel 5 die weitere Konzeption und Ausarbeitung. Dabei erfolgt die Beschreibung der Spielwelt, des Ablaufs und der grundsätzlichen Gestaltung. Darüber hinaus sollen die relevanten Aspekte der Interaktion und Darstellungsperspektive bezüglich des Surface erarbeitet werden und in das Konzept einfließen.

In Kapitel 6 soll auf Grundlage des zuvor erarbeiteten Konzepts die technische Umsetzung eines Prototyps erfolgen, sowie die Erfahrungen beschrieben werden, die dabei gemacht wurden.

4 Grobkonzepte für Spielideen / Kreation I

Der folgende Abschnitt beschreibt beispielhaft ausgewählte Spielideen und wie sie für eine Realisierung auf dem Surface denkbar wären. Dabei sollen die grundlegenden Surface-spezifischen und spielkonzeptionellen Aspekte dargestellt werden, die für die jeweilige Realisierung relevant sind. Im Anschluss wird eines der Spielkonzepte ausgewählt und im Rahmen des weiteren Verlaufs der Arbeit ausgestaltet und prototypisch umgesetzt.

4.1 Speed - Actionspiel

„Speed“ ist ursprünglich ein Kartenspiel, bei dem 2-6 Spieler ihre Spielkarten, die sie auf der Hand halten, möglichst schnell „loswerden“ müssen. Dazu können sie eine ihrer Karten auf einen offenen zentralen Stapel ablegen, wenn sie „kompatibel“ zu der obersten Karte des Stapels ist, d.h. die gleiche Zahl oder Farbe aufweist. Hält der Spieler keine passende Karte auf der Hand, besteht die Möglichkeit, eine Karte von einem zweiten zugedeckten Stapel aufzunehmen. Die Besonderheit bei dem Spiel ist, dass **simultan** gespielt wird, also jeder Spieler zu jeder Zeit seine passenden Karten auf dem offenen Stapel ablegen darf. Die Spieler **konkurrieren** also darum, möglichst schnell ihre Karte auf dem zentralen Stapel abzulegen. Der Spieler, der zuerst keine Karten mehr auf der Hand hat, gewinnt das Spiel. Eine Variation ist, dass die weiteren Platzierungen bis zum letzten Spieler ausgespielt werden.

Das Spiel ist, wie auch der Name schon vermuten lässt, in das Genre der **Action-Spiele** einzuordnen. Bis auf die zufällige Verteilung der Spielkarten ist das Spiel **symmetrisch** ausgelegt, d.h. es gelten für alle dieselben Spielregeln. Da das „Gameplay“ insbesondere **motorische und einfache kombinatorische Fähigkeiten** sowie **Reaktionsvermögen** erfordert, wird die Spielbalance und -geschwindigkeit vornehmlich durch die Fertigkeiten der beteiligten Spieler bestimmt. Ursprünglich als Kartenspiel konzeptioniert, wird Speed verdeckt (auf der Hand) und an einem Tisch o. ä. gespielt wird. Angesichts der Geschwindigkeit im Ablauf kann man aber öffentliche „Blätter“ zulassen, was dann sogar Teil der kombinatorischen Leistungen wird, die zu erfüllen sind.

Als Zielgruppe wird für das Kartenspiel Personen im Alter ab 8 Jahren angegeben. Vor allem Jugendliche und Kinder sind aufgrund der besonderen Herausforderung und Interaktion bei gleichzeitig einfachem Spielprinzip häufig sehr gut in dem Spiel. Da eine Spielrunde i.d.R. sehr schnell vorüber ist, bereitet das Spiel ein kurzweiliges Vergnügen (Casual-Game). Um es spannender und wettbewerbsintensiver zu gestalten, könnte man die Platzierung, bzw. Punkte aus vergangenen Spielrunden für jeden Spieler sammeln und ausgeben, sie für besonders gute Runden extra belohnen und dergleichen mehr.

Die Darstellungsperspektive wäre ausreichend und wie für TableTop-Spiele üblich, zwei- bzw. 2.5-dimensional mit Sicht von oben („**Top-Down**“) auf das gesamte Spielfeld („**Single-Screen**“). Das Spielfeld zeigt dabei im Wesentlichen den offenen zentral platzierten Stapel zur gemeinsamen Ablage in der Mitte an, sowie den zweiten verdeckten Stapel zum Ziehen von Karten auf die Hand durch die Spieler. Aus Symmetriegründen könnte man auch mehrere verdeckte Kartenstapel platzieren, um sie für alle Spieler gleich gut erreichbar zu machen. Weiterhin ist für jeden Spieler die eigene „Hand", auch oft „Deck“ gennant, darzustellen.

Der Interaktionsmodell ist „**omnipresent**", d.h. alle Spieler sollten zu jeder Zeit alle Spielelemente erreichen können. Dabei bedienen die Spieler die Spielkarten direkt durch Touch und über Gesten wie z. B. „Flick", wobei die Spielelemente ein realistisches **physikalisches Verhalten** bezüglich

Beschleunigung und Massenträgheit aufweisen sollten. Weiterhin sollte das „Deck“ flexibel auf der Oberfläche verschiebbar sein, damit die Anordnung der Spieler rund um den Tisch frei geschehen kann. Anstelle typischer Spielkarten könnten für die virtuelle Version des Spiels beliebige Spielsteine verwendet werden. Anstelle von Farben oder Zahlen, die übereinstimmen müssen, könnten auch Symbole und die Form der Spielsteine selbst als Kriterien dienen. Wichtig zu beachten ist hierbei insbesondere, dass die Spielsteine von allen Seiten rund um den Tisch gut erkennbar sind.

Eine Implementation mit WPF wäre gut denkbar, da ein ereignisgesteuerter Ablauf eher den Anforderungen entspricht: Ohne eine Benutzerinteraktion treten keine besonderen Ereignisse auf, die nebenläufig zu berechnen wären, und die abzubildende Spiellogik ist einfach und unaufwändig. Die Realisierung könnte mit Hilfe der ScatterView- und ScatterItem-Komponenten und ihrer vordefinierten Gesten und physikalisch korrekten Verhalten relativ schnell erfolgen.

4.2 Poker - Kartenspiel

Poker hat stark an Popularität hinzugewonnen und ist auch als Online-Spiel sehr beliebt, wobei auch dort teilweise um echtes Geld oder andere limitierte Einsätze gespielt wird. Die Regeln von Pokerspielen variieren in Abhängigkeit von der jeweiligen Spielart. Im Wesentlichen geht es beim Pokern darum, ein besseres „Blatt", d.h. Set von Karten, als die Gegner zu haben oder zumindest die Gegner das glauben zu machen. Im Laufe einer Spielrunde (Hand) setzen dabei die Spieler reihum einen „wertvollen“ Einsatz, i.d.R. Spielchips, auf die Gewinnchance ihres Blattes. Es besteht allerdings auch die Möglichkeit, für die Spielrunde auszusteigen, wenn der Spieler das Maximalgebot der Gegenspieler nicht mitgehen möchte. Am Ende einer Hand gewinnt der verbleibende Spieler mit dem höchsten Blatt die Einsätze der Gegenspieler („Pot“). Bei vielen Poker-Varianten kann der Spieler sein Blatt durch Austauschen von Karten oder durch die Nutzung der „offenen“ Karten (Stack), die für alle Spieler gleichzeitig gelten, zusammenstellen. Das Spiel ist also stark **kompetitiv,** wird **runden-basiert** und **nacheinander** gespielt.

Die Karten werden zu jeder Runde zufällig aus einem anglo-amerikanischen Kartenset mit 52 oder 104 Karten verteilt. Legal (in Deutschland) ist Poker dabei kein Glücksspiel, da mit gewissen Karten auf der Hand und auf dem Spielbrett die Berechnung der Gewinnchance auch ohne das Wissen um die Karten der Gegenspieler zumindest teilweise möglich ist. Neben den **kombinatorischen** Fähigkeiten fordert das Prinzip wertvoller/limitierter Einsätze zusätzlich **taktische** und **strategische** Spielentscheidungen, wie z. B. gelungenes Bluffen auch Gewinne mit einem vergleichsweise schwachen Blatt ermöglicht.

Poker ist kein sehr schnelles Spiel und eignet sich auch aufgrund der an die Spieler gestellten Anforderungen eher für ausgedehntere Spielsitzungen zwischen jugendlichen oder erwachsenen Spielern. Die Spielgeschwindigkeit wird dabei gänzlich durch die einzelnen Spieler selbst bestimmt, was Zeit für die Kommunikation und das gegenseitige Beobachten der Spieler untereinander lässt. Insbesondere Letzteres geht beim Online-Poker aufgrund der (lokalen) Entfernung der Spieler fast vollkommen verloren und birgt gerade besondere Reize für eine lokale Spiel-Runde. Größere Turniere werden oftmals über mehrere Tische und Abende gespielt.

Die präferierte Darstellungsperspektive ist **2.5-dimensional** mit „**Top-Down**“ Ansicht auf die gesamte Spieloberfläche („**Single-Screen**“). Neben der Darstellung der (verdeckten) Karten der Spieler sollten die Spielchips der Spieler und des Pots visualisiert werden. Ein besonderes Problem ergibt sich dabei aus der grundsätzlichen Sichtbarkeit von Spieler-eigenen Karten für alle Anwesenden, da die eigenen Spielkarten eine private Information darstellen, was wesentlich für

das Spielprinzip ist. Dieser Problematik könnte beispielsweise durch Verdeck- bzw. Aufdeck-Gesten begegnet werden, wie sie das „Surface Academy 2009 Toolkit“ liefert. Eine andere, aber aufwendigere Lösung wäre die Einbindung externer Geräte wie Smartphones, die private Informationen der Spieler anzeigen.

Das Interaktionsmodell ist bei Poker ebenfalls „**omnipresent**“ vorgesehen. Die Interaktion mit den Spielkarten beschränkt sich zumeist auf das „Darunterschauen", Weglegen oder Aufdecken der eigenen Karten. Für den Spieler der am Zug ist, müssen besondere Aktionen mit den Spielchips, z. B. zum bequemen Setzen oder Erhöhen, zur Verfügung gestellt werden. Auch hier sollten bekannte Gesten und physikalische Eigenschaften für bewegliche Spielelemente Einsatz finden. Es muss weiterhin klar herausgestellt werden, welcher Spieler gerade am Zug ist und was seine aktuellen Optionen sind. Nur der Spieler, der am Zug ist, sollte eine sinnvolle Aktion durchführen können, die sich auf den weiteren Spielablauf auswirkt.

Die Spielmechanik sollte vornehmlich die Spielregeln zumindest einer bekannten Pokervariante implementieren und für den korrekten Ablauf der Spielrunden sorgen. Die Wahrung der Reihenfolge der Spieler während einer Hand, aber auch Spielrunden übergreifend, ist wichtig, da der beginnende Spieler mit jeder Hand wechselt, so dass alle Spieler rotierend an allen Positionen spielen.

Gerade da Poker-Sitzungen länger andauern können, sollten die Spieler ihre Position um den Tisch ändern oder während des Spiels pausieren (aussetzen) können, ohne ihre virtuellen Chips zu verlieren. Für die Erhaltung der Zugabfolge und Verwaltung des Spieler-Kontos könnte ein echter Chip (Jeton) mit einem persönlich für den Spieler erstellten „ID-Tag“ als „Avatar“ dienen. Mit Hilfe dessen könnte sich der Spieler in eine Spielrunde ein- und ausklinken oder seine Position am Tisch, bzw. die seiner Karten und Chips, steuern.

Einige Ideen für den weiteren Ausbau wären z. B.:

- Spielmechanik auslegen für verschiedene Pokerarten und Variationen
- Implementation verschiedener Session- und Turniermodi
- Der Einsatz von Computergegnern (KI)
- Die Vernetzung von Tischen, Internetspiel oder Verbindung mit Online-Poker-Systemen

Für die Realisierung der Spielmechanik kann WPF dienen, wie auch eine bereits verfügbare[1] OpenSource „Poker Engine“ zeigt. Durch die Nutzung des ebenfalls in WPF umgesetzten „Surface-Academy-Toolkit“ erhält man interaktive Spielkarten, die speziell auf die Bedürfnisse des Surface angepasst sind.

4.3 Musik-/Rhythmusspiel

Insbesondere für Konsolen sind in letzter Zeit vermehrt so genannte Rhythmus- und Musik-Spiele, wie z. B. „Guitar Hero“ oder „Let's Tap“ für Nintendo Wii sehr populär geworden. Hierbei wird der Spieler in die Rolle eines Musikers oder „Rockstars“ versetzt, in der er zumeist populäre Lieder nach spielen und darbieten muss. Verschiedene Mehrspieler-Modi, die oftmals **simultanes** und **kooperatives** Spielen erforderlich machen, stärken das Gruppen- und Zusammengehörigkeitsgefühl beim Spielen.

Der Spielablauf besteht im Wesentlichen darin, dass der Spieler möglichst fehlerfrei vorgegebene

1 URL: http://wpfpoker.codeplex.com, Online 24.11.2009

Rhythmen oder Melodien (Patterns/Muster) nach spielen muss, die zumeist abstrakt visualisiert in Form von Tonspuren oder Sequenzen und als Audio-Feedback wiedergegeben werden.

Häufig kommt bei diesen Spielen an Konsolen besondere Controller-Hardware in Form nachgebildeter Schlagzeuge oder Gitarren zum Einsatz. Die Instrumente erlauben eine an die Realität angelehnte, vereinfachte und schnelle Eingabe und vermitteln eine besondere Haptik und Bindung an das Spielgeschehen, was in dieser Form am Surface nicht geschehen kann. Zugleich sind sind diese Spiele jedoch „traditionell“ auf **physische** und **motorische Leistungen** ausgelegt und die Konzentration liegt während des Spielens vornehmlich auf der Ausgabe/Vorgabe vom System und der Handhabung des Instruments (zumindest bei Anfängern/Casual-Gamern); nur selten interagieren die Spieler direkt miteinander. Weitere Anforderungen an den Spieler, neben **Timing** und **Geschick**, sind **Reaktionsvermögen**, **Takt- und Rhythmusgefühl** oder auch Erinnerungsvermögen.

Für eine Implementation auf dem Surface sollte der Vorteil genutzt werden, dass die Spieler rund um den Tisch angeordnet sind und direkt sehen, was die anderen Spieler tun. Um die Interaktion der Spieler untereinander zu fördern, könnten sich deren Eingaben direkt auf das Spiel, z. B. des Nachbarn auswirken. Die Spieler könnten darauf Einfluss nehmen, was abgespielt wird, und vielleicht sogar selbst Muster vorgeben, die reihum gehen und in gewisser Zeit angeschlagen werden müssen. Weiterhin sollten die Spieler auch die Zeit haben zur Interaktion untereinander, die Spielgeschwindigkeit und Schwierigkeit für den Einzelnen ist also gut zu balancieren. Dabei sollte die Bedienung selbst einfach sein und keine große Anstrengung oder Konzentration erfordern. Dazu sollte ein relativ „freier“ (keine „Knöpfe“) Eingabereich für Touch-Anweisung und einfache Gesten für alle Spieler zur Verfügung stehen. Dieser Bereich muss **360°-View** und -Interaktion unterstützen und so flexibel sein, dass Spieler einfach ihren Ort ändern oder neue Spieler teilnehmen können, daher ist eine **omnipresente** „**Single-Screen**“ Lösung anzustreben. Da die Darstellung insgesamt, und insbesondere die von Tonmustern oder -abfolgen, relativ abstrakt und „zurückhaltend“ erfolgen kann, eignet sich eine zwei- bzw. 2.5-dimensionale Perspektive.

Aufgrund der hohen physischen Anforderungen an den Spieler ist das Spiel vornehmlich in das **Action**-Genre einzustufen. Wie schon beschrieben, sollten allerdings bei der weiteren Gestaltung des Gamplay für das Surface diese Anforderungen etwas in den Hintergrund rücken. Vielmehr sollten die **Kreativität** des Spielers und die **Kommunikation** untereinander gefördert werden.

Als Zielgruppen kommen prinzipiell Menschen jeden Alters in Betracht, insbesondere auch Jugendliche. Die Inhalte sollten möglichst für alle Geschlechter mehr oder minder „gleich“ ansprechend und neutral gestaltet sein.

Eine Implementation könnte bei nicht zu komplexer Visualisierung, von der prinzipiell abgeraten wird, und „einfacher“ Ausgabe von Ton und Musik durch WPF erfolgen. Ist die Ausgabe von Ton durch das Gameplay aufwendiger gestaltet, etwa weil Tonspuren gemischt oder mit Effekten belegt werden (müssen), bietet sich auch die Nutzung von XACT unter XNA an, was speziell für solche Dinge ausgelegt ist (vgl. Abschnitt 2.2.2.4).

4.4 Sportsimulation Curling

Beim traditionellen Curling versuchen zwei Mannschaften abwechselnd ihre Spielsteine näher als die Gegner an den Mittelpunkt eines Zielkreises am Ende einer 45m langen Eisbahn zu platzieren. Curling ist eine olympische Disziplin und wird offiziell von zwei Teams zu je 4 Spielern gespielt. Jeder Spieler hat 2 Spielsteine, die er im Laufe einer Spielrunde („End“) über das Eis gleiten lässt,

wobei sich die Teams und die Spieler innerhalb der Teams abwechseln.

Sind alle Steine eines Ends gespielt, bekommt das Team so viele Punkte gutgeschrieben, wie es Steine innerhalb des Zielgebiets platziert hat, die näher am Ziel liegen, als der bestplatzierte Stein des gegnerischen Teams. Dann beginnt jeweils das Siegerteam das nächste End und gibt dem Verliererteam so den Vorteil des letzten Steins (Hammer) im End. Gesamtsieger ist das Team, das nach allen gespielten Ends in der Summe die meisten Punkte erzielt hat. Weiterhin haben die Mannschaften die Möglichkeit, durch Wischen auf der Eisfläche mit einem Besen die Laufbahn der Steine zu beeinflussen.

Das Spiel ist generell **rundenbasiert** und in erster Linie **kompetitiv**. **Kooperatives** Spiel entsteht durch die Bildung von Teams, deren Mitglieder abwechselnd spielen, um gemeinsam Punkte zu sammeln. Durch das Element „Wischen“ während der Gleitphase eines Steins entstehen ein **simultanes Gruppenspiel** und Interaktion zwischen den Spielern.

Die besonderen Herausforderungen des Spiels sind **Geschick** und **Präzision**. Die Spieler benötigen in der Realität eine gute Hand/Arm-Auge Koordination, um den schweren Stein am anderen Ende der relativ langen Eisbahn genau zu positionieren. Hinzu kommen **taktische Elemente** und Überlegungen wie z. B. der Vorteil des Letzten Steins, der sich – bei Profispielern - in Form einer offensiven oder defensiven Spielweise innerhalb eines Ends und auch End-übergreifend auswirkt.

Die Umsetzung des Spiels sollte eine Mischung aus **Action-Spiel** und **Sport-Simulation** ergeben. D.h. es sollten die wesentlichen Regeln des Curling-Sports umgesetzt und ein möglichst **realistische Abbildung** der physikalischen Umgebung in der Spielwelt geschaffen werden, um die elementaren Herausforderungen dieser Sportart zu erhalten. Gleichzeitig sollte die Steuerung es erlauben, Steine genau und taktisch zu platzieren, ohne dass der Spieler ein Profi-Curler zu sein hat. Schnelle Erfolge und gutes Gameplay sind hier für den Spieler wichtiger als eine realistische Simulation. Da auch aufgrund der fehlenden Benutzererkennung der Plattform einige der Spielregeln nicht ohne weiteres durch das Surface kontrollierbar sind, muss die Regelprüfung und Auslegung teilweise frei durch die Spieler geschehen, ein Umstand der allerdings auch konzeptionell durchaus gewünscht sein kann

Prinzipiell erscheint ein **omnipresentes Interaktionsmodell** sinnvoll, da mehrere Spieler zur gleichen Zeit Aktionen ausführen (wischen) und zu jeder Zeit rund um den Tisch den Spielablauf und -stand betrachten können sollen. Jedoch hat nur ein Spieler zu einer Zeit die Kontrolle über den Spielstein, der gerade gespielt wird, so dass zumindest in dieser Phase eines Zugs auch in einer ihm (und seinem Team) zugewandte Perspektive dargestellt werden könnte.

Ein besonderes Problem stellt dabei die Größe des Spielfelds dar, dass u. U. zur sinnvollen Steuerung nur ausschnittsweise gezeigt werden kann, bei gleichzeitig prinzipiell gewünschter 360°-View/-Interaktion. Durch eine leicht geneigte Perspektive von schräg oben ergäbe sich mehr Raum zur Darstellung, aber je flacher der Winkel wird, desto verwirrender ist das Bild für den Betrachter auf der gegenüberliegenden Seite. Zusätzlich könnte eine flexible Kamera aus der **3rd Person-Perspektive** (vom Stein aus) oder **Kontext-sensitiv** zum aktuellen Spielgeschehen navigieren.

Als **Zielgruppe** für das Spiel kommen sowohl jüngere als auch ältere Personen in Frage. Das Spiel soll kurzweilig sein und Casual-Game-Prinzipien folgen.

Eine Realisierung könnte mittels XNA erfolgen, was sich für die Darstellung einer 3D-Welt aus flexibler Kameraperspektive anbietet. Die besonderen Herausforderungen liegen schließlich in der

Umsetzung der perspektivischen Darstellung bei möglichst gleichzeitiger Unterstützung der 360°-View und in der Abbildung der Aktionen des Steinspielens und des Wischens. Weitere Anforderung ist die physikalische Simulation z.B. von Massenträgheit, Reibung, Kollision (Impuls), etc.

5 Ausarbeitung des Spielkonzepts „Curling“ / Kreation II

Für die weitere Ausarbeitung eines Spielkonzepts ist die Auswahl aus Kapitel 4 auf „Curling“ gefallen, da es insbesondere die verschiedenen Problembereiche und Aspekte der Entwicklung für das Surface auch unter XNA zeigt. Darüber hinaus ist die dreidimensionale Darstellungsperspektive und Kameraführung unüblich im Vergleich zu den existenten Spielen und so eine besondere Herausforderung. Zudem handelt es sich bei Curling um eine Sportsimulation, die so ebenfalls bisher noch nicht für das Surface umgesetzt wurde und daher noch keine diesbezüglichen Erfahrung bekannt sind. Jedenfalls können die Spieler die physikalische Umgebung uns simulierte Verhaltensweisen erkunden und können dabei schnell die Bedienung erlernen. Außerdem sind neben kompetitivem und abwechselnden Spiel auch Elemente von simultanem und kooperativen, bzw. Gruppenspiel enthalten.

In Abschnitt 4.4 wurden das Grobkonzept für Curling und die grundsätzlichen Anforderungen bereits beschrieben. In diesem Kapitel soll das Konzept weiter ausgearbeitet werden, wobei zunächst das genaue Regelwerk, die Begrifflichkeiten und besonderen Herausforderungen an die Spieler dieser Sportart erklärt werden. Aufbauend darauf sollen die Abbildung und Darstellung der virtuellen Spielwelt, ihrer Regeln und insbesondere die Interaktions- und Steuerungsmöglichkeiten für die Spieler näher definiert werden. Dies stellt die konzeptionelle Basis für den im weiteren Verlauf zu entwickelnden Prototyp dar.

5.1 Curling: Regelwerk und Begriffe

Die folgenden Abschnitte 5.1.1 bis 5.1.5 präsentieren auszugsweise das offizielle Regelwerk des Deutschen Curling Verbandes [DCV], sowie Spielelemente, Begriffe und Besonderheiten des Curling. Die Informationen sollen für ein einheitliches Verständnis für die besonderen Anforderungen an die Umsetzung der Spielwelt im weiteren Verlauf dieser Arbeit sorgen.

5.1.1 Das Spielfeld

Abbildung 22 zeigt das Spielfeld (gespiegelt), welches beim Curling „Rink“ genannt wird. Es ist ca. 42m lang und 4,30m breit. Es ist von einer Eisfläche bedeckt und wird vor dem Spiel vom Eismeister mit Wasser besprenkelt, um den „Curl“-Effekt zu erhöhen. Die Beschaffenheit der Eisfläche ist ohnehin maßgeblich für das Verhalten der Steine (Reibung) und sorgt so auch für Varianzen innerhalb eines Spiels. Der Spieler spielt seinen Stein von der „Hack“-Linie, spätestens aber bei der „Hog“-Linie, ab. Er muss ihn möglichst nahe an den „Button“ bringen, der innere Kreismittelpunkt der von der „Central“-Linie und der „Tee“-Linie gebildet wird. Die Kreise um den Button beschreiben die 4, 8 und 12 Fuß Ringe, die zusammen das „House“ bilden. Am Ende eines Ends kommen nur die Steine in die Wertung, die innerhalb des „House“ liegen oder es zumindest schneiden. Wie in der Abbildung ist beim offiziellen Curling das Spielfeld symmetrisch gespiegelt, da nach jedem End die Spielrichtung gewechselt wird.

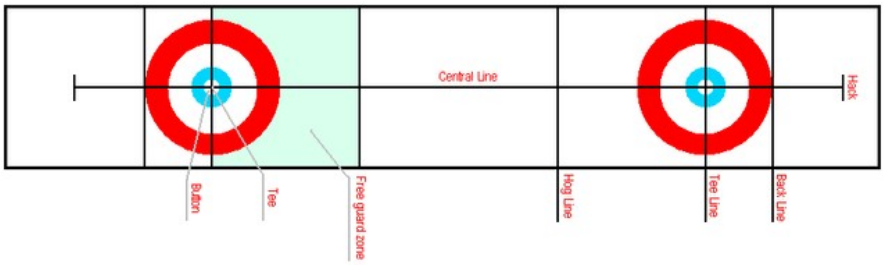

Abbildung 22: Curling Spielfeld (Rink) [CurlingRink]

Abbildung 23 zeigt das Rink maßstabsgetreu, um die Dimension und Verhältnisse der realen Spielwelt aufzuzeigen. Der kleine Punkt, den man beim rechten House sieht, oder vielmehr kaum sieht, zeigt einen Spielstein im korrekten Verhältnis unter Abbildung realer Dimensionen.

Abbildung 23: Curling Spielfeld maßstabsgetreu

5.1.2 Die Spieler

Beim offiziellen Curling besteht ein Team aus 4 Spielern, wobei Männer und Frauen und Jung und Alt gemischt spielen dürfen. Der „Skip“ ist der Kapitän der Mannschaft. Er gibt die taktische Ausrichtung der Mannschaft vor und bestimmt die Reihenfolge der Spieler, etc.

Die besonderen Anforderungen an alle Spieler sind Geschick und Präzision, wobei der Spieler generell zwei grundlegende Aktionen durchführen kann: Entweder er ist an der Reihe und spielt einen Stein, oder er darf wischen.

Der Vorgang des Steinspielens selbst dauert nicht lange an, der Spieler stößt sich von der „Hack“-Linie an einer Befestigung ab und gleitet mit dem Stein zusammen wenige Meter über das Eis. Dabei reguliert er seine Geschwindigkeit und Richtung, visiert das Zielgebiet an und lässt den Stein dann einfach weiter gleiten. Gegebenenfalls versieht der Spieler den Stein beim Loslassen mit einer leichten Rotationsbewegung („Curl“), so dass er einen Bogen in seiner Laufbahn erfährt.

Um das Gleiten zusammen mit dem Stein zu ermöglichen, tragen die Spieler besondere Schuhe, die Sohle des Stand-Fußes ist glatt. Um bei dem Gleitvorgang das Gleichgewicht besser halten zu können, kommt manchmal auch ein Besen als Stütze zum Einsatz, wie es ein Curling-Spieler im Moment des Abwurfs in Abbildung 24 zeigt.

Abbildung 24: Curling-Spieler mit Stein und Besen [DCV]

5.1.3 Spielsteine

Neben dem Spielfeld sind die wichtigsten Elemente des Spiels die Steine (s. Abbildung 24). Sie bestehen zum größten Teil aus Granit, sind rund mit einem Durchmesser von ca. 30cm und wiegen ca. 20 kg. Auf der Oberseite des Steins ist ein Griff angebracht, mit dem sich der Stein führen und im Moment des Abspielens (Abgabe) in eine Drehbewegung („Curl“) versetzen lässt. Die Drehbewegung führt zu einem seitlichen Drift in der Laufbahn des Steins in die Richtung der Drehbewegung. Ist der Stein abgegeben, darf er nicht mehr weiter berührt werden.

5.1.4 Besen

Der Besen ist ein weiterer wichtiger Bestandteil des Spiels. Er dient den Spielern vornehmlich zum Wischen der Eisbahn während des Laufs eines Steins. Durch das Wischen vor der Laufbahn des Steins wird diese beeinflusst. Durch das Wischen wird der Reibungswiderstand der Eisfläche verringert und somit die Laufbahn des Steins verlängert. Hat der Stein einen Curl-Effekt, also seitlichen Drift, wird auch dieser verringert, d.h. der Stein verläuft weniger stark gekrümmt. Dabei darf prinzipiell nur die Mannschaft den Stein „wischen“, der ihr gehört. Erst ab der „Tee“-Linie dürfen auch gegnerische Steine gewischt werden. Der Effekt des Wischens kann die Reichweite des Steins um bis zu 3 Meter erhöhen.

5.1.5 Spielablauf

Welche Mannschaft beginnt, wird vor dem Spiel dem ausgelost („Toss“). Die zweite Mannschaft hat den Vorteil des letzten Steins („Hammer“) im aktuellen End.

Dann macht jede Mannschaft abwechselnd ihren Zug. Die Spieler wechseln sich dabei auch innerhalb des Teams in fester Reihenfolge ab, so dass jeder Spieler einmal einen Stein „geworfen“ hat, bevor der erste seinen zweiten Stein spielt. Während ein Stein gespielt wird, kann vor diesem von den Mitgliedern des eigenen Teams mit dem Besen „gewischt“ werden, um seine Laufbahn nach dem Loslassen ggf. noch zu beeinflussen.

Haben beide Mannschaften alle ihre Steine gespielt, wird das End ausgewertet. Dabei werden generell nur die Steine gezählt, die im oder am Haus liegen. Punkte bekommt nur die Mannschaft, die mit ihren Steinen näher am „Button“ liegt, als der nächste Stein die gegnerische Mannschaft.

Der Gewinner des letzten Ends beginnt das nächste End und bekommt somit ein Handicap zum Ausgleich für den Sieg, wenn der Gegner den Vorteil des letzten Steins auszunutzen weiß. Dieser Vorgang wird wiederholt, bis alle Ends (offiziell 10) gespielt sind. Der Gesamtsieger ergibt sich aus der Summe der Punkte aus den einzelnen Ends. Steht es zu diesem Zeitpunkt unentschieden, wird ein zusätzliches End gespielt,

Der Vorteil des letzten Steins hat nicht unerhebliche Auswirkungen auf die Spielstrategie und Taktik, wie ein Auszug aus den Erläuterungen des Deutschen Curling Verbandes [DCV] zeigt:

- *„Das Team mit letztem Stein („Hammer“) spielt grundsätzlich offensiv und versucht mindestens zwei Steine zu „schreiben“.*
- *Das Team ohne letzten Stein spielt grundsätzlich defensiv und versucht entweder den Gegner zu einem Stein zu zwingen oder einen Stein zu stehlen (also am Ende des Ends mindestens einen Stein näher am Kreismittelpunkt („Button“, „Dolly“) zu platzieren, als die gegnerische Mannschaft).*

- *Die Strategie ist im Spitzencurling spielentscheidend.*
- *Wie im Schach muss im Curling der Skip die Spielzüge des Gegners voraussehen und somit einerseits seinem Team Chancen aufbauen und andererseits dem gegnerischen Team diese verbauen.*
- *Sehr oft entscheidet der allerletzte Stein in einem Spiel über Sieg oder Niederlage.*
- *Die Psyche ist ein zweiter spielentscheidender Faktor im Spitzencurling.“*

5.2 Spielwelt

Als Sportsimulation sollte die Spielwelt zunächst bezüglich der physikalischen und zeitlichen Dimension möglichst nah an die Realität angelehnt sein. D.h. die Zeit und wie sie während der Simulation vergeht sollte auf Echtzeit basieren und die physikalischen Aspekte wie Raum, Dimension und Maßstab originalgetreu nachgebildet sein. Auf dieser Basis können später noch Optimierungen folgen, in dem Aspekte herausgenommen oder modifiziert werden, um eine bessere Benutzbarkeit zu erreichen. Die Abbildung des physikalischen Verhaltens der Steine kann im zweidimensionalen Raum geschehen, da das Spielfeld eben ist und die Interaktion der Steine (Kollision) ebenfalls nur in zwei Dimensionen stattfindet.

Wie die „User Experience Guidelines“ für das Surface es vorschreiben, sollte die Elemente in der Spielwelt visuelle hochwertig gestaltet sein, aber zugleich zurückhaltend anmuten und nicht überfrachtet wirken. Die Darstellung der Spielwelt soll sich daher zunächst auf ein einfaches Curling-Spielfeld und die Steine darauf reduzieren. Ein Ausbau der Spielwelt, z. B. durch Abbildung der Spieler selbst, von Zuschauern oder eines ganzen Eisstadions ist denkbar und kann gegebenenfalls später erfolgen.

5.3 Gameplay

Das wesentliche Gameplay, also die Herausforderungen, die sich dem Spieler stellen und seine diesbezüglich möglichen Aktionen sollen im Wesentlichen am realen Curling-Spiel bzw. -Regelwerk angelehnt sein, welches in den vorangegangenen Abschnitten bereits ausführlich beschrieben wurde. Im Folgenden sollen weitere Ideen und Anregungen zur Ausgestaltung des Gameplay in das Spielkonzept einfließen, um den Ablauf für den Spieler interessanter zu gestalten:

Um den Reiz des Spiels und den Wettbewerb zu erhöhen, können intelligente „Einblendungen“ zum aktuellen Spielgeschehen weiter die Spannung erhöhen. Mögliche Ausgaben zur jeweiligen Situation sind z. B.:

- Der Status über den Stand des aktuellen Ends, also das Anzeigen des nächstgelegenen Steins vom Ziel, des zweiten, etc.
- Die Anzeige der Anzahl noch verfügbarer Steine oder Einblendung „Letzter Stein“ im End
- Die Anzeige des aktuellen Punktestands aller Ends
- Einblendungen während ein Stein ins Ziel gleitet, wie Distanz zum „Button“, aktuelle Rangfolge der Steine, etc.
- Kommentare zum Wurf je nach Qualität (auch gut über Audio-Feedback möglich)

Dieses Statusanzeigen müssen nicht zwingend in Form von Text oder Zahlen dargestellt werden, sondern können/sollten innerhalb der Spielkomponenten visualisiert oder durch symbolische Elemente und Metaphern vermittelt werden.

Dies gilt auch für die Anzeige möglicher Aktionen, für die visuelle Hilfestellung gegeben werden kann. Z. B. könnte interaktiv kenntlich gemacht werden, wann ein Stein losgelassen werden muss. Andererseits kann man den Spieler diese Art der Regeln konform zu den „User-Experience-Guidelines“ auch „erkunden“ lassen. Denkbar wäre hier auch das optionale Ein- und Ausschalten der (automatischen) Regelprüfung.

Die ausführbaren Aktionen sollten möglichst an die realistischen Bewegungsabläufe angelehnt sein. So sollte der Spieler den Stein als digitalen Inhalt mit dem Finger direkt manipulieren/schieben und damit seine Geschwindigkeit und Richtung regulieren können. Das Gefühl der Massenträgheit des in der Realität 20 kg schweren Steins sollte vermittelt werden, auch wenn dieser auf dem Eis nur einen geringen Widerstand (Reibungszahl) aufweist. Der Curl-Effekt soll ebenfalls realisiert werden, in dem der Spieler eine Drehbewegung mit dem Finger/der Hand im Moment des loslassen ausführt.

Zu Gunsten des Gameplay sind voraussichtlich Abstriche bei der Umsetzung der Steuerung und der physikalischen Simulation hinsichtlich der Realitätstreue der Abbildung zu machen.

Dies trifft auch für das zu implementierende Regelwerk zu, insbesondere was die Anzahl der Spieler/Steine pro Team und Vorgaben der Spieler-Reihenfolge, Anzahl der zu spielenden Ends, etc. betrifft. Prinzipiell ist die Möglichkeit einer flexibleren Regelauslegung durch die Spieler erstrebenswert. Die Simulation sollte nur kommunizieren, welches Team aktuell an der Reihe ist und der wievielte Stein innerhalb des Ends von dem aktuellen Team gespielt wird. Die Spieler haben so jederzeit selbst in der Hand, wie die Aufteilung der Spieler in Teams erfolgt und wer welchen Stein spielt.

Speziell der Einsatz von realen, mit ID-Tags ausgestatteten Spielbesen im Rahmen der „Wischen“-Aktion könnte dem Spiel mehr „Haptik“ verleihen. Die Verwendung teamspezifischer ID-Tags, stellt zugleich eine Möglichkeit dar, in dieser Spielphase die Benutzerzuordnung durch das System zu überprüfen. Zumal hier weitere Regelprüfungen erforderlich werden könnten, da eigentlich nicht zu jeder Zeit jedes Team wischen darf. Im Rahmen der aktuellen Prototypentwicklung soll es jedoch zunächst ausreichen, durch einfache Touch-Bewegungen auf dem Spielfeld das Wischen zu ermöglichen und es bzw. den daraus resultierende Effekt auf die Laufbahn der Steine zu visualisieren. Das Wischen wird also zunächst keiner Regelprüfung unterzogen, d.h. alle Wischaktionen auf dem Spielfeld, egal von welchem Spieler/Team, wirken sich auf alle sich bewegenden Steine aus und sollten zu jeder Zeit möglich sein.

5.4 Core-Mechanics

Die Kernmechanismen bilden die Spielregeln, Logik und das Verhalten der Elemente in der Spielwelt ab. Sie halten den Status über den aktuellen Spielstand und implementieren die Simulation der Welt und den Fortschritt des Spielablaufs. Im Einzelnen gehören die im Folgenden beschriebenen Aufgaben zu den Core-Mechanics:

- **Verwaltung des Spielstandes**: Dies umfasst den Punktestand der gespielten Ends beider Mannschaften, sowie den Status der aktuellen Spielphase (Bestimmung des aktuellen Ends, des aktuellen Spielers, der am Zug ist, bzw. der Nummer des Steins, der gespielt wird, und des „Hammer“-Spieler des Ends. Aus diesen Informationen kann ermittelt

werden, welcher Spieler im nächsten End beginnt, ob ein End oder das Spiel beendet ist und wer der Sieger ist.

- **Auswertung und Vergabe von Punkten**: Die punktbezogene Auswertung eines Ends, Die Punktevergabe anhand der Positionen der Steine auf dem Rink und die Ermittlung des Siegers sind in den Core-Mechanics implementiert.

- **Simulation des physikalischen Raums** und der interaktiven Spielelemente in Abhängigkeit von **der (Real-)Zeit** insbesondere in der Phase des „Steinspielens“. Zum Einsatz kommt hierbei einfache Physik aus dem Bereich der Mechanik, insbesondere aus der Kinetik.

- Implementation und Simulation der **Aktion „Stein spielen“**: Simuliert werden soll die Massenträgheit des Steins in der Bewegung, seine Gleitreibung auf der Eisfläche und die damit verbundene Verzögerung. Weiterhin sollen die Auswirkungen des Curl-Effekts abgebildet werden, also eine seitliche Translation des Stein in Abhängigkeit von der Rotationsgeschwindigkeit und -richtung beschrieben werden. Die Masse des Steins, der Reibungszahl und die Stärke des Curl-Effekts sollen zunächst mit realen Einheiten und Werten versehen und getestet werden. Auch die Simulation der Kollision zweier Steine soll realistisch abgebildet werden, in dem die resultierenden Kräfte in Abhängigkeit von der Masse, der Beschleunigung und dem Winkel der Kollision der involvierten Steine verteilt werden. Dies sollte ebenso für den Drehimpuls gelten, der sich in Abhängigkeit von der Rotationsgeschwindigkeit der Steine überträgt. Die Haftreibung eines Steins kann grundsätzlich vernachlässigt werden, da sie sich aufgrund der Beschaffenheit der Flächen (Granit und Eis) kaum zur Gleitreibung unterscheidet.

- Implementation und Simulation der **Aktion „Wischen“**: Das Wischen hat insbesondere den Effekt, dass die Reibung des Steins an der bearbeiteten Stelle des „Eises“, je nach Intensität, Dauer und Geschwindigkeit des Wischvorgangs, temporär vermindert wird. D.h. insbesondere je länger an einer Stelle gewischt wird, verringert sich bis zu einem gewissen Grad der Reibungswiderstand zwischen Stein und Spielfeld. Nach einer Weile gefriert die Stelle auf dem Eis wieder und der normale Reibungswiderstand stellt sich wieder her. Die Rate, mit der sich der Widerstand durch Wischen vermindert und wieder herstellt, sowie der Grad der Auswirkung des Effekts auf den Stein, sollte zunächst mit geschätzten Werten aus der Realität belegt und anhand des Prototyps getestet werden.

- Ermittlung des **Zustands** und Behandlung von **Status-Übergängen** in Spielablauf und Simulation: Insbesondere während des Simulationsvorgangs muss das Eintreten verschiedener Ereignisse, z. B. wenn der aktuelle Spielstein die „Hog“-Linie überquert oder außerhalb des Spielfelds gerät, erkannt und behandelt werden. Erst dadurch kann die Überprüfung der Regeln und das Voranschreiten im Spielablauf gewährleistet werden. Die Ermittlung konkreter Zustände der Spielelemente im jeweils aktuellen Kontext des Spielablaufs (z. B. Distanz eines Steins zum „Button“) kann darüber hinaus bei der Visualisierung oder als Feedback für den Spieler notwendig werden.

Zugunsten des Gameplay und um das Spiel generell kurzweilig zu gestalten, sollten schließlich einige Abwandlungen zum offiziellen Regelwerk vorgenommen werden:

- Der Stein sollte frei bis zur ersten „Hog“-Linie geführt, d.h. egal von welcher Seite (links

oder rechts der Center-Linie), und auch nach dem Loslassen wieder berührt werden dürfen, solange er die Hog-Linie noch nicht überschritten hat

- Das Ausspielen von 10 Ends kann mit jeweils insgesamt 16 Steinen sehr lange dauern. Für das kurzweilige Spiel eignen sich eher 2 oder 4 Ends. In einer finalen Version sollte die Anzahl der Ends auch von den Spielern einstellbar sein.
- Die Teilnahme von insgesamt 8 Spielern gleichzeitig am Surface ist unrealistisch und wenig praktikabel. Es sollten 2-4 Spieler das Spiel gut spielen können
- Dementsprechend könnte die Anzahl der Steine pro Team und End auch auf 4 (evtl. 6 bei zwei Spielern pro Team) reduziert werden.

5.5 User-Interface

Die grundlegende Problematik der Darstellungsperspektive, in der die Spielwelt in ihren Dimensionen dem Spieler präsentiert wird, wurde bereits eingangs und im Grobkonzept in Abschnitt 4.4 kurz erläutert. In den nächsten beiden Abschnitten zur Gestaltung des User-Interface sollen insbesondere die visuellen Elemente und ihre Präsentation näher beschrieben und diskutiert werden.

5.5.1 Interaktionsmodell und Perspektive

Aufgrund der Größe der Spielwelt ist diese prinzipiell nicht in einer Top-Down-Ansicht vollständig darstellbar („Single-Screen“) *und gleichzeitig* (gut) spielbar, wenn man die realen Größenverhältnisse von Stein und Spielfeld beibehält. Beim ausreichend weiten Herauszoomen würde der Stein allerdings zu klein abgebildet werden.

Die dreidimensionale Darstellung aus einem schrägen Betrachtungswinkel schafft zwar wesentlich mehr Raum zur Visualisierung der Spielwelt, da aber das Bild für einen Betrachter auf der gegenüberliegenden Seite des Surface auf dem Kopf steht, ist durch die Perspektive die räumliche Darstellung für ihn kaum zu erkennen. Fälschlicherweise erscheinen für den Betrachter nahe Objekte kleiner, als entfernte; die räumliche Perspektive ist zerstört. Die folgende Abbildung 25 veranschaulicht diesen Sachverhalt: Die Skizzen a) und c) zeigen jeweils einen Ausschnitt des Zielgebietes, a) aus einer schrägen Perspektive und c) in einer Draufsicht von oben (zweidimensional). Skizzen b) und d) zeigen dieselben Abbildungen wie a) und c), nur von der anderen Seite des Tisches.

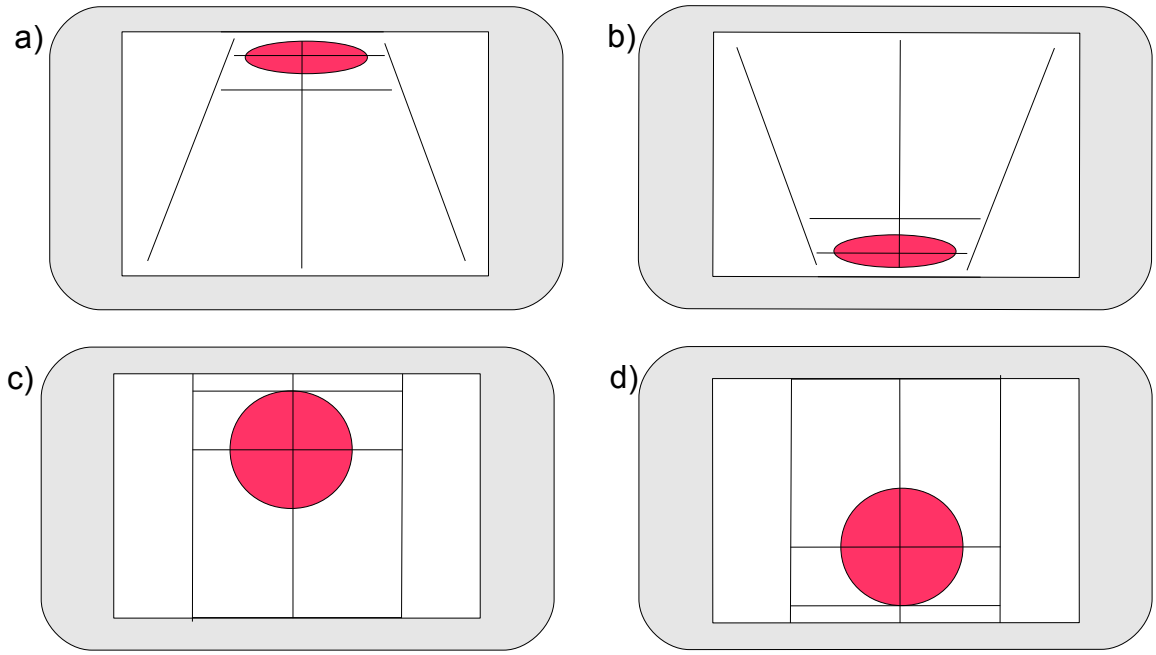

Abbildung 25: Darstellung des Spielfelds von verschiedenen Perspektiven und Seiten

Zum Zeitpunkt des Steinspielens hat nur ein Spieler (auf einer Seite des Tisches) Kontrolle über die Steuerung (des Steins). Dabei benötigt er auch direkte Sicht auf das Zielfeld, um seinen Stein dort präzise platzieren zu können, was einen recht flachen Sichtwinkel produziert. Der Sichtwinkel sollte stets so steil wie möglich sein, um gleichzeitig den Stein und das Ziel anzuzeigen, auch wenn der Stein bereits läuft. Eine flexible Kamerafahrt mit dem Stein kann dies bewerkstelligen. Dieser Sachverhalt sollte anhand eines Prototyps veranschaulicht und geprüft werden.

Im Sinne der Lösung dieses Zielkonfliktes (realitätsgetreue Abbildung, 360°-View/-Interaktion und Spielbarkeit) könnten alternativ auch die Maßstäbe und Verhältnisse der Spielwelt verändert werden, jedoch könnte dies Folgen für das Gameplay haben, die schwer abzuschätzen sind. Es müsste ebenso überprüft werden, ob die Herausforderungen bestehen und die Aktionen noch attraktiv (nicht zu leicht/schwer) bleiben.

Eine andere denkbare „Lösung“ wäre eine „Splitscreen“-Darstellung, also die Trennung der Sichtbereiche für beide Seiten. Dies halbiert allerdings die verfügbare Darstellungsfläche, schafft auch keine 360°-View und sollte nach „User-Experience-Guidelines“ vermieden werden.

Grundsätzlich wäre eine vertikale Nutzung der Darstellungsfläche möglich, zumal diese Ausrichtung dem Seitenverhältnis des Spielfeldes entsprechen und so mehr Fläche davon gleichzeitig darstellbar würde. Allerdings ändert sich damit wahrscheinlich auch die Position der Spieler um den Tisch (es sei denn, sie würden seitwärts spielen) und damit auch ihr Interaktionsradius, wie Abbildung 26 veranschaulicht. Inwiefern diese Lösung praktikabel/besser ist, sollte anhand von Prototypen und Usability-Tests überprüft werden.

Unabhängig davon, ob eine horizontale oder vertikale Ausrichtung gewählt wird, muss, sofern sich die beiden Teams gegenüber sitzen, nach jedem gespielten Stein die Ausrichtung des Spielfelds gedreht werden, so dass Start und Zielposition dem aktuellen Spieler entsprechend ausgerichtet ist.

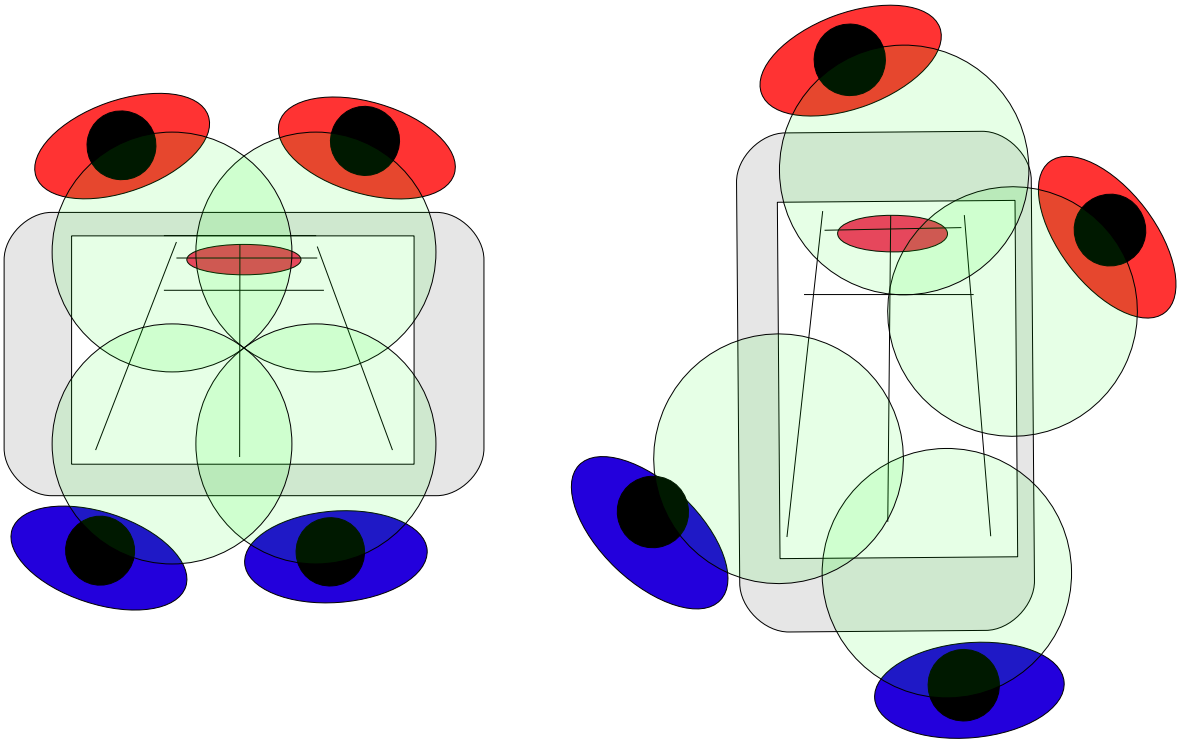

Abbildung 26: Horizontale und vertikale Ausgabe und Interaktionsraum der Spieler

5.5.2 Visuelle Elemente und grafische Effekte

Für die visuelle Darstellung des Spiels wird vornehmlich das Spielfeld abgebildet, auf dem alle Handlungen der Spieler stattfinden. Dies soll naturgetreu nachgebildet werden mit den Zielkreisen, Linien und Markierungen, wie es in Abbildung 22 gezeigt wird. Dabei sollten hochwertige Texturen benutzt werden, die auch bei näherem Heranzoomen mit der Kamera ausreichend hoch aufgelöst sind. Des Weiteren können auch Muster und Strukturen im Eis mittels Texturen dargestellt werden. Um den visuellen Eindruck der glatten Oberfläche zu verstärken, könnten Lichteffekte wie die Spiegelung der Deckenbeleuchtung oder Reflexionen der Spielsteine auf der Eisoberfläche Anwendung finden.

Zur Visualisierung der Spielsteine sind diese ebenso als 3D-Modell zur Verfügung zu stellen, bzw. müssen modelliert werden. Auch hierfür sollte hochwertige Texturen z. B. zur Nachbildung des Granit-Materials verwendet werden. Zur Unterscheidung der Spielerzugehörigkeit sind diese bspw. in den gut sichtbaren und unterscheidbaren Curling-Standardfarben Rot, Blau oder Gelb darzustellen. Neben der Spiegelung in Eisfläche können auch die Steine Lichtreflexionen u. Ä. aufweisen. Visuelles Feedback und Indikation besonderer Ereignisse kann am Spielstein z. B. durch Ein- und Aus-“faden“, Blinken oder Ähnliches realisiert werden.

Visualisiert werden soll auch der Effekt des Wischens. Dabei sollte der Zustand des aktuellen „Wischgrades“, also wie sehr eine Stelle bereits angewärmt ist, auf der Spielfeldfläche direkt sichtbar sein.

Weiterhin sollten zumindest nach Ends und zum Spielende die aktuellen Punktestände ausgegeben werden. Während eines Ends ist der aktuelle Spieler bereits durch die Farbe seines Spielsteins gekennzeichnet. Darüber hinaus könnte zu Beginn jedes Zuges eine Einblendung erscheinen, welches Team am Zug ist und der wievielte Stein im End gespielt wird.

Damit der Spieler seinen Zug/seine Ablage besser einschätzen kann, könnte eine so genannte „Mini-Map“ zum Einsatz kommen, die eine vergrößerte Darstellung des Zielgebiets und der darin bereits enthaltenen Steine wiedergibt. Die Mini-Map sollte an einer Position platziert sein, wo sie nicht stört und sich gegebenenfalls ausblenden.

Entsprechend den Surface „User Experience Guidelines“, sollten alle visuellen Übergänge immer fließend sein, nichts sollte abrupt erscheinen oder verschwinden. D.h. auch die visuellen Komponenten des Spiels und Textinformation, usw. sollten animiert ein- oder ausgeblendet werden. Das trifft z. B. auf Spielsteine zu, die das Spielfeld verlassen, aber prinzipiell auch auf die interaktive Kamera, die nicht abrupt die Perspektive oder Position ändern, sondern vielmehr zum nächsten Kontext „gleiten“ sollte.

5.5.3 Audio

Die Verwendung von Ton und Musik (Sound) ist zur Unterstützung der Atmosphäre während des Spielens unerlässlich. Die „User-Experience-Guidelines“ schlagen, ähnlich der visuellen Gestaltung, Einfachheit und Minimalismus vor, zumal man nicht weiß, wie hoch der Geräuschpegel in der jeweiligen Spielumgebung ist. Hintergrundmusik ist so nicht zwingen notwendig, aber es könnte eine Geräuschkulisse bestehen, die an das Publikum in einer Eishalle erinnert. Insbesondere zu den Aktionen des Spielers sollte es Feedback durch Soundeffekte geben. Im Einzelnen sollte dies zumindest die folgenden sein:

- Sound beim Wischen: kann abhängig von der Geschwindigkeit in Höhe und/oder Lautstärke variieren.
- Das Gleiten von Steinen sollte ein schleifendes Geräusch ebenfalls in Abhängigkeit von ihrer Geschwindigkeit erzeugen.
- Die Kollision von Steinen sollte mit Soundeffekten belegt sein.

Zusätzlich könnten die Sound-Effekte in ihrer Lautstärke variieren, je nachdem wie weit die aktuelle Betrachtung (Anzeige) vom Ort der Geräuschentstehung entfernt ist, wie es in Spielen mit 3D-Welten üblich ist. Um den räumlichen Eindruck noch zu verstärken, kann die Ausgabe auch auf die Stereokanäle verteilt werden, je nachdem wo sich die Geräuschquelle zum linken oder rechten Lautsprecher der letztlichen Darstellung auf dem Surface befindet.

Weiterhin könnte auch Feedback über Sprache und andere Audio-Formen zu gewissen Spielständen und -situationen gegeben werden, z.B. durch

- Publikum, welches klatscht, schreit, raunt, etc.
- Ein Stadionsprecher, der aktuelle Spielstände und Ereignisse durchsagt

Da die Simulation die Umgebung einer Eishalle oder eines Stadions darstellt, könnten alle Sounds auch mit einem Hall- oder Echo-Effekt versehen, bzw. erzeugt werden.

6 Anwendung / Implementation

Nachdem in Abschnitt 5 bereits die grundlegenden fachlichen Aspekte der umzusetzenden Spielsoftware festgehalten wurde, soll in diesem Abschnitt die technische Realisierung des ersten funktionalen Prototyps beschrieben werden. Dazu wird zunächst in Abschnitt 6.1 die technische Architektur erläutert, die durch Hard- und Softwareumgebung und im speziellen durch das XNA-Framework gegeben ist. Daraus ergibt sich der wesentliche Aufbau der zu entwickelnden Komponenten und wie diese mit der XNA-Umgebung zusammenspielen. Aufbauend auf der Architektur erfolgt in Abschnitt 6.2 ein Systementwurf (Design), in dem die konkreten Komponenten für den Prototyp modelliert und erläutert werden. Der Systementwurf bildet die Basis für die technische Umsetzung, die im Anschluss durchzuführen ist, und in Abschnitt 6.3 näher beschrieben wird.

6.1 Technische Architektur

Das Surface als zugrunde liegende Zielplattform wurde in seinen Hard- und Softwarebestandteilen und den besonderen Anforderungen an die Entwicklung von Software bereits ausgiebig in den Grundlagen beschrieben (vergleiche Abschnitt 2.2.1 und 2.2.2.4). Da XNA für die konkrete Umsetzung eingesetzt wird, liefert es den Anwendungsrahmen und gibt die Strukturen vor, in die sich die zu erstellende Anwendung und ihre Komponenten eingliedern müssen.

6.1.1 Systemschichten der Zielplattform

Abbildung 27 zeigt ein Modell des Gesamtsystems und die Einteilung in die elementaren logischen Hard- und Softwareschichten. In der obersten Schicht in Blau dargestellt ist die XNA-Anwendung, also das prototypische Spiel. In Gelb dargestellt sind die Schichten, mit denen die Spielanwendungen direkt kommuniziert. Die Spielanwendung läuft im Rahmen des „Application Model“ (Extended Framework) ab und nutzt dessen Strukturen. Das Spiel kommuniziert weiterhin mit Objekten und Diensten aus der “Core-Framework“ Schicht, weiter darunter liegende Schichten bleiben transparent/verborgen. Zur Verarbeitung der Touch-Informationen auf dem Surface muss die Spielanwendung zusätzlich auf das Surface-SDK zurückgreifen, welches als Bibliothek (Library) in die XNA-Anwendung eingebunden wird.

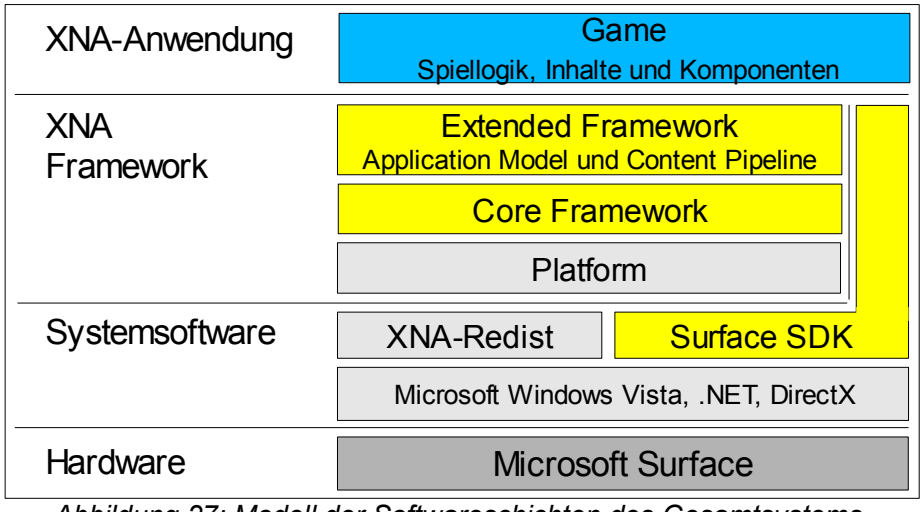

Abbildung 27: Modell der Softwareschichten des Gesamtsystems

Das Schichtenmodell in Abbildung 27 bietet Wiederverwendbarkeit von Code und Komponenten, Plattform- und Hardwareunabhängigkeit bei der konkreten Implementation und stellt einheitlich Dienste und Ressourcen zur Verarbeitung von Grafik, Sound und Inhalten bereit.

6.1.2 Komponenten-Modell von XNA

Die Klasse, die das Spiel und den Einstiegspunkt für die Implementation eigener Anwendungslogik darstellt, ist von XNA vorgegeben und wird von der Klasse Microsoft.Xna.Framework.Game vererbt. Die Game-Klasse ist in Abbildung 28 mit ihren wesentlichen Eigenschaften und Methoden abgebildet. Sie bietet Zugriff auf Funktionen für die grafische Ausgabe über das „GraphicsDevice“ und den Dateizugriff auf Mediendaten wie Sound, Grafiken, usw. und bildet den Rahmen, in dem die komplette Logik der eigenen XNA-Anwendung stattfindet.

6.1.2.1 „Application Model“ - GameComponents, Gameloop und Services

Das „Application Model“ innerhalb des Extended Framework, stellt die Steuerung der „Game-Loop“ zur Verfügung, welche für die zyklische Ausführung der Routinen des Spiels sorgt. Dieser Zyklus ist in 2 Phasen unterteilt: Aktualisierung der Logik (Update) und Ausgabe (Draw), welche in der Game-Klasse als Hook-Methoden bereitgestellt und dort überschrieben werden können. Die Wiederholrate der Gameloop, also die Durchführungen pro Sekunde (Frames per Second), kann in der eigenen Game-Klasse festgelegt werden. Das XNA-Framework sorgt transparent für die Steuerung und Einhaltung der Aufrufe. Es kann dabei die i. Allg. länger andauernde Draw-Phase, aber auch die Update-Phase, innerhalb eines Zyklus auslassen, falls der Vorgang der grafischen Ausgabe (Rendering) zu lange dauert, bzw. das System mit den Berechnungen innerhalb eines Zyklus nicht alle Operationen durchführen kann, d.h. überlastet ist. Um die Echtzeitaspekte eines Spielablaufs zu wahren, wird bei jedem Update- und Draw-Vorgang die vergangene Echtzeit seit dem letzten Aufruf mitgeliefert, welche in die Berechnung des Spielfortschritts einkalkuliert werden kann.

Weiterhin gibt das „Application Model“ mit dem Konzept der „GameComponents“ ein Verhaltensmuster (Template-Pattern) vor, mit dessen Hilfe leicht eigene Komponenten entwickelt und in den Ablaufzyklus integriert werden können. Abbildung 28 zeigt ein Modell dieses Aufbaus: Im Wesentlichen beinhaltet die Game-Klasse eine Kollektion von GameComponents, die über die Implementation der Interfaces IUpdateable und IDrawable in den Update- und Draw-Zyklus eingehängt werden. Prinzipiell wird also zwischen „visuellen“ und „nicht visuellen“ Komponenten innerhalb der XNA-Anwendung unterschieden. Darüber hinaus können die eigenen Komponenten frei gestaltet und implementiert werden.

Über die „Services“-Schnittstelle der Game-Klasse besteht weiterhin die Möglichkeit, Objektinstanzen zur Laufzeit in Abhängigkeit von ihrem (Klassen-)Typ zu registrieren und so zentral verfügbar zu machen. Dieser Aufbau entspricht dem Singleton-Pattern und ermöglicht eine lose Kopplung der Instanzen von Anwendungs-Objekten und -Komponenten an die Game-Klasse, um sie für andere Komponenten als Dienst bereitzustellen. Die Eigenschaften „GraphicsDevice“ und „ContentManager“ der Game-Klasse sind bereits fest verdrahtete Dienste, die i. Allg. von den Spielkomponenten benutzt werden.

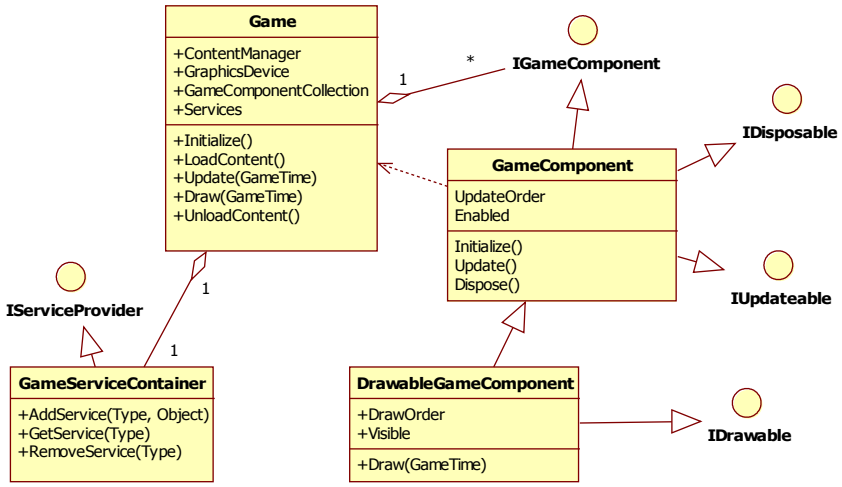

Abbildung 28: Modell der XNA „GameComponents“-Struktur

6.1.2.2 Vorgang der Gameloop

Abbildung 29 zeigt den Vorgang der Initialisierung (5) der eigenen XNA-Anwendung (Klasse „Game“), sowie den Ablauf der Gameloop (ab 15), der ständig wiederholt wird, solange die Anwendung aktiv ist.

Zum Zeitpunkt der Initialisierung (5) können GameComponents erzeugt und registriert werden, die damit in den Initialisierungsprozess einbezogen werden (7, 9, 11). Danach besteht die Möglichkeit, digitale Inhalte und Materialien laden und initialisieren zu lassen (13). Weiterhin ist anhand der drei verschiedenen Typen von GameComponents der Prozess der Gameloop in Abhängigkeit von den implementierten Schnittstellen veranschaulicht (18, 20, 24), der automatisch für registrierte und aktivierte Komponenten aufgerufen wird.

In Orange eingezeichnet sind die Klassen, die vom XNA-Framwork vorgegeben und deren Funktionalitäten vererbt werden. Die Abbildung veranschaulicht, wie die Aktualisierung und Ausgabe registrierter Komponenten automatisch geschieht. Dabei hat man zu jeder dieser Phasen Kontrolle über diesen Vorgang, da es spezielle Einsprungspunkte für die eigene Klasse „Game“ gibt (2, 5, 13, 16, 22), die im Diagramm blau gefärbt ist.

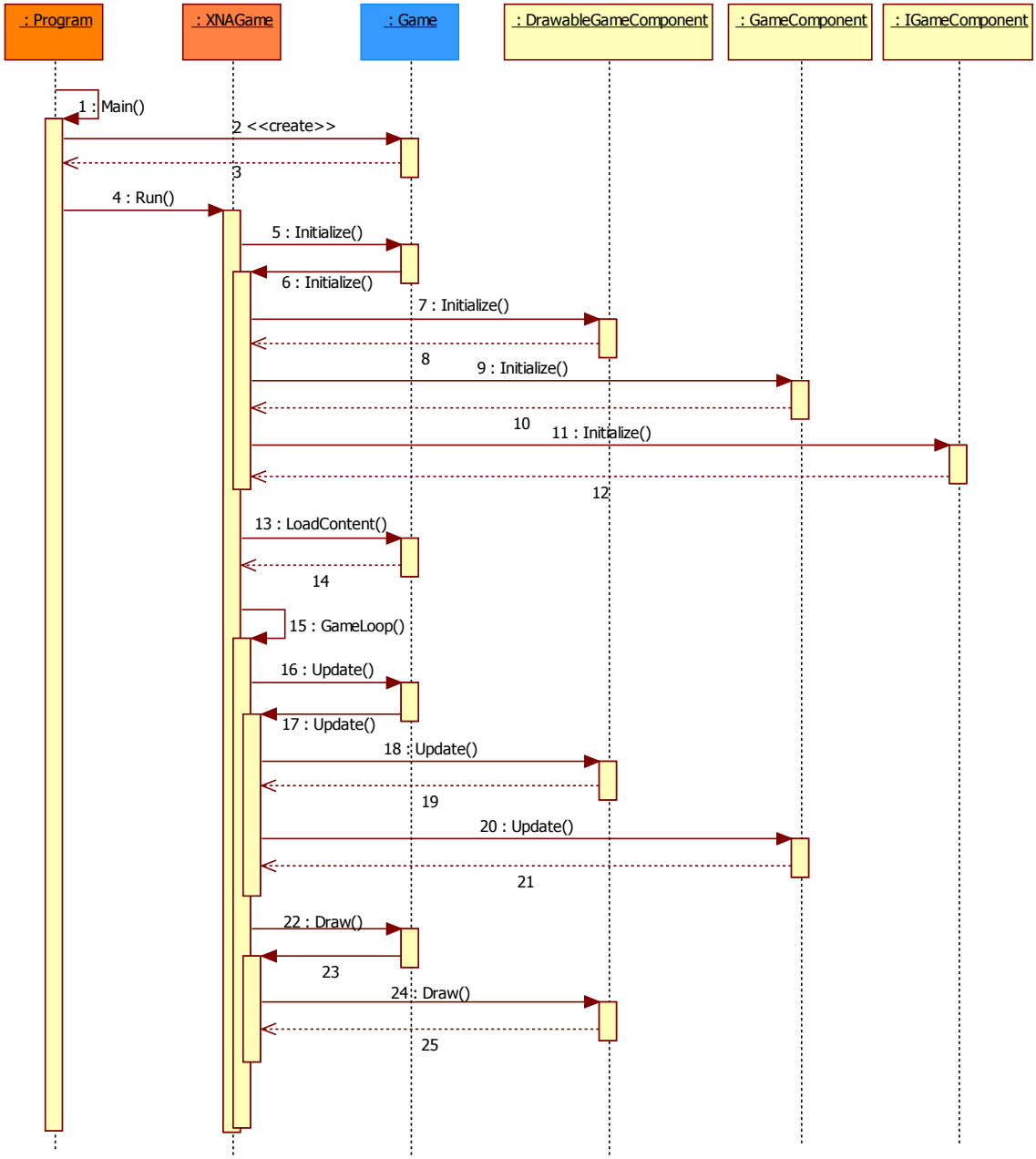

Abbildung 29: Vorgang der Initialisierung und Ablauf der Gameloop

6.1.3 Komponenten des Surface SDK

Wie beschrieben, müssen zur Nutzung der Dienste und Strukturen des Surface-SDK die benötigten Bibliotheken in die XNA- Anwendung eingebunden werden. Unter XNA sind vornehmlich die Komponenten aus den .NET Namensräumen „Microsoft.Surface“ und „Microsoft.Surface.Core“ nutzbar.

Die Funktionen der Klasse Microsoft.Surface.ApplicationLauncher stehen für Surface-XNA-Projekte direkt zur Nutzung bereit. Eine Instanz des ApplicationLauncher arbeitet im Hintergrund als Prozessmanager für alle Anwendungen, die aktuell auf dem Surface laufen, und kann die Eingabe auf ein Anwendungsfenster fokussieren, bzw. dies auf Systemereignisse hin informieren.

Die Klassen aus dem Namensraum Microsoft.Surface.Core beinhalten vornehmlich Strukturen und Funktionen zur Verarbeitung der Berührungsinformationen. Die Klasse „Contact“ hält mit einer Instanz die Informationen zu Position, Größe (Fläche) und Ausrichtung (Rotation) einer Berührung bereit. Des Weiteren ist für jeden Kontakt festgelegt, ob ein Blob, ein Finger oder ein ID-Tag erkannt wurde. Einen Schnappschuss der Kontaktinformationen zum Zeitpunkt der Abfrage erhält man über die Struktur ReadOnlyContactsCollection des Microsoft.Surface.Core Namensraumes. Das bedeutet, es gibt für XNA keine direkte Ereignisverwaltung zur Verarbeitung von Kontaktinformationen durch die Nutzung des Surface-SDK.

6.2 Softwareentwurf

Mit den gegebenen Informationen aus den vorangegangen Abschnitten kann nun ein grundlegender Entwurf erstellt werden, welcher innerhalb des Prototyps exemplarisch umzusetzen ist. Da die Software aufgrund der gegebenen Anforderungen agil zu entwickeln ist, soll die Modellierung zunächst nur die wesentlichen Klassen und deren Aufgaben umfassen.

6.2.1 Umfang und Ablauf des Prototyps

Besonderer Fokus bei der Umsetzung soll zunächst auf den speziellen Anforderungen bezüglich der Benutzerinteraktion und der Darstellungsperspektive liegen, die im Spielkonzept zu Curling in Abschnitt 5.5.1 erläutert wurden. Dazu ist es notwendig, zumindest den rudimentären Spielablauf, den Vorgang des Stein-Spielens und die Visualisierung der Spielelemente aus der definierten Perspektive abzubilden. Im Einzelnen sollen daher die folgenden Szenarien im Prototyp realisiert werden:

- Interaktion des Steinspielens durch Direct-Touch: Von der Startposition aus soll ein Spielstein durch Touch zu bewegen und in Richtung des Zielfeldes zu beschleunigen sein. Der Stein sollte, bei freier Bewegung, Massenträgheit und Reibungswiderstand aufweisen.
- Interaktion des Wischens auf dem Spielfeld durch Direct- und Multi-Touch: Durch Berührung des Spielfeldes soll der Effekt des Wischens ausgelöst werden, wie in dem Spielkonzept in Abschnitt 5.3 beschrieben.
- Interaktive Kamerafahrt: Bei der Interaktion mit dem Stein und auch während der Gleitphase soll eine Kamerafahrt den aktiven Stein verfolgen, wie es im Spielkonzept unter Abschnitt 5.5.1 beschrieben ist.

Zur Umsetzung dieser Szenarien ist die Visualisierung des Spielfeldes und der Spielsteine in Form einer 3D-Welt notwendig. Außerdem bedarf es einer flexiblen Kamera, von deren Position und Sichtwinkel aus die Spielwelt zur Ansicht projiziert wird. Durch die virtuelle Kamera sollte sich die Spielwelt in ihrer Darstellung auch um die Z-Achse drehen lassen (360°-Interaktion), wie es das Konzept nach jedem gespielten Stein zum Seiten-, bzw. Team-Wechsel vorsieht (vgl. Abschnitt 5.5.1).

Weiterhin sollte der grundlegende Spielablauf implementiert werden, der vornehmlich für die andauernde Simulation und gegebenenfalls das Zurücksetzen der Umgebung sorgt, um die Interaktion mit dem Stein und die virtuelle Kamerafahrt testen zu können.

Für spätere Tests und Optimierungen des Gameplay sollte der komplette Spielablauf zu Curling und die physikalischen Verhalten wie Kollisionen weiter ausgebaut, sowie das visuelle Element der „Mini-Map“ prototypisch realisiert werden. Dies wird im ersten Prototyp zunächst nicht umgesetzt

werden, soll aber bereits in die Modellierung der Anwendungskomponenten einfließen.

6.2.2 Curling Komponenten

Die zu entwickelnden Komponenten zur Umsetzung der Anwendungslogik lassen sich anhand des Spielkonzepts identifizieren. Im Einzelnen sind dies insbesondere die Bestandteile der Core-Mechanics (vgl. Abschnitt 5.4):

- **„CurlingEngine“**: Organisiert den grundlegenden Spielablauf und hält den Status über den aktuellen Spielstand. Diese Komponente ist auch für die Zustandsübergänge und die andauernde Simulation der Welt verantwortlich. Sie bildet den Rahmen für die Implementation der Anwendungslogik des Curling-Spiels.
- **„Rink“**: Stellt die Repräsentation eines Spielfeldes dar und ist auch visuelle Komponente. Auf dem Spielfeld werden die Steine platziert und simuliert, d.h. die Rink-Klasse „weiß“, welche Steine sich gerade auf dem Spielfeld befinden und sorgt für die Aktualisierung und Kollisionsbehandlung der Steine. Die Interaktion des Benutzers mit dem Rink erfolgt insbesondere bei der „Wischen“-Funktionalität durch Multi-Touch.
- **„Stone“**: Stellt eine Einheit eines Spielsteins im Spiel dar und hat Eigenschaften wie Position (auf dem Rink), Geschwindigkeit, Rotation und Winkelgeschwindigkeit sowie Masse. Ein Spielstein bietet ebenfalls Benutzerinteraktion durch Touch und muss visualisiert werden.
- „**CameraActor**“: Bietet die Funktionalität, die virtuelle Kamera zur Projektion der Spielwelt in dem Kontext zu steuern, den die CurlingEngine-Komponente in Abhängigkeit vom Spielzustand vorgibt. Im Wesentlichen sollte dies die Position und Ausrichtung des aktuellen Spielsteins sein, den die Kamera aus günstigem Winkel und adäquater Distanz verfolgen soll.

Neben den Anwendungskomponenten sind einige technische Komponenten erforderlich, die unabhängig von der Anwendungslogik des Curling-Spiels grundlegende Dienste bereitstellen sollten, wie z.B. die Verwaltung der Kontaktinformationen. Diese stehen seitens des Surface-SDK nur als Schnappschuss für den jeweiligen aktuellen Zustand zum Zeitpunkt der Abfrage bereit und müssen ständig gepolled und aufbereitet werden, wofür die Klasse „InputManager“ verantwortlich sein soll.

Die Klasse „**InputManager**“ wird von GameComponent vererbt und in die automatische Gameloop eingehängt und soll so bei jedem Durchlauf den aktuellen Zustand der Berührungen aktualisieren. So kann eine einfache Ereignisverwaltung über neue, bestehende und verlorene Kontakte für jeden Durchgang des Gameloop-Zyklus realisiert werden. Weiterhin kann eine Instanz des InputManager als Service in der Game-Klasse registriert werden, um als zentraler Dienst von anderen Komponenten befragt zu werden.

Die Positionsangaben der Kontaktinformationen werden immer relativ zum Bildschirm in so genannten Screen-Koordinaten geliefert. Als zweidimensionales Koordinatensystem hat es den Ursprung in der linken oberen Ecke des Bildschirms (Window). Um zu erkennen, ob ein Touch ein interaktives Spielelement der dargestellten Welt betrifft, muss zunächst ermittelt werden, wo der Kontakt innerhalb der Welt stattgefunden hat. Bei der Visualisierung einer 3D-Welt ist dies von der aktuellen Kameraposition und -perspektive abhängig, aus der die Projektion geschieht. Daher soll als weitere Komponente der „ViewManager“ für die Verwaltung der Projektionseinstellungen und

3D-Modelle der interaktiven Elemente der Spielwelt sorgen.

Die Klasse „**ViewManager**“ ist ebenfalls eine GameComponent. Sie hält die Projektionsinformation (Matrix) in Form eines „**Camera**“ Objektes, welches Position, Sichtwinkel/-richtung und Sichtweite der anzuzeigenden Szene definiert. Die ViewManager-Komponente benutzt den „InputManager“-Dienst, um durch Rückprojektion die Screen-Koordinaten aus den Kontaktinformationen in Weltkoordinaten umzuwandeln. Weiterhin soll sie Kenntnis haben über die aktuell auszugebenden interaktiven Elemente und deren Ausmaße in der 3D-Welt, um gegebenenfalls Touch-Ereignisse gesammelt an die Elemente weiterzuleiten.

Um ViewManager und InputManager flexibel und relativ unabhängig von der Anwendungslogik implementieren und nutzen zu können, werden außerdem die Klasse „Touch“ und „Actor“ eingeführt.

Die Klasse „**Touch**“ dient der Repräsentation einer Contact-Information und ihrer Behandlung in der spielinternen Logik. In einer Instanz können lokale und Welt-Koordinaten, auch über mehrere Gameloop-Zyklen hinweg, gehalten werden.

Die Klasse „**Actor**“ stellt eine abstrakte Form eines Spielelements dar und soll Basisklasse der interaktiven Elemente der Anwendungskomponenten sein. Im Wesentlichen soll sie der einheitlichen Behandlung von Darstellung und Verteilung der Touch-Ereignisse im ViewManager dienen.

Abbildung 30 zeigt die wesentlichen im Spiel involvierten Komponenten in Form eines Klassendiagramms. Die grün dargestellten Klassen sind die als erstes in diesem Abschnitt behandelten Anwendungskomponenten. In Beige eingefärbt sind die technischen Komponenten, die im Verlauf dieses Abschnitts beschrieben wurden. Orange eingefärbt sind die Basisklassen der GameComponents und die XNA-Game-Klasse, die XNA mitbringt und deren Funktionalitäten vererbt werden (vgl. Abschnitt 6.1.2). In Blau dargestellt ist die eigene Game-Klasse, in der für die Instanzierung und Initialisierung aller beteiligten Komponenten gesorgt werden muss (sofern diese nicht später/woanders erzeugt werden).

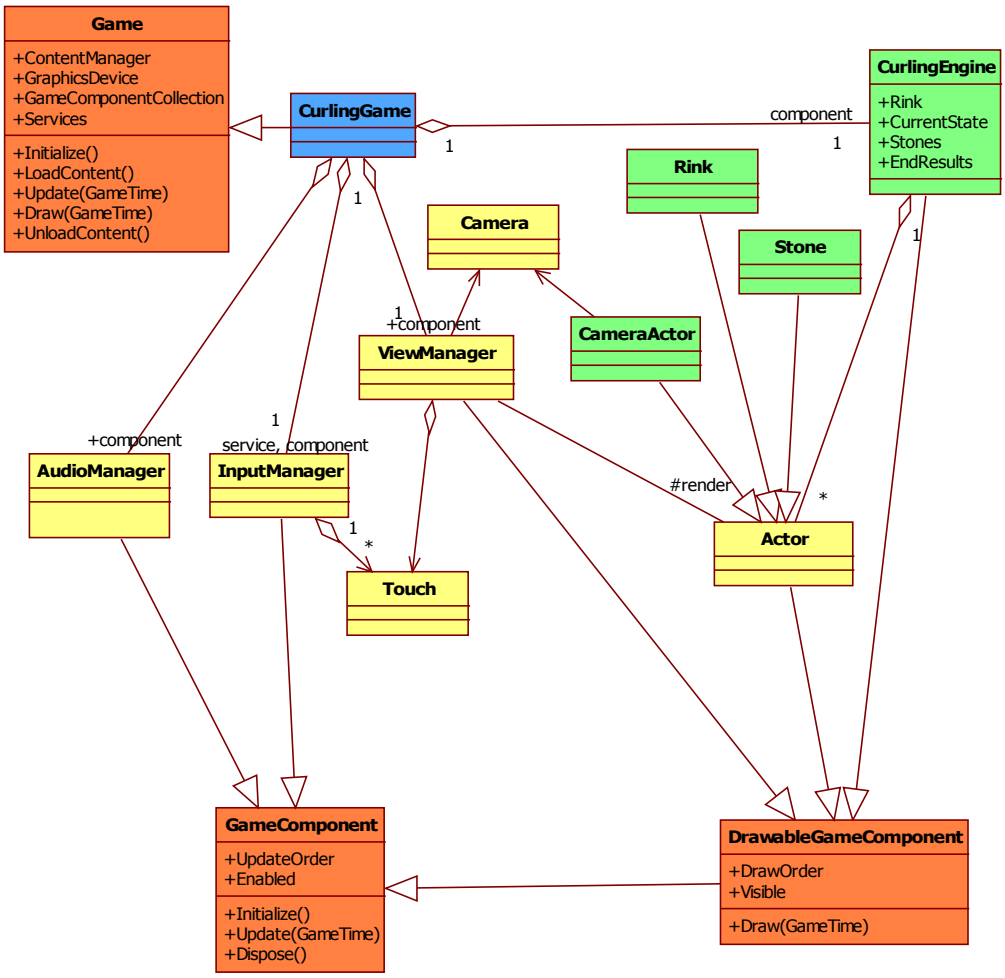

Abbildung 30: Curling Komponenten

6.2.3 Zusammenspiel der Komponenten

Innerhalb eines Zyklus der Gameloop müssen alle aktiven Komponenten ihre Update- und Draw-Routinen durchlaufen. Dabei muss eine GameComponent nicht beim Game registriert werden, wobei das Verhaltensmuster von Update und Draw dennoch genutzt werden kann, indem die Aufrufe von verantwortlichen/übergeordneten Komponenten „manuell“ abgesetzt werden. Dies ist vor allem dann sinnvoll, wenn mehr Kontrolle über Durchführung und Reihenfolge der Aktualisierung der Komponenten erreicht werden soll, oder z. B. die Aktualisierungen gruppiert passieren sollen.

6.2.3.1 Update-Sequenz der Gameloop

Abbildung 31 zeigt einen schematisierten Ablauf der Update-Sequenz der Gameloop für das Curling-Spiel und die wichtigsten Komponenten, die darin involviert sind. Der Vorgang wird wie folgt beschrieben:

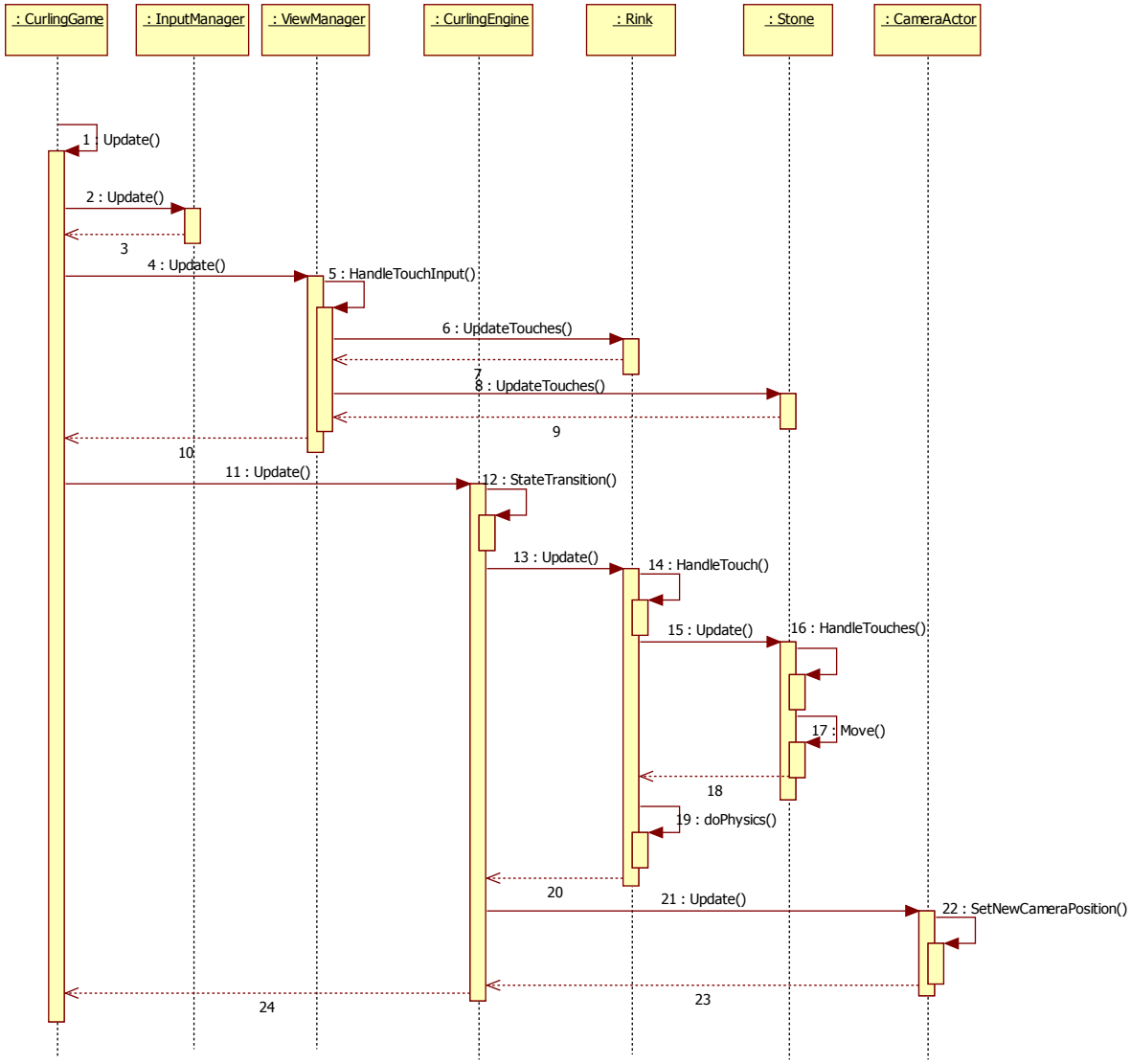

Abbildung 31: Sequenz des Update-Zyklus der Curling-Simulation

Bei (2) holt sich der InputManager den neusten Stand der Kontaktinformationen über das Surface-SDK ab und bereitet diese Daten in Form von Touches auf. Bei (4) wird die Update-Routine des View-Manager aufgerufen, die wiederum (5) die Behandlung der Kontaktinformationen aufnimmt. Dabei erfolgt die Berechnung und Aktualisierung der Weltkoordinaten unter Einbezug der Kamera-Einstellung, die beim letzten Render-Vorgang angewendet wurde. Weiterhin wird festgestellt, ob ein Touch ein Spielelement (Actor) berührt. Gegebenenfalls wird der entsprechende Actor über jedes dieser Ereignisse informiert (6+7), d.h. zu diesem Zeitpunkt werden die Informationen nur gesammelt und später zusammen per Actor (14+16) abgewickelt.

Bei (11) beginnt die Ausführung der Anwendungslogik für das Curling-Spiel. Die CurlingEngine selbst gibt dabei nur den Rahmen für die Spielelemente vor und implementiert den groben Spielablauf und Status, der zunächst in (12) geprüft wird. Vornehmlich wird die CurlingEngine dann das Rink-Objekt aktualisieren (13), das den eigentlichen Vorgang der Simulation der Welt implementiert. Dies verarbeitet zunächst seine Touch-Informationen (14) und aktualisiert danach alle Steine, die sich aktuell auf dem Spielbrett befinden (15+).

Für jeden Stein auf dem Rink beginnt die Verarbeitung der Kontaktereignisse (16), sofern welche vorliegen, die sich normalerweise in Veränderung der Positionsdaten oder der Beschleunigungswerte des Steins äußern. In (17) wird der Stein entsprechend seiner aktuellen Beschleunigungswerte bewegt, welche danach, zur Simulation des physikalischen Effekts des Reibungswiderstandes, ebenfalls angepasst werden.

Nachdem alle Steine für die Sequenz bewegt wurden, überprüft das Rink (19) die Positionen der Steine und ob eine Kollision zwischen ihnen vorliegt. Gegebenenfalls müssen die Positionen und Beschleunigungswerte nach Auswertung der Kollision neu gesetzt werden.

Abschließend aktualisiert die CurlingEngine (21) den CameraActor, der die Darstellungsperspektive günstig zum aktuellen Spielgeschehen in Szene setzen soll (22).

6.2.3.2 Draw-Sequenz der Gameloop

Die Draw-Phase ist in einer ähnlichen Kaskade aufgebaut, wie die Update-Aufrufe in der Update-Sequenz (vgl. Abbildung 31, ab (13)). Hierbei werden nun lediglich die Draw-Routinen der visuellen Komponenten aufgerufen. Diese schreiben ihre visuelle Ausgabe in Form von Zeichenanweisungen an das GraphicsDevice, welches nach Beendigung aller Draw-Routinen das Grafik-Subsystem aktualisieren und die Szene rendern lässt. Zu dieser Phase sollte keine „Anwendungslogik“ mehr ausgeführt werden, es wird bloß die visuelle Darstellung der Szene des aktuellen Spielzustands erzeugt.

Eine Besonderheit in der Draw-Phase ist der CameraActor, der keine direkte visuelle Ausgabe erzeugt. Stattdessen muss er spätestens zu Beginn dieser Phase die Kameraeinstellung des ViewManagers aktualisieren, mit der die aktuelle Szene projiziert wird. Dadurch nimmt der CameraActor indirekt Einfluss auf die Darstellung des gezeigten Ausschnitts und der Elemente. Zur Steuerung muss der CameraActor die Instanz der Kamera des ViewManagers kennen, bzw. der ViewManager, der dem CameraActor ebenfalls bekannt sein muss, könnte eine entsprechende Schnittstelle anbieten.

Ist die neue Kameraeinstellung übernommen, können alle visuellen Elemente ihre Draw-Routinen anwenden und damit ihre grafische Ausgabe erzeugen.

Das Ändern der Kameraperspektive hat zur Folge, dass sich die Weltkoordinaten bestehender Touch-Informationen ändern, obwohl die Position des Kontakts auf der Bildschirmoberfläche konstant ist. Auch wenn dies i. Allg. ein gewünschter Effekt ist, kann es die Umsetzung der Steuerung von Spielelementen erheblich verkomplizieren. Schließlich wird die Welt, bzw. der sichtbare Ausschnitt, unter dem Kontakt verschoben (oder auch gedreht), was zunächst seine Position relativ zum Element, das er betrifft, verändert. Auf die Problematik und den konkreten Ablauf der Steuerung der Spielsteine wird in Abschnitt 6.3.3 nochmals eingegangen.

6.3 Implementation des Prototyps

In diesem Abschnitt soll die Umsetzung der konkreten Funktionalitäten näher erklärt werden, die für die Realisierung des Prototyps besondere Relevanz haben. Dazu wurde der Abschnitt in 3 Bereiche geteilt, die jeweils die Implementationsdetails, bzw. die technische Realisierung zu

1. Spielablauf, Simulation und physikalisches Verhalten
2. Visualisierung der Spielwelt und Elemente sowie der Kamera
3. Interaktion durch Touch

beschreiben. Die Implementationsdetails werden weiterhin anhand der beteiligten Komponenten ausgeführt, die im Entwurf in Abschnitt 6.2.2 und 6.2.3 erläutert wurden.

6.3.1 Spielablauf und Simulation

Wie bereits angedeutet, ist die CurlingEngine-Komponente für die Implementation des wesentlichen Ablaufs des Curling-Spiels, der Spielregeln und des andauernden Fortschritts der Simulation verantwortlich. Zu den Aufgaben der Komponente gehören insbesondere:

- Die Bereitstellung der Spielkomponenten, welche sie selbst instanziert. Hierzu gehören im Wesentlichen das Spielfeld (Rink) und die Spielsteine (Stone) sowie die CameraActor-Komponente.
- Die Aktualisierung der Spielkomponenten innerhalb der Interaktions- und Simulationsphasen, was den eigentlichen Vorgang des Curlings abbildet.
- Die Verwaltung des Zustands des Spiels wie die Punktewertung der Ends, das aktuelle Team und End, Stein im End und insbesondere die aktuelle Phase innerhalb des Spielablaufs.
- Das Überprüfen des aktuellen Spielzustands und gegebenenfalls der Übergang in die nächste Phase des Spielablaufs.

Für die Implementation der CurlingEngine-Komponente bietet sich als Lösung ein Zustandsautomat an, da der wesentliche Vorgang innerhalb der Klasse das Verwalten des Spielstatus und der Übergänge im Ablauf ist.

6.3.1.1 Spielzustände und Übergänge

Abbildung 32 veranschaulicht die identifizierten Zustände in der Simulation des gesamten Spielablaufs einer Curling-Partie, die innerhalb der CurlingEngine Unterscheidung finden. Die Zustände, ihre Aufgabe und die Bedingungen zum Wechsel (Übergang) werden im Weiteren wie folgt beschrieben:

- „**Starting**“: Der Anfangszustand setzt zunächst den aktuell zu spielenden Spielstein auf das Spielfeld. Weiterhin besteht zu diesem Zeitpunkt Gelegenheit für einen visuellen Übergang, beispielsweise einer Kamerafahrt zu diesem Element, der Anzeige des aktuellen Teams und dergleichen mehr. Ist dieser Vorgang beendet, kann der Status in den nächsten Zustand „Curling“ übergehen.
- „**Curling**“ und „**Simulating Move**“: Beide Zustände zusammen bilden den Hauptvorgang

der eigentlichen Simulation, der in den Abschnitten 6.3.1.2 ff beschrieben wird. Dabei unterscheiden sich die beiden Zustände im Wesentlichen darin, dass zum Zeitpunkt des „Curling“ der Spieler noch Einfluss auf den aktiven Stein hat und zugehörigen Kontaktinformationen verarbeitet werden. Diese logische Trennung könnte auch anders abgebildet werden, prinzipiell soll zusätzliche Zustand dazu dienen, diesen besonderen Umstand nach Außen kenntlich zu machen, um z. B. eine andere Kamerafahrt oder andere grafische Ausgaben zu realisieren.

Die Implementation der Spielregeln entscheidet, wann der Übergang von „Curling“ zu „Simulating Move“ stattfindet. Für die hier angestrebte Lösung soll zunächst eine freie Steuerung des Steins im Startbereich des Spielfeldes möglich sein. Bis zur ersten Hog-Linie soll der Spieler diesen frei manövrieren können, darüber hinaus soll die Bewegung des Steins mit seiner zuletzt bekannten Beschleunigung fortgesetzt werden. Es muss also überprüft werden, ob der aktive Stein die Linie überschritten hat.
Die Phase „Simulating Move“ endet schließlich, wenn alle Steine, die sich auf dem Spielfeld befinden, sich nicht mehr bewegen, bzw. ihre Beschleunigungswerte Null sind. Steine, die außerhalb des Spielfelds geraten, werden entfernt, was ebenfalls während dieser Phase überprüft und durchgeführt werden soll.

- „**Evaluating Move**“: Nachdem die Simulation der Steinbewegungen abgeschlossen ist, kann der aktuelle Zug ausgewertet werden. Dabei soll zunächst geprüft werden, ob ein End vorüber ist, also alle Spieler ihre Steine für das End gespielt haben. Ist dies der Fall, wird in den Zustand „EndFinished“ übergegangen. Ist das End noch nicht vorüber, ist das nächste Team mit dem nächsten Stein an der Reihe und der Vorgang beginnt wieder beim Zustand „Starting“.

- „**EndFinished**“: Ist ein End vorüber, werden zunächst der Gewinner und die Punkte des Ends ermittelt, was anhand der Reihenfolge der Distanzen der Spielsteine im House zum Button geschieht, wie es im Curling-Konzept beschrieben ist. Das Ergebnis für das End kann nun ausgegeben werden. Weiterhin gilt es zu überprüfen, ob bereits das letzte End gespielt wurde, woraufhin gegebenenfalls in den Zustand GameFinished übergegangen wird. Ist das Spiel noch nicht beendet, kann das nächste End begonnen werden. Dazu wird der Beginner ermittelt und das Spielfeld zurückgesetzt. Der Vorgang beginnt wieder beim Zustand „Starting“.

- „**GameFinished**“: Ist das Spiel beendet, d.h. sind alle Steine aller Ends gespielt, kann die Gesamtauswertung über alle gespielten Ends erfolgen. Dieser wird angezeigt und schließlich die komplette Spiellogik zurückgesetzt. Das Spiel beginnt wieder komplett von vorne beim ersten End und Stein im Zustand „Starting“.

Der Zustandsautomat und die Steuerung des Ablaufs soll in der Update-Sequenz der CurlingEngine implementiert und so ständig durchlaufen werden. Deswegen sollten die Berechnungen, die innerhalb eines kompletten Durchgangs stattfinden, nicht zu komplex sein, bzw. lange andauern, um eine flüssige Darstellung zu erreichen. Dasselbe gilt im Übrigen für die Draw-Sequenz der Gameloop.

Da die Curling-Engine (bisher) höchste Instanz zur Ausführung der Anwendungslogik des Prototyps ist, wurde kein Endzustand in dem Diagramm vermerkt. Die Anwendung läuft schließlich unendlich lange, bis der ApplicationLauncher sie pausiert oder beendet.

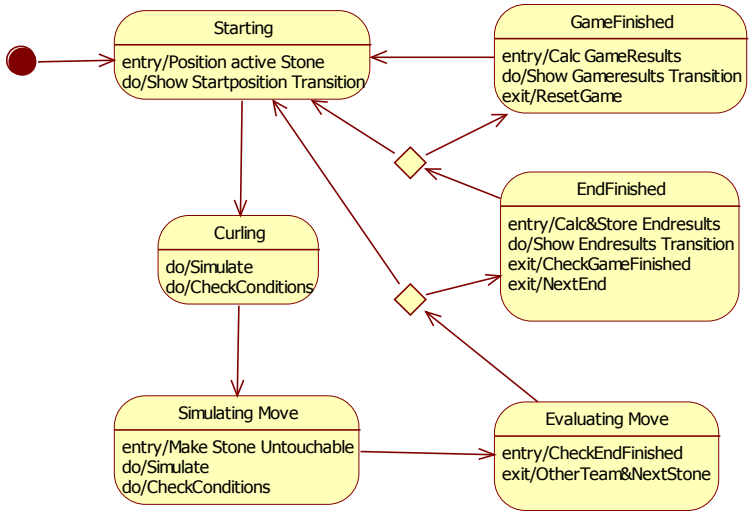

Abbildung 32: Spielzustände und Übergänge

6.3.1.2 Ablauf der Simulation

Die vergangene Zeit seit der letzten Update-Sequenz wird standardmäßig an die Update-Routinen der Komponenten übermittelt und ist in Sekunden (oder ms) angegeben. Um die Formeln der Mechanik anwenden zu können, sind einheitliche und sinnvolle Dimensionen und Maßeinheiten für die Konstanten und Variablen der Abbildung zu wählen. Dies sollte auch dabei unterstützen, mit Werten für Konstanten, z. B. der Reibungszahlen, zunächst ein reales Abbild des Verhaltens zu erzeugen.

In der Curling-Welt sind Einheiten in den Dimensionen von Metern (oder Zentimetern) für Strecken, Sekunden für Zeit und Kilogramm für Masse sinnvoll. Da die Spielsteine als einzig relevante Elemente alle dasselbe Gewicht aufweisen, kann die Eigenschaft der Masse in den Berechnungen prinzipiell vernachlässigt werden.

Wie bereits in Abschnitt 6.2.3.1 beschrieben, finden die Simulation und der Fortschritt im Ablauf des Steinspielens (Phasen „Curling“ und „Simulating Move“) vornehmlich in der Update-Sequenz der Rink-Komponente statt. Sie soll zunächst ihre Kontaktinformationen zur Erzeugung des Wischen-Effekts verarbeiten, wie in Abschnitt 6.3.1.4 erläutert wird. Weiterhin lässt sie alle sichtbaren Steine aktualisieren, die gegebenenfalls Kontaktinformationen verarbeiten (vgl. Abschnitt 6.3.3.3) und wiederum ihre Position entsprechend ihrer Geschwindigkeit und der vergangenen Zeit anpassen (vgl. Abschnitt 6.3.1.3). Dies stellt den ersten Teil der physikalischen Simulation dar.

Den zweiten Teil bildet die Kollisionserkennung und -behandlung, die anschließend durch das Rink geschieht. Im Wesentlichen besteht die Erkennung aus der Iteration über alle sich bewegenden Steine auf dem Spielfeld und der jeweiligen Prüfung, ob diese einen anderen Stein in ihrer neuen Position schneiden/berühren. Der Test ist hierbei trivial, da der zu prüfende Raum zweidimensional ist und die Spielsteine eine runde Form haben: Ist der Abstand der Mittelpunkte der Steine kleiner oder gleich ihrer addierten Radien, liegt eine Kollision vor.

Bei der Behandlung von Kollisionen sind die Geschwindigkeitsvektoren der Steine in Abhängigkeit von ihrer aktuellen Bewegung und dem Aufprallwinkel zu berechnen. Dasselbe gilt für die

Rotationsgeschwindigkeiten, die sich als Drehimpuls auf die Kollisionspartner auswirken.

Es stehen bekannte Algorithmen zur Kollisionserkennung und -behandlung zur freien Verfügung, insbesondere für den zweidimensionalen Raum und feste Körper. Für den Prototyp wird bei der Behandlung der Kollision auf eine frei verfügbare Lösung, bzw. Algorithmus, z. B. aus der XNA-Community zurückgegriffen.

Nach diesem Vorgang erfolgt noch die Einstellung der aktuellen Betrachterkamera.

6.3.1.3 Physik der Spielsteine

Position und Geschwindigkeit der Steine im Raum werden jeweils als 3D-Vektor dargestellt, wobei der Geschwindigkeitsvektor Richtung und Stärke der Bewegung vorgibt. Für die logische Abbildung des Steins wäre prinzipiell auch ein zweidimensionaler Positions- und Geschwindigkeitsvektor sowie ein Skalar für die Rotation ausreichend, da sich die Bewegungen des Steines nur auf die zweidimensionalen Ebene des Spielfelds beziehen. Da jedoch die Visualisierung dreidimensional realisiert werden soll, soll auch die logische Positionierung einheitlich im dreidimensionalen Raum erfolgen, so dass zumindest hierbei keine weiteren Transformationen notwendig werden.

Zur Ermittlung der neuen Position eines sich frei bewegenden Steins wird der Geschwindigkeitsvektor mit der vergangenen Zeit multipliziert, die in der Update-Routine mitgeliefert wird. Dies ergibt die in der Zwischenzeit zurückgelegte Strecke, welche zur ursprünglichen Position hinzu addiert wird:

Position + = Geschwindigkeit * vergangene Zeit.

Erfährt der Stein eine Drehbewegung, muss er, analog zur Position, seinen Rotationswinkel anpassen, wobei die Rotation im Winkelmaß (Radiant) und die Rotationsgeschwindigkeit in der Winkelgeschwindigkeit (Radiant/Sekunde) vorliegen:

Rotation + = Rotationsgeschwindigkeit * vergangene Zeit.

Weiterhin wirkt sich die Rotation eines Steins auch auf dessen Position aus, in dem der Effekt eines seitlichen Drifts entsteht, der aus der Reibung und gleichzeitigen Fortbewegung des Steins resultiert. Dieses Verhalten beschreibt eine Kurve in der Laufbahn des Steins, deren Krümmung von beiden Geschwindigkeiten und dem Reibungswert des Steins abhängig ist. Eine physikalisch korrekt simulierte Lösung ist an dieser Stelle nicht notwendig. Zunächst soll es ausreichen, eine Annäherungsfunktion für die Beschreibung der gewünschten Kurve zu finden, die später flexibel angepasst werden kann.

Die Richtung der Drift-Bewegung entspricht der Drehrichtung und verläuft orthogonal zur Bewegungsrichtung des Steins, den der Geschwindigkeitsvektor beschreibt. Der orthogonale Vektor zur Bewegungsrichtung und seine Länge soll für die X/Z-Ebene bestimmt werden, die Y-Komponente wird hierbei nicht verwendet. Dazu wird der Geschwindigkeitsvektor im Uhrzeigersinn gedreht und die Strecke ermittelt:

Weg_{ortho} = (- $Geschwindigkeit_Z$, 0 , $Geschwindigkeit_X$) * vergangene Zeit,

ergibt den maximal zurückzulegenden Weg auf der Orthogonalen, der weiterhin mit der normalisierten Rotationsgeschwindigkeit und einem konstanten Drift-Faktor multipliziert wird. Daraus ergibt sich eine Formel für die Positionsveränderung, bzw. die neue Position durch Drift:

Position += (Rotationsgeschwindigkeit / Rot_{Max} * vergangene Zeit) * DriftFaktor * Weg_{ortho},

wobei hier ein lineares Verhältnis zwischen Rotationsgeschwindigkeit und Drift-Effekt vorliegt. Die Konstante „DriftFaktor“ kann zur Justierung der Stärke des Effekts genutzt werden und hat einen Wertebereich zwischen 0 (keine Einwirkung) und 1 (volle Einwirkung). Weiterhin ist die Rotationsgeschwindigkeit auf einen Maximalwert (Rot_{Max}) zu begrenzen, was schon zuvor, z. B. bei der Behandlung und Verarbeitung der Kontaktinformationen, passieren kann.

Nachdem die Position und Rotation eines Steins neu berechnet wurden, wird außerdem seine Geschwindigkeit angepasst, um den Effekt des Widerstandes der Gleitreibung zwischen Stein und Eis zu simulieren. Haftreibung und Luftwiderstand sind dabei zu vernachlässigen. Der Widerstand der Gleitreibung ergibt sich auf ebener Oberfläche, nach der allgemeinen Kinetik, aus der Gewichtskraft G des Steins und der Reibungszahl μ_R : $F_R = \mu_R * G$.

Dies ist der Vektor der Kraft in Newton (N), die einer Bewegung beim Gleiten entgegenwirkt. Die Gewichtskraft G ergibt sich aus dem Produkt von Masse m (20 kg) und der konstanten Erdbeschleunigung (9,8 m/s²). Für μ_R gibt es empirische Werte für spezifische Materialien und Eigenschaften und deren Kombinationen. Im hier behandelten Fall Eis auf Granit kann man von einem realistischen Wertebereich von ca. 0,01 (Stahl auf Eis) bis 0,025 (Eis auf Eis) ausgehen. Das ergibt, in die Formel eingesetzt, minimale und maximale Werte für

$F_{Rmin} = 0{,}01 * 9{,}8 * 20\ N = 1{,}96\ N$, bzw.

$F_{Rmax} = 0{,}025 * 9{,}8 * 20\ N = 4{,}9\ N$.

Dieser Wert F_R, bzw. μ_R, ist insofern wichtig für das Gameplay und die Steuerung, als dass er bestimmt, wie stark der Stein in der Fahrt verzögert wird und damit, wie hoch die Startgeschwindigkeit des Steins sein muss, um das Ziel zu erreichen. Ist der Reibungswiderstand höher, muss der Stein am Anfang mehr beschleunigt werden, was den gesamten Ablauf der Gleitphase beschleunigt.

Da eine konstante Verzögerung vorliegt, kann leicht die Startgeschwindigkeit errechnet werden, die notwendig ist, um das Ziel zu erreichen. Die Verzögerung a_R ergibt sich aus Newtons Zweitem Satz (umgestellt):

$a_R = F_R / m$, eingesetzt:

$a_{min} = 2N / 20kg = 0{,}1\ m/s^2$

$a_{max} = 5N / 20kg = 0{,}25\ m/s^2$

Wie man sehen kann, ist die Masse hierbei zu vernachlässigen, da sich Gewichtskraft und. Um die Startgeschwindigkeit zu ermitteln, kann die folgende Formel angewendet werden:

$v_1 = SQRT (v_2^2 – 2 * a_R * s)$.

Die Endgeschwindigkeit v_2 soll, am Ende der Bewegung am Button, Null sein, und die zurückgelegte Strecke s misst von der ersten Hog-Linie bis zum Button (ca. 32m). Mit den Beispielwerten eingesetzt ergeben sich Startgeschwindigkeiten von

$v_{min} = SQRT (- 2 * - 0{,}1 * 32) = 2{,}5\ m/s = 9{,}1\ Km/h$ und

$v_{max} = SQRT (- 2 * - 0{,}25 * 32) = 4\ m/s = 14{,}4\ Km/h$.

Der Beschleunigungswert a_R wird hier invertiert, da es sich um eine Verzögerung (Gegenkraft) handelt. Die Dauer für das zurücklegen des Weges ergibt sich aus Startgeschwindigkeit und Verzögerung, also

$t_{min} = v_{min} / a_{min} = 2{,}5 / 0{,}1 = 25s$ und

$t_{max} = v_{max} / a_{max} = 4 / 0{,}25 = 16s.$

Um für einen Schritt die neue Geschwindigkeit des Steins zu ermitteln, können aktuelle Geschwindigkeit und konstante Verzögerung direkt verrechnet werden. Der Geschwindigkeitsunterschied ist das Produkt aus Verzögerung und vergangener Zeit:

Geschwindigkeit += Verzögerung * vergangene Zeit, oder

$v_2 - v_1 = -a_R * (t_2 – t_1)$.

Da die Geschwindigkei im Falle des Steins ein Vektor ist der die Richtung vorgibt, soll die Verzögerung im gleichen Maße auf die Komponenten des Vektors und entgegengesetzt seiner Richtung erfolgen. Dazu wird der normalisierte und invertierte Geschwindigkeitsvektors benutzt, der mit der Verzögerung skaliert wird.

Für die Rotationsverzögerung kann analog verfahren werden.

6.3.1.4 Simulation des „Wischen“

Die Wisch-Informationen müssen hinterlegt und auf das Spielfeld abgebildet werden. Diese Informationen soll daher das Spielfeld (Rink) selbst tragen, aktualisieren und gegebenenfalls Auskunft darüber geben.

Das Wischen hat, wie bereits im Konzeptteil beschrieben, Auswirkungen auf den Gleitvorgang eines Steines, welcher in Abschnitt 6.3.3.1 definiert wurde. Im Wesentlichen reduziert das Wischen den dort beschriebenen Wert der Reibungszahl μ_R, bzw. der Verzögerung a_R eines betroffenen Steines, was sich in einer längeren Gleit- und Rotationsphase äußert. Weiterhin reduziert sich der Einfluss des Drifts, so dass eine weniger gekrümmte Laufbahn entsteht, falls der Stein rotiert.

Zur Realisierung des Effekts muss also die Physik der Spielsteine erweitert werden. Sie müssen in ihrer Update-Phase die Information über den Grad der Wisch-Intensität in Bezug zu ihrer aktuellen Position beziehen und diese in die Kalkulation der Verzögerungswerte einfließen lassen. Dies kann mithilfe eines weiteren Faktors in den entsprechenden Formeln bewerkstelligt werden, der ähnlich zum beschriebenen Drift-Faktor, die Stärke des Effekts steuert. Für die Intensität der Wisch-Information kann, genauso wie beim Wisch-Faktor, von einem Wertebereich zwischen Null (kein Einfluss) und 1 (voller Einfluss) ausgegangen werden, so dass hierbei keine weitere Normalisierung notwendig ist.

Das Erzeugen von Wisch-Informationen geschieht durch Touch/Multi-Touch innerhalb des Spielfelds, wie es in Abschnitt 6.3.3.4 beschrieben ist. Dabei soll, je länger ein Touch an einer Stelle auf dem Spielfeld andauert, der Wert für diese Stelle hochgezählt werden.

In der Update-Sequenz des Rink müssen alle Wisch-Informationen iteriert und reduziert werden, um den Effekt des Wischens wieder abklingen zu lassen.

Die Wisch-Informationen und Werte müssen auf verschiedene Arten zugänglich sein:

- Zur Abfrage eines Werts über die Spielfeld-Koordinaten bei der Aktualisierung der Steine
- Zur Erhöhung eines Wertes an einer Spielfeld-Koordinate durch Touch
- Bei der stufenweisen Reduzierung aller vorhandenen Wisch-Informationswerte
- Als Gesamtstruktur zur grafischen Ausgabe

Für die Speicherung der Wisch-Informationen bietet sich - zur Abfrage mit Koordinaten und zur Visualisierung - ein zweidimensionales Raster an. Dieses sollte allerdings nicht zu viele Elemente enthalten, da die Daten häufig aktualisiert werden müssen. Ein Rastermaß von z. B. 15 cm sollte ausreichend sein, was aber bei der Spielfeldgröße von ca. 45 m x 4,5 m bereits 9.000 (300*30) Einheiten ergibt. Hilfreich könnte hier sein bei der Aktualisierung nur die Bereiche reduzieren zu müssen, die überhaupt erhöhte Werte tragen. Dazu wäre eine weitere Liste zur Verwaltung der Belegungen notwendig. Dies lohnt sich nur, wenn genug Speicherplatz zur Verfügung steht und nicht zu viele Felder im Raster gleichzeitig erhöht sind.

6.3.2 Visualisierung

Die Qualität der grafischen Elemente und überhaupt der visuellen Effekte ist für diesen Prototyp nicht weiter relevant. Auch auf die Ausgabe von Status und Punkteanzeigen sowie Textausgaben kann zunächst verzichtet werden. Es sollen jedoch das Spielfeld und die Spielsteine als solche erkennbar sein und die Größe und Dimensionen den realen Verhältnissen entsprechen.

Die grafische Ausgabe in XNA geschieht durch eine Schnittstelle über Direct3D und erlaubt dessen komfortable Nutzung im Wesentlichen über die Eigenschaft „GraphicsDevice“ der Game-Klasse. Direct3D sorgt für den Rendering-Vorgang einer Szene im dreidimensionalen Raum, die aus der Sicht und Perspektive einer virtuellen Kamera aus dargestellt (gerendert) wird.

Weiterhin können durch den ContentManager-Service von XNA Texturen und 3D-Modelle geladen und in den Draw-Sequenzen benutzt werden.

Die visuellen Komponenten müssen innerhalb der Draw-Sequenz ihre grafische Ausgabe an das GraphicsDevice in Form von Zeichenanweisungen für Linien und Polygone bzw. Flächen, die sich daraus ergeben, übermitteln.

6.3.2.1 Spielsteine

Ein Spielstein wird von der Stone-Komponente repräsentiert. Zur Visualisierung soll ein einfaches 3D-Modell erstellt und über den ContentManager geladen werden. Die Draw-Sequenz der Steine besteht im Wesentlichenim Wesentlichen aus der Iteration über alle Flächen dieses Modells, der Anwendung der Texturen und Effekte, etc. und ist weitgehend vorgegeben.

Eine Besonderheit ist die Rotation der Steine um die eigene Achse, was visualisiert werden soll. Dazu muss in der Draw-Sequenz eine Rotationsmatrix auf das 3D-Modell angewendet werden, die der aktuellen Rotation des Steins entspricht. Damit die Rotation auch um die Hochachse (Y) des Steins geschieht, muss dieser vor der Rotation an den Ursprung des Koordinatensystems verschoben und nach der Rotation wieder zurück verschoben werden.

6.3.2.2 Spielfeld

Das Spielfeld wird von der Rink-Komponenten dargestellt und soll zunächst als weiße Fläche erscheinen und im Ursprung des Koordinatensystem liegen. Weiterhin sollen das Zielgebiet mit den Kreisen als Textur und die Markierungslinien eingezeichnet werden. Beides kann mit wenigen Draw-Anweisungen an das GraphicsDevice geschehen.

Eine Besonderheit ist die Visualisierung der Wisch-Informationen, welche in 6.3.1.4 beschrieben sind. Eine Möglichkeit der Umsetzung, die auch am Prototyp versucht werden soll, ist die schnelle Umwandlung des Rasters der Wisch-Informationen in eine Textur, die auf das gesamte Spielfeld projiziert werden kann.

6.3.2.3 Kamerafahrt

Für den ersten Prototyp soll die Kamerafahrt möglichst einfach gestaltet sein, wie es in Abschnitt 5.5.1 beschrieben wurde. Sie verfolgt zunächst den Stein mittig aus einer Perspektive von schräg oben dahinter. Neigungswinkel und Abstand zum Stein sollten in Abhängigkeit von der z-Position (Tiefenachse) des Steins definierbar sein, so dass anhand des Prototypen eine Funktion entwickelt werden kann, die das gewünschte Verhalten für jede z-Position liefert.

Nachdem die Update-Sequenz der Spielelemente vorüber ist, bzw. bevor die nächste Draw-Phase beginnt, muss der CameraActor die Kamera des ViewManagers justieren, mit der die Szene gerendert wird. Dies soll in Abhängigkeit vom aktiven Stein geschehen, den die CurlingEngine dem CameraActor zuvor mitteilen muss, was in dem Zustand „Starting“ passiert.

6.3.2.4 Mini-Map

Die Mini-Map ist ein visuelles Element, das einen Bildausschnitt des Zielgebiets darstellen soll. XNA bietet hierzu die Möglichkeit, den so genannten Viewport einzustellen. Das ist der Bereich auf dem Bildschirm, in den eine 3D-Szene in der Draw-Sequenz gerendert wird. Zur Realisierung der Mini-Map soll nach der eigentlich Draw-Sequenz nochmals die Szene gerendert werden, wobei der Viewport zuvor auf den Bereich eingestellt wird, an dem die MiniMap erscheinen soll. Des Weiteren wird eine zweite Kameraposition für den Render-Vorgang der Mini-Map benötigt, nämlich eine Einstellung mit Draufsicht auf das Zielgebiet von oben. Alle Elemente der Spielwelt müssen mit der Sicht aus dieser „Mini-Map-Kamera“ gerendert werden. Nach dem Vorgang muss der Viewport der normalen Ansicht für die nächste Draw-Sequenz wiederhergestellt werden.

6.3.2.5 Visuelle Übergänge

Spiel- und Interaktionskonzept sehen vor, dass die visuelle Änderung des Zustands in der Spielwelt, oder eines Elements nicht abrupt geschieht, sondern ein visueller Übergang geschaffen wird. Dies wird beim Curling-Spiel insbesondere bei der Situation des Steinspielens durch die Kamerafahrt realisiert, welche durch den Prototyp zu veranschaulichen ist.

Für weitere Vorgänge im Spielablauf, z. B. der Anzeige der Punktestände, dem Entfernen eines Spielsteins oder dem Zurücksetzen des Spielfeldes sind ebenfalls flüssige visuelle Übergänge vorzusehen, diese sind aber zunächst nicht relevant für die Umsetzung des Prototyps.

Prinzipiell ist darauf zu achten, dass ein visueller Übergang vollständig geschieht und nicht plötzlich beendet wird. Dazu muss unter Umständen, bevor der eigentliche Spielablauf fortgesetzt werden kann, auf die Übergangsphase abgewartet werden, was die Kenntlichmachung des Zustandsübergangs erfordert. So ist dies bereits für den grundlegenden Spielablauf und die Zustände vorgesehen, wie in Abschnitt 6.3.1.1 beschrieben wird.

Ein Stein z. B. sollte später die Funktionalität haben, stufenweise ein- und auszublenden. Auf diesen Vorgang muss gewartet werden, bevor der Stein (auch logisch) vom Spielfeld genommen wird, da dies mit der Visualisierung des Elements verbunden ist.

6.3.3 Interaktion / Eingabe durch Touch

Der Prozess der Eingabeverarbeitung durch Touch wurde im groben Ablauf in den Abschnitten 6.2.2 und 6.2.3 beschrieben. Im Wesentlichen sind die technischen Komponenten „InputManager“ und „ViewManager“ dafür verantwortlich, zu Beginn einer Update-Sequenz den aktuellen Stand der Kontaktinformationen vom System zu beziehen. Im weiteren Verlauf werden die

Kontaktinformationen aufbereitet und den relevanten Komponenten zur weiteren Behandlung zur Verfügung gestellt.

6.3.3.1 InputManager- und Touch-Komponente

Der InputManager benutzt direkt die Klassen des Surface-SDK, um einen Schnappschuss des Standes der Kontaktinformationen zu erhalten. Diese Informationen bereitet er so auf, dass zunächst Listen von neuen, bestehenden (und zu aktualisierenden) und verlorenen Kontakten vorliegen. Da vom Surface-SDK bei jeder Abfrage neue Objektinstanzen der „Contact“-Klasse geliefert werden, muss der Abgleich von vorhandenen und neuen Kontaktinformationen anhand der vom Surface für jeden Kontakt generierten ID erfolgen. Um einen schnellen Zugriff für den Abgleich zu erreichen, soll eine Hash-Tabelle, bzw. ein Dictionary zur Speicherung der bestehenden Kontakte verwendet werden.

Für jeden neuen Kontakt erzeugt der InputManager eine Instanz eines Touch-Objekts, welches zur weiteren Verarbeitung der Kontaktinformation innerhalb der Anwendungslogik verwendet wird. Neben den wesentlichen Kontakt-Daten, die ein Contact-Objekt aus dem Surface-SDK enthält, werden weitere Werte, wie z. B. die Weltkoordinaten, initiale Rotation/Ausrichtung oder Änderungen zur letzten Update-Sequenz in einem Touch-Objekt gehalten.

Die Listen über die neuen, aktuellen und verschwundenen Kontakte stellen die Basis für den Aufbau einer einfachen Ereignisverwaltung. Mit ihrer Hilfe soll auf die elementaren Ereignisse Berührung (onTouch), Veränderung (updateTouch) und Loslassen (unTouch) eines Kontaktes reagiert werden.

Eine Instanz des InputManagers wird als Service in der Game-Klasse registriert und so zentral zur Verfügung gestellt. Weiterhin führt der InputManager zu jeder Update-Sequenz eine Aktualisierung und den erneuten Abgleich der Kontaktinformationen durch.

6.3.3.2 Viewmanager- und Actor-Komponente

Die Rolle des ViewManagers bei der Behandlung der Kontakt-Informationen und das Zusammenspiel mit den Actor-Komponenten wurde in Abschnitt 6.2.2 erläutert.

Da die Information der Tiefe/Höhe des Kontakts innerhalb des visualisierten dreidimensionalen Raums durch die Screen-Koordinaten nicht gegeben ist, muss diese ermittelt werden. Mit den Einstellungen der letzten Projektion (Kamera) können die Weltkoordinaten aus den zweidimensionalen Screen-Koordinaten berechnet werden. Die Funktion „UnProject“ des GraphicsDevice kann dies unter Angabe der Projektionseinstellungen bewerkstelligen und liefert den zurück projizierten Punkt „auf der Kameralinse“. Aus Startpunkt und Ziel kann weiterhin die Richtung des virtuellen Strahls durch den Raum berechnet werden, mit dem die Kollision der zuletzt gezeigten interaktiven Spielelemente geprüft wird. Dabei werden zur Kamera näher gelegene Objekte zuerst untersucht.

XNA bietet für die Abbildung des Strahls eine Klasse (Ray), die Ausgangspunkt und Richtung in zwei Vektoren speichert. Sie bietet die Möglichkeit zum Kollisionstest mit elementaren geometrischen Körpern wie Kugel (BoundingSphere), Quader (BoundingBox) und Ebene (Plane), die ebenfalls durch XNA implementiert sind. Zur Feststellung der Kollision des Strahls mit einem Spielelement sollte die Actor-Komponente über solch eine „Hülle“ verfügen, die als Eigenschaft im Weiteren BoundingObject genannt wird. Für die Abbildung des Steins und seine Bedienung soll eine Kugel als BoundingObject genutzt werden, für das Spielfeld ein flacher Quader oder eine

ganze Ebene. Die Einbeziehung der tatsächlichen Kontaktfläche oder ihre Ausrichtung findet hierbei nicht statt. Bei der Lösung wird lediglich der Mittelpunkt einer Kontaktfläche zur Ermittlung der Kollision genutzt.

Bei der Durchführung des Kollisionstests wird vom ViewManager für jeden neuen oder bestehenden und noch nicht mit einem Spielelement verbundenen Kontakt durch alle visualisierten Spielelemente iteriert. Dabei wird jeweils die Kollisionsabfrage mit dem Strahl durchgeführt, die im Erfolgsfall die Distanz (Radius) vom Ausgangspunkt des Strahls zum Punkt auf dem getroffenen BoundingObject liefert. Aus den Informationen des Strahls und der Distanz werden letztlich die Weltkoordinaten des Kontakts auf der Oberfläche des BoundingObjects berechnet (Strahl-Position + Richtung * Distanz) und im Touch-Objekt hinterlegt. Außerdem wird die Verbindung des Touches zum Spielelement verzeichnet.

Wird ein neuer Touch verarbeitet, besteht bei der erstmaligen Berührung (onTouch) eine Differenz (Offset) zwischen dem Bezugspunkt zur Positionierung des Elements und dem Berührungspunkt. Der Offset-Wert wird zusätzlich abgelegt, um bei beweglichen Spielelementen und Verschiebung der Kontakt-Position den Abstand wahren zu können. Bestehende Touch-Instanzen müssen vom ViewManager aktualisiert werden, da ebenfalls die Aktualisierung der Weltkoordinaten notwendig ist (updateTouch). Touches, die verloren gehen, müssen ebenso als Ereignis behandelt werden wie neue oder aktualisierte, nur dass die Touch-Informationen selbst keine Rolle mehr spielen. Zumindest müssen Spielelemente über das unTouch-Ereignis informiert werden, wofür der Eintrag der Verbindung zwischen Spielelement und Touch im ViewManager hinterlegt wird.

Die Actor-Klasse gibt für die drei Ereignisse (onTouch, updateTouch, unTouch) abstrakte Methoden vor, die in den Klassen der Spielelemente Rink und Stone implementieren werden können, um individuelles Verhalten zu realisieren. Die abstrakte Actor-Klasse gibt in ihrer Update-Sequenz die Behandlung und Abarbeitung der Kontaktereignisse vor und ruft für jeden einzelnen Kontakt die entsprechende Ereignisbehandlung in der konkreten Actor-Klasse auf.

Der beschriebene Vorgang liefert ein einfaches Ereignismodell zur elementaren Verarbeitung der Kontaktinformationen in den interaktiven Spielelementen.

6.3.3.3 Interaktion „Stein spielen“

Das physikalische Verhalten des Steins wurde in Abschnitt 6.3.1.3 ausführlich beschrieben. Bei der Steuerung des Steins während der Curling-Phase der Simulation spielen die physikalischen Eigenschaften zunächst jedoch keine Rolle. Der Spieler soll eine möglichst freie und direkte Kontrolle über den aktiven Stein besitzen.

Auf die Ereignisse onTouch und updateTouch hin soll die Position des Steins entsprechend angepasst werden. Dies kann zum einen direkt geschehen, indem die neue Position (und Rotation) des Steins entsprechend der Informationen des Touches gesetzt werden. Der Stein würde dann direkt immer mit dem Kontakt positioniert, egal wie sich die Perspektive oder der Kontakt im Wesentlichenändern.

Der Stein kann aber auch indirekt bewegt werden, indem durch die Touch-Information nicht die Position, sondern die Geschwindigkeit (oder Beschleunigung) des Steins beeinflusst wird. Die neue Positionierung des Steins mit dem Einfluss des Touch erfolgt dann erst in der regulären Update-Sequenz des Steins, in der das physikalische Verhalten weiterhin angewendet wird. Bei dem Ereignis unTouch setzt der Stein seine Bewegung dann mit der letzten Geschwindigkeit fort.

Für den Prototyp soll die indirekte Variante realisiert werden, so dass der Eindruck entsteht, man

ziehe den Stein an einem Gummiseil oder einer Feder; der Stein folgt dem Kontakt schrittweise. Dabei ist die Geschwindigkeitsänderung des Steins in Abhängigkeit von der Distanz und Richtung zum Kontakt anzupassen. Wie stark die Distanz in die Änderung einfließt, sollte mit einer Konstanten flexibel einstellbar sein. Diese beschreibt sozusagen, mit wie viel Kraft (und damit Beschleunigung) sich die Distanz auf den Stein auswirkt (physikalisch „Federkonstante“).

Diese Umsetzung zeigt zwei wesentliche Problematiken auf:

- Position von Touch und Stein liegen bei stärkeren Bewegungen nicht mehr übereinander. Der ViewManager soll jedoch bei der Berechnung der Weltkoordinaten weiterhin Werte liefern, die auf der Bewegungsebene des Steins liegen. Dazu wird der Actor mit der Eigenschaft „TouchPlane“ ausgestattet, die die Ebene der Bewegung des Steins im Raum darstellt. Diese wird bei dem onTouch-Ereignis in Bezug zur Position des BoundingObjects einmalig gesetzt und von da an vom ViewManager bei der Aktualisierung der Weltkoordinaten für den Touch, respektive den Actor, benutzt.
- Die Kamerafahrt wirkt sich auf die Weltkoordinaten aus. Gerade da der aktive Stein die Führung der Kamera beeinflusst, entsteht ein dynamisches System, welches stark vom Bewegungsablauf der Kamera abhängig ist. Diese beeinflusst wieder die letztliche Darstellungsposition des Steins, usw. Im Prototyp soll die Verfolgung durch die Kamera stets hinter dem Stein und von schräg oben erfolgen. Wird ein Stein berührt und der Kontakt nach vorne geschoben, beschleunigt der Stein und mit ihm die Kamera. Da sich auch der Kontakt (in Weltkoordinaten) mit bewegt, entsteht eine (konstante) Geschwindigkeit, mit der sich der Stein andauernd fortbewegt.

Des Weiteren sollte eine Grenze für die maximale Geschwindigkeit der Steine schon bei der Berechnung der neuen Geschwindigkeit geprüft und nicht überschritten werden. In Abschnitt 6.3.1.3 werden die Startgeschwindigkeiten der Gleitphase in Abhängigkeit von der gewählten Reibungszahl beschrieben.

Für die Rotation soll ein ähnlich „träges“ Verhalten wie für die Positionierung erfolgen. Schließlich soll der Spieler hierbei die Möglichkeit haben, durch Drehen eines Fingers kurz vor der Abgabe des Steins den „Curl“ anzuwenden. Dazu soll der Rotationswert (Orientation) des Kontakts benutzt werden, der ebenfalls in Radiant gemessen wird.

6.3.3.4 Interaktion „Wischen“

Wie das Wischen simuliert und visualisiert wird, wurde in Abschnitt 6.3.1.4 und 6.3.2.2 beschrieben. Die Interaktion des Wischens geschieht über die Rink-Komponente, die als interaktives Spielelement (Actor) ihre Touch-Ereignisse über den ViewManager erhält. Zum onTouch- und updateTouch-Ereignis soll die Wisch-Information an der Koordinate des Touches auf dem Spielfeld aktualisiert werden.

Ist das Spielfeld in der Welt verschoben, bzw. ist seine Ausrichtung oder seinen Bezugspunkt zur Positionierung nicht im Ursprung und entsprechend des Koordinatensystems der Welt, müssen die Weltkoordinaten in lokale Koordinaten des Spielfelds transformiert werden. Dazu müssen die Weltkoordinaten mit der invertierten Transformationsmatrix multipliziert werden, die das Spielfeld in der Welt positioniert und ausrichtet. Für die Umsetzung des Prototyps soll diese Problematik umgangen werden, indem das Spielfeld entsprechend des Systems der Darstellung ausgerichtet und skaliert ist, und in dessen Ursprung liegt.

Die Funktionalitäten in Abhängigkeit von der Kontaktfläche einen größeren Bereich auf dem

Spielfeld auf einmal zu wischen oder der Einsatz von realen, mit ID-Tags versehene „Schrubber“ per Team, kann mit der gegebenen Basis später geschehen.

6.4 Status quo und Erfahrungen

Die in Abschnitt 6.3 zu gegebene Beschreibung des Prototyps wurde weitestgehend umgesetzt. Der grobe Spielablauf und Zustandsübergänge, die Interaktion mit den Steinen sowie das Wischen, das wesentliche physikalische Verhalten und die Visualisierung der Kamerafahrt wurden realisiert. Anhand des Prototyps lässt sich die grundlegende Interaktion und Darstellung des Spiels testen und optimieren, was die Basis für die weitere Ausarbeitung des Steuerungskonzeptes darstellt.

Bei der Verwendung von XNA ist insbesondere positiv aufgefallen, dass es einen relativ schnellen Einstieg und direkten Weg zur Umsetzung der Anwendungslogik eines Spiels bietet. Dabei gibt es einen Rahmen und Ablauf vor, in dem man jedoch frei entwickeln und die Umsetzung logischer und visueller Komponenten vornehmen kann.

Die Bereitstellung der Dienste für Grafik, Audio und Content erfolgt durch XNA ebenso direkt und einfach, wie deren flexible Nutzung. Die Dienste bieten umfangreiche Funktionen und eine breite Unterstützung von Medienformaten für Sound, Grafik, Texturen und 3D-Modelle.

Bisher gibt es nur sehr rudimentäre Unterstützung aus dem Surface SDK für XNA, insbesondere bei der echten dreidimensionalen Darstellung ist man auf eigene Routinen zur Behandlung von Kontaktinformationen und -ereignissen angewiesen. Dabei gibt es noch viel Raum zum Ausbau der implementierten Komponenten, die der Behandlung der Kontaktereignisse dienen, beispielsweise bei der Einbeziehung der Kontaktfläche oder zur Feststellung und Behandlung von (komplexeren) Gesten mit einem Kontakt, als Gruppe aus mehreren Kontakten bestehend, usw.

Sinnvolle Tests zur Performance und Optimierungen sollten insbesondere erst dann erfolgen, wenn die Qualität oder Komplexität der Visualisierung erhöht wird, da die Darstellung i. Allg. der rechenintensivere Vorgang ist.

Die Ausführung des Prototyps hat jedoch bereits gezeigt, dass für die Bereinigung der Objekte explizit zu sorgen ist, da aufgrund der hohen Rate, mit der die Update-Sequenzen stattfinden, unter Umständen sehr schnell viele Objektinstanzen erzeugt werden können. Dies ist insbesondere bei der Behandlung der Kontaktinformationen der Fall. Wird für die Bereinigung unnötiger Objektinstanzen nicht selbst gesorgt, geht zu viel Rechenleistung in die Verwaltung der Garbage-Collection, was sich zur Laufzeit durch unregelmäßiges Ruckeln in der Visualisierung äußert.

Nicht zuletzt sollten die Tests zur Bedienung von Touch unbedingt am Surface direkt vorgenommen werden, da viele Aspekte wie Latenzen, Kontaktfläche, Ungenauigkeiten und Rotation im Simulator mit der Maus nicht gut nachgestellt werden können.

7 Fazit und Ausblick

Abschließend möchte ich ein finales Fazit ziehen und einen Ausblick für die weitere Entwicklung geben.

7.1 Fazit

Prinzipiell ist das Surface, wenn auch nicht ausschließlich, für Unterhaltungssoftware wie Spiele konzipiert. Die Hardwareausstattung bezüglich CPU, GPU und RAM sind zwar mittlerweile bereits veraltet, die Plattform bringt jedoch die notwendige Hard- und Softwareausstattung mit, um multimediale, nicht zu komplex gestaltete Lösungen zu realisieren.

Wie in Kapitel 3 zusammenfassend erläutert wurde, gibt es nicht *die* ultimative Spielidee oder *das* Konzept, welches prädestiniert ist zur Anwendung auf dem Surface. Existente Spiele zeichnen sich insbesondere durch kurzweilig gestaltetes Spiel aus und weisen eher für den Casual-Spielertyp ausgelegte Eigenschaften auf.

Konzeptionelle Auswirkung auf die Entwicklung eines Spiels zeigt vor allem die besondere Benutzerinteraktion, die das Surface durch Touch, Multi-Touch und Gesten bietet. Diese Art der direkten Steuerung muss auf die Aktionen in der Spielwelt abgebildet werden, was entsprechende Konzepte für Benutzerschnittstellen und deren Implementation erforderlich macht.

Die Touch-Technologie eröffnet neue Möglichkeiten der natürlichen Bedienung und bietet die Erfassung von Informationen über die Fläche und Ausrichtung (Rotation) eines Kontakts sowie die Erkennung von Fingern und Objekten durch Tags. Dabei liegt eine relative Ungenauigkeit schon in der Natur der Berührung selbst, die aus verschiedenen Winkeln und abhängig von der Intensität des Drucks, unterschiedliche Kontaktflächen erzeugt.

Besondere Herausforderungen an Konzeption und Gestaltung der Interaktion und Darstellung stellt auch der Multi-User Betrieb, da u. A. technisch keine Zuordnung von Kontakten zu Benutzern möglich ist.

Die Anordnung der Nutzer rund um das Surface und die gemeinsame Interaktion mit einer Ein- und Ausgabeschnittstelle bietet neue Möglichkeiten für soziale Aspekte und lokales Gruppenspiel gegenüber traditionellen Konsolen- und Computerspielen. Position und Ausrichtung der Nutzer bringen allerdings auch besondere Abhängigkeiten mit sich, die schon in der Konzeptionsphase in Form der 360°-View und -Interaktion Beachtung finden sollten.

Diese und mehr Richtlinien und Vorschläge, wie sie von Microsoft in den „User Experience Guidelines“ für das Surface formuliert werden, unterstützen dabei, ein gutes „Surface-Erlebnis“ zu erzeugen. Sie wirken sich bei Einhaltung auf die Gestaltung der Visualisierung und des Verhaltens der Spielwelt und deren Elemente, sowie auf die Interaktion aus und sorgen für eine einheitliche Repräsentation des Bedienkonzepts und des Erscheinungsbilds.

Im praktischen Teil der Arbeit wurde unter Anwendung der erarbeiteten Anforderungen ein eigenes Spielkonzept entwickelt, welches prototypisch umgesetzt wurde. Dies hat im Wesentlichen gezeigt, dass die Bereitstellung von Berührungsinformationen seitens des Systems sehr rudimentär ist. Insbesondere bei der Darstellung im dreidimensionalen Raum ist die Zuordnung von Kontakten zu virtuellen Objekten selbst herzustellen. Abgesehen davon bietet die Softwareplattform mit Windows, DirectX und XNA eine relativ leicht zu erlernende, komfortable Umgebung und großen Funktionsumfang. Erste eigene Tests und Modifikationen am Prototyp machen bereits einen viel versprechenden Eindruck, leider konnte das Spiel im Rahmen dieser Arbeit nicht vollständig

implementiert werden.

Es bleibt abzuwarten, ob und wann Geräte dieser Art für den Hausgebrauch erschwinglich werden und sich etablieren.

7.2 Ausblick

Die Bestrebungen von Microsoft sind groß, die Surface-Plattform an immer mehr Partner in vielen Ländern zu liefern. Für die Zukunft plant Microsoft, weitere Modelle des Surface-Konzepts mit moderner Hardware für den breiten Mark erschwinglich zu machen. Weiterhin sind bei Windows 7 bereits die grundlegenden Touch-Funktionalitäten des Surface in das Betriebssystem integriert worden und auch andere große Hersteller wie Apple befassen sich schon lange mit Touch und Multi-Touch.

Da die Entwicklung des Spielkonzepts und die Umsetzung ein positives Resultat ergeben hat, stünde einer Weiterentwicklung des Spiels nichts im Wege. Nach ausgiebigen Tests und Optimierungen der Interaktion anhand des Prototyps kann das Spiel weiter ausgestaltet und umgesetzt werden. Ideen für interessante Erweiterungen im Sinne des Surface wären z. B.:

- Ausbau von Spielregeln und Gameplay, z. B. durch Wischen mit einem realen Besen oder der Kamerasteuerung für Spieler
- Ausbau der Touch-Funktionalitäten: Multi-Touch und komplexe Gesten, Einbeziehung der Kontaktfläche
- Ausgestaltung der Spielwelt, z. B. durch Visualisierung eines Eisstadions, der Spieler, Zuschauer, Bandenwerbung, etc.
- Intelligenteres Spielgeschehen durch visuelles Feedback, Statusanzeigen und Sound entsprechend der aktuellen Spielsituation
- Ausarbeitung der grafischen Elemente und Effekte, Sound und Animation; flüssige Übergänge von Zustandsänderungen
- Usability-Tests mit Personen aus der Zielgruppe.

Nicht behandelt oder nur kurz angesprochen werden, konnten einige weitere interessante Themen im Zusammenhang mit der Spielentwicklung auf dem Surface. Insbesondere waren dies

- Vernetzung: Lokale Inter-Tisch-Kommunikation in besonderen Lokationen, entferntes verteiltes Gruppenspiel, Nutzung von Internetservices , etc
- Kommunikation mit Smartphones, PDAs und andere „intelligentere“ Devices und Integration in ein Spiel
- „Adapter“ oder „Bridges“ zum Surface-SDK zur Nutzung in anderen „Umgebungen“
- Praktische Usability-Tests mit dem Surface im Labor

Literatur-und Quellenverzeichnis

[Adams:2006] Adams, E. and Rollings, A. Fundamentals of Game Design (Game Design and Development Series), Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[ADLR:SPTP] August de los Reyes, Surface: Predicting the past, Microsoft 2008, URL: http://www.microsoft.com/emea/msdn/spotlight/sessionh.aspx?videoid=965&PUID=00030000898C3C0C [Online: 29.09.2009].

[AMD:Stream] AMD-Stream-Computing, AMD, URL: http://developer.amd.com/gpu/ATIStreamSDK/Pages/default.aspx [Online: 29.09.2009].

[APPLE:IPHONE] Apple iPhone, Apple, URL: http://www.apple.com/iphone/ [Online: 01.12.2009].

[ATI1650] ATI X1650-Spezifikation, AMD, URL: http://ati.amd.com/products/RadeonX1650/specs.html [Online: 01.12.2009].

[ATI4670] ATI 4670-Spezifikation, AMD, URL: http://www.amd.com/us/products/desktop/graphics/ati-radeon-hd-4000/hd-4600/Pages/ati-radeon-hd-4600-specifications.aspx [Online: 01.12.2009].

[ATI4870] ATI 4870-Spezifikation, AMD, URL: http://www.amd.com/us/products/desktop/graphics/ati-radeon-hd-4000/hd-4870/Pages/ati-radeon-hd-4870-specifications.aspx [Online: 01.12.2009].

[Benko:2006] Benko, H., Wilson, A. D. and Baudisch, P. "Precise selection techniques for multi-touch screens"'CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems', ACM, New York, NY, USA, 2006, pp. 1263--1272.

[Blender] Blender, URL: http://www.blender.org [Online: 01.12.2009].

[Bourg:2001] Bourg, D. M. Physics for Game Developers, O'Reilly Media, Inc., 2001.

[Bourg:2004] Bourg, D. M. and Seemann, G. AI for Game Developers, O'Reilly Media, Inc., 2004.

[CircleTwelve] Circle-Twelve Inc., DiamondTouch, URL: http://www.circletwelve.com/ [Online: 29.09.2009].

[CurlingRink] Curling Spielfeld, Wikimedia, URL: http://commons.wikimedia.org/wiki/File:RikkuPakkala.jpg [Online: 29.09.2009].

[DarkGAME] DarkGAME-Studio, URL: http://www.darkgamestudio.com/ [Online: 29.09.2009].

[DaVinci] Razorfish Davinci Physics Illustrator, 2009, URL: http://emergingexperiences.com/2009/05/davinci-microsoft-surface-physics-illustrator/ [Online: 29.09.2009].

[DCV] Deutscher Curling Verband, URL http://www.curling-dcv.de/ [Online: 20.11.2009].

[DiamondTouch:2001] Dietz, P. and Leigh, D. "DiamondTouch: a multi-user touch technology"'UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology', ACM, New York, NY, USA, 2001, pp. 219--226.

[Edelman:2009] Edelmann, J., Schilling, A. and Fleck, S. "The DabR - A multitouch system for

intuitive 3D scene navigation"'Proc. 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video', 2009, pp. 1--4.

[Entertaible:2006] Philips Entertaible News 2006, Phillips, URL http://boardgames.about.com/od/news/ig/Entertaible-from-Philips.--09/index_g.htm [Online: 29.09.2009].

[ENTERTAIBLE] Philips Entertaible Image, www.joystiq.com, 2006, URL: http://www.joystiq.com/media/2006/08/entertable.jpg [Online: 29.09.2009].

[FalseProphets:2002] Mandryk, R. L. and Maranan, D. S. "False prophets: exploring hybrid board/video games"'CHI '02: CHI '02 extended abstracts on Human factors in computing systems', ACM, New York, NY, USA, 2002, pp. 640--641.

[Gamasutra] Gamasutra, URL: http://www.gamasutra.com/ [Online: 29.07.2009].

[Han:2006] Han, J., TED-Talks Konferenz, 2006, URL: http://www.ted.com/talks/jeff_han_demos_his_breakthrough_touchscreen.html [Online: 29.09.2009].

[Harrahs] Harrah's Entertainment and Surface News, Microsoft ,Jun 2008, URL: http://www.microsoft.com/presspass/press/2008/jun08/06-11HETSurfacePR.mspx [Online: 20.07.2009].

[Mindstorm:iBar] iBar Image, Mindstorm Interactive Surface Solutions, URL: http://www.i-bar.ch/pictures/ [Online: 20.07.2009].

[Mindstorm] Mindstorm Interactive Surface Solutions, URL: http://www.mindstorm.com [Online: 20.07.2009].

[Morris:2004] Morris, M.R., M. D. and Winograd, T. "Individual Audio Channels with Single Display Groupware: Effects on Communication and Task Strategy.," Proceedings of CSCW (), 2004, pp. 242-251. .

[Morris:2006] Morris, M. R., Huang, A., Paepcke, A. and Winograd, T. "Cooperative gestures: multi-user gestural interactions for co-located groupware"'CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems', ACM, New York, NY, USA, 2006, pp. 1201--1210.

[MS:Cebitnews:2009] Microsoft Surface Press Release, März 2009, URL: http://www.microsoft.com/presspass/press/2009/mar09/03-02SurfaceCebitPR.mspx [Online: 20.07.2009].

[MS:DirectX] Microsoft DirectX, Microsoft, URL: http://msdn.microsoft.com/en-us/library/bb219737(VS.85).aspx [Online: 29.11.2009].

[MS:FireFly] Firefly, Microsoft, 2008, URL: http://blogs.msdn.com/surface/archive/2008/02/28/firefly-game-hits-the-web.aspx [Online: 29.11.2009].

[MS:Ghosts] Ghosts Images, URL: http://www.flickr.com/photos/jldavid/sets/72157604553910218/ [Online: 20.09.2009].

[MS:MobileConnect] Mobile Connect Application, Microsoft, 2009, URL: http://community.surface.com/blogs/develop_sample_applications/archive/2009/04/07/mobile-connect.aspx [Online: 29.11.2009].

[MS:NATAL2009] Project Natal, Microsoft 2009, URL: http://www.xbox.com/en-US/live/projectnatal/ [Online: 03.12.2009].

[MS:PaddleBall] Surface PaddleBall, Microsoft, 2008, URL: http://community.surface.com/blogs/develop_sample_applications/archive/2008/06/18/paddle-ball-sample-application.aspx [Online: 29.11.2009].

[MS:PIX] PIX DirectX Debugger, Microsoft, URL: http://msdn.microsoft.com/en-us/library/bb173085(VS.85).aspx

[MS:ScatterPuzzle] Microsoft Surface ScatterPuzzle, 2008, URL: http://community.surface.com/blogs/develop_sample_applications/archive/2008/06/18/scatterpuzzle-sample-application.aspx [Online: 29.11.2009].

[MS:SDS] Surface-Datasheet, White Paper, Microsoft, 2008, URL: http://download.microsoft.com/download/7/2/9/729fc97d-692d-4231-abf3-(20b6a1de8a43/Surface_Berlin_Datasht_fnl.pdf [Online: 20.07.2009].

[MS:SGP] Surface Game Pack, Microsoft, 2009, URL: http://community.surface.com/blogs/develop_sample_applications/archive/2009/03/19/games-pack-sample-applications.aspx [Online: 20.07.2009].

[MS:SH2007] Surface History, Microsoft Press, 2007, URL: http://www.microsoft.com/presspass/presskits/surfacecomputing/docs/SurfaceHistoryBG.doc [Online: 20.07.2009].

[MS:SST2008] Surface SDK Start Here, Microsoft 2008, Microsoft Surface

[MS:SUEG] Surface User Experience Guidelines, Microsoft 2009, Microsoft Surface Workstation Edition

[MS:SURFACE] Surface Image, URL: http://tjspcservices.com/images/microsoft_surface.jpg [Online: 20.07.2009].

[MS:SurfaceCommunity] Microsoft Surface-Community, Microsoft, URL: http://community.surface.com [Online: 20.07.2009].

[MS:TechReport:2009] Benko, H., M. M. B. A. and Wilson, A. "Insights on Interactive Tabletops: A Survey of Researchers and Developers Microsoft Research Technical Report", Technical report, Microsoft ResearchMicrosoft Research, 2009.

[MS:TOUCHWALL] Microsoft Project Touchwall, Microsoft 2008, URL: http://www.officelabs.com/projects/touchwall/Pages/default.aspx [Online: 29.11.2009].

[MS:VistaSR] Vista System Requirements, Microsoft, URL: http://www.microsoft.com/windows/windows-vista/get/system-requirements.aspx [Online: 20.07.2009].

[MS:WPF] Windows Presentation Foundation, Microsoft, URL: http://msdn.microsoft.com/wpf [Online: 20.07.2009].

[MS:XAML] XAML, Microsoft, URL: http://msdn.microsoft.com/en-us/library/ms752059.aspx [Online: 20.07.2009].

[MS:XNA:Creators] XNA Creators Club, Microsoft URL: http://creators.xna.com/en-US/ [Online: 29.11.2009].

[MS:XNA:DEV] XNA Developer Center, Microsoft, URL: http://msdn.microsoft.com/xna [Online: 20.07.2009].

[NINTENDO:WII] Nintendo Wii, Nintendo, URL: http://nintendo.de/NOE/de_DE/systems/technische_details_1072.html [Online: 20.07.2009].

[NUIGroup] NUIGroup, URL: http://nuigroup.com/ [Online: 20.07.2009].

[NUIGroup:2009] NUI-Community-Book, NUIGroup, URL: http://nuicode.com/attachments/download/112/First_Edition_Community_Release.pdf [Online: 20.07.2009].

[PerceptivePixel] Perceptive Pixel, URL: http://www.perceptivepixel.com/solutions.html [Online: 20.07.2009].

[Pinelle:2008] Pinelle, D., Wong, N. and Stach, T. "Using genres to customize usability evaluations of video games"'Future Play '08: Proceedings of the 2008 Conference on Future Play', ACM, New York, NY, USA, 2008, pp. 129--136.

[PQLabs] PQ Labs, URL: http://multi-touch-screen.net/ [Online: 29.09.2009].

[PQLabs:iTable] iTable, PQ Labs, URL: http://multi-touch-screen.net/ [Online: 29.09.2009].

[REACTABLE] Reactable Systems, URL: http://www.reactable.com/ [Online: 20.09.2009].

[Reed:2009] Reed, A., Learning XNA 3.0, O'Reilly Media, Inc., 2009.

[Rubin:2008] Rubin, N. "GPU evolution: will graphics morph into compute?"'PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques', ACM, New York, NY, USA, 2008, pp. 1--1.

[Ryall:2004] Ryall, K., F. C. S. C. and Morris, M. "Exploring the Effects of Group Size and Table Size on Interactions with Tabletop Shared-Display Groupware," Proceedings of CSCW, 2004, pp. 284-293.

[Saffer:2008] Saffer, D. Designing Gestural Interfaces: Touchscreens and Interactive Devices, O'Reilly Media, Inc., 2008.

[SurfaceAcademy:2009] Surface Academy 2009 Toolkit – Opensource-Projekt, URL: http://surfaceacademy2009.codeplex.com/ [Online: 29.09.2009].

[TableToss] Razorfish Table-Toss Game, 2009, URL: http://emergingexperiences.com/2009/02/microsoft-surface-table-toss/ [Online: 02.12.2009].

[TZI:MrT:2009] Mixed Reality Table, Uni Bremen Tzi, 2009, URL: http://medien.informatik.uni-bremen.de/de/research/hci/mrt/ [Online: 29.09.2009].

[Unity3D] Unity3D, Unity Technologies, URL: http://unity3d.com/ [Online: 02.12.2009].

[Vectorform] Vectorform, URL: http://www.vectorform.com/ [Online: 02.12.2009].

[Weathergods:2007] Bakker, S., Vorstenbosch, D., van den Hoven, E., Hollemans, G. and Bergman, T. "Weathergods: tangible interaction in a digital tabletop game"'TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction', ACM, New York, NY, USA, 2007, pp. 151--152.

[Wobbrock:2009] Wobbrock, J. O., Morris, M. R. and Wilson, A. D. "User-defined gestures for

surface computing"'CHI '09: Proceedings of the 27th international conference on Human factors in computing systems', ACM, New York, NY, USA, Wobbrock:2009, 2009, pp. 1083--1092.

[WPF3.5] Wegener, J., Windows Presentation Foundation - .NET WPF, Hanser, 2009.

[XNA:Schichten] Konerow, J., XNA Framework, Entwickler Press, 2007.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den 04.12.2009 _____

Sandro Wolf