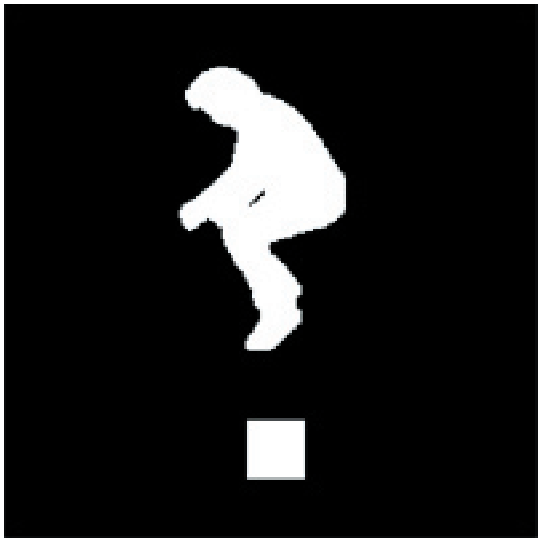
# Unencumbered Full Body Interaction in Video Games

**Jonah Warren**
MFA Design and Technology
Parsons School of Design
NewYork City, New York
April 2003

Supervisor: Golan Levin



Thesis URL: http://a.parsons.edu/~jonah/thesis/jonah_thesis.pdf
URL: http://a.parsons.edu/~jonah
email: jonah@parsons.edu

## Abstract

This thesis offers an alternative to the stationary, hand-centric experience that most existing video games provide. It proposes a scenario in which the player can affect action in the game by using his or her entire body, free of wires and controllers. Through the use of computer vision technology, this thesis attempts to develop an interactive vocabulary to aid in the creation of full body based video games. It then proposes a series of functional game scenarios, or "gamelets," that demonstrate how these interactions can be incorporated into game contexts.

# Acknowledgements

I would like to thank the following people for their support, encouragement and inspiration.

Devjit Basu
Melissa I. Bermudez
Ross Bochnek
Carrie Burgener
Eddy Chai
Perry Chan
Wen-Hsuan Chen
Haesung Chung
Catherine Herdlick
Juan Herrara
Joseph Chi Huang
Bo Yeon Joo
Yu Jin Jung
Elaine Castillo Keller
Peter Lee
Barbara Morris
Paula Pelletier
Alina Podkolinska
Katie Salen
Steven Sanborn
Yoomee Song
Eddie Teng
Wade Tinney
David Warren
Tessa Warren
Loretta Wolozin
Mateo Zlatar

A special thanks goes to Zach Lieberman for the C++ OpenCV template. Also, a special thanks to Golan Levin. His limitless enthusiasm, rigorous critique and creative vision served as invaluable resources throughout this thesis process.

# Contents

# Figures

# 1. Introduction

## 1.1. Motivation

As a child growing up at the same time video games were emerging as a medium, I became fascinated with this new form of gaming. I threw myself into their bleeping, pixelated worlds at every opportunity. These exciting new worlds usually consisted of a blocky representation of a character on a screen that could be moved around in a virtual space. The character's movement was constrained to a limited number of predefined actions. While these game actions were usually produced by some combination of mashing buttons and pushing joysticks, these quick hand movements were only a small percentage of the action that would occur during an afternoon of video game playing. I would spend hours jumping up and down in front of television screen, as if somehow, my physical actions would help the jagged heroes run, jump and shoot their way through the virtual worlds they inhabited.

Although the scene I just described may be a bit exaggerated, it is not far from the truth. While not all video gamers jump up and down while they play a game, most move their body more than is needed. They bob their head, lean left and right in sync with the character's movement and dodge virtual bullets. Though it seems we have a natural urge to use our entire bodies when playing video games, very few game designers have attempted to incorporate these more expressive physical actions into their games.

Why is this the case? It certainly isn't because of a lack of technology. Over the past 20 years, technological advances in computing (in fields such as computer vision, speech processing, and wireless technologies) have made possible interactive scenarios that could only be dreamed about in the early days of Nintendo and Atari. However, the video game industry has for the most part ignored these advances. As a result, the interactive vocabulary of video games have remained relatively unchanged for the past 30 years. The image of the hunched over video gamer in front of a television set, tethered to a gaming system frantically moving their fingers, still applies.

This thesis attempts to challenge this traditional image of video games. It does so by attempting to show how video games can use these new interactive possibilities to create games that not only incorporate our natural urges for physical action when playing games, but encourages them. Which of the new interactive possibilities are we to use? What type of input device can allow the flexibility needed to create such games?

One of the interactive possibilities that the gaming industry has failed to successfully incorporate into a gaming experience, but has enormous potential for the creation of playful interactions, is video processing and computer vision. While these technologies are not exactly new (experiments of its use date back to the 1970's), advances in computing have made it so that now, in 2003, the average household PC is fast enough to process video in real-time and thus utilize computer vision technologies. Because computer vision allows for an unencumbered full body interactive experience and can recognize a enormous amount of possible interactions, it is an obvious solution for creating games that incorporate dramatic physical movement. It is for these reasons I chose to use computer vision as an interactive solution in this thesis.

It is my hope that this thesis can start to challenge our conceptions of what a video game experience can be. I hope to show that through the use of computer vision based interactions, game experiences can be created that encourage our natural urges to engage in physical activity during play.

## 1.2. Overview of Thesis

The rest of this thesis is divided into four chapters. Chapter 2, *Background and Context*, describes and evaluates existing games and projects that incorporate computer vision technologies and full body movement. The four projects investigated range from art projects to interactive installations at museums to commercial game systems. They are: Myron Krueger's *Videoplace*, the Vivid Group's *Mandala Gesture Xtreme System*, Reality Fusion's *GameCam* and Intel's *Me2Cam*. This section is concluded by presenting five important advantages of using computer vision in the creation of game experiences.

Chapter 3, *Methodology*, describes the process of formulating a concept and the final project itself. The first section, *Initial Interests*, describes the initial interests I had in video games which drove my creative process. The second section, *Abstracting Action*, presents how this project was initially envisioned and the prototypes that accompanied it. It then describes how these prototypes were critically evaluated and how the project was reformulated. The final section, *Unencumbered Full Body Interaction* presents the final project itself. After identifying my design process,  a basic set of computer vision based interactions to serve as building blocks for the creation of more complex game interactions is presented. I then describe the game interactions that were developed from these building blocks, and finally, the gamelets that show how these interactions can be used in game contexts.

Chapter 4,  *Evaluation and Analysis*, provides a detailed evaluation and analysis of the project. This chapter is divided into three sections. The first section, *General Gamelet Experiences and Playability*, describes the experience of playing each gamelet, critically evaluates them and suggests possible improvements. In the second section, *Comparative Evaluation*, I comparatively evaluate how successfully each gamelet incorporates the advantages of computer vision, as well as their overall game play. The final section, *Project Evaluation*, discusses the successes and failures of the project as a whole, and the lessons learned.

In Chapter 5, *Conclusions*, I summarize the main conclusions arrived at after the completion of this project and discuss future possibilities and directions.

## 1.3. Contributions

This thesis proposes to help broaden our understanding of the interactive possibilities in gaming. It attempts to do this by conducting a study of the interactive possibilities that computer vision technologies allow in gaming, and creating a series of gamelets that realize these potentials. The contributions of this thesis include:

1. A study of existing projects related to the creation of computer vision based interactions and video games.

2. A list of the advantages of using computer vision as an interface in video games.

3. A vocabulary of computer vision based interactions that incorporate these advantages.

4. A set of "gamelets," or partial games, that show how these interactions can be used in game contexts.

5. A detailed analysis and evaluation of these gamelets.

# 2. Background

The following section represents the background research done in the preparation for the creation of the final project. This section is divided into two parts. The first section, *Related Projects*, investigates projects throughout the history of video games and interactive media that inform the creation of computer vision based game interactions. This section is organized chronologically. It starts with an investigation of Myron Krueger's installation *Videoplace* (1975), one of the first systems to allow participants to use their whole body to interact with graphic objects. It continues with a description and evaluation of the Vivid Group's, *Mandala Gesture Extreme System* (1996), an interactive system which allows users to participate in graphically rich computer vision games. It then considers two commercial computer vision game systems: Reality Fusion's *GameCam* (1999) and Intel's *Me2Cam Virtual Game System* (1999).

The second section, *Advantages of Using Computer Vision in Video Games*, identifies five major advantages of using computer vision as an interactive strategy in the creation of computer games. These five advantages, identified during my research, will serve as a guide for the creation and subsequent evaluation of the interactions and gamelets I propose in this thesis.

## 2.1. Related Projects

### 2.1.1. Myron Krueger's *Videoplace*

Working in the 1970's, Myron Krueger was one of the first artists to seriously ask the question: "What are the various ways in which people and machines might interact, and which of these are most pleasing?" [Krueger 1993] Blurring the boundaries between art and science, Krueger defined a new medium, which he called interactive computer art,

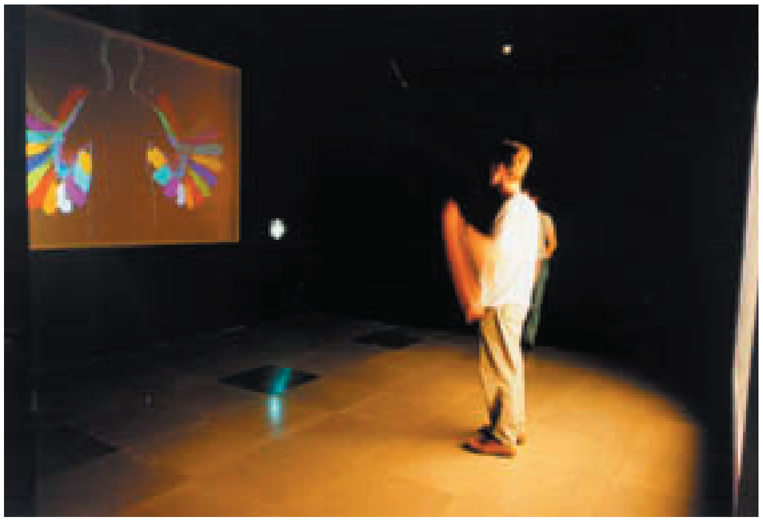


Figure 1: A user interacting with *Videoplace* [ Ars Electronica 1999].

whose primary goal was to explore possible answers to this very question. In his book *Artificial Reality II*, Krueger discusses this new medium and describes a series of installations he created in the 1970's and 1980's that allow participants to interact with computers in novel ways. These experiments were quite possibly the first attempts to incorporate full body movement in human computer interaction.

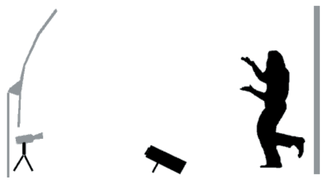


Figure 2: *Videoplace's* installation setup [Kreuger 1993].

One of works described in *Artificial Reality II*, *Videoplace*, is Krueger's most famous work and one of the inspirations for this project. *Videoplace*, shown in 1975, was an installation that allowed participants to use their whole bodies to interact with computer generated graphic objects. In the installation, the participant faces a projection screen and a video camera. Behind the user is placed a backlit piece of translucent plastic to help the computer distinguish the participant from the background. The live video image of the participant is fed through a computer which extracts the participant's silhouette, which is then projected in front of them along with various graphic objects.

*Videoplace* was designed as an interactive medium and not an individual art piece. As such, its goal was not to provide a single interactive experience, but to provide as many different and varied interactive experiences as possible so as to demonstrate its potential richness as a medium. To do this, Kreuger designed over 50 different interactive experiences. The installation was designed to change between the various interactive experiences when the user stepped away from the piece. Thus, by repeatedly stepping away from and back into the installation, the user could experience multiple interactions.

In order to better understand the types of interactions Krueger created for the *Videoplace* system, I will quickly describe the three of its interactions that are the most playful, and thus most relevant to my project.

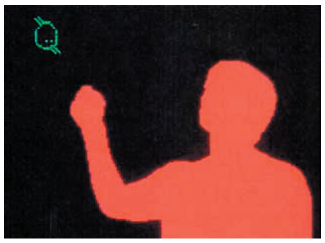


Figure 3: Krueger's *Critter* [Kreuger 1993].

***Critter:*** This was the most popular of the interactive experiences developed in *Videoplace* [Krueger 1993]. In Critter, the participant interacts with a small sprite with a playful personality. The critter is represented by a circle the size of a participants' fist, with two miniature arms, legs and eyes. The critter's behavior changes depending on the situation. At first, the critter chases the participant around the screen. When the critter catches the participant, it attempts to climb to the top of their silhouette, after which it does a little dance. Other critter actions include: chasing an open hand, exploding when the participant captures it, and dangling from an outstretched finger.

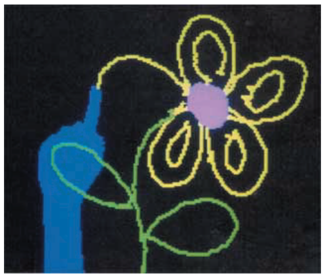


Figure 4: Krueger's *Digital Drawing* [Kreuger 1993].

***Digital Drawing:*** This interaction allows the participant to draw on the video screen using his or her finger. If there is more than one person in the environment, each person is assigned a different color to draw with. The user can clear the screen by opening their hand and erase portions of drawing by "rubbing" two fingers held close together over part of the drawing [Krueger 1993].

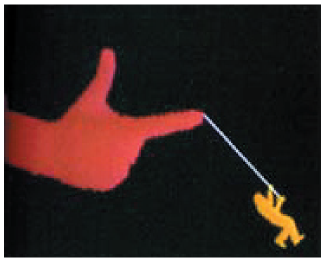


Figure 5: Krueger's *Hanging by a Thread* [Kreuger 1993].

***Hanging by a Thread:*** This is a interaction that allows two users to interact with each other in a playful environment. This interaction incorporates two of Krueger's environments: *Videoplace* and *Videodesk*. *Videodesk* is very similar to *Videoplace*, the crucial difference being that only the user's hands and arms are used to manipulate graphic objects, not the whole body. In this interaction, the silhouette of the participant in the *Videoplace* environment is displayed dangling on a string. The string is being held by the hand of the participant of the *Videodesk* environment. The participant dangling on a thread can swing back and forth by moving and leaning left and right. The *Videodesk* participant can also affect the swinging silhouette by moving their hand in rhythm to the swaying.

Krueger's *Videoplace*, though created over 20 years ago, is still incredibly relevant. It is a testament to the strength of the *Videoplace's* system that art projects such as Camille Utterback's *Text Rain* are still impressing audiences. As an initial exploration of a new interactive medium, its comprehensiveness is unparalleled.

Despite its importance to the history of interactive media, there are a few shortcomings of *Videoplace* in relation to the goals of this project. One of these is perhaps related to its biggest strength: its comprehensiveness. While *Videoplace* does identify a incredible number of interactions that can be performed within the system, Krueger fails to organize a taxonomy or a system for understanding these interactions. Without such a system to help us understand this incredible number of studies, it becomes difficult to suss out the lessons learned the creation of this piece and how to build upon it. As such, future interactive artists and designers are doomed to repeat the mistakes Krueger made years ago.

Another shortcoming of the *Videoplace* system in relation to this project is that it fails to demonstrate how these interactions can be used in a game context. While this was not a goal of the *Videoplace* system, it is an important distinction to make. While many of the interactive studies Krueger made are very playful in nature and fun to interact with, there are no defined goals to allow for competitive play.

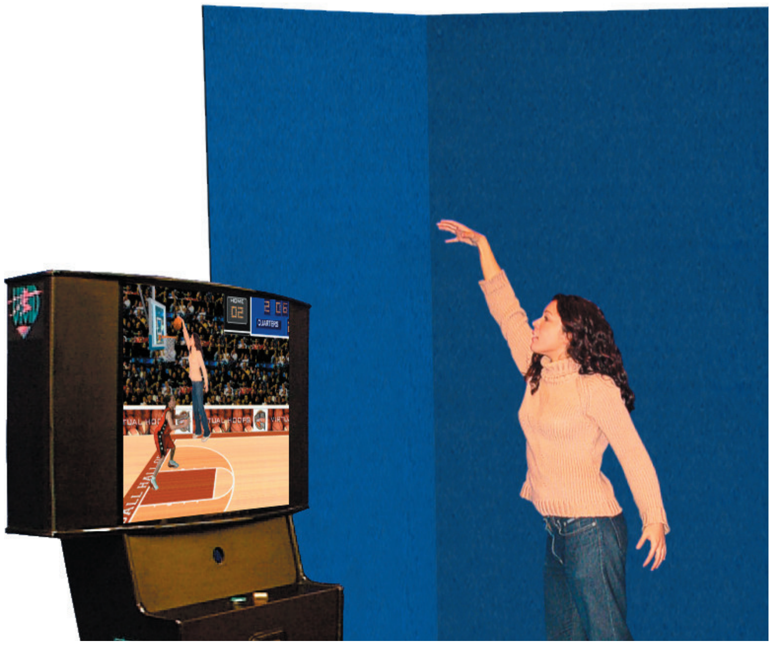


Figure 6: A user interacts with the *Mandala GX System* against a chromakey background [The Vivid Group 1996].

### 2.1.2. The Vivid Group's *Mandala Gesture Extreme System*

The *Mandala Gesture Xtreme (GX) System*, created in 1996, is possibly the most comprehensive computer vision based gaming system available today. Created by the Vivid Group, a Toronto based computer entertainment company, the *Mandala GX System* is specifically designed for what the company calls "location based entertainment facilities," [The Vivid Group 1996] or in other words, Museums, Science Centers, Halls of Fame and TV Productions. The *Mandala GX System* consists of a 1.6 Ghz computer, CCD Video Camera and cables, Nvidia Video card with TV output, Video Capture board, PCI sound card and a license to their custom Gesture Xtreme software. They also offer a variety of lighting and chromakey installation solutions.

The *Mandala GX System*, as described on their web site, is a virtual reality system that uses a video camera to place a participant's image into a variety of "fast action games and virtual environments" [The Vivid Group 1996]. The installation setup is similar to the setup used in Krueger's *Videoplace*. As in *Videoplace*, the user stands in front of a camera which records their movement. The user's image is computationally separated from the background image, which is then displayed in front of them with either a projector or large screen television. Instead of a backlit plastic panel, a chromakey background is used to isolate the user's image. In the *Mandala GX System*, the video image taken from the background is a full color image of the user unlike *Videoplace*, where the processed video image is a binarized version of the user's image. As in *Videoplace*, the user's video image is then placed in a variety of different virtual environments that allow users to interact with various graphic objects.

The Vivid Group has created a large number of software titles to run in its *Mandala GX System*. It has created five entertainment titles, seven educational titles, nine sports titles, and six "virtual theatre," [The Vivid Group 1996] or multiplayer games. I will quickly describe a few of the games that are most relevant to the project so as to get a better understanding of the types of experiences these games provide and to be able to accurately critique them.

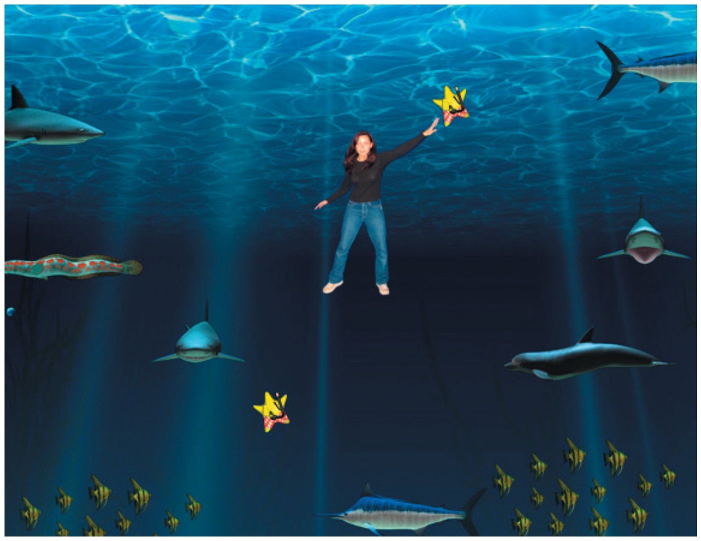


Figure 7: A user reaches out to move in Vivid Group's *Sharkbait* [The Vivid Group 1996].

***Sharkbait:*** This game allows the user to "swim" around an underwater virtual environment. The object of the game is to collect as many stars as possible. While attempting to gather the sinking stars, the user most also avoid creatures such as electric eels and sharks that swim across the screen. The user is approximately one fifth the size of the playing area and may move around the screen by either pointing up, left or right, to go in the corresponding direction, or by ducking to move down.

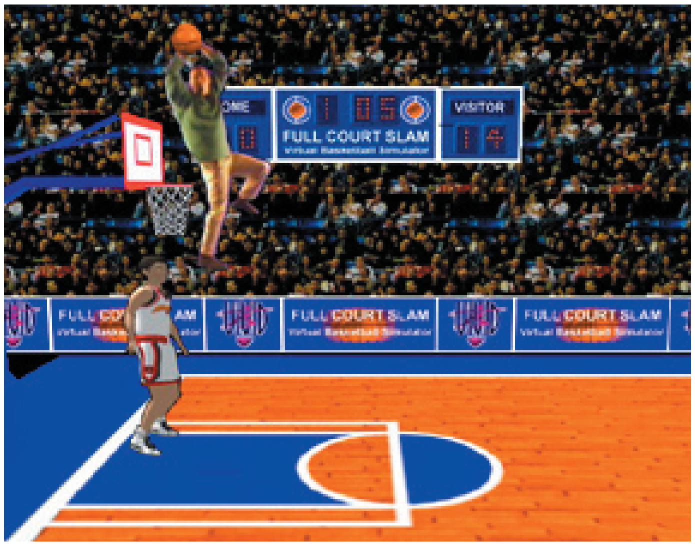


Figure 8: A player dunks in the Vivid Group's *Full Court Slam* [The Vivid Group 1996].

***Full Court Slam:*** This game is a virtual reality full-court basketball game. When on offense, the user attempts to dribble a ball down the court against a virtual opponent and shoot the ball in the basket. As you do this, your opponent attempts to steal the ball from you and block your shot. When on defense, the user must attempt to steal the ball and block the computer player's shots.

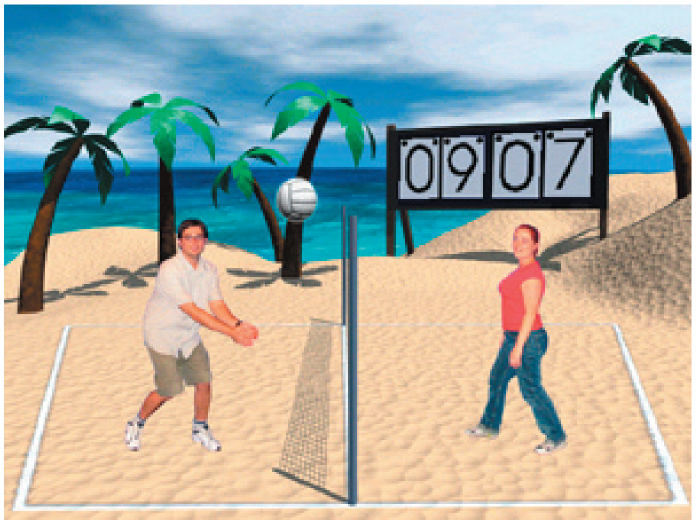


Figure 9: Two users play the Vivid Group's *Volleyball* [The Vivid Group 1996].

***Volleyball:*** This game is a virtual volleyball game. The user attempts to continually hit a volleyball over the net to a virtual robot opponent. The ball moves a bit slower than real time, in order to make the game easier to play. The direction the ball bounces off the user is dependent upon the angle of the user's edge at the point of contact.

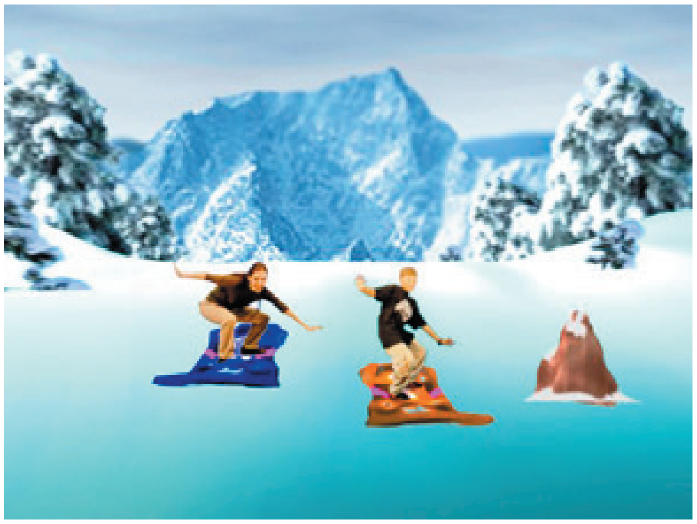


Figure 10: Two users play the Vivid Group's *Snowboarding* [The Vivid Group 1996].

***Snowboarding:*** This game is a virtual snowboarding game, which can be played by up to four people. The user's travels down the slope of a mountain head-first attempting to be the first to cross the finish line. While boarding down the mountain, the user must avoid obstacles such as trees, rocks and stumps and attempt to fly off jumps. The user can lean left and right to control the direction of movement, and squat to control the speed at which he or she travels down the mountain. The first user to the finish line wins.

The biggest strength of the *Mandala GX System* is that its installation setup allows for a wide range of movement. The camera is set far enough away from the user to provide the computer with a large video image within which movement can be detected. The games, for the most part, take advantage of this by encouraging dramatic movement.

Another of the system's strengths are the games' graphics and sounds. Each game places the user's live video image in a in a rich 3D graphical environment. A lot of attention has been given to the visual treatment of the graphic objects with which the user interacts. One of the most impressive of these environments is *Sharkbait*, where the user must chase realistically rendered and animated dolphins around the screen, while attempting to avoiding sharks that bite and electric eels that shock. All of the action takes place in a beautifully rendered underwater environment.

The system also has many extras that add to the overall experience. Some of these extras include the user's image being "beamed" into each game environment, an easy to use interface for choosing between games, and a high score board that takes a victory photo of the player rather than asking them to enter their name or initials.

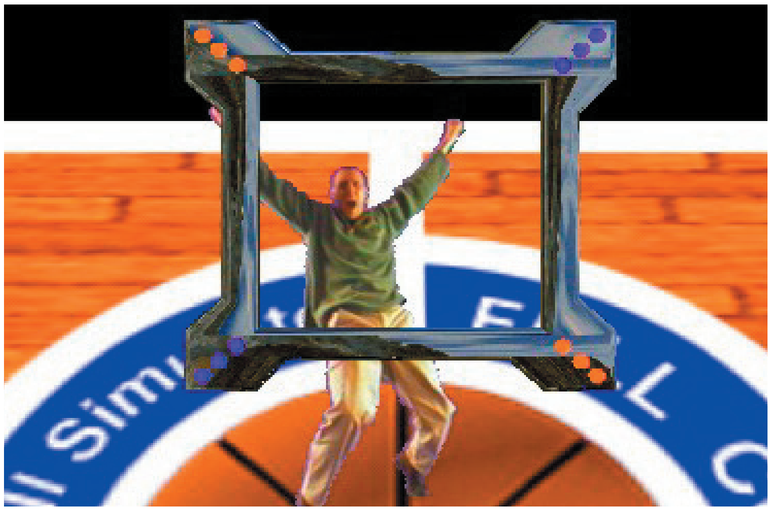


Figure 11: *The Mandala GX System's* takes a screenshot of the user for the high score board [The Vivid Group 1996].

Despite these strengths, there are quite a few aspects of the system that could be improved upon. Most of these weakness are related to the specifics of the of interactive experiences the games provide. While these issues are crucial to the lasting playability and entertainment value of the system, they are probably not of primary concern to the creators of the system. Since this system was designed to exist in a science center, museum or hall of fame type environment, where there is high traffic and few repeat users, issues relating to the specifics of the interactions and playability of the games come second to the ability of the system to impress audiences within a short period of time. However, the issues of lasting playability are crucial to my project, and as such, I will address them at some length.

The biggest weakness of the game experiences, which quickly became apparent after watching a few first time users play, is that overall, the interactions are not very intuitive. There are a few reasons for this. One is due to the fact that many of the games set up false interactive expectations. Many of the games designed for the system are based on popular real world games and sports such as basketball, volleyball, soccer and snowboarding. While creating this type of game makes sense when designing for first time users (so that they don't have to learn a different set of rules for each new game), this strategy becomes less useful when the actions involved in successfully playing the real world game have little to do with the actions that are successful within the video game.

This is most apparent in *Mandala GX System's* basketball game: *Full Court Slam*. In attempting to recreate the experience of playing basketball, the designers of *Full Court Slam* came up with a vocabulary of interactions that attempted to accurately match the real world actions of playing basketball. This resulted in a large vocabulary of game interactions consisting of: jumping, dribbling, stealing, blocking, shooting and dunking. While most of these interactions are somewhat similar to their corresponding real world actions, each one of them has significant differences that must be learned. In many video games, learning of the specifics of an interaction and perfecting its control can be a source of pleasure. However, this learning becomes irksome in games where there is a preexisting expectation of how the action should be performed which doesn't match the actual interactive experience.

Preexisting expectations of how an action should be performed occur often in simulation based computer vision games. This is because of a few reasons. In most computer games, the input device of the game usually provides clues to how the interaction should occur (e.g. if there are buttons, they probably should be pressed). As an input device, a camera provides little information for the user about how the actual interaction should occur. As such, all of the user's expectations of how the interaction should occur in the game come from the game itself. In simulation games, this expectation is based wholly on the real world game experience. When this is combined with an input device that allows for a natural and free range of motion, the user naturally expects the game interaction to match the real world interaction.  Problems occur when this expectation is not met.

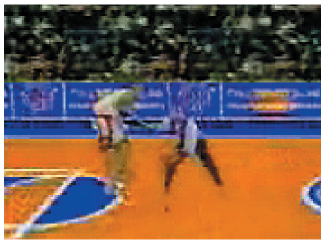


Figure 12: A user dribbles against a virtual opponent in *Full Court Slam* [The Vivid Group 1996].

Another problem with designing computer vision games that attempt to keep the interactive experience as close to the real world experience as possible, is that often certain interactions don't translate successfully across media. This can be due a number of issues, such as: the translation of 3D interactions into 2D ones or the importance of tactile and force feedback to the interaction. These issues of translation are especially apparent in the dribble interaction in *Full Court Slam*. In real world basketball, dribbling is an essential interaction. Dribbling is a learned skill, whose pleasures are closely related to control, movement in 3D space and tactile feedback. When translated into a 2D computer vision interaction, the dribbling interaction is stripped of these essential features. Without the ability to dribble around a defender or feel the force of the ball return into your palm, the dribbling interaction is reduced to an awkward necessary patting that takes away from the flow of the game experience.

Another crucial problem with the games' interactions are that a large number of them map a real world action to a different character action in the game world. This type of interaction can be thought of as a "representational interaction," where one action stands for another. Because of the difference between action and outcome in representational interactions, they tend to be less intuitive. The user must spend some time learning the specifics of how their real world action corresponds to action in the game world. One of the biggest advantages of computer vision technologies is that it allows for a more direct type of interaction to occur. This more direct type of interaction that can be thought of a "literal interaction." In a literal interaction, the user can directly affect objects in the game world. As in virtual reality, the participant in the real world actually becomes the character in the video game. If the user wants to jump over something, the user simply jumps, without having to worry about the translation of their action into a character action.

Most of the games designed for the *Mandala GX System* fail to take advantage of the potential of literal interactions allowed by computer vision.

A good example of how the *Mandala GX System* uses representational interaction is in its game *Sharkbait*. In *Sharkbait*, as well as a number of its other games, the basic way the user moves around the game environment is by pointing in the direction the user wants to move. When the user points up, the user floats to the upper part of the screen, when the user points left, they float to the left side of the screen, and when the user points right, they move to the right side of the screen. To move downwards, the user ducks down low. These interactions are representational interactions. The gesture "point left" in the real world stands for a completely different action in the game world: move left. As such there is still a significant disconnect between action and outcome, and as a result, a lack of direct control is felt.

## 2.1.3. Reality Fusion's *GameCam*

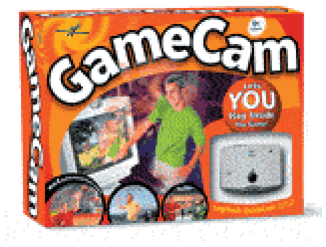


Figure 13: Reality Fusion's *GameCam* box [Reality Fusion 1999].

One of the first commercially available products to use video as input in a video game context was Reality Fusion's *GameCam*, released in October of 1999. The GameCam bundle comes with a Logitech QuickCam PC video camera and a software suite consisting of six different interactive experiences that place to the user's video image in various interactive scenarios.

After connecting the webcam to your personal computer and installing the software, the user stands in front of the camera which displays the user's video image in a game context. All of the six experiences included in the software suite are designed to allow interaction between graphic objects and the user's upper body.

Of the six titles that come in the bundle, four are games, and two are playful interactive experiences. These six different titles are briefly described below:

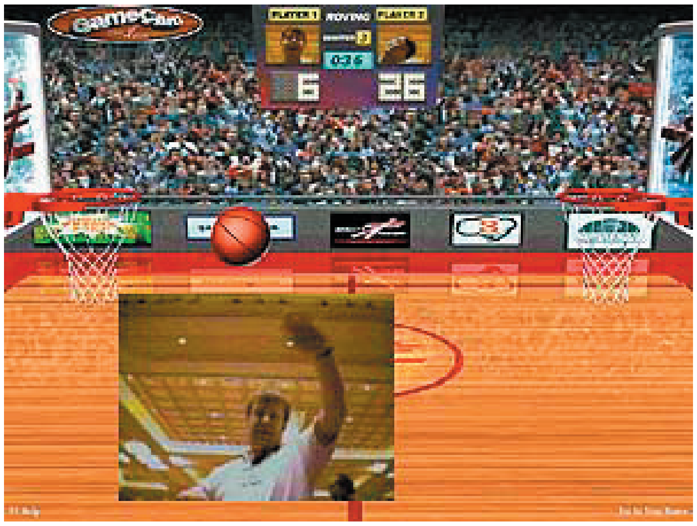


Figure 14: A user interacts with Reality Fusion's *Basketball*. The user tries to bounce the ball off their body into a basket [Reality Fusion 1999].

***Basketball:*** A one on one game where each user tries to tap a basketball into the net on their side of the screen. The physics of the basketball moves more like a beach ball than a basketball.

***Horse:*** A game similar to the schoolyard basketball variety. Two to four people take turns shooting a ball into a basket. If the user fails to make the shot the previous player made, you receive a letter. When you spell the word HORSE you are eliminated.

***Karate:*** A game that where user spars with an animated virtual fighter. The user punches and kicks to score points against a virtual opponent, which punches and kicks back.

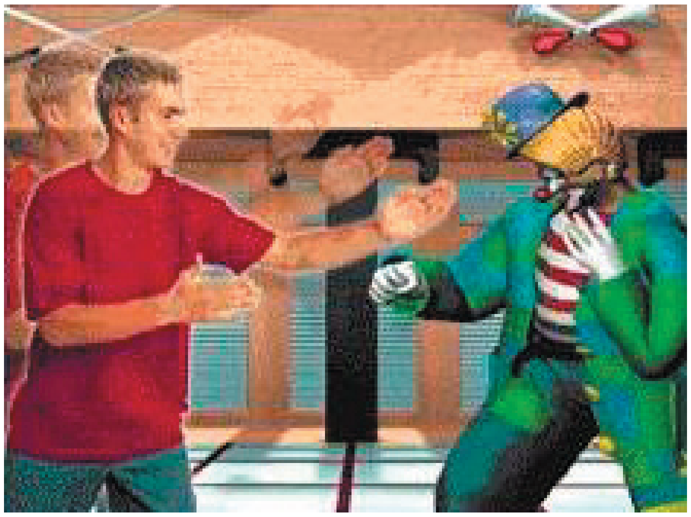


Figure 15: A user fights a virtual clown in Reality Fusion's *Karate* [Reality Fusion 1999].

***Volleyball:*** A game which allows a user to play beach volleyball against a virtual opponent.

***JumpIn Video:*** An interactive experience that allows users to create a "kaleidoscopic light show" to accompany the music of your favorite CD.

***Be There:*** An interactive experience that allows the user to take pictures of their own faces superimposed in funny places.

While the *GameCam* may be the first commercially available video game system to incorporate video as input, it is not without its faults. The following review by ZDNet articulates the experience of playing with the software, and the conveys the general feeling one gets after reading a number of product reviews.

> "Well, perhaps fun is not exactly the right word. Amusing might be more accurate. *GameCam* is an interesting idea whose time has not yet come. First off, these games are really just high-tech *Pong* when you get right down to it. Technology has come a long way when you can use your hands as the paddles in a game of *Pong*; but just being able to do this does not mean that it has lasting value... Given more powerful technology in the future and better applications to work with, *GameCam* could open new avenues for gaming. For now, however, it is little more than a mildly amusing diversion" [CNET Networks 2000].

This quote is most critical of the design of the game experiences themselves, which is a common theme seen in the reviews found on the Internet. Many users described the experience of playing the games as interesting and exciting at first, but that the novelty quickly wears off. As one reviewer put it, the games are simply "amusing for a short amount of time." [Epinions 2000].

The other most common complaint about the system is the inconsistency of the technology. One reviewer commented, "it does not work well in rooms that are too bright or too dark, and it will miss your movements if you move too quickly" [CNET Networks 2000]. Another stated "the images are full of static, and can be really slow-moving" [ZMedia 2000]. Other reviewers complained that often the interactions did not work as expected, and at times a graphic object would float over their video image rather than reflecting off of it.

Finally, a few reviewers commented on the fact that while these games did provide a somewhat aerobic experience, they felt the experience to be somewhat restrictive. The games set up an expectation by the user that they could freely move and jump around in the games, but the actual game experience requires that the user stay within the frame of action. This frame of action is somewhat limited do to the proximity the user needs to be to the screen in order see the action. As one reviewer states: "(the games don't) let you interact properly without moving way back from your desk, not easy in most spaces" [ZMedia 20000].

Besides all of its faults, the *GameCam* is a breakthrough from a technological point of view. This is the reason why most of the reviews of the software start out by pointing out the potential of this type of interaction in gaming environments, and why most felt frustrated when it didn't live up to this potential. It is also very appropriate that on two different occasions it was compared it to *Pong*. While both *Pong* and the *GameCam* have a limited entertainment value in terms of their staying power, they both show the potential of a medium to creating engaging interactive experiences.

### 2.1.4. Intel's *Me2Cam Virtual Game System*

Figure 16: The box for Intel's *Me2Cam Virtual Game System* [D'Hooge 1999].

The *Me2Cam Virtual Game System* developed by Intel as part of its Intel PlayTM product line is very similar to the *GameCam* system created by Reality Fusion. Like the *GameCam*, the *Me2Cam* is a combination hardware and software package that comes with a webcam and suite of interactive experiences that use video as input. Intel also released the *Me2Cam* in October of 1999. It is unclear whether the Intel's *Me2Cam* or the Reality Fusion's *GameCam* was the first to be released.

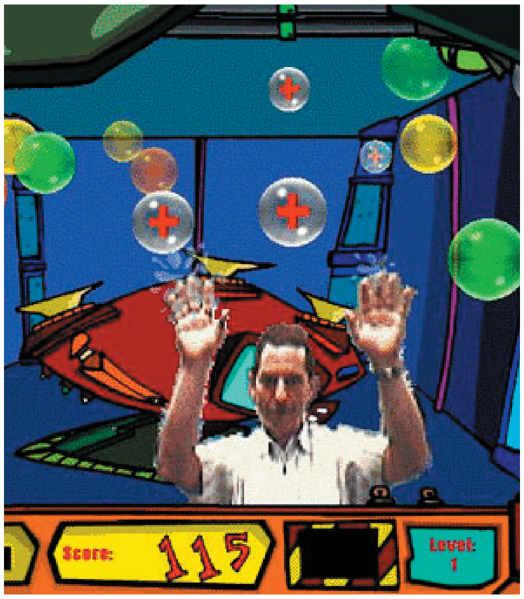
Just as with the *GameCam*, the user connects the *Me2Cam* to their personal computer and installs the software in order to play the games. Also like the *GameCam*, the all of the interactive experiences in the software suite are restricted to the user's torso. The software suite consists of five different interactive experiences: three games and two interactive experiences. The five titles are briefly described below:

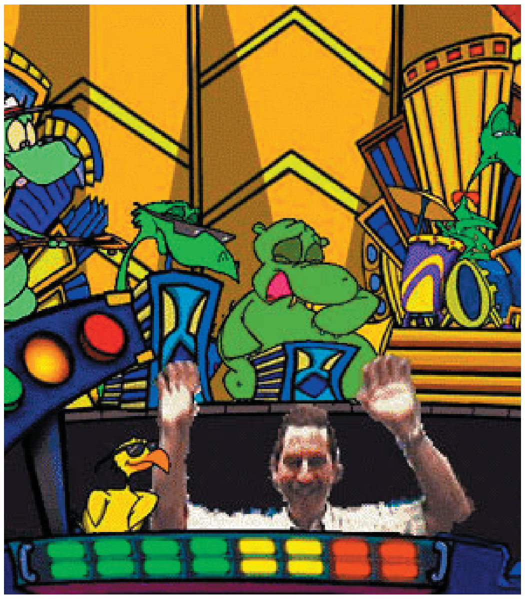
***Bubble Mania:*** A game where the user, positioned on the bottom of the screen, attempts to either reach out and pop or avoid bubbles that float down from the top of the screen. Whether the bubble should be popped or avoided depends on the type of bubble falling.

***Club Tune:*** An interactive experience where a band plays music in the background, while he user dances in the foreground. The more the user dances, the faster the band will play.

***Pinball:*** A pinball game where in place of paddles are copies of the user's image. By moving around, the user can affect how the ball reflects off of them. The direction the ball bounces off the user is depen-

Figures 17 and 18: The figure on the left shows a Me2Cam's Bubble Mania in action. The figure on shows a user playing with Club Tune [D'Hooge 1999].





dent upon the angle of the user's edge at the point of contact.

***Fun Zone:*** An interactive experience that allows the user to see their live video image stretched, morphed and generally just distorted in a variety of different ways.

***Snow Surfin':*** A snowboard game which allows the user to slide down a virtual mountain. By leaning left and right the user can control the movement of the snowboard as it travels down the mountain. The user must avoid obstacles that reduce the user's speed.

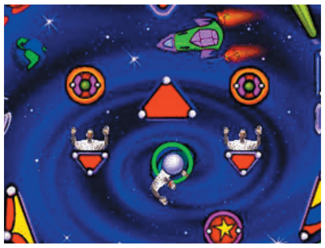


Figure 19: A user playing *Me2Cam's Pinball* [D'Hooge 1999].

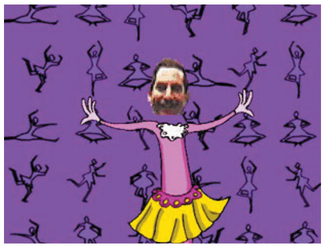


Figure 20: A user interacting with *Me2Cam's Fun Zone* [D'Hooge 1999].

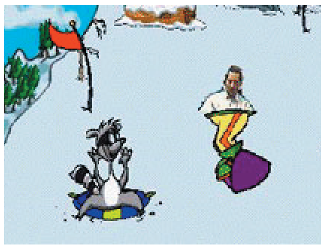


Figure 21: A user playing with *Me2Cam's Snow Surfin'* [D'Hooge 1999].

The *Me2Cam* and the *GameCam* are strikingly similar: they both have approximately the same price, they both are a combination webcam and software system, they both come with a music based interactive experience, and they both include an image manipulation based interaction. While they seem identical, there are a few subtle differences. One difference between the *Me2Cam* and the *GameCam* are its target audiences. Intel's *GameCam* is designed specifically for children 4-8, unlike the *GameCam*, which attempts to appeal to a wide audience. As such, the *Me2Cam* has no fighting game, and its emphasis is more on the fun interactive experiences rather than on the design of the games.

Another difference between the systems is that the *GameCam* allows for multiple player games, while the *Me2Cam* is restricted to single player action. Perhaps the most significant difference, when reading its reviews, is the quality of the technology. While there were complaints in the quality of the video, as there were with the *GameCam*, there were no complaints about the speed of the system. The following review mentions this fact, as well as addresses the weakness of the game's design: "I didn't think the picture quality was the best, but it had close to true-time motion... no one my age would play it after the novelty wore off. They might, though, if the games were more complicated and the software more sensitive to finer movements" [Able Minds 2000].

Also, like the *GameCam*, many users of the *Me2Cam* complained of feeling restrained by limited by the range of motion the webcam captures. One user states, "Unfortunately, the movements that can be made in the game are somewhat restrictive. You cannot move outside the boundaries of the camera's view without interfering with the feel and flow of the program" [Berger 2000].

Aside from a few technological advantages, the *Me2Cam* adds little to the medium of video as input based games that was not already explored by Reality Fusion. The reviews of these two systems are in most cases interchangeable, and as such, reinforce the valuable lessons for the future design of camera based games I've identified in my critiques.

## 2.2. Advantages of Using Computer Vision in Video Games

While researching the projects described earlier, I compiled a list of five important advantages of using computer vision in video games over more traditional interfaces. This list was formed partially from the successes of the games and interactions I researched, and partially from the potentials I saw unfulfilled. I created this list to have as a reference to help guide the creation of my own computer vision interactions and games.

### 1) Athletic Activity

One advantage of using computer vision in video games is its ability to provide athletic gaming experiences. Since the only input device is a video camera, the user is not restricted by wires, or wearable or held devices. This encourages a more natural, unencumbered interactive experience, perfect for allowing for more athletic and physical activity. Only recently, with the creation of games such as *Dance Dance Revolution*, *Para Para Paradise*, and *Police 24/7*, has the gaming industry begun to explore the potentials of athletic interaction. Even so, these physically orientated games are almost all dance based games, and are fairly limited in terms of the type of interactions they recognize. The lack of exploration of athletic action in video gaming beyond dance based games is extremely puzzling when considering the natural human tendency to associate athletic activity with competition.

**2) More Expressive / Performative Interactions**

Unlike most video games, where the players' physical performance is limited to discrete button presses and joystick taps, video capture allows for a continuous interactive experience with a large range of expression. This is range of expression is due to the interactive bandwidth that video capture provides when compared to more traditional gaming interfaces. The amount of information that passes from user to computer per second is significantly higher. This facilitates a different type of interactive experience, one which accentuates the players' differences in personality and allows for more creative expression.

**3) Closer Interactive Mappings**

As I discussed in section 2.1.2., in almost all video games, there is a mapping of the player's physical real world action to a character action that happens inside the virtual game world. This mapping creates a gap between physical action and virtual outcome that must be learned. This mapping is almost always accompanied by a translation of scale and position. With computer vision games, this mapping between the real and virtual is greatly diminished, which allows users to interact game objects more intuitive and naturally, and in relation to their body. This can be thought of a "literal interaction", in contrast to the "symbolic interactions" that usually take place on a computer. Because of this, in computer vision games, one will rarely ever find a user taking their eyes off of the action on the screen. At no point does the user miss a button, drop a controller or look where a button is on a keypad.

**4) Use of the Whole Body**

Using the video camera as an input device allows an enormous amount of flexibility in terms of the type of physical action that can be interpreted. Most traditional gaming input devices, such as controllers, joysticks, and foot pads are designed for a specific part of the body and is accompanied by a preconceived idea of how that action is to be executed. Since computer vision can recognize the user's whole body as an input device, the game designer has much more flexibility in terms of the type of interaction they want to occur. Extremely different interactive experiences can be designed without the need for an entirely new input device.

**5) Large Vocabulary of Actions / Creative Solutions**

Perhaps the most important advantage of computer vision games is that their interactive vocabulary is not as limited to a small predefined set of actions. The only limitation of action is that which can be performed by the human body and be recognized by the computer. If utilized effectively in a game context, this larger vocabulary of possible action can allow the user more creativity to come up with different interactive solutions to game problems. While allowing such interactive flexibility in a video game context is not an easy task, when accomplished it can be very powerful.

# 3. Methodology

This chapter is divided into three sections. The first section, *Initial Interests*, explains the initial interests that guided my thought process in realizing a final project. The second section, *Abstracting Action*, explains the thought behind and the imagined form of the project as it was defined in late 2002, as well as how and why it evolved to its final form. The third section, *Unencumbered Full Body Interaction*, describes the final project. The section begins by presenting the project's goals, as well as the design process which guided its implementation. The technical setup is then addressed, from the specifics of the installation, to a description of the computer vision algorithms used. After this, a basic vocabulary of computer vision interactions is identified. This is followed descriptions of the game interactions that were created from this basic vocabulary. Finally, a series of game studies, or "gamelets," are presented to show how these interactions can be used in a game context.

## 3.1. Initial Interests

My interest in computer and video gaming is fueled by an intuitive feeling that there are enormous potentials in the field that have been left unexplored, especially in the field of interaction. Though the gaming industry is often touted as being on the forefront of experimentation in technology and computing, this claim really only applies when considering animation and graphics. If we look at the evolution of video gaming since its beginnings in the 1960's from the perspective of interaction, the lack of innovation is striking. When one compares the controllers of today to the controllers of 30 years ago, the only difference is more buttons and joysticks. This realization inspired me to investigate the process of creating interactions in games. Perhaps this would lead me to discover new interactive possibilities, or at the very least, discover an explanation for the industry's lack of experimentation in the field.

When looking at the multi-billion industry gaming is today, where it can easily take a team of 50 designers, programmers, artists and musicians to produce a commercially successful console game, one can quickly lose sight of what gaming is about. In order to concentrate on the basic issues of interaction involved in creating video games, I decided to examine the history of video games from its very beginnings, back when games consisted of knobs, dots and blips rather than 3D characters, in depth narratives and multiple soundtracks.

In looking back at the earliest video games, it quickly becomes obvious that all of these games were simply simulations of natural, physical phenomenon which allowed user interaction. The first two games ever created (*Tennis for Two* at Brookhaven Nation Laboratories [Burnham 2000] and *Spacewar* at M.I.T. [Burnham 2000] ) were developed at large research centers. Both started out as simply simulations of objects moving in space, and both added user interaction in order to make the experience more engaging for novice users.

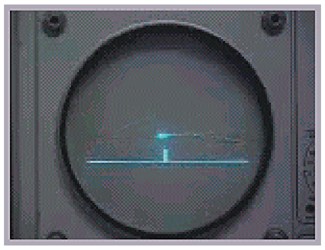


Figure 22: The oscilloscope from *Tennis for Two* [Hunter 2000].

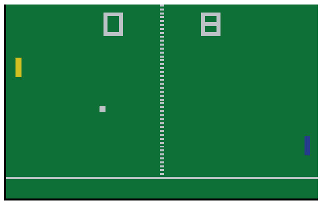


Figure 23: A screenshot of Atari's classic *Pong* [Hunter 2000].

However, since the computational and interactive possibilities of these early machines were limited, these early game designers had to selectively choose what aspects of the action to allow the user to control. These decisions had an enormous effect on the user's experience of the simulation. For example, though *Pong* and *Tennis for Two* were both simulations of the game tennis and both were created with extremely limited computational power and interactive possibilities, the experience of these two games are quite different. These differences arose from the decisions the designer made about what aspects of real world tennis to simulate, both visually and interactively. *Tennis for Two* simulated tennis from a side view and allowed the user to control not only when to block the ball back (with a button), but also the angle of the racket (with a knob). Pong, on the other hand, simulated tennis from a top down view, and represented the tennis player with a vertical line. The user was allowed to control the position of the vertical line along one axis in order to block the tennis ball back to the opponent's side of the court (with a knob or slider).

The effect of the game designer's choices about what actions and what aspects of action to represent in a video game experience seemed to be one of the most essential aspects in the creation of a video game. This discovery encouraged me to further explore the concept of translating interactive experiences across media, which lead me to think about the concept of "abstracting action."

## 3.2. Abstracting Action

As I claimed previously, all of the earliest video games could be seen as interactive simulations of physical phenomena. In fact, all games can be considered simulations [Salen and Zimmerman 2003] in that they all borrow their ideas of interaction and play from the real world. The game designer's challenge is to effectively translate engaging game interactions from the real world into the digital. Because this translation is necessarily a translation from a more expressive medium to a less expressive one, the game designer must selectively choose what actions to represent and what to leave out. This process of reducing an experience down to its essential interactions can be seen as a process of abstraction.

This process of abstraction is also intimately related to the input device the game designer is designing for. The fewer actions an input device can understand, the more abstract and symbolic the representation of action must be. Conversely, the more actions an input device allows, the more potential there is for a literal translation of action. The number of actions an input device can recognize can be thought of as its "interactive bandwidth."

Interactive bandwidth can be seen as a measure of the amount of information passing between the user and the computer. Factors that go into computing an input device's interactive bandwidth are whether the interaction is continuous or discrete, the update rate, the number of degrees

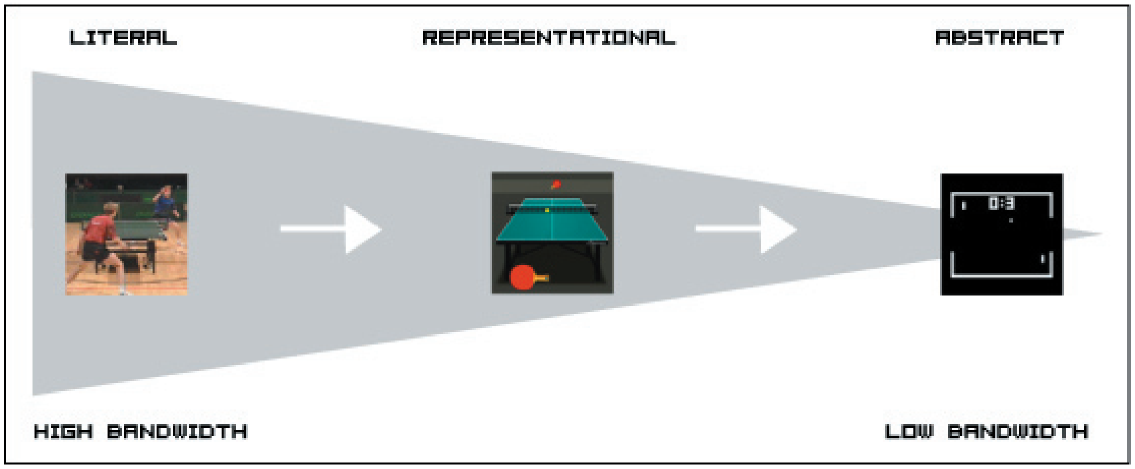


Figure 24: This diagram organizes game interactions on a continuum from literal translation of action to abstract translation of action. Three versions of ping pong are placed on this continuum: real world ping pong, *Table Tennis* by shockplay.com and Atari's *Pong*.

of freedom, and the number of bits per degree of freedom. Investigating these variables can give us a good idea of the interactive bandwidth of a particular input device, and its potential for allowing literal translations of action.

Thus we can understand all interactions as existing on a continuum corresponding to both their level of abstraction and interactive bandwidth. A representation of this continuum can be see in the Figure 24. The left side of the diagram represent interactive experiences that have a more literal translation of action and have a higher interactive bandwidth. The right side of the diagram represents games that have a more abstract or symbolic translation of action and a lower interactive bandwidth. On this diagram, I have placed three different digital versions of ping pong. The game on the left side of the continuum has the most literal representation of action, real world ping pong. The game on the right side of the diagram has the most abstract translation of action, where the actions of ping pong are abstracted into one slider that the user vertically scrolls up and down.

I envisioned my thesis to be an exploration of the space this continuum maps, in order to better understand the creation of game interactions. In order to effectively study this process of translation, I came up with a series of questions I thought a successful thesis addressing this topic would cover: When a video game designer is creating an game based on a real world game, what aspects of the physical actions involved are represented and what are left out? How are these actions controlled by the user physically? To what extent does the user have control over these actions? How are these decisions made? How do these decisions affect the design of the rest of the video game experience? How do these decisions affect the user's experience of the game?

### 3.2.1. Case Studies

Before I set out to design a project that would address these questions, I first decided to explore how game designers have traditionally translated physical actions from the real world into game interactions. To do this, I conducted a study of  real world games that have been translated into video games in a variety of different ways.

In video gaming's short history, there have been a handful of real world games that have translated well to the digital realm. Since these digital games have proved generally successful from a game play perspective, they have become popular testing grounds for experimenting with new technologies. As a result, a game such as ping pong has dozens of digital counterparts, each providing a drastically different experience, and in some cases, interactive possibilities. Studying these games from a purely interactive perspective allowed me to look at different existing strategies for translating real world actions into video game actions and how these decisions affect the experience of that game.

The two actions games I chose to study were ping pong and paintball. I chose these games because both games are reliant on the pleasures of coordination of movement and sport, and both game have multiple digital counterparts that provide a wide range of interactive alternatives. In order to understand how each game translated its interactive experience from the real world to the digital, I first established a vocabulary of physical actions that made up each real world game experience. While this process was not an exact science, it did give me a general idea of the crucial actions involved each game, and what actions would make sense to represent in a video game simulating that experience. For each game I came up with a list of general descriptions of the actions involved such as: move, look, shoot, dodge, and swing. I then broke these actions down to more

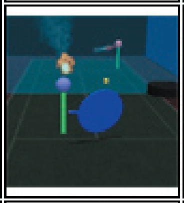
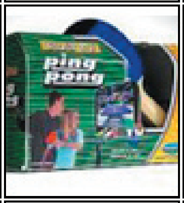
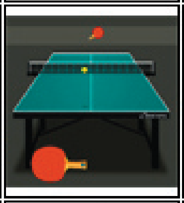
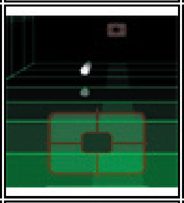
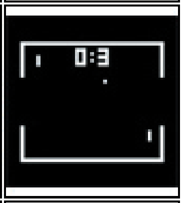
| INTERACTIONS | | Haptic Pong | TV Pong | TableTennis | 3D Pong | Pong |
|---|---|---|---|---|---|---|
| | |  |  |  |  |  |
| move | left | cont. | cont. | cont. | cont. | cont. |
| ---- | right | cont. | cont. | cont. | cont. | cont. |
| ---- | forward | cont. | cont. | cont. | ---- | ---- |
| ---- | back | cont. | cont. | cont. | ---- | ---- |
| swing | back | cont. | cont. | cont.(lr) | ---- | ---- |
| ---- | forward | cont. | cont. | cont.(lr) | ---- | ---- |
| ---- | up | cont. (lr) | ---- | cont.(lr) | cont. | ---- |
| ---- | down | cont. | ---- | cont. (lr) | cont. | ---- |
| ---- | angle | ---- | ---- | ---- | ---- | -- |
| look | up | ---- | ---- | ---- | ---- | ---- |
| ---- | down | ---- | ---- | ---- | ---- | ---- |
| ---- | left | ---- | ---- | ---- | ---- | ---- |
| ---- | right | ---- | ---- | ---- | ---- | ---- |

Figure 25: This diagram compares various versions of digital ping pong.

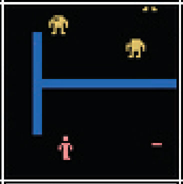
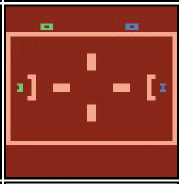
specific actions that could potentially be represented by a human computer interaction.

The first game I looked was ping pong. I chose five different versions of digital ping pong to study, each of which provided a significantly different interactive experience. The versions I chose to look at were: *Haptic Battle Pong* by Dan Morris and Neel Yoshi which uses the phantom haptic mouse as an input device, *PlayTV Ping Pong* by Radica Games which uses a paddle fitted with sensors, *TableTennis* by shockplay.com which uses the mouse, 3D Pong by albinoblacksheep.com which also uses the mouse, and the classic *Pong* by Atari which uses either a slider or a dial.

To investigate how each video game recreated the experience of ping pong, I first identified what I thought to be the essential vocabulary of actions that make up the experience of real world ping pong: moving, swinging and looking. I then broke these actions down into more specific actions and identified which of these real world actions are represented in each digital version.  I then created a chart to compare the games' interactive experiences. The horizontal axis lists the five digital versions of ping pong I chose to investigate, while the vertical axis lists the specific actions I identified as the interactive vocabulary of ping pong. The chart identifies which of the list of possible actions each digital version represents. It also identifies whether the control provided for each action was discrete, continuous, or a continuous within a limited range.

I also looked at the game paintball. I purposely decided to employ a fairly broad definition of the game in my search for digital versions. Though at first glance it might seem inappropriate to consider a game like *Tank* (a game where the user are maneuvers a tank around a battlefield in order to shoot an enemy tank and avoid being shot) a version of paintball, if examined closely, it is essentially the same game. In both paintball and *Tank*, the user is a character

Figure 26: This diagram compares various versions of digital paintball.

| INTERACTIONS | | Police 911 | Quake III | Wolf. 3D | Berzerk | Tank |
|---|---|---|---|---|---|---|
| | | | | |  |  |
| move | left | ---- | discrete | discrete | discrete | discrete |
| ---- | right | ---- | discrete | discrete | discrete | discrete |
| ---- | forward | ---- | discrete | discrete | discrete | discrete |
| ---- | back | ---- | discrete | discrete | discrete | discrete |
| shoot | aim | cont. | ---- | ---- | ---- | ---- |
| ---- | trigger | discrete | discrete | discrete | discrete | discrete |
| ---- | reload | ---- | ---- | ---- | ---- | ---- |
| dodge | duck | cont. (lr) | discrete | ---- | ---- | ---- |
| ---- | jump | ---- | discrete | ---- | ---- | ---- |
| ---- | step l | cont. (lr) | discrete | discrete | ---- | ---- |
| ---- | step r | cont. (lr) | discrete | discrete | ---- | ---- |
| look | up | ---- | cont. | ---- | cont. | ---- |
| ---- | down | ---- | cont. | ---- | ---- | ---- |
| ---- | left | ---- | cont. | ---- | ---- | discrete |
| ---- | right | ---- | cont. | ---- | ---- | discrete |

that moves around a given world with various obstacles, where the goal is to shoot other characters while attempting to avoid being shot. The only difference is the representation of the character and the environment in which it takes place.

The versions of paintball that I included in my study are: *Police 911* by Konami, an arcade game where the user uses a laser gun to shoot enemies and physically move to dodge enemy fire, *Quake III* by Id Software, a game where the user explores a 3D space inhabited by aliens in the first person with a keyboard or controller, *Wolfenstein 3D* by Id Software, a game similar to *Quake* except with a more limited vocabulary of actions, *Berzerk* by Stern, where a character explores a simple two dimensional world viewed from above, and *Tank* by Kee Games, a game similar to *Berzerk*, except the user are controls an tank rather than a person. I identified the essential actions to the game of paintball as moving, shooting, dodging and looking (see diagram).

While these studies proved useful in gaining a better understanding of the different interactive possibilities when translating real world game actions into game interactions, it was difficult to gain any further understanding of how these different translations of action affected the experience of the game without conducting a user study of these games. However, since my interest lies more in the design of games and the creation of interactions than the study of their psychological effect, this investigation just strengthened my urge to attempt to formulate a design project, so I could study the actual creation of games and game interactions, rather than just study their effects.

## 3.2.2. Initial Project Proposal

In order to study the process of translating physical actions from the real world to the digital first hand, I decided to focus on a simple real world game whose pleasures are closely tied to coordination and accuracy of movement. I decided to create a series of digital games that were all based on the same real world game, but whose interactions were drastically different. In such a way, I would be able to compare the different games not only from the perspective of the user, but from the perspective of game designer as well.

The game I chose to study was dodgeball. I chose dodgeball because its rules are simple and easy to learn, and more importantly, the pleasures of the game are derived primarily from athletic activity: dodging, catching and throwing. In order to focus the study even further, I decided to concentrate on just one of dodgeball's essential interactions, the act of dodging.

I decided on creating three different digital dodging games, each of which translated the real world act of dodging into game interactions differently. Ideally, one game would have a fairly abstract translation of action, one would have a more representational translation of action, and one would have a literal translation of action. For example, if I were conducting this study for a ping pong game, I would hope to create a series of games similar to the ones used in the diagram previously mentioned: *Pong* (abstract), *Table Tennis* (representational), and *PlayTV Ping Pong* (literal).

I decided to keep the initial game concept as simple as possible, so as to be able to concentrate on the actual interactive experience. The initial game concept I came up with was called *Duck and Jump*. The game starts with a character positioned on the right side of a screen. When the game starts, blocks of varying size and speed introduced at the left side of the screen would scroll horizontally at the character. The blocks would scroll at differing heights. Some blocks would approach the user's head, while others would scroll towards their feet. The goal of the game would be simply for the user to avoid the blocks scrolling at them by ducking, jumping, lying down, stepping over, or whatever action is most appropriate.

The three different interactive scenarios for the *Duck and Jump* games can be seen in Figure 26. The game with the most abstract translation of action would allow the user to control the character in the game with just a keyboard. There would be keys available to perform each of the following actions: move forward (left on the screen), move back (right on the screen), jump, duck and lie down.

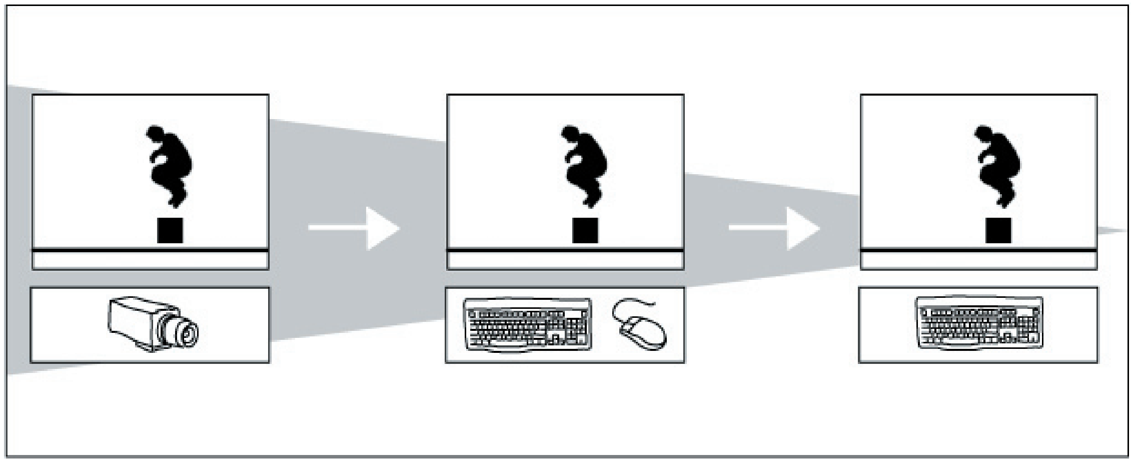


Figure 27: This diagram shows the three different versions of Duck and Jump envisioned places on the continuum.

The second game, with a more representation translation of action, would allow the user to control the game character by using the mouse and the keyboard combined. The action mappings would be similar to the previous game, except for the fact that all the ducking and jumping would be controlled by the mouse. The user could duck by clicking the mouse to establish a starting position and then drag down to make the user duck. The further down the user dragged from the initial starting point, the further the character would crouch down. To make the character  jump, the user could simply let go of the mouse while the character in the game was in a crouching position. The further crouched the character was when the button was released, the higher the user would jump. In this way, the user could have continuous control over the height the game character actually ducked and jumped, which could cut down on recovery time, but would perhaps be more difficult to master.

The final game, with the most literal translation of action, would allow the user to actually become the character in the game. By using computer vision and a setup similar to Krueger's *Videoplace*, where a camera recorded the user's image which was processed and then projected in front of them. In such a way, the user would jump over blocks and duck under them by actually physically ducking and jumping.

I envisioned a few different experiments to conduct after I finished the creation of these three different digital dodging games. The first experiment would simply be a comparison of the user experience of each game. I would have a user play each game and describe in detail the interactive experience of each. I would then analyze these results and see if the user reactions could be generalized along the continuum of representation of action I established earlier. The second experiment would entail the creation of a final version of the game which would allow two users to simultaneously compete against each other using different interactive methods. I thought that this model might provide different insights and perhaps provoke interesting questions about the effect the different interactions have on the game experience. The last of these experiments would be to develop each game further separately. I envisioned that the decisions about how to make each game more engaging would differ based on each game's interactive experience. I hoped that this process might help me develop some general rules for the creation of engaging video games based on the type of interactive experience one is designing for: abstract, representation or literal.

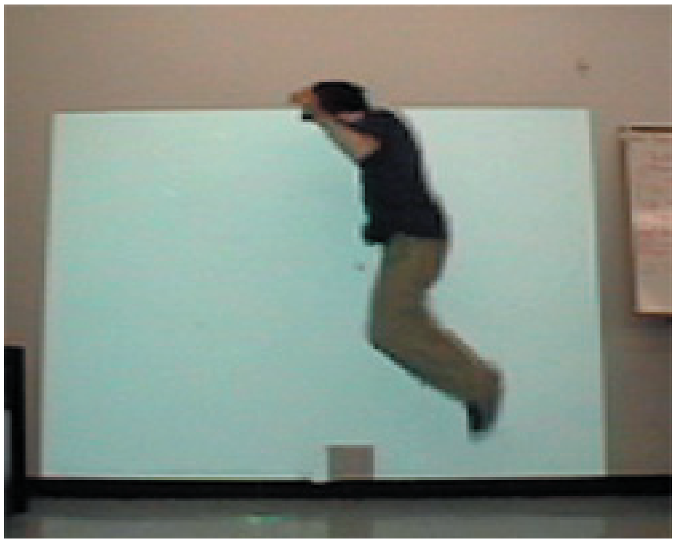
## 3.2.3. Initial Prototypes

### 3.2.3.1. Prototype I

The first of these three games I decided to prototype was the *Duck and Jump* game with the most literal translation of action: the computer vision based game. I prototyped this game first for a few reasons. In order for my study to be a useful one, each game must be somewhat engaging, and be able to hold a user's attention. The computer vision based game was by far the most original interaction of the three games, and as such, the most unpredictable in terms of its game experience. I wanted to be sure that this interaction could hold a user's attention.  Also, this game would be by far the most challenging to create. Because of this fact, I needed to establish early on whether on not the time making this game was going to be well spent.

In order to test if this game would be successful or not, I came up with an experiment that would let me test the essential interaction of the game without actually creating it. This experiment was conducted with a projector and

Figure 28: A frame of the movie produced during the creation of the prototype for the computer vision version of *Duck and Jump*.



computer. The object was basically to see how fun it would be to jump over projected animated graphic blocks.

A projector was placed approximately 20 or so feet away from a blank white wall, pointed directly at it. The height of the projected image was adjusted so that the image was a foot or so taller than the average person. The projector was positioned so that the bottom of the projected image aligned flush with the bottom of the wall. A short looped Flash animation was then played on the projector. The animation consisted of blocks that repeatedly scrolled horizontally across the screen, from left to right. The blocks were made so that their projected size was approximately one foot by one foot. The scrolling blocks appeared at two different heights, one that would animate across the bottom of the projected image, and one would animate across the projected image four to five feet up from the floor.

After this was set up, I started the animation and positioned myself in the far right side of the projected image. As the blocks scrolled towards me, I attempted to avoid the blocks. I jumped over the blocks scrolling towards my feet, and ducked under those that were approaching my head. I repeated this a few times with the blocks scrolling at different speeds. After finding an appropriate speed, I set up a camera and videotaped myself jumping over and ducking under the blocks. I did this so I could analyze the experience later, as well as to be able to quickly and effectively communicate my idea to others. While the exact setup of the prototype does not exactly match the installation setup I envisioned for the final game (where the game action is projected in front of them) the interaction was the same.

As an initial prototype, this experiment was extremely successful. Even though the blocks appeared in a regular pattern, and there was no detection of contact when a block hit the user, simply the interaction of ducking and jumping in time with the animated blocks proved to be quite engaging. As an interactive experience, it showed a large potential for use in a video game context.

The creation of this prototype proved to be quite an exciting and important step forward in the development of the project. It was my first realization of the potentials for using full body interaction in video games. After the creation of this prototype I found myself constantly thinking about other potential ideas for full-body computer vision based games.

### 3.2.3.2. Prototype II

After the first prototype, I decided to create a prototype for the version of *Duck and Jump* with the most abstract translation of action: the keyboard based game. Grabbing stills

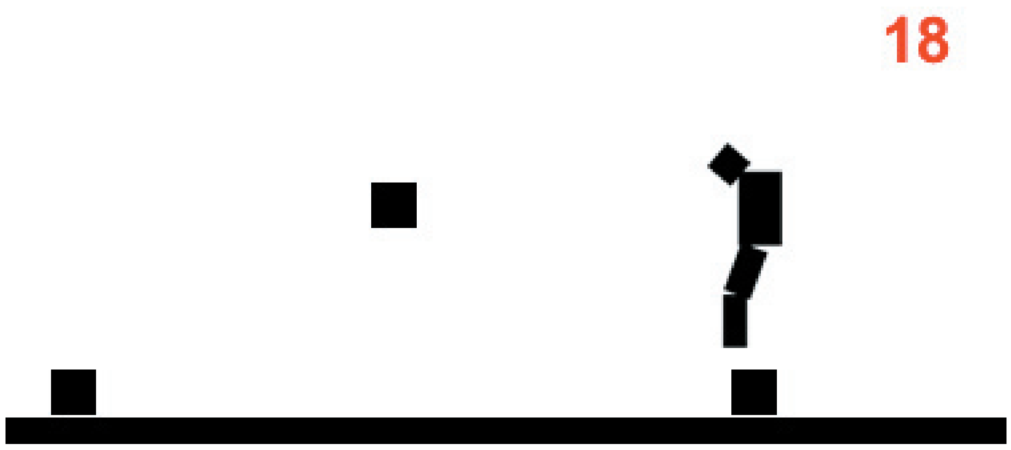


Figure 29: A screenshot from the second *Duck and Jump* prototype. The game character is shown in mid-jump.

from the movie made in the creation of the first prototype and tracing over them, I made jumping and ducking sequences in order to create a realistic moving animated character for the game. Using Macromedia Director, I allowed the user to play these animated sequences depending on the button pressed. When the user pressed the spacebar, a character located on the right side of the screen would jump, when the "d" key was struck, the character would duck.

The final iteration of the prototype existed as an online shockwave game. When the user pressed start, blocks, introduced at random intervals, would scroll across the screen. As in the first prototype, the blocks would either scroll across the screen at the user's head or at their feet. The game would continue indefinitely as long as the user could successfully avoid all the blocks, by jumping and ducking appropriately. The game would end after a given amount of blocks were detected to have touched the game character.

This prototype, while successfully in providing a somewhat engaging experience, proved to be quite a bland exercise in game translation compared to potentials revealed in the creation and testing of the previous prototype. As a result of this, I began become more and more focused on the issues related to the first prototype, and the development of computer vision based games and interactions.

### 3.2.4. Project Reformulation

After the completion of these two prototypes and the first semester presentation, I began to critically reevaluate my initial project proposal: an exploration of various methods of abstracting action in the creation of game interactions. While I believed this concept was based on a strong set of assumptions that could potentially provide a valuable framework for understanding video game interactions, I wasn't sure the scope of the proposed products was comprehensive enough to be useful in supporting my claims.

The project's main weakness was the *Duck and Jump* game that was conceived to have an interaction based on a representational translation of action. While the proposed abstract and literal versions of *Duck and Jump* were both appropriate, the mouse and keyboard based game envisioned for the representational version fell far too close to the abstract side of the continuum. To create a game with a more "representational" interactive experience, I would most likely have had to create my own input device that could somehow approximate ducking and jumping actions. However, when considering my strengths, interests, and the scope of the project, learning how to create such a device seemed inappropriate. As such, this weakness in the project as currently proposed was unavoidable.

The combination of this critical evaluation of my project with the success of the first prototype, motivated me to shift the project's focus. I decided to take a more design based and exploratory approach. Because of the success of the first prototype, I decided to continue with the implementation of the computer vision based *Duck and Jump* game while at the same time, encouraging thought about other possible computer vision based games and interactions. This approach proved successful and lead to a complete reformulation of the project.

## 3.3. Unencumbered Full Body Interaction

### 3.3.1. Project Goals and Design Process

After my initial frustration with the mismatch between the concept behind my first semester's project and the project itself, I decided to take a more open ended approach to the project, one which would be motivated and driven by design, production and intuition. I imagined that a clear concept could be arrived through a process of building and critical evaluation, rather than just research. Instead of having a clearly articulated concept and attempting to create a project that fit to that concept, I decided to follow my intuition and simply explore what appealed to me most from my first semester's project: the potentials of computer vision based interaction in games.

While my production process did proceed somewhat intuitively and without a clear idea of what my exact goals were, my intuition was informed by the research I had done from first semester concerning game interactions, the research I had done into Krueger's *Videoplace* and the experience of playing the Vivid Group's *Mandala GX System*. Because my intuition was based on a fair amount of knowledge about previous projects that attempted to create playful computer vision based experiences, my process followed a fairly straight path. Though I often wasn't directly conscious of the many decisions I was making while creating these interactions, these decisions were all informed by this knowledge.

This following section, which discusses the design goals, design process and the self applied constraints of the project, can be seen as an attempt to explain the reasons behind the intuitive decisions that drove my design process. Although the topics addressed in the following section were not all fully articulated until after the creation of my interactions were complete, it does not mean that they were arbitrary or were inconsequential to the design process. These goals and constraints that follow were simply brought out and made more clear by the creative process itself.

#### 3.3.1.1. Design Goals

As I stated previously, these design goals were formulated during my production and design process. The following design goals were created as result of investigating the intuitive decisions I was making during production and responding to its successes and failures.

1) Develop an interactive vocabulary to aid in the creation of simple and engaging computer vision based games.

2) Create interactions that are simple and intuitive to understand.

3) Create interactions that encourage creative, athletic and expressive movement.

4) Create interactions that encourage the use of the entire body.

5) Make a series of gamelets that show how these interactions can be used in a game context.

These goals were articulated at different points of the design process. Goals #2, #3, and #4 were defined fairly early on in the design process, after the advantages of using computer vision in gaming were identified. Goal #1 was identified as a general vocabulary of interactions was naturally formed through experimentation. Goal #5 was arrived at last. I wanted to be able to show how each of the interactions I created could be used in a game context. However, due to the number of interactions, creating a complete game to contextualize each interaction was unrealistic. Thus, I decided to create a series of "gamelets" to demonstrate their potential. These "gamelets" would not be fully realized games, but would simply place the interaction in a competitive context. These "gamelets" can simply be thought as one level of a game created to demonstrate the game's essential interaction.

If all of these goals were met, this project would be a success. The larger the vocabulary of interactions, and the more gamelets I could create to contextualize these interactions, the more useful my project would be to future game developers attempting to create computer vision games.

#### 3.3.1.2. Design Process

Based on these design goals, I developed a process for the creation of these interactions and their contextualization. Though this design process was articulated after the creation of the project's design goals, this process accurately describes the intuitive process that drove the creation of the developed computer vision interactions and gamelets. The process is as follows:

1) Create a simple and intuitive computer vision based interaction.

2) Make sure it generalizes to the whole body. That is, make sure it is general enough a rule that it makes sense to be initiated by not only a user's head and hands, but also foot or any other part of the body.

3) Place the interaction in a game context that encourages creative use of the body to initiate the interaction.

Though all the interactions and gamelets did follow this general process, some interactions were more successful at being placed in a game contexts that encourage the use of the whole body than others.

#### 3.3.1.3. Self Applied Constraints

In creating interactions that would effectively satisfy the goals identified previously, some self applied constraints were applied to the design of these interactions. These constraints were created to simplify the large task at hand as well as to concentrate the energy on the most important design decisions in relation to this project. Limiting the amount of design decisions needed to be made is an effective way to do this.

The first of these constraints was on the interactions themselves. In order to create interactions that were as intuitive and natural as possible, I focused my studies on the different types of action that can take place at the boundaries of the player's silhouette and graphic objects. Though it might seem the case, this is not the only type of computer vision interaction computationally possible. An example of another type of interaction can be seen in the *Mandala GX System* game *Sharkbait*. In this game, the user controls their movement on the screen by pointing in the direction they want to move. While such an interaction seems simple enough, it is still an interaction that must be learned through experience. It is my claim that there is a huge potential for computer vision games that don't need to use this kind of interaction, but instead rely on a more direct method of interaction, where users simply touch graphic objects to affect their movement.

The main reason this type of interaction is most intuitive and easy to understand is because physically touching a given object in order to affect its action is the most common way we interact with objects in the real world. While the difference between touching a block and pointing in a given direction might not seem very different in terms of learnability, such subtle differences should not be overlooked in interaction design. The difference between directly touching a block to affect its movement, and pointing in a given direction to move that way is the difference between a literal interaction and a symbolic one. Why use a more complex interaction when a simpler one can be used just as effectively?

The other self applied constraint concerned the visual treatment of the interactive experiences I created. In order to focus on the interactions happening, I decided to restrict my visual language to user's binarized silhouette, colored blocks, balls and lines. By creating this constraint, I force both me as the designer as well as the user to focus on the types of action occurring, rather than being distracted by the visual treatment of the graphic objects, or the user's video image. This constraint was inspired by Krueger's *Videoplace*, which employs a similar visual language.

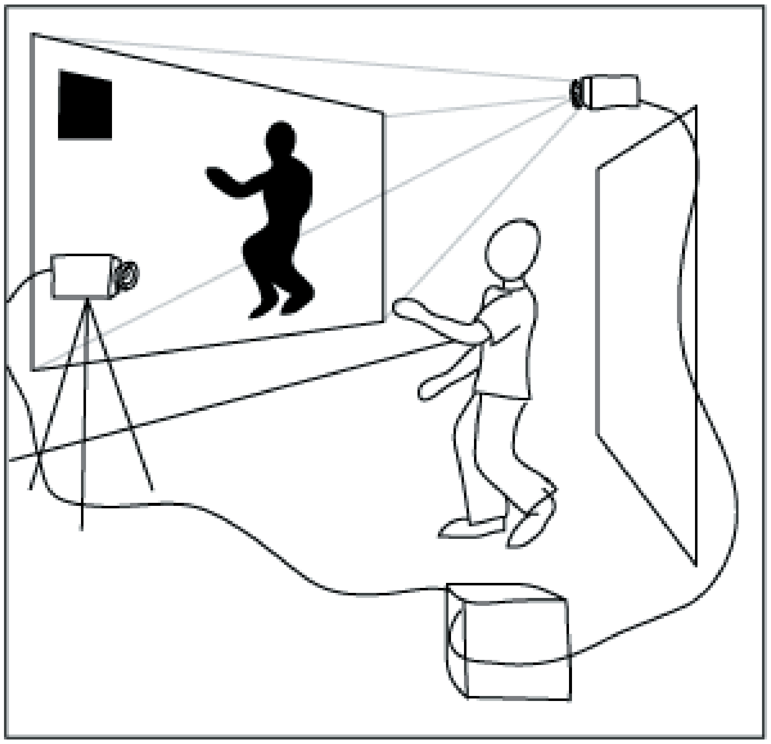


Figure 30: A sketch of the envisioned interactive setup. As in *Videoplace*, the user stands in front of a camera which captures their live image. This video data is then processed, and the user's silhouette is extracted. The user's silhouette and graphic objects with which they can interact are then projected in front of them.

### 3.3.2. Technical Configuration

#### 3.2.2.1. Envisioned Installation Setup

The first step I took in the creation of the *Duck and Jump* game, my initial objective, was to design the setup of a system that would allow me to test my computer vision interactions. The arrangement for this setup was inspired by Kreuger's *Videoplace*. As in *Videoplace*, I envisioned a scenario where the user would stand in front of a video camera and a display screen of some kind. The video image captured by the camera would be fed to a computer which would process the image and subtract the background, leaving just the user's silhouette. This silhouette would then be projected in front of the user creating a sort of "magic mirror," where the user could perceive their own video image moving in real-time. With this setup, I could then go about defining various relationships between the graphic objects and the user's silhouettes, like *Duck and Jump's* essential interaction: detecting when a graphic block touches a participant's silhouette.

### 3.2.2.2. Video Processing

To create such a system, I faced a number of challenges. The first of these challenges was simply to get a live video stream into my computer to be processed. After doing some research into some webcams and other types of cameras, I decided upon using an analog VHS camera. I chose to use an analog camera because of the speed of transfer that component video allows when compared to USB and other computer interfaces. To feed the video into my PC, I purchased a Hauppauge WinTV card, a card specifically designed to allow video image capture using component video. This setup fed live video into my computer at a rate faster than 30 frames a second, fast enough for *Duck and Jump*.

The next challenge was to find a tool that would allow me to capture live video and process it in real-time. I investigated a few options. The first options I looked at were Macromedia Director extras that facilitated video capture. The two available options I found were Danny Rozin's Track Them Colors and Josh Nimoy's WebCamXtra. While these options both captured live video and allowed for object tracking, they both had fairly slow frame rates even without introducing animation, collision detection or advanced image processing algorithms. Because a high frame rate was crucial to being able to recognize a user's jump, and thus the success of *Duck and Jump*, these options were eliminated.

I also researched video libraries for Max, a graphical programming environment for music and media applications for the Macintosh. These libraries consisted of Jitter, NATO and softVNS. In order for these options to achieve an acceptable frame rate, I would need high end Macintosh computers. Even so, introducing collision detection and other image processing algorithms would most likely slow the frame rate down to unacceptable levels. Also, the Max programming environment was not designed for the creation of large scale applications such as video games. The organization of such a large program in a graphical programming environment would prove tremendously time consuming.

In order to achieve a high enough frame rate and be able to program in an environment suited to the production of large scale applications, I decided to use C++, and Intel's open source computer vision library, OpenCV. OpenCV is a library created to assist in the creation real-time computer vision applications. It provides optimized code to assist developers working in many areas of computer vision including: Human-Computer Interaction, Object Identification, Segmentation and Recognition, Face Recognition, Gesture Recognition, Motion Tracking, and Motion Understanding; Structure From Motion, and Mobile Robotics. Using OpenCV's functions related to Segmentation and Recognition to create a background subtraction algorithm would allowed me to separate the user's silhouette from the background.

In order to use OpenCV's image processing and computer vision functions, I needed to be able to access the video data that was constantly being updated from the live video provided by the camera. To do this, I used Video for Windows, a set of functions provided by Microsoft to enable an application to process video data.

The last component in the system's pipeline was the output of the video image and graphic objects. To do this, I used OpenGL, a comprehensive and robust graphics library for C++. OpenGL is one of the most powerful libraries for the creation of rich real-time 2D and 3D graphics. This would allow me a tremendous amount of flexibility, power and speed in the creation and animation of the graphic objects with which my users would interact.

After the underlying structure of this pipeline was complete, I performed some simple tests in order to make sure the frame rate was acceptable for recognizing a user's jump, the essential interaction in my *Duck and Jump* game. I discovered, to my surprise, that the frame rate was not acceptable, and that a large amount of frames were being dropped. After some investigation, I realized that the graphics card that was installed with my computer was not OpenGL accelerated, which greatly decreased the performance. After purchasing and installing an OpenGL accelerated GeForce Ti 4200 graphics card into my computer, the frame rate rose to an acceptable rate.

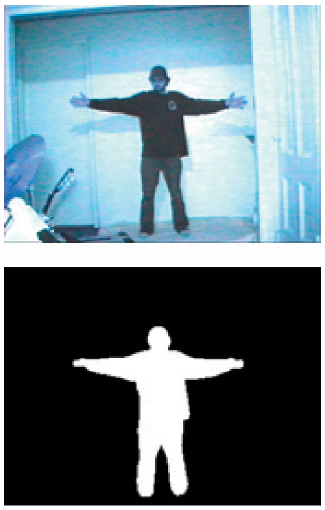


Figure 31: The top image is the live video image before processing. The bottom image is the resultant image after my background subtraction algorithm is applied .

### 3.2.2.3. Background Subtraction

The last large technological hurdle I had to overcome before I could start creating computer vision interactions was the development of the background subtraction algorithm to apply to the live video image data. This background subtraction algorithm would, ideally, allow me to separate the pixels that make up the background from the pixels that make up the user's image. I could then use the user's image pixel data in a collision detection algorithm to detect when a user came into contact with a graphic object.

The background subtraction algorithm I developed is fairly simple and straightforward. As such, the algorithm only works effectively in ideal lighting conditions. However since the final manifestation of this project will exist as an installation space with controlled lighting conditions and an established background, such an algorithm is sufficient. The development of optimal background subtraction routines is in itself a large field of research within computer vision. Since this is not the subject of my thesis, I decided to devote my time to the creation and investigation of computer vision interactions rather than the development of a robust background subtraction algorithm. If the project needs to exist in an environment with unpredictable lighting conditions or an inconsistent background, alterations to the algorithm can easily be made.

The background subtraction algorithm works by taking first taking an average of the pixels in the first 100 frames. This is done to create an static image that represents an average background image. This is more effective than using a single frame as a background in that it reduces the effect of slight changes in lighting conditions. However, using this technique necessitates that when the program is first run, for the first 100 frames of video capture, the user must be out of the video camera's field of vision so that the user's image is not factored into this image.

After the background image is computed, the algorithm compares this image with the image of the current frame. The algorithm goes through the images pixel by pixel. If the absolute value of the difference between a pixel in the background image and the corresponding pixel in the current

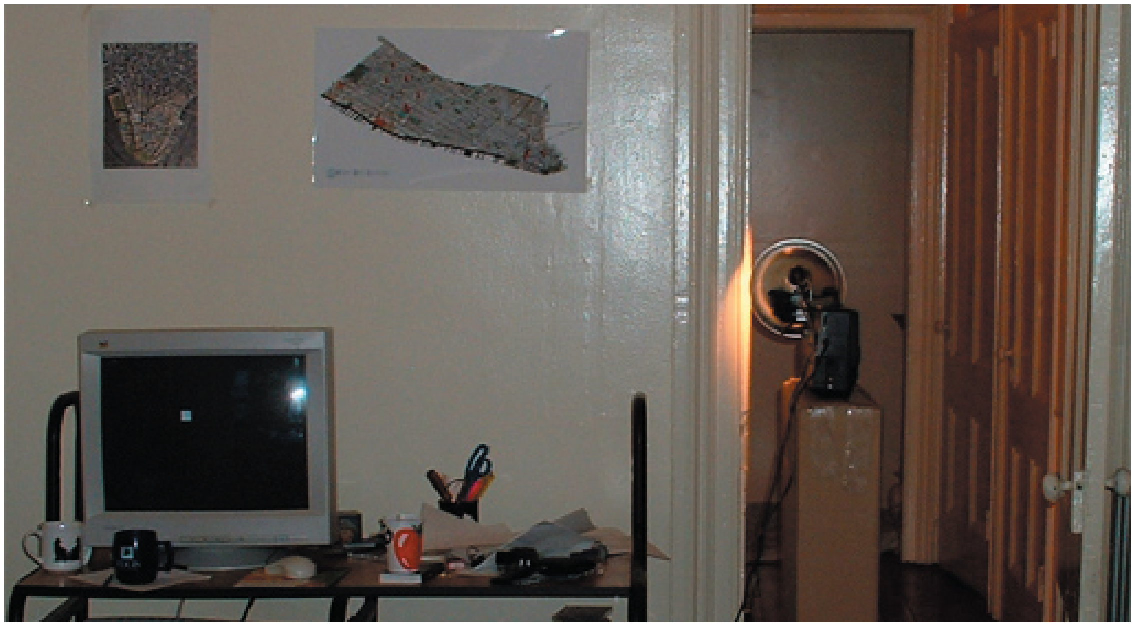


Figure 32: This was the testing environment. The camera and light record action in the far room.

frame is over a given threshold, it is considered part of the foreground. All foreground pixels are turned white. In such a way, an image where the user appears as a white silhouette over a black background is computed.

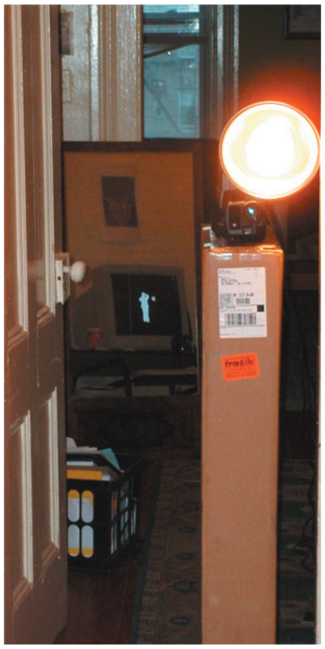


Figure 33: The user looks in a mirror to see the computer screen and be able to interact with graphic objects.

### 3.2.2.4. Testing Setup

In order to test and critically evaluate the computer vision interactions I would develop, I needed to create a testing environment. This space had a number of constraints. The setup needed to be in an environment with a neutral background (preferably all the same color) and controlled lighting conditions. The space also needed to be at least 15 feet long, in order to give enough space between the camera and user so as to allow the user's entire silhouette to be captured. Finally, the computer screen also had to be visible from the user's perspective, so they would be able to view the graphics objects with which they were to interact.

I set up an environment in my apartment with these constraints in mind. I pointed my camera at a blank white wall, and covered the red floor with a white sheet in an attempt to keep the background somewhat uniform. I positioned the camera 16 feet away from the wall and attached a flood lamp to the top of the camera so as to increase the contrast between the user and the background while minimizing resulting shadows that could be mistaken for part of the user. The one problem with this setup was that the white wall was in one room, and the camera and the computer was in another. This made it impossible for the user to see their image on the computer screen, and thus make it impossible for them to see the graphic blocks with which they were to interact. To remedy this problem, I positioned a full length mirror so that the screen could be viewed in the mirror from in front of the white wall, where the user was supposed to stand. This proved sufficient for testing purposes.

### 3.3.3. Basic Interactive Building Blocks

Now that I had a developed a pipeline for the creation and testing of computer vision interactions, I began implementing and testing my initial ideas. I began with developing a set of building blocks with which I could create other, more complex interactions. All of these interactions were designed following the design process I identified earlier. As such, they all are simple, intuitive interactions that can be initiated by any part of the user's body. They also all take place at the silhouette's edge.

#### 3.3.3.1. *Contact*

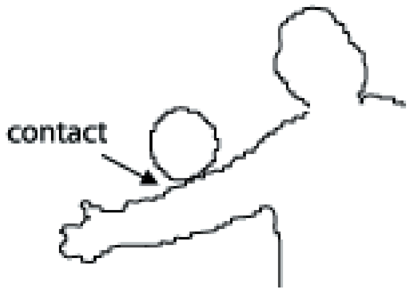


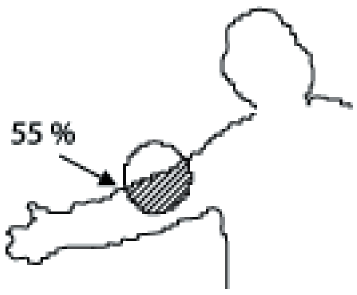
Figure 34: The *Contact* interaction simply detects when a user comes into contact with a graphic object.

The *Contact* interaction is the simplest of the building blocks. This was the first interaction I created after I set up my system. A stationary white block on the screen would simply turn red when the user's silhouette came into contact with it. This discrete interaction became the fundamental building block for all of my other interactions. As soon this simple interaction was built, a huge possibility space was opened up.

The algorithm I used to detect contact is fairly straightforward and could very easily be optimized. Since I never faced any frame rate issues even when more than 15 or so blocks were using the algorithm at once, an optimization of this algorithm was never needed. To detect contact, the algorithm first grabs any video region the block is currently lying over, and stores it in an array. This image data is then passed to a function that simply loops through the array and counts the number of foreground pixels. If the number of foreground pixels detected is over a given threshold, contact is detected. To minimize overlap, this threshold is set at a low number, usually somewhere between 1 and 10.

### 3.3.3.2. *Overlap*

Figure 35: The *Overlap* interaction detects the amount of overlap when a user moves their silhouette over a graphic object.



The second interactive building block was called *Overlap*. *Overlap* identifies the percentage overlap that occurs when the user moves their silhouette over a graphic object. While a small step from contact when considering my algorithm for detecting contact, it does allow for significantly different types of interactions to occur. While the second interaction I created didn't actually take the percentage of overlap of a block, it did identify the amount of overlap occurred with a graphic object. This interaction associated the amount of overlap with a given "trigger block" on the screen with the height of another "target block" in the environment. The more the user's silhouette overlapped the "trigger block," the higher the "target block" rose. This produced a continuous, levitating type of interaction.

The algorithm for detecting overlap is the same as that for detecting contact, except there is no threshold number of pixels. This number of pixels determines the amount of overlap.

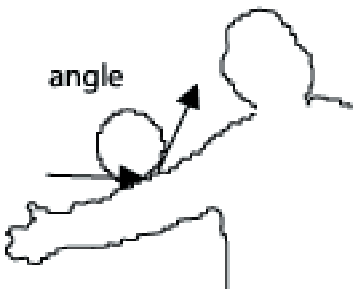


Figure 36: The *Reflect* interaction calculates the angle of reflection when a moving object strikes the user's silhouette.

### 3.3.3.3. *Reflect*

The last of these building blocks I identified was *Reflect*. *Reflect* was actually created after all of the other interactions and gamelets were completed. As such, it was not used in any of the gamelets, though it does have a large potential for use in game environments. The idea for this interaction was identified after realizing all the other types of interactions I was creating occurred at the silhouette's edge. This was another obvious edge related interaction which could be initiated by any part of the user's body.

*Reflect* identifies the normal angle of the user's silhouette at a point of contact with a graphic object. As such, it gives the user control over the motion of a traveling object. The current algorithm for determining the angle of reflection is not yet complete. As it stands now, the graphic object to be reflected must be moving one pixel at a time (very slow) in order for the normal angle of the silhouette's edge to be computed correctly. To remedy this problem, a more effective collision detection is needed. The algorithm currently works by first identifying the point of contact along the user's silhouette. This point of contact is then used in a Sobel edge detection algorithm in order to compute the normal angle of the silhouette's edge. This angle is then used in combination with the current direction of the graphic object to compute the angle of reflection.

### 3.3.4. Game Interactions

After I established this initial set of building blocks with which to work, I started to create more complex interactions. These new interactions were created by taking these basic interactive building blocks and associating a resulting action to it. These actions could affect the graphic object that was touched, another one in the environment, or the user's silhouette. In the following section I will describe three interactions that were developed from the *Contact* building block, and three that were developed from the *Overlap* building block.

#### 3.3.4.1. *Contact* Interactions - *Hold, Redirect, Shoot*

While there is an infinite number of possible interactions that could be developed from the *Contact* building block, I will discuss the three most successful I developed. The first of these three interactions is called *Hold*. In this interaction, the user is presented with a graphic block that is moving across the screen in a given direction. When the block reaches the edge of the screen, the block wraps, and appears on the other side of the screen. When the user's silhouette touches the block, the block stops. The use can then "let go" of the block by moving out of its path. In such a way, the user can control the movement of the block.

The second interaction I developed based on the contact building block was the *Redirect* interaction. This interaction is exactly the same as the hold interaction, except for one thing. When the user lets go of a held block, the block continues its movement in the opposite direction. This gives the effect of the block "bouncing" off of the user. This simple change produced an interesting and engaging emergent behavior. The interaction allowed the user to perform a sort of dribbling action, where the user could touch the block with one hand and redirect it to their other hand, which would touch it and redirect it back to the original hand. This could be repeated indefinitely to stall the block's forward movement.

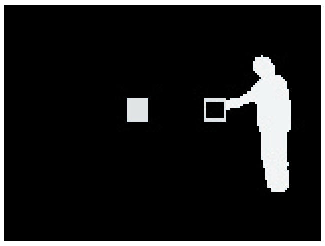


Figure 37: The *Shoot* interaction creates a block and propels it across the screen when you touch the "trigger block."

The last interaction I developed from the *Contact* building block was a the *Shoot* interaction. This interaction was very simple. When the user touches a colored block, a block of the same color and size would be propelled across the screen in a given direction.

### 3.3.4.2. *Overlap* Interactions - *Float, Grow, Push*

I also created three different interactions that were based on the *Overlap* building block. The first of these interactions, *Float*, was discussed earlier when I introduced the overlap building block. This interaction simply associates the amount of overlap when the user touches a graphic block to the height of another graphic block.

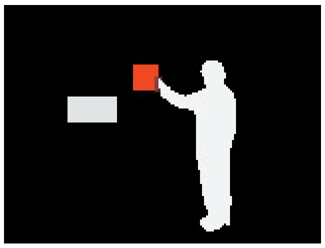


Figure 38: The *Float* interaction associates the amount of overlap with a "trigger block" (red) with the height of a "target block" (white).

The second of these *Overlap* based interactions is very similar to the *Float* interaction. In this interaction, the amount of overlap when a user touches a graphic block simply affects the size of another block. The more the user's silhouette overlaps the trigger block, the larger the other block will get.

The final *Overlap* based interaction that I developed was

*Push*. This interaction allows the user to push a block around the screen. The block can be pushed upwards, downwards, left and right depending on which side its pushed from. To create this interaction, I altered the contact detection algorithm. To determine which side the block is being pushed from, I divide the block into four sections and count the amount of pixels that overlap each section. The direction is determined by identifying the section with the most overlapping pixels. The block is then moved in the appropriate direction a given distance, dependent on how deep the overlap is.

## 3.3.5. Gamelets

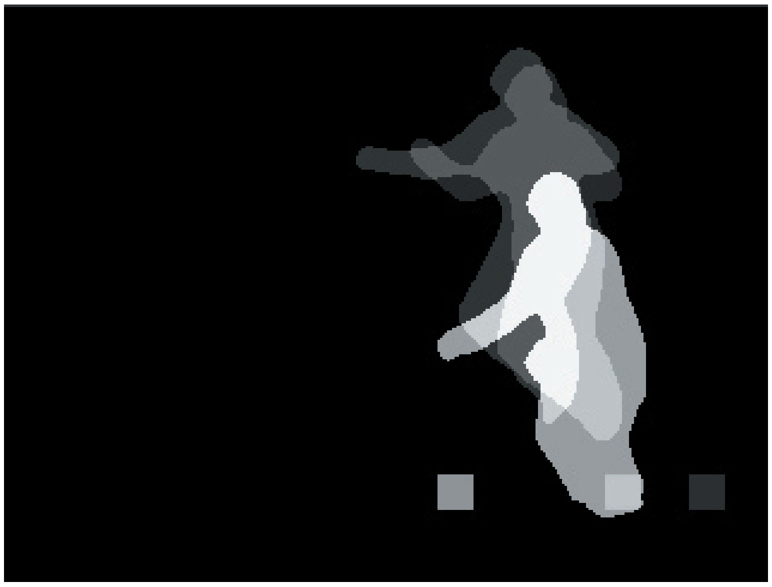
The following section will describe the gamelets that were developed in order to show how these interactions could be used in a game context. While *Color Match*, *Two Touch* and *Color Shooter* are all completely implemented, *Duck and Jump* and *Collide* have yet to be fully completed. Though these games need additions such as timers or displayed scores, their essential interaction of the experience is full functional. For clarity's sake I will describe the envisioned final version of each.

### 3.3.5.1. *Duck and Jump*

The idea for *Duck and Jump* was the spark that convinced me of the potential in creating unique computer vision based games. This game was discussed in the previous section *Abstracting Action*. It was the first game I thought of after seeing a few videos of Krueger's *Videoplace*. The premise of the game is very simple. The user simply must avoid touching moving graphic blocks.

To start the game, the user touches a start block located above their head on the far right of the screen. When the user touches the block, a timer located at the bottom of the screen appears, counting upwards. This also starts the generation of blocks introduced at the left edge of the screen which scroll horizontally across screen. These blocks are

Figure 39: In *Duck and Jump*, the user attempts to avoid blocks that travel horizontally across the screen. This image shows a few consecutive frames from the game overlaid.



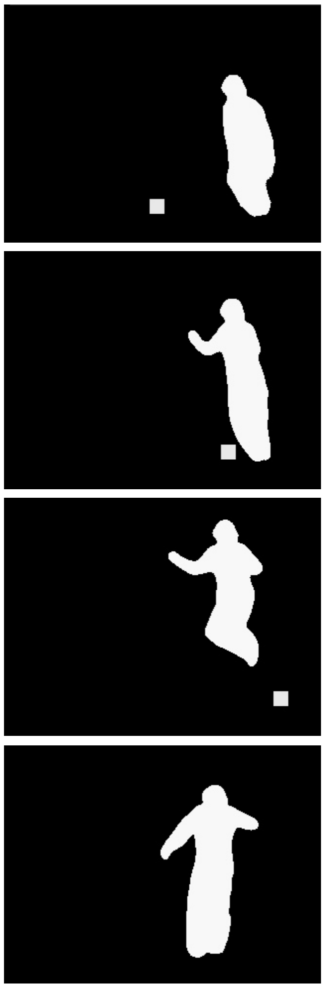


Figure 40-43: A sequence of frames from *Duck and Jump*.

generated at random intervals. However, a given amount of time must pass after one is generated before another can be generated. This is done in order to prevent two blocks from appearing in quick succession. These graphic blocks are approximately a fifth the size of the user's silhouette and all travel across the screen at the same speed. Their vertical position on the screen can be one of two possibilities: head or foot level. This is randomly determined when the block's is initially generated.

As stated earlier, the object of the game is simply to avoid touching the scrolling graphic blocks. The easiest way to accomplish this task is to stand in the far right hand corner of the screen and watch the left side of the screen as blocks appear. If a block appears at the bottom, the user must then jump or step over the approaching block at the appropriate time in order to avoid it. If the block is approaching the user's head, the user must duck under the block. When the user touches three blocks, the game ends. As the game progresses, the speed at which the blocks are generated gradually speeds up, making the task more difficult.

The object of this game is simply to stay alive for as long as possible. Thus the higher the time that appears at the end of the game, the better the player has done.

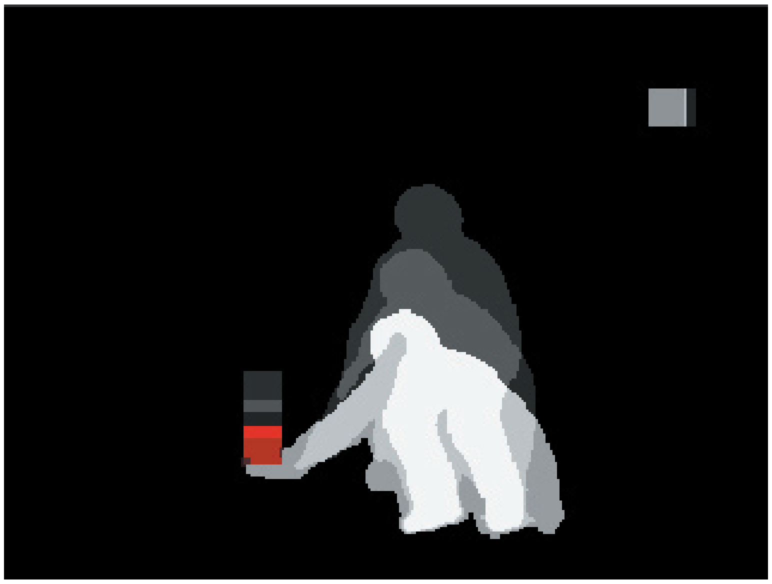


Figure 44: In *Collide*, the user attempts to collide moving blocks by reflecting them into each other. In this screenshot, the user is attempting to reflect the red block to the top of the screen in order to collide with the block now located at the top right of the screen.

## 3.3.5.2. *Collide*

The idea for *Collide* was developed naturally through the development of the *Hold* and *Redirect* game interactions. While this game could exist using either one of these interactions, the more engaging version uses the *Redirect* interaction.

The user starts the game by touching a block located above their head, in the middle of the screen. When the user touches the start block, a timer located at the bottom of the screen starts counting down from 30 seconds. Blocks then appear two at a time, each of which is introduced at a random point on the edge of the screen. In each pair, one block moves horizontally, and one moves vertically. The blocks wrap, so when they move off one side of the screen, they reappear on the other. When the player touches a block, the block reverses directions. When two blocks collide, they disappear. After the two blocks collide, a point is gained, and two more blocks randomly appear. This process continues until the 30 seconds has expired. The object of the game is to collide as many blocks, and thus score as many points, as possible in 30 seconds.

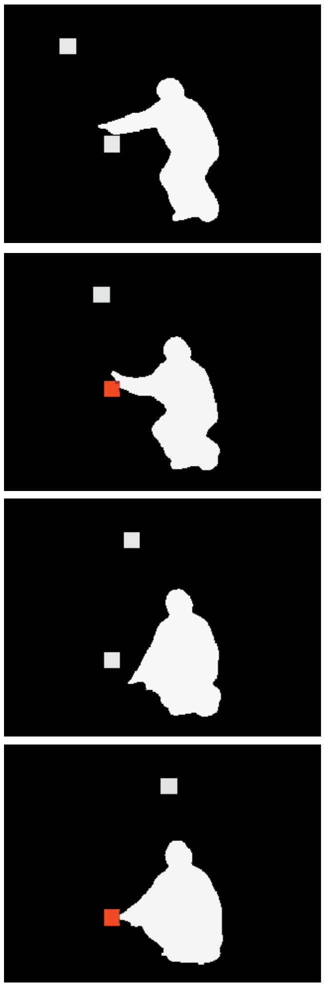
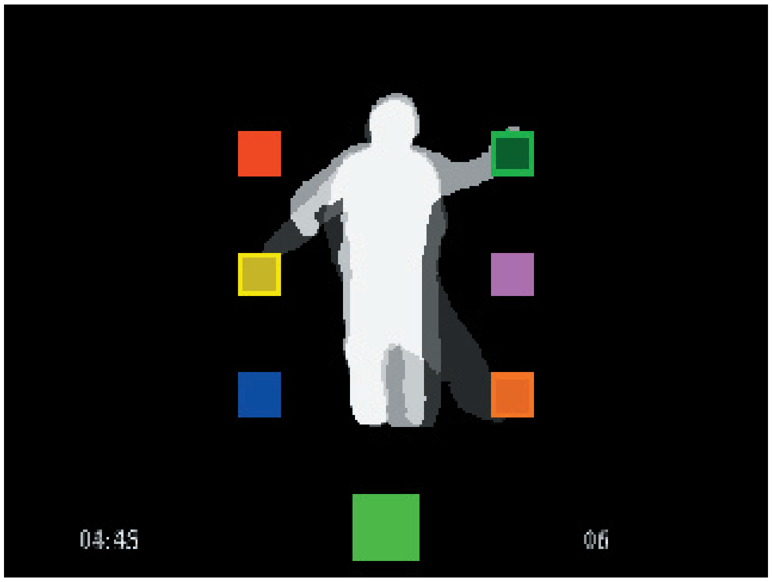


Figure 45-48: A sequence of frames from *Collide*.

Figure 49: In *Color Match*, the user attempts to touch the colored block that matches the "target block" located at the bottom of the screen as fast as possible. When the correct block is selected, the "target block" changes color and the process is repeated. This image shows a few consecutive frames from the game overlaid.



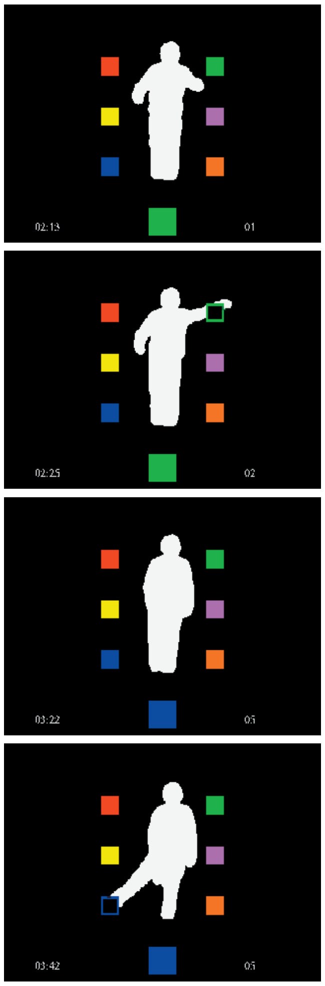


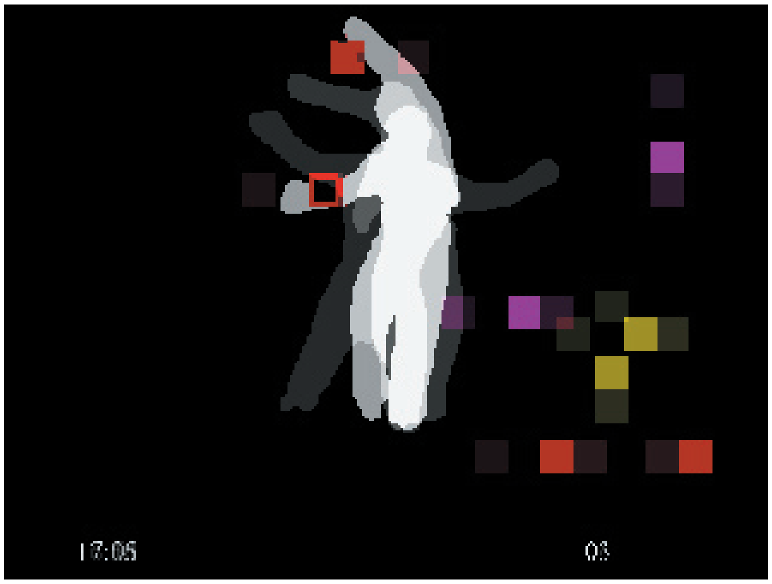
Figure 50-53: A sequence of frames from *Color Match*.

### 3.3.5.3. *Color Match*

The idea for this simple game came from thinking about how to develop a computer vision based *Dance Dance Revolution* or *Simon* type game where the user has to touch various points on the screen rather than positions on a footpad.

To start the game, the user touches the start block located above their head, in the middle of the screen. When this block is touched, a timer appears at the bottom of the screen which counts down from 30 seconds. Six different colored blocks also appear around the user's silhouette. Red, yellow and blue blocks appear to the user's left, and green purple and orange blocks appear to the user's right. The user is presented with a "target block" located on the bottom of the screen who's color matches the color of one of the six blocks around the player. When the user touches the block with the same color as the target block, the target block changes color and a point is gained. The game ends after the 30 seconds expire. The object of the game is to touch as many target colored as blocks as possible in these 30 seconds.

Figure 54: In *Two Touch*, the user attempts to two blocks of the same color at the same time. The blocks travel horizontally and vertically across the screen.



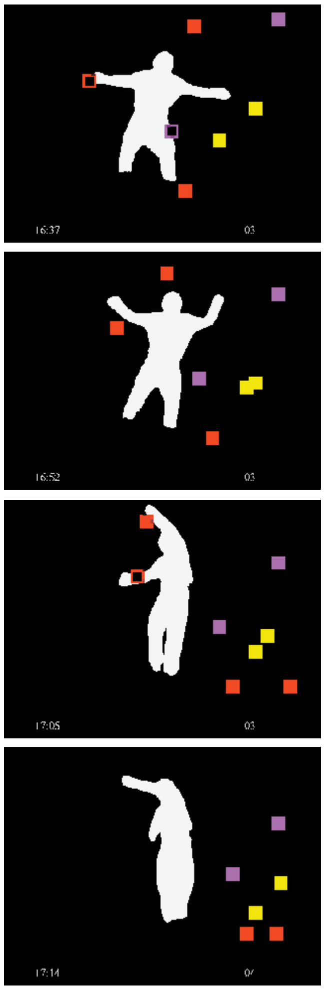


Figure 55-58: A sequence of frames from *Two Touch*.

### 3.3.5.4. *Two Touch*

The idea for the two touch game came after the creation of *Color Match*. In a way, *Two Touch* can be seen almost as a combination of the ideas that created *Color Match* and Collide. *Two Touch* is a game where the blocks move similar to the way the do in *Collide*, but whose essential interaction is closer to that in *Color Match*.

The user starts the game by touching a block located above their head, in the middle of the screen. When the game is started, a timer appears at the bottom of the screen that counts down from 30 seconds. Different colored blocks are introduced at random positions along the screen edges. When these blocks are initially generated, their color is also randomly determined as one of three different colors: yellow, red or purple. The colored blocks scroll vertically and horizontally across the screen, and wrap as in *Collide*. When the user touches two blocks of the same color, they disappear and a point is gained. The game ends after the 30 seconds expire. The object of the game is to collect as many points possible in these 30 seconds.

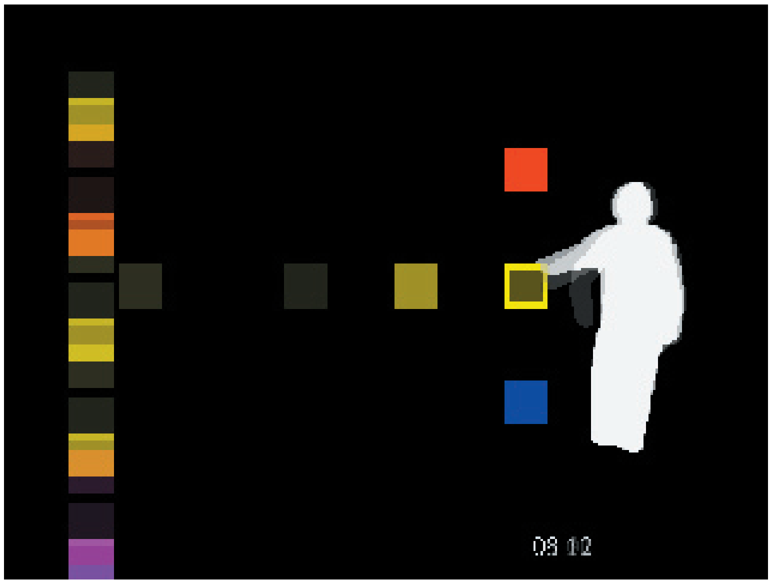


Figure 59: In *Color Shooter*, the user attempts shoot colored blocks at a line of vertically scrolling blocks on the left side of the screen.

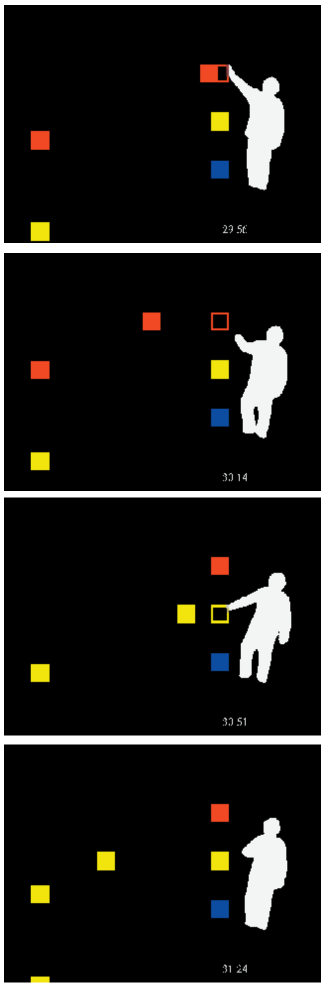


Figure 60-63: An sequence of frames from *Color Shooter*.

### 3.3.5.5. *Color Shooter*

The idea for *Color Shooter* was devised after the creation of *Color Match*. The game was initially imagined to have the same initial setup as *Color Match*, with six blocks surrounding the user. However, this setup proved too complicated, and the game was simplified by eliminating the blocks to the user's right. It was during the creation of this game that the *Shoot* interaction discussed previously was first developed.

To start the game, the user touches a block located above their head, on the far right of the screen. After this block is touched, a timer appears at the bottom of the screen counting upwards. Three different colored blocks also appear to the user's right. These blocks are colored yellow, red and blue. When a colored block is touched, a block of that color is propelled across the screen. On the left side of the screen, a series of blocks scroll upwards. The series of blocks wraps to the bottom of the screen. When a block in the series is shot with a block of the same color, it disappears. When a block in the series is a secondary color and is shot by a color that makes it up, that block turns into the other primary color that makes it up (an orange block hit with a yellow block turns red). The object of the game is to get rid of all of the blocks in the series as fast as you can. The lower

# 4. Evaluation and Analysis

In this chapter, I will evaluate the five gamelets introduced in the previous section, as well as the project as a whole. I will attempt to address questions such as: What is the experience of playing these gamelets like? How long do they keep a player's interest? Do they successfully incorporate the advantages of using computer vision? In order to answer these questions, I've divided the following chapter into three sections. The first of these sections, *General Gamelet Experiences and Playability*, will describe my experiences playing each gamelet, critically evaluate these experiences from a game play perspective, and suggest features to be considered if each were to be turned into a fully realized game. The next section, *Comparative Evaluation*, will comparatively evaluate how successfully the gamelets incorporated the advantages of computer vision I identified in the background chapter of this paper, as well as how successful they were at providing an engaging game experience. Finally, in *Project Evaluation*, I will evaluate the project as a whole, including both process and product.

## 4.1. General Gamelet Experiences and Playability

### 4.1.1. *Duck and Jump*

*Duck and Jump* is definitely the most exhilarating and exhausting of the gamelets created. I found myself becoming winded after a minute or so of playing. Though its athleticism is a crucial aspect to the fun of the experience, it brings up interesting questions about the design of such games. How much should aerobic fatigue be factored into the gameplay? How important should fitness and aerobic stamina be to achieving success?

From my experience of playing different versions of the game, I think the most successful version of *Duck and Jump* would balance the importance a user's skill and aerobic stamina. In the first version of *Duck and Jump* I created, the blocks that were propelled at the user were too large, and they scrolled to slowly. In this version I became winded a bit too quickly. While the game was fun, it bordered on feeling like a workout. To remedy this problem, I reduced the block size, sped up the speed at which the blocks traveled and shrunk the size of the user so as to give user more time to see the approaching blocks. After these alterations, the game became much more enjoyable. While the experience of the game was still a fairly aerobic one, I found myself attempting to figure out how to expend the least amount of energy while still successfully dodging the blocks. If timed correctly, the user could successfully jump over a quickly traveling block with minimal effort. In such a way, an accurate jump and ducker could limit his physical exertion and still be successful. While this quick fix did greatly improve the overall game experience, I barely scratched the surface in terms of investigating this issue of fitness. Were I to create this into a fully realized game experience, many more experiments and tweaking of variables would need to be carried out.

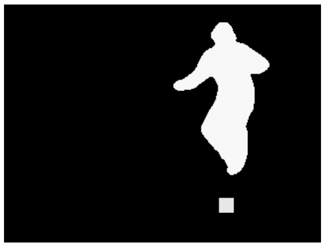


Figure 64: A screenshot from *Duck and Jump*. The user jumps just high enough to clear the block.

Another problem with the game is that it is a bit too one dimensional. The only interaction involved is *avoiding*. The game's premise is simple enough that more activities could be incorporated into the game experience without overwhelming the user. Since the blocks scroll so quickly across the screen and are introduced at random intervals, there exists points in the gameplay where the user is simply waiting for blocks to appear. While this represents important recovery time, another activity could be encouraged to happen during this time. One possibility is to have different colored blocks slowly fall from the top of the screen. These blocks could contain bonuses of some kind that could be obtained if touched before they scroll off the screen. These colored blocks would tempt the player to other parts of the screen besides the right corner. If caught in the middle of the screen while attempting to catch a falling block, the user would have less time to react to approaching blocks. Possible bonuses could be extra lives or possibly time towards a block that, when touched, would turn your silhouette invisible.

Another addition to the game that could possibly change the game experience for the better would be a small macro view of the experience that could show the user far in advance when and where blocks were approaching.

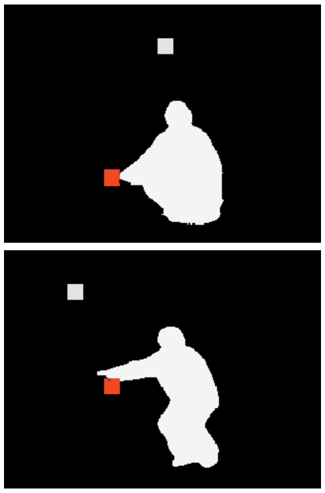


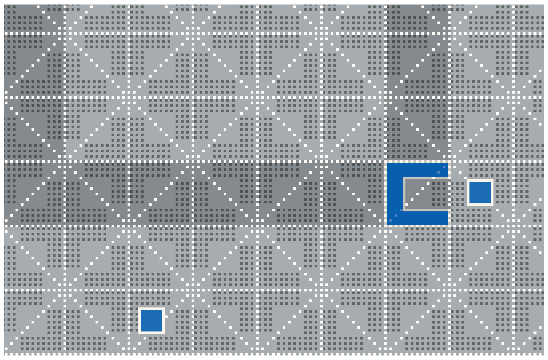
Figure 65 and 66: Two screenshots from *Collide*. The user "dribbles" a block back and forth.

### 4.1.2. *Collide*

The core mechanic of *Collide* is quite a satisfying one. Learning how to control the scrolling blocks by repeatedly redirecting them from one hand to the other proves challenging and fun to master. While this interaction is perhaps one of the most engaging the interactions I've incorporated into a game context, the design of the game itself is perhaps the least developed. As imagined now, the gamelet would repeatedly present the user with two blocks they must collide together. This is perhaps too simple too keep a user's interest if it were to exist as a fully realized game.

One way to complexity to the game experience would be simply to have more than two blocks on the screen at once. Perhaps four to eight blocks could exist on the screen at once. The blocks would be different colors, and would only disappear if they ran into a block of the same color. For this model to be implemented successfully, many tests and experiments would need to be conducted to determine the amount of blocks that can exist on the screen at once before the user gets completely overwhelmed.

Such an addition to the game would add another aspect to the game experience that doesn't exist in the current version of *Collide*. In order to explain the aspect of the gameplay of which I am speaking, I will describe how it is successfully incorporated into the experience of two existing games. These two games are gameLab's *Crash* and *Blix*. Like *Collide*, both of these games allow the user to coordinate the timing of predictably moving objects in order to achieve a given goal. In *Crash*, the user, by clicking on cars in order to slow them down and speed them up, attempts to prevent crashes between cars (a sort of anti-*Collide*). When first playing *Crash*, a novice user intensely will watch as two cars pass each other to make sure that the cars are set at speeds so that they don't run into each other. As a player gets better, the point at which he knows when two cars will crash and when they won't becomes earlier and earlier. As such, this lets the user set the speeds of two cars so they won't crash, and then move on to deal with the next two cars. In such a way, the user can start to handle more and more cars. In *Blix*, where the user has to direct horizontally and vertically squares into goal cups by placing walls in the environment, a similar thing occurs. A novice player of *Blix* will direct squares to their goal cups one at a time, and won't start directing the next block until he watches the first one go in the cup. As a user gets better, he will start dealing with other blocks before the previous one reaches the cup. The user knows he or she has set up a situation where the block must eventually reach the cup successfully.



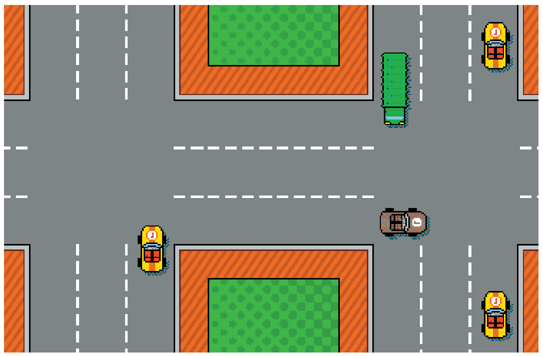


Figure 67 and 68: On the left, a screenshot from *Blix*, on the right, a screenshot from *Crash*.

With more than two blocks on the screen at one time, a similar situation could occur in *Collide*. A experienced user will know that two blocks are set up to hit each other and be able to concentrate on colliding the other blocks. Allowing for this type of interaction to occur tends reward more experienced players, and provide more lasting gameplay. Such an experience could not occur in a version with where only two blocks appear on the screen at once.

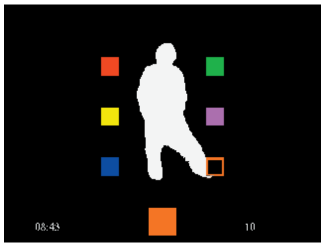


Figure 69: A screenshot from *Color Match*.

### 4.1.3. *Color Match*

As it exists now, *Color Match* is only engaging for a very limited amount of time. The primary reason for this is that a user quickly reaches a point where there is very little room for improvement. Though my score increased the first three or four times I played the gamelet, after this point, I couldn't improve upon my score. There is always a small amount of time it takes to recognize what color you are being presented with. Attempting to limit this amount of time tends to be more frustrating than engaging after a certain point.

However, despite its current problems, I think the gamelet has a lot of potential for improvement. There are a few small changes that could immediately add the its gameplay. One of these is simply changing the target block from an individual block to a scrolling list of target blocks. This list would allow to see the next target block in advance which would give more advanced users the opportunity to prepare for it earlier. Another small change that could drastically improve its gameplay, would be to allow two target blocks to appear at the same time. The user would then have to touch both blocks at the same time in order to move on the next block.

Another large potential in this game lies in the effective incorporation of sound and music. One option is to create a *Dance Dance Revolution* type game, where there is a soundtrack, and the user must sync the touching of the blocks to the beat in the music. Another perhaps more interesting option would be to try and utilize a more generative model, where the user actually creates sound through touching the blocks. One option would be to create a game where the user has to actually play a beat or composition at a certain speed before he can advance to a more difficult one.

### 4.1.4. *Two Touch*

*Two Touch* provides an interesting interactive experience. While for the most part the experience of playing it is

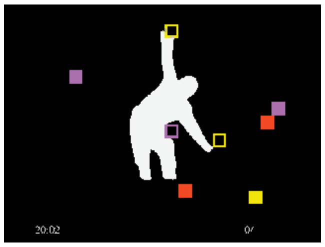


Figure 70: A screenshot from *Two Touch.*

somewhat frustrating, it does provides some unique and dramatic interactive moments that point to its potential. The gamelet's main problem is that most of the blocks that scroll on the screen are either out of the user's reach or they just happen to run into the user's silhouette and disappear. The fun moments come on the rare occasions when two similarly colored blocks are within reach of the user's silhouette, but don't actually run into it. These situations, while somewhat rare, can leas to some very interesting an unique motions and movements. For the majority of the game as it exists now, the user waits for these kinds of moments to occur.

A few adjustments could be made that would, hopefully, make these moments occur more frequently. This could be accomplished by restricting the positions where blocks can be introduced. When a new block is generated, the program could find the dimensions of the user's silhouette as well as the user's range of motion and randomly select a position between the user and the edge of the user's range of motion.

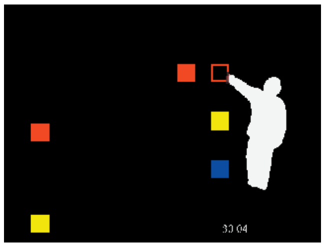


Figure 71: A screenshot from *Color Shooter.*

### *4.1.5. Color Shooter*

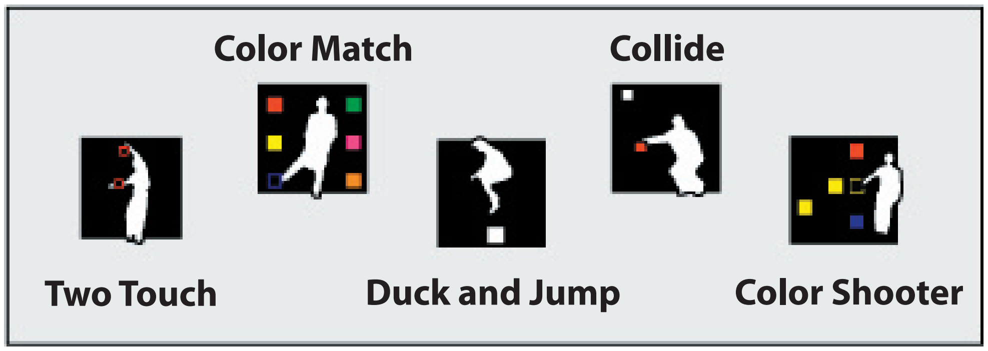
*Color Shooter* is the most complete of the gamelets and is by far the most fun to play. The shoot interaction in itself is a very satisfying one, and coordinating its timing to hit an other blocks adds to this. Another reason for its success is how the trigger blocks are positioned in relation to the user's body. While a similar type of game could be created as a game played with a mouse, where the user would click on the appropriate trigger block, it would not be nearly as engaging. One reason for this is because of the pleasure that is gained from being able to punch and kick the blocks. Coordinating the timing of a punch or kick is much more challenging and fun than coordinating the timing of a mouse click. Another advantage of the computer vision based version is the user's ability to trigger more than one block at the same time, by kicking and punching at the same time.

While the game does have many strengths, it also has some significant weaknesses. The main weakness of the game is that it fails to fully incorporate the advantages of using computer vision as a gaming interface. Though it does provide a full body interactive experience, there isn't much flexibility or variation in the type of interactivity that occurs. As such, it limits the amount of creative solutions possible. This is due to the limitations of the shooting interaction. Because all the trigger blocks are all stationary, and these trigger blocks are where all of the game's interaction occurs, it tends to encourage a fairly uniform type of interaction.

## 4.2. Comparative Evaluation

In the following section I compare the gamelets' success in two different respects. I will compare their success in incorporating the advantages of using computer vision as a gaming interface, as well as their success from a general gameplay perspective. I've identified five different axes to compare the different gamelets along. These axes were taken from the five advantages of using computer vision in video games identified in the background chapter, except for *Closer Interactive Mappings*. This is left out because my design process necessitated that all of the interactions I developed would be triggered by direct manipulation. As such, all games would be equally successful along this axis. The axes identified are: Athletic, Expressive, Full Body, Vocabulary of Action, and Playability. I've created a series of icons for the five different gamelets in order to facilitate a visual analysis.

Figure 72: A key to the icons used in the comparative analysis of the gamelets.



### 4.1.1. Athleticism

How successfully does each gamelet incorporate the potential for athletic interaction that computer vision provides? While each game allows for a full body interaction, some encourage faster and more sustained movement than others. *Duck and Jump*, by far, provides the most athletic experience of all the games. *Color Shooter*, because of the nature of the shooting interaction and its stationary interactive blocks, necessitates the least amount of athletic activity.

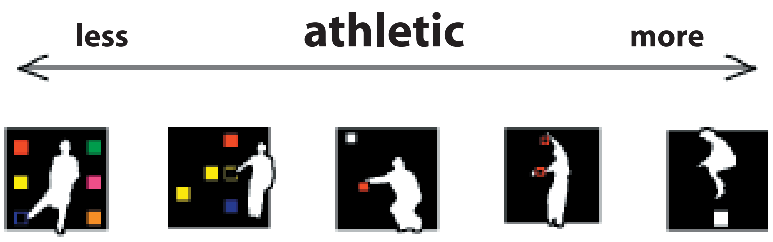


Figure 73: How athletic of an experience each gamelet provides.

### 4.1.2. Expressivity

This axes addresses the dramatic and performance nature of the interactive experiences the gamelets provide. The games I have rated the most successful on this axis encourage the most dramatic and expressive interactive experiences. They are also the gamelets that would be most apt to accentuate differences in users' personalities. *Two Touch*, *Duck and Jump*, and *Collide* all encourage a fair amount of unique interaction that could potentially allow for more user expression. Because of their limited interactive range and their stationary interactive blocks, Color Shooter and Color Match both encourage a fairly uniform interactive experience.

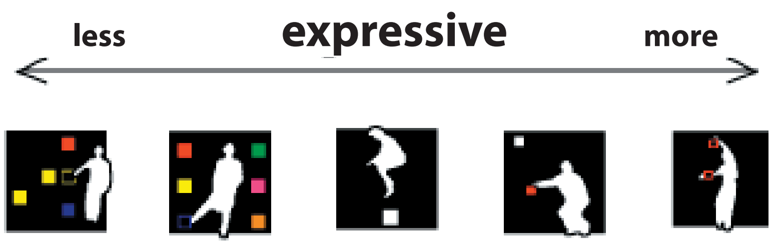


Figure 74: How much each gamelet encourages expressive interaction.

### 4.1.3. Whole Body

This axis analyzes how successfully each game encourages the players use of their entire body. Because of the design process I identified in the *Methodology* section, all of the actual interactions in each gamelet should be able to be triggered by any part of the user's silhouette. However, some games are more successful then others at encouraging this type of interaction from the user. *Two Touch* is the most successful of the games at encouraging full body interaction. *Two Touch* user's often end up contorting and stretching their bodies in order to touch two blocks at the same time.

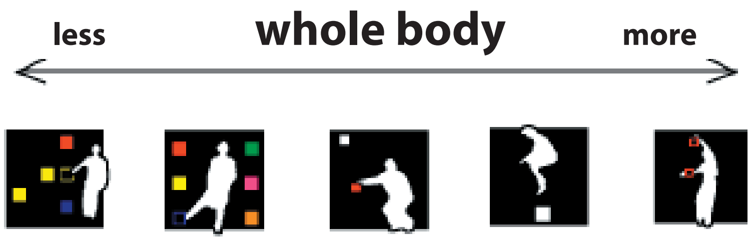


Figure 75: How much each gamelet encourages full body interaction.

### 4.1.4. Vocabulary of Action

Vocabulary of Action refers to the number of significantly different types of interactions that can be recognized by each gamelet. This axis is to help me get a general idea of which gamelets encourage a wide variety of action and which gamelets provide a more one dimensional interactive experience. *Two Touch* and *Collide* were the most successful gamelets at encouraging a wide range of interactions. This is probably partially due to the fact that in both these gamelets, the user interacts with blocks that are constantly moving.

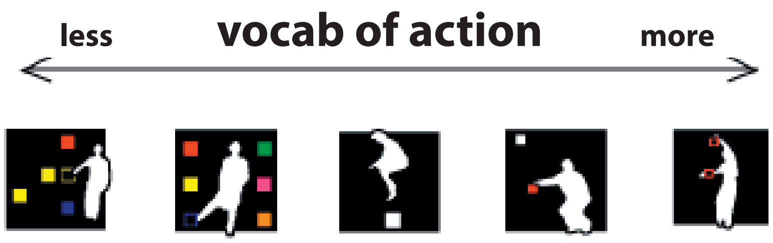


Figure 76: How many different types of interactions are available in each gamelet experience.

### 4.1.5. Playability

I've defined Playability in this context as a combination of how long each gamelet holds my interest and how enjoyable the experience of playing it is. While this is definitely the most subjective and slippery of the axes, it is also perhaps the most important. Providing engaging gameplay and fun are the ultimate goals in creating any game experience. As such, any gamelet I produce that does not provide an engaging experience of some sort, or point to potential directions that could provide such an experience, is useless to my study. I found the most engaging of the gamelets as they exist now, to be *Color Shooter*.

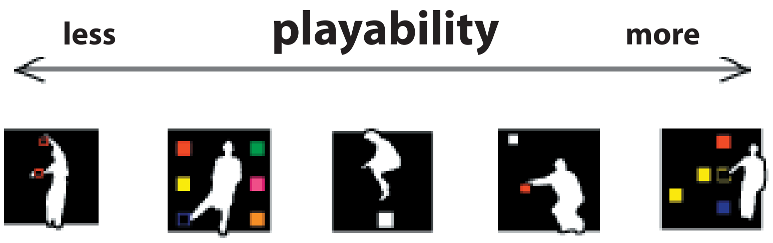


Figure 77: How fun is each gamelet to play.

### 4.3. Project Evaluation

In order to evaluate this project as a whole, I will investigate my project from a few different perspectives. I will first evaluate the process I went through to formulate my concept. I will then evaluate the process of creating the game interactions, and the gamelets themselves. Finally, I will address the evaluation of the gamelet experiences as a whole.

As evident by the length of the *Initial Interests* and *Abstracting Action* sections of the *Methodology* section, the development of a solid conceptual base for this project was one of the most challenging aspects of this project. While the process I went through to formulate the concept for this project proved to be a valuable personal learning experience.

Having said this, overall, I view this project as a success. The main goal of the project, which was to show the potential of full body interactions in video games was, in my opinion, successfully communicated through the number of engaging interactions and gamelets proposed by this thesis. I think this is partially due to the creative process I developed for myself. This process entailed breaking down the problem into smaller and smaller problems, and constraining my design process to focus on the issues I was most concerned with. Restricting my visual language as well as the types of interactive experiences I could create (only those that occur at the edge of a user's silhouette), facilitated experimentation and the implementation of my ideas by dramatically reducing the scope of the problem. I think that this process reduced the overwhelming nature of the questions I was asking and allowed me to focus on the basic interactive issues that were crucial to their solutions.

The last evaluation missing from this project will occur by watching the user response to the final installation piece. While I had a few people play the gamelets I created in the testing setup I developed at my apartment, this installation was significantly flawed, and as such, it was difficult to evaluate whether any confusion that occurred during their play was due to the gamelets themselves, or the setup. With a free range of motion, a backlit background, and most importantly, a large projected display, the experience of these gamelets will be completely different.

# 5. Conclusions

## 5.1. Conclusion

The underlying goal of this project from its conception has been to challenge the traditional image of the video game experience. It attempts to do this by offering an alternative to the stationary, hand-centric experience that most existing video games provide. It proposes a scenario where the player can affect action in the game by using their entire body, free of wires and controllers. In order to facilitate such an unencumbered interactive experience, a video camera is used as an input device. In a setup similar to Krueger's *Videoplace*, the user stands in front of a video camera, which records their live video image. Using computer vision, this image is then processed, and the user's silhouette is extracted. The user's silhouette, as well as game objects with which they can interact, is then projected on a surface in front of them.

While there exists a few systems that use computer vision in a video game context, such as the Vivid Group's *Mandala GX System*, Reality Fusion's *GameCam* and Intel's Me2Cam, each of these systems fail to fully incorporate the its potentials as a gaming interface. Most of the games developed for the *Mandala GX System* utilize symbolic interactions rather than literal ones, which increases the distance between action and outcome, and thus makes them less intuitive. The *GameCam*, and the *Me2Cam* both use USB webcam technologies which, while cheap, causes a low frame rate that distracts from the experience. Also, the games designed for these systems we created for young children and as such, can only hold a user's attention for a limited amount of time.

To aid in the creation of more engaging and intuitive full body games, this thesis identifies a vocabulary of possible computer vision based game interactions. A series of functional game scenarios, or "gamelets," were then created to demonstrate how these interactions can be incorporated into game contexts. These gamelets were then evaluated based on how successfully each incorporated the advantages of using computer vision as an interface and on how fun they are to play.

## 5.2. Future Directions

Perhaps the biggest success of this thesis are the opportunities for further investigation and development it has revealed. During the creation of this project, new avenues of exploration were constantly being discovered, not only by myself, but by peers and advisors as well. When presenting my project to others, suggestions for new features, potential interactions, and game ideas came by the dozens. In order to stay focused enough to demonstrate any one concept in depth, many of these ideas needed to be tabled. Since many of these ideas and suggestions were extremely good ones, the documentation of these ideas became an important exercise. In the following section, I will discuss the most important of these documented ideas.

One of these future directions entails the incorporation of dramatic positive and negative feedback. For example, in the game *Collide*, as it currently exists, when two blocks collide, they simply disappear. This feedback is somewhat subtle, and doesn't provide the user with a satisfying enough response to the successful completion of a task. A small animation when two blocks collide could add a lot to the game experience, as would sound effects. Such an animation could simply be a short animation of the blocks getting larger and fading to black, rotating the blocks and fading to black, or simple a flash of the appropriate color. Sounds could be anything from a gunshot, to a atari type noise, to an exclamation. Similar types of feedback using animation and sound could be incorporated into each gamelet.

Another future direction involves the creation of games that utilize the *Push* and *Reflect* interactions I identified in the *Methodology* chapter. Both of these interactions are engaging simply as interactive experiences, but have yet to be shown how they could be used in a game context. One game idea involving the *Push* interaction that has yet to be implemented involves pushing a block around in order to redirect a constantly growing line towards various goal positions. While the *Reflect* interaction has enormous potential for use in computer vision games (and has been used in existing games like the Vivid Group's *Volleyball*), since it was one of the last interactions developed, I didn't have the opportunity to show its use in a game context. A simple game that could incorporate the *Reflect* interaction would be one where the user attempts to redirect randomly generated colored balls traveling across the screen into similarly colored stationary goal positions.

Issues related to the specifics of its installation experience have also yet to be fully fleshed out. The biggest challenges in creating an installation that allows users to play the multiple gamelets include: designing an intuitive interface to allow the user to select between gamelets, educating the users on how to play the gamelets, and recording high scores for each gamelet.

Perhaps the largest potential area for future exploration is the development of computer vision games that allow multiple users to play at once. The incorporation of multiple users in such a gaming environment has enormous potential for creating interesting social dynamics. Because of the area's huge potentials, and the fact that the design of such games involves a significantly different type of thinking, I felt as if I didn't have the time to sufficiently address this topic in this thesis. However, it is an area I will be sure to be explore in the future development of this project.

# Appendix A. Gamelet Pseudocode

## A.1. *Duck and Jump*

```
void mainloop(){
        if (frameCt > 100) {
                if (startBlock->isActive) {
                        glReadPixels(startBlock->xPos,
                        startBlock->yPos, startBlock-
                        >height, startBlock->width, test_
                        block);

                        startBlock->intersect(test_
                        block);

                        if (startBlock->numHit==1) {
                                startGame();
                        }
                }
                if (gameState == 1) {
                        createBlocks();
                        updateBlocks();
                }
        }
        frameCt++;
}

void createBlocks(){
        int randNum = 0;
        int nextBlock;

        randNum = int(rand() % blockFreq);
        nextBlock = findNextFreeBlock();
        if (createBuffer < frameCt - lastCreated) {
                if (randNum == 0) {
                        blocks[nextBlock]->isActive=true;
                        lastCreated=frameCt;
                        randNum = int(rand() % 2);
                        if (randNum == 0) {
                                blocks[nextBlock]-
                                >setPos(-40, 155);
                        }
                        else {
                                blocks[nextBlock]-
                                >setPos(-40, 415);
                        }
                }
        }
}

void updateBlocks(){
       for(int i=0;i<10;i++) {
              if (blocks[i]->isActive) {
                     glReadPixels(blocks[i]->xPos,
                     blocks[i]->yPos, blocks[i]-
                     >height, blocks[i]->width, test_
                     block);

                     blocks[i]->intersect(test_block);
                     blocks[i]->move(20,0);
              }
       }
}
```

## A.2. *Collide*

```
void mainloop(){
       if (frameCt > 100) {
              if (gameState == 0) {
                     test_block = getScreenRegion(
                     startBlock->xPos, startBlock-
                     >yPos, startBlock->width,
                     startBlock->height);

                     startBlock->intersect(test_
                     block);

                     if (startBlock->isHit == true) {
                            startGame();
                            blocks[0]->isActive=true;
                            blocks[0]->setPos(0, 400);
                            blocks[1]->isActive=true;
                            blocks[1]->direction=2;
                            blocks[1]->setPos(300, 0);
                     }
              }
              else {
                     updateBlocks();
                     collisionDetect();
              }
       }
       frameCt++;
}

void updateBlocks(){
       for(int i=0;i<10;i++) {
              if (blocks[i]->isActive) {
                     test_block = getScreenRegion(
                     blocks[i]->xPos, blocks[i]->yPos,
                     blocks[i]->width, blocks[i]-
                     >height);
```

```
                        blocks[i]->intersect(test_block);
                        if (blocks[i]->isHit == false)
                                blocks[i]->moveWrap(7);
                }
        }
}
```

## A.3. *Color Match*

```
void mainloop(){
        if (frameCt > 100) {
                if (gameState == 0) {
                        test_block = getScreenRegion(
                        startBlock->xPos, startBlock-
                        >yPos, startBlock->width,
                        startBlock->height);

                        startBlock->intersect(test_
                        block);

                        if (startBlock->numHit==1) {
                                startGame();
                        }
                }
                else if (gameState == 1) {
                        checkForMatch();
                        createBlocks();
                        updateBlocks();
                        if (!theTimer->isActive)
                                gameState = 2;
                }
        }
        frameCt++;
}

void createBlocks(){
        int randNum = 0;
        randNum = int(rand() % 6);
        if (!target_blocks[0]->isActive) {
                target_blocks[0]->isActive = true;
                if (randNum == 0)
                        target_blocks[0]->
                        changeColor(0.0f, 0.0f, 1.0f);
                else if (randNum == 1)
                        target_blocks[0]->
                        changeColor(1.0f, 1.0f, 0.0f);
                else if (randNum == 2)
                        target_blocks[0]-
                        >changeColor(1.0f, 0.0f, 0.0f);
                else if (randNum == 3)
                        target_blocks[0]-
                >changeColor(1.0f, 0.5f, 0.0f);
            else if (randNum == 4)
                target_blocks[0]-
                >changeColor(1.0f, 0.0f, 1.0f);
            else if (randNum == 5)
                target_blocks[0]-
                >changeColor(0.0f, 1.0f, 0.0f);
        }
}

void updateBlocks(){
        for(int i=0;i<6;i++) {
            if (active_blocks[i]->isActive) {

                test_block =
                getScreenRegion(active_blocks[i]-
                >xPos, active_blocks[i]->yPos,
                active_blocks[i]->height, active_
                blocks[i]->width);

                active_blocks[i]->intersect(test_
                block);
            }
        }
}
```

## A.4. *Two Touch*

```
void mainloop(){
        if (frameCt > 100) {
            if (gameState == 0) {
                test_block = getScreenRegion(sta
                rtBlock->xPos, startBlock->yPos,
                startBlock->width, startBlock-
                >height);

                startBlock->intersect(test_
                block);

                if (startBlock->numHit==1) {
                    startGame();
                }
            }
            else if (gameState == 1) {
                checkForMatch();
                createBlocks();
                updateBlocks();
                if (!theTimer->isActive)
                    gameState = 2;
            }
        }
        frameCt++;
}
```

```
void createBlocks(){
       int randNum = 0;
       int nextBlock;

       randNum = int(rand() % blockFreq);
       nextBlock = findNextFreeBlock();
       if (createBuffer < frameCt - lastCreated) {
              if (randNum == 0) {
                     blocks[nextBlock]->isActive =
                     true;

                     lastCreated = frameCt;
                     randNum = int(rand() % 4);
                     blocks[nextBlock]->direction =
                     randNum;

                     if (blocks[nextBlock]->direction
                     == 0) {

                     randNum = int( (rand() %
                     int(pixZoomX*320)) + vidOffsetX);
                     blocks[nextBlock]-
                     >setPos(randNum, 0);
                     }
                     else if (blocks[nextBlock]-
                     >direction == 1) {

                     randNum = int( (rand() %
                     int(pixZoomY*240)) + vidOffsetY);
                     blocks[nextBlock]->setPos(0,
                     randNum);

                     }
                     else if (blocks[nextBlock]-
                     >direction == 2) {

                     randNum = int( (rand() %
                     int(pixZoomX*320)) + vidOffsetX);

                     blocks[nextBlock]->setPos(-
                     randNum, 600);

                     }
                     else if (blocks[nextBlock]-
                     >direction == 3) {

                     randNum = int( (rand() %
                     int(pixZoomY*240)) + vidOffsetY
                     );

                     blocks[nextBlock]->setPos(800,
                     randNum);
                     }
                     randNum = int(rand() % 6);
                     if (randNum == 0) {
                     blocks[nextBlock]-
                     >changeColor(1.0f, 0.0f, 0.0f);

                     }
                     else if (randNum == 1) {

                     blocks[nextBlock]-
                     >changeColor(1.0f, 1.0f, 0.0f);

                     }
                     else if (randNum == 2) {

                     blocks[nextBlock]-
                     >changeColor(1.0f, 1.0f, 0.0f);

                     }
                     else if (randNum == 3) {

                     blocks[nextBlock]-
                     >changeColor(0.0f, 1.0f, 0.0f);

                     }
                     else if (randNum == 4) {

                     blocks[nextBlock]-
                     >changeColor(1.0f, 0.0f, 1.0f);

                     }
                     else {

                     blocks[nextBlock]-
                     >changeColor(1.0f, 0.5f, 0.0f);

                     }
             }
     }
 }

 void updateBlocks(){
     for(int i=0;i<20;i++) {
             if (blocks[i]->isActive) {

                     test_block = getScreenRegion(bl
                     ocks[i]->xPos, blocks[i]->yPos,
                     blocks[i]->height, blocks[i]-
                     >width);

                     blocks[i]->intersect(test_block);
                     blocks[i]->moveWrap(15);
             }
     }
 }
```

## A.5. *Color Shooter*

```
void mainloop(){

	if (frameCt > 100) {
		if (gameState == 0) {
			test_block = getScreenRegion(sta
			rtBlock->xPos, startBlock->yPos,
			startBlock->width, startBlock-
			>height);

			startBlock->intersect(test_
			block);

			if (startBlock->numHit==1) {
				startGame();
			}
		}
		else if (gameState == 1) {
			checkForMatch();
			createBlocks();
			updateBlocks();
			collisionDetect();
			if (!theTimer->isActive)
				gameState = 2;
		}
	}
	frameCt++;
}

void createBlocks(){
	int randNum = 0;

	randNum = int(rand() % 6);
	if (!target_blocks[0]->isActive) {

		target_blocks[0]->isActive = true;

		if (randNum == 0)
			target_blocks[0]-
			>changeColor(0.0f, 0.0f, 1.0f);
		else if (randNum == 1)
			target_blocks[0]-
			>changeColor(1.0f, 1.0f, 0.0f);
		else if (randNum == 2)
			target_blocks[0]-
			>changeColor(1.0f, 0.0f, 0.0f);
		else if (randNum == 3)
			target_blocks[0]-
			>changeColor(1.0f, 0.5f, 0.0f);
		else if (randNum == 4)
			target_blocks[0]-
			>changeColor(1.0f, 0.0f, 1.0f);
		else if (randNum == 5)
```

```
                        target_blocks[0]-
                        >changeColor(0.0f, 1.0f, 0.0f);
        }
}

void updateBlocks(){
        for(int i=0;i<3;i++) {
                if (active_blocks[i]->isActive) {
                        int hitCount = active_blocks[i]-
                        >numHit;

                        test_block =
                        getScreenRegion(active_blocks[i]-
                        >xPos, active_blocks[i]->yPos,
                        active_blocks[i]->height, active_
                        blocks[i]->width);

                        active_blocks[i]->intersect(test_
                        block);

                        if (hitCount < active_blocks[i]-
                        >numHit && !shot_blocks[i]-
                        >isActive) {

                                shot_blocks[i]->isActive =
                                true;
                        }
                }
        }
        for(int i=0;i<3;i++) {
                if (shot_blocks[i]->isActive) {
                        shot_blocks[i]->move(20);
                }
        }
        for(int i=0;i<10;i++) {
                if (target_blocks[i]->isActive) {
                        target_blocks[i]->moveWrap(5);
                }
        }
}
```

# Bibliography

Able Minds, Inc. “Me2Cam.” <http://www.cyberkids.com/cy/so/mul/html/me2cam.html>, 2000.

Ars Electronica 99. *LifeScience*. <http://www.aec.at/lifescience/pressepic/pic_installations2.html>

Berger, Sandy. “Me2Cam.” <http://www.compukiss.com/populartopics/computercenterhtm/review93.htm>, 2000.

Burnham, Van. *Supercade: A Visual History of the Videogame Age*, 1971-1984. Cambridge, Mass: The MIT Press, 2001.

CNET Networks. “Put Yourself into the Game.” <http://zdnet.com.com/2100-11-518799.html?legacy=zdnn>, 2000.

Crawford, Chris. *The Art of Computer Game Design*. Osborne McGraw-Hill, 1984.

D’Hooge, Herman and Goldsmith, Michael. “Game Design Principles for the Intel® Play™ Me2Cam Virtual Game System.”

<http://www.intel.com/technology/itj/q42001/articles/art_4.htm>, 1999.

Epinions. “Reality Fusion GameCam.” <http://

www.epinions.com/cmhd-MiscPeripherals-All-Reality_Fusion_GameCam/display_~reviews>, 2000.

Freeman, W. T. et al, "Computer Vision for Interactive Computer Graphics," *IEEE Computer Graphics and Applications*, Vol. 18, No. 3. May-June 1998

Freeman, W. T. et al, "Computer Vision for Computer Games," *2nd International Conference on Automatic Face and Gesture Recognition*. Killington, VT, 1996.

Herz, J.C. *Joystick Nation*. Boston, Mass: Little, Brown and Company, 1997.

Hunter, William. "From Pong to Pac-Man." <http://www.designboom.com/eng/education/pong.html>, 2000.

Holmquist, Lars Erik. "The Right Kind of Challenge: Game Design and Human-Computer Interaction." *In Proceedings of IRIS 20*. <http://iris.informatik.gu.se/conference/iris20/16.htm>, 1997.

Jenkins, Henry. "Games, the New Lively Art," *Handbook of Computer Game Studies*. Cambridge, Mass: The MIT Press, 2003 (forthcoming).

Krueger, Myron W. *Artificial Reality II*. Reading, Mass: Addison-Wesley, 1991.

Myron Krueger. “Environmental technology: Making the real world virtual.” *Communications of the ACM*, 36(7):36 37, July 1993.

Laurel, Brenda. *Computers As Theatre*. Reading, Mass: Addison-Wesley, 1991.

Myler, Harley R. and Weeks, Arthur R. *The Pocket Handbook of Image Processing Algorithms in C*. Upper Saddle River, NJ: Prentice Hall, 1993.

Newman, James. “The Myth of the Ergodic Videogame,” Game Studies. *The International Journal of Computer Game Research*. <http://www.gamestudies.org//0102/newman/>, 2001.

Salen, Katie and Zimmerman, Eric. *Rules of Play: Fundamentals of Game Design*. Cambridge, Mass: The MIT Press, 2003 (forthcoming).

Sengupta, Kuntal; Bing, Wong Hon and Kumar, Pankaj. “Computer Vision Games Using A Cheap (< 100 $) Webcam.” <www.ece.nus.edu.sg/stfpage/eleks/ICARCV’2000.pdf>.

Reality Fusion. *GameCam*. <http://www.realityfusion.com/corp/products/gamecam>, 1999.

Vajpeyi, Praveen. *Myron Krueger.* <http://bubblegum.parsons.edu/~praveen/thesis/html/wk05_1.html>, 2002.

The Vivid Group. *The Mandala Gesture Xtreme System*. <http://www.vividgroup.com/products_main.html>, 1996.

ZMedia. “Reality Fusion.” <http://www.digitalduo.com/221_dig.html>, 2000.