NARAYANAN KUTTY, CHAYA. Toward Video Games on Video. (Under the direction of
Dr. Robert St. Amant).

The focus of this thesis lies on developing a novel interactive application we call
video game on video (VGV). This new class of video games integrates the logic and interac-
tion mechanisms of a video game with a video of the real environment, to create a playable
game. The research is aimed at creating VGVs using publicly available image processing li-
braries and off-the-shelf implementations of image processing algorithms to obtain a system
which fuses the virtual entities with a video of the real-world in a consistent manner.

As a part of this research, we have developed two games as a proof-of-concept, one
AugFrog and the other Human Pinball. AugFrog is the game of Frogger integrated with a
video stream of traffic on a freeway. Human Pinball is a game of Pinball whose field of play
is a corridor of a shopping mall with people walking by.

An additional novel aspect of this thesis is a modeling approach to representing
games such as Frogger and Pinball, which interprets interaction between virtual and real
objects in a game in terms of simple physics. We believe that this approach has the potential
for adapting other similar games into video game on video.

One of the main results of our research is a negative finding: although there
are numerous options available for open source image processing libraries and there is an
enormous variety of user-generated video online, it has been difficult to find a mapping
between videos and image processing algorithms, such that the integration might plausibly
be automated. There are several possible reasons for this, but we believe that current
libraries of image processing algorithms are simply not yet mature enough to support end
user applications without knowledge of the domain of the video and specialized expertise
in image processing.
Toward Video Games on Video

by

Chaya Narayanan Kutty

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2008

APPROVED BY:

__________________________  ________________________
Dr. Steffen Heber             Dr. R. Michael Young

__________________________
Dr. Robert St. Amant
Chair of Advisory Committee
DEDICATION

To
Mom, Dad and Maya,
For being my strength and driving force.
BIOGRAPHY

Chaya Narayanan Kutty was born in Bangalore, India. She received her undergraduate degree in Computer Science and Engineering from Visveswariah Technological University, Belgaum, India. She is currently pursuing her Masters at the department of Computer Science, North Carolina State University. During the course of her graduate work, she has interned at Cisco Systems Inc. at San Jose. She is also an active member of WiCS (Women in Computer Science) and served as a member of the Computer Science Graduate Student Association in North Carolina State University.

After graduation, she will be joining Cisco System Inc., San Jose, CA.
ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Dr. Robert St. Amant. Without your guidance, positive outlook and encouragement, this thesis work would never have been possible. I would also like to thank Dr. Michael Young and Dr. Steffen Heber for their valuable insights and suggestions.

I would like to thank my Dad, for teaching me to be passionate about my work; Mom, for teaching me to be a better person and my sister, Maya, for always believing in me. I could have never come this far without your advice and moral support. I would also like to thank my friends for all their encouragement and support.

I would also like to thank all my friends in the Knowledge Discovery Lab who have helped me throughout the duration of my thesis work. Special thanks are in order to those who participated in the user evaluation of the VGVs implemented as proof-of-concept for this research work. I would also like to thank Thomas Horton for proofreading my thesis in great detail.
# TABLE OF CONTENTS

## LIST OF TABLES

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>vii</td>
<td>LIST OF TABLES</td>
<td></td>
</tr>
</tbody>
</table>

## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>viii</td>
<td>LIST OF FIGURES</td>
<td></td>
</tr>
</tbody>
</table>

## 1 Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Video Games on Video (VGV)</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Novelty / Contribution</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Thesis Organization</td>
<td>6</td>
</tr>
</tbody>
</table>

## 2 Background

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Video Game Theory</td>
<td>7</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Basic elements of a video game</td>
<td>7</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Taxonomy of video games</td>
<td>8</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Abstraction in video games</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Merging virtual and real worlds</td>
<td>15</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Advantage of using a real-world environment in a game</td>
<td>16</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Challenges</td>
<td>16</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Is VGV an Augmented Reality system</td>
<td>17</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Related Work</td>
<td></td>
</tr>
</tbody>
</table>

## 3 Modeling approach based on QPT

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Modeling of Frogger</td>
<td>22</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Representation of Frogger based on QPT</td>
<td>23</td>
</tr>
<tr>
<td>3.2</td>
<td>Modeling of Pinball</td>
<td>38</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Representation of Pinball based on QPT</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>Advantages of the modeling approach</td>
<td>47</td>
</tr>
<tr>
<td>3.4</td>
<td>Limitation of the modeling approach</td>
<td>47</td>
</tr>
</tbody>
</table>

## 4 System Design

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Design</td>
<td>49</td>
</tr>
<tr>
<td>4.2</td>
<td>Implementation</td>
<td>50</td>
</tr>
<tr>
<td>4.2.1</td>
<td>AugFrog</td>
<td>51</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Human Pinball</td>
<td>56</td>
</tr>
</tbody>
</table>

## 5 Analysis

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Technical Aspects</td>
<td>60</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Algorithms used for scene analysis in AugFrog</td>
<td>60</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Algorithms used in Object Management in AugFrog</td>
<td>62</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Algorithm used for performing scene analysis in Human Pinball</td>
<td>63</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Algorithms used for Object Management in Human Pinball</td>
<td>65</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 3.1 Mapping between interactions and processes that are based on QPT in Frogger 34
Table 3.2 Mapping between interactions and processes that are based on QPT in Pinball 44
Table 5.1 Summary of algorithms used in the implementation of AugFrog. ............. 63
Table 5.2 Summary of algorithms used in the implementation of Human Pinball. .... 69
Table 6.1 Users’ feedback on the interesting aspects in AugFrog ......................... 71
Table 6.2 Users’ feedback on the interesting aspects in Human Pinball ............... 71
Table 6.3 Users’ feedback on the difficulties while playing AugFrog ................... 71
Table 6.4 Users’ feedback on the difficulties while playing Human Pinball ......... 71
Table 6.5 Users’ suggestions for the improvement of AugFrog ......................... 72
Table 6.6 Users’ suggestions for the improvement of Human Pinball ............... 72
LIST OF FIGURES

Figure 1.1 A system that can automatically generate VGVs .......................... 2
Figure 1.2 AugFrog during game-play ............................................... 3
Figure 1.3 Human Pinball during game-play ................................. 4

Figure 2.1 Milgram’s Virtuality Continuum ................................. 17
Figure 2.2 A camera maps objects in the world coordinate system onto the display
coordinate system by perspective mapping [1] ............................... 18

Figure 3.1 Prediction of Player-lose behavior based on frog-y-position .......... 35
Figure 3.2 Prediction of Player-win behavior based on frog-y-position ........... 36
Figure 3.3 Histories and evolution of ball-y-position .............................. 46

Figure 4.1 System Design .......................................................... 49
Figure 4.2 Design of AugFrog ...................................................... 52
Figure 4.3 Design of Human Pinball ............................................. 57

Figure 5.1 Blob Tracker Auto ...................................................... 66
Figure 5.2 Collision Response ...................................................... 68
Chapter 1

Introduction

1.1 Video Games on Video (VGV)

At some time, almost everyone imagines what it would be like to have ordinary objects and environments that were augmented in an interesting or unusual way - as in a video game. While driving in heavy traffic, I might imagine ray guns in the headlights of my car that can zap away other cars; walking through a crowd I might wish for the ability to freeze everyone in place while I pass through them. We see such capabilities in movies and on television, and they are part of the repertoire of some virtual and augmented reality systems. Our focus, however, is more specialized: We are interested in the feasibility of integrating the logic and interactions of video games with video of real environments. We call the resulting game, a video game on video(VGV).

The long-term goal of this project is to create a system that generates VGVs automatically given a game model and a video game of the end user’s choice. Such a system must have the capability to define mapping between the elements of a video game and the video, to create a playable game. This is achieved using a toolkit of image processing algorithms and libraries for graphics. This system also consists of a database of models of video games in abstract representation and a set of videos. A high level overview of such a system is as shown in the figure 1.1.

The current work is only a first step towards this long term goal. Here, we are attempting to create a playable VGV, given a game model and a pre-recorded video. This research presents two implementations as proof-of-concept of VGVs and an outline of modeling and software design considerations for a familiar set of video games. We begin with
two motivating examples. The first VGV we developed, AugFrog, is based on the classic video game Frogger, developed by Konami in 1981. In Frogger, a user must move a virtual character, represented as a frog, across several lanes of traffic. The frog moves from the bottom of the screen to the top, avoiding collisions, each move being a jump in one of the four cardinal directions. Once the traffic has been passed, the frog must cross a river by jumping onto logs and turtles. This description is incomplete, but it is sufficient to make the game identifiable to most who have played it.

AugFrog, the VGV, follows analogous rules by design. We recorded a video of a nearby highway for the game environment. A virtual frog moves through the traffic, as shown in figure 1.2. Lacking an appropriate video of a river, we dispensed with that component of the game, but we arranged for the frog to be able to jump onto nearby passing vehicles.

The second proof-of-concept implementation is Human Pinball, which is based on the arcade game of Pinball. The rules of the Pinball game are widely known: The game takes place in a playing field that is usually the area of the table. A ball is launched via a plunger and this ball collides with various objects on the table. The player’s aim is to keep the ball in the playing field as long as possible, using a pair of flippers to deflect the ball from the bottom of the field, while scoring as many points as possible in the process.

Human Pinball adopts the same basic concept as the Pinball game, but with the difference that it involves the ball bouncing off people walking in a corridor of a shopping mall instead of being deflected by objects on a table. The video data has been taken from the CAVIAR project (IST 2001 37540 See [2]). In Human Pinball, the virtual entities i.e. flippers, ramps and the ball, augment the video frames. A screenshot of Human Pinball,
Figure 1.2: AugFrog during game-play
Figure 1.3: Human Pinball during game-play

which depicts a virtual ball bouncing off people, is as shown in the figure 1.3

This may seem a simple exercise; we are taking video games inspired by physical environments and mapping the logic back to the same kinds of environments. Nevertheless the task poses some interesting questions for modeling and software design, as discussed in the chapters to follow.

1.2 Novelty / Contribution

Arcade simulation games are only able to imitate portions of real world experience [3]. Immersive virtual environments such as Krueger’s VIDEOPLACE [4] are able to provide a simulated experience with limited interaction. The idea of merging virtual entities and real objects is seen in work as early as the 1980s in Lippman’s Movie Maps [5] and introducing
interaction between the real and virtual entities is seen in VIDEOPLACE [4], although limited. Extracting 2D models of objects from the video of a real scene as achieved in VGV is explained in detail in Tani et.al’s Object Oriented Video [1]. An augmented reality system is one where virtual entities are superimposed onto a scene from the real world. The idea of extending existing arcade or desktop games into an environment where virtual entities are superimposed into a real scene is derived from several augmented reality games such as Human Pacman [6], ARQuake [7] and PingPongPlus [3].

Augmented Reality Games face several challenges as described in Human Pacman [6] such as imposing a limit on the duration of game play due to problems in powering the mobile computing device and usage of Head Mounted Displays (HMDs) brings in problems such as hassles in using head gear, low resolution, eye fatigue and requirement of dim lighting conditions [6].

2D interaction techniques are attractive because of their proven acceptance and widespread use on the desktop [8]. Desktop systems typically use a combination of a keyboard and mouse to allow the user to interact with some kind of Window, Icon, Menu, Pointer (WIMP) interface. It takes any user a short learning period to become extremely proficient in these interaction techniques.

The question we are trying to address is whether it is possible to introduce an Augmented Reality-like realism into video games using readily available software and PC hardware?

The novelty of VGVs lies in the following:

• VGVs provide a mapping of the model of arcade games onto real 2D environments, thus giving a degree of realism without significant design and implementation effort in this area.

• Interaction techniques from the original 2D arcade games are retained, making it easier for players to transfer knowledge.

• VGVs offer a cost effective means of providing Augmented Reality-like effects, with respect to mixing real and virtual entities, employing simpler image processing techniques than are generally required in other approaches.

• VGVs avoid all the problems of using HMDs and constraints due to the usage of such similar equipments that are required to play AR games.
The contribution of this thesis is in two parts:

- We have reproduced video games in a novel medium. VGVs are similar to Augmented Reality games but are far less computationally demanding. Even though the development of VGVs has proved far more difficult than expected, we believe that the general idea is promising and worth further exploration.

- We have developed a novel modeling approach for a class of video games, one that is more interaction-centric than existing modeling approaches.

The important aspect to note here is that VGVs are not Augmented Reality games. VGVs differ from Augmented Reality games mainly due to the fact that it does not attempt to immerse its users into the real world. VGVs are just an attempt to adapt the mixed reality environment into video games that can be played on desktops.

One of the main results of our research is that it has been difficult to find a mapping between videos and image processing algorithms despite the availability of several open source image processing algorithms and videos.

1.3 Thesis Organization

In Chapter 2, we delve into the details of the work in related areas that have influenced VGVs. The novel modeling approach is presented in detail in Chapter 3. In Chapter 4, we discuss the design and implementation of the two VGVs we developed as proof-of-concept. In Chapter 5, we present a detailed description of the various algorithms and attempts, some successful and others failures, that were adopted during the implementation of the two VGVs - AugFrog and Human Pinball. Chapter 6 presents the results obtained from the user evaluation of AugFrog and Human Pinball. We end this thesis with a discussion of the limitations of our current work, insights we have gained and the conclusion of this thesis in Chapter 7.
Chapter 2

Background

Work on VGVs has been greatly influenced by prior work in video game abstraction as well as work in augmenting real and virtual worlds. In this chapter, we first delve into the basic elements of a video game, the taxonomy for video games and the various approaches in modeling a video game, which in turn influenced the development of a modeling approach for VGVs. We then move on to considering several aspects of merging reality and virtuality - the advantage of building worlds using real environments, the various challenges, a discussion on whether a VGV is an Augmented Reality system, and related work in the area.

2.1 Video Game Theory

In this section, we look into various aspects of the basics of video games and the various approached of abstraction with respect to video games. Throughout this section, we use an example of Frogger, which has been introduced in section 1.1, to make it easier for the reader to understand the concepts.

2.1.1 Basic elements of a video game

Wolf et.al [9], describe four fundamental elements which are the core of what makes a video game unique. Every video game comprises graphics, player activity, interface and an algorithm.

- Graphics involves pixel-based imaging which creates the various visual effects in any video game
• The interface occurs at the boundary between the player and the video game itself and is a portal through which player activity occurs.

• Player activity deals with what the player is physically doing during game-play to achieve a certain result (Extradiegetic) activity and what a player’s avatar does as a result of player activity (Diegetic) activity.

• The Algorithm ties all the elements of a video game together. The tasks of an algorithm can be divided into representation, rules, randomness and responses. Representation mostly deals with the graphics rendering in a typical video game. The game rules impose limitations on the gaming activity. Randomness introduces a factor of unpredictability into a video game. Responses deal with the reaction to changing situations in a video game.

    In the game of Frogger, Graphics comprises entities such as the frogs, motor vehicles, the median, logs, turtles, and frog-homes. The GUI of Frogger acts as the interface between the player and the video game. Player activity occurs whenever the player uses one of the arrow keys, which initiates the movement of the frog in the game. Some of the game rules of Frogger consists of the occurrence of the death of the frog when it is hit by a vehicle, ends up in water, frog is run off the screen and so on. The varying distances that exists between vehicles, varying speeds of the logs, and disappearing turtles bring in the unpredictability factor into the game of Frogger. The player loses a turn as a response to the death of the frog or when the player runs out of time.

2.1.2 Taxonomy of video games

    In The Art of Computer Game Design, Crawford gives a taxonomy for computer games whose basis of division is chronological order [10]. He divides computer games into two categories

• Skill and Action Games: This class of games is characterized by real-time play, heavy emphasis on graphics and sound, and use of joysticks or paddles. The primary skills demanded of the player are hand-eye coordination and fast reaction time. Skill-and-action games comprises six categories: combat games, maze games, sports games, paddle games, race games, and miscellaneous games.
• **Strategy Games**: These games emphasize cogitation rather than manipulation. The major distinguishing factor between strategy games and Skill and action games is the emphasis on motor skills. All skill-and-action games require some motor skills; strategy games do not. These games can be divided into Adventures, D&D games, war games, games of chance, educational games, and interpersonal games.

Based on this taxonomy, the games of Frogger and Pinball fall into the category of Skill and Action games.

### 2.1.3 Abstraction in video games

In this section, we look into different approaches to abstraction in video games that have been previously proposed.

**Abstraction based on entities identified on the gaming-screen**

One of the first few things a player learns while playing a new video game is to identify and understand the functionality of various entities seen on the gaming-screen. Wolf et al. [9] divide these elements into four categories:

- **Player-Character** indicates the presence of the player in the game.
- **Computer-controlled characters** are entities indicating the presence of the computer.
- **Objects** are entities that can be manipulated by game characters.
- **Background environment** refers to the setting of the game which is not manipulated by any of the entities in the game.

On applying this abstraction to the game of Frogger, we see that the **Player-Character** is the frog. The vehicles, logs, and turtles are the **computer-controlled characters**. In Frogger, the frog can catch a bug and hence this falls into the category of **Objects**. The median and the frog’s homes comprises the **Background environment**.

**Abstraction based on Game Mechanics and Game Patterns**

A game mechanic is defined as any part of the rule system of a game that covers one and only one possible kind of interaction that takes place during the game, be it
general or specific [11]. A game may comprise several mechanics or a particular mechanic may be part of many games. Lundgren et al. [11], employing game mechanics as a means of describing interaction for computer-augmented games, identify several game mechanics relevant to computer augmented games, such as superimposed game world, pervasive gaming, espionage, body mapped avatar and so on. Although mechanics can be used as a way to summarize game rules and to categorize games, they do not explicitly describe how they affect the experience of the game and interaction between players, thus making it difficult to use it as a designing tool for games.

Some of the game mechanics seen in Frogger are victory condition mechanics, movement mechanics and turns. Frogger is a turn based games. The player is given three turns within which she has to guide the three frogs safely to their homes. Victory condition mechanics specify how a player wins the game. A victory condition is defined in terms of goals. For example, in the game of Frogger, the victory condition is to move the frogs such that they reach their homes safely. Movement mechanics defines when and how elements in the game can be moved by the player. In the game of Frogger, the frog can be moved in the four cardinal directions. The player can initiate movement by using the arrow keys.

Bjork et al. suggest an interaction-centric model for games which consists of a structural framework and game design patterns. The structural framework describes the components of the game and game design patterns are an extension of game mechanics, that describe the player interaction while playing [12]. The structural framework divides a game into three categories as follows

- **Bounding**: Specifies activities that are allowed to occur and those which are not allowed to happen, in terms of goals, rules and game modes.
- **Temporal**: Refers to the temporal execution during game play such as actions, events, end conditions, evaluation functions and closures.
- **Objective**: Denotes the physical/virtual components such as players, interfaces and game elements (which describe how players can affect the game and describe the knowledge of the game state)

A game design pattern consists of a name, description, consequences, usage and relation to other patterns. Using this model involves identifying and generating a collection of suitable game design patterns. However, the process of making a pattern collection is difficult
and time consuming. Making one large collection containing all identified patterns in an 
encyclopedic endeavor may solve this problem by containing all possible sets of required 
patterns, but finding the specific patterns in the day to day design work may be too time 
consuming especially as identified patterns may be linked to many patterns that are not 
relevant to a particular case [12].

The game of Frogger can be divided on the basis of the structural framework 
described by Bjork et al. as illustrated below.

**Bounding Category**

Goal: The frogs reach their homes safely  
Rules: The frogs are allowed to leap onto logs and turtles.  
    The player is responsible for frog movement.  
    The frog dies if it is hit by a vehicle or if it ends up in water.  
    The player loses a turn if she is not able to achieve the goal within the 
    duration allotted for every turn.  

Game modes: Single player mode.

**Temporal Category**

Actions: Player enables frog movement (Left, Right, Up and Down).  
Events: Collision (Vehicle hitting a frog)  
    Drown (Frog in contact with water)  
    Leap Onto Log (Frog jumping onto a log)  
    Leap Onto Turtle (Frog jumping onto a turtle)

**Objective Category**

Player: Player controls the frog.  
Interfaces: GUI of the game.  
Game elements: The player possess the ability to move the frog using the arrow keys.

Some of the game design patterns described by Bjork et al. [13], that are found in 
the game of Frogger are Obstacles (refers to game elements that hinder player’s movement), 
Avatars ( a game element through which the player can affect the game world), Movement 
deals with the movement of game elements in the game world), Predefined Goals (goals 
that are preset by the game designer), Single-Player game, Turn-Based games and Player
Elimination (involves bringing the player session to an end due to the player being unable to achieve a goal).

**Abstraction of games in terms of problem spaces**

Newell [14] introduced the problem space principle as the rational activity in which people engage to solve a problem. It can be described in terms of

1. a set of **states** of knowledge
2. **operators** for changing one state into another
3. **constraints** on applying operators
4. control **knowledge** for deciding which operator to apply next [14].

Thus, the problem space is represented in terms of problem states and problem solving involves generating these states by applying legal transitional operators. The problem consists of an initial state(s) and a goal state(s). The initial state is generated based on the observations of the external environment. The task of problem solving comprises finding the sequence of operations that form a path within the state space, from the initial state to the goal state via one of the intermediate states. Since problem spaces can be huge, heuristics can be used to select appropriate operators.

Representing the game of Frogger in terms of problem spaces is as follows:

**States** - These represent the positions of the various entities identified in the game such as the frog, vehicles, logs and turtles as shown below, at a given point of time.

Frog:
- X: integer
- Y: integer
- Mode: Road / Median / log / turtle
- Alive / Dead: boolean

Each vehicle present in the play-field at a given point of time can be represented as

Vehicle1:
- X: integer
- Y: integer
Direction: East or West

Vehicle2:
  X: integer
  Y: integer
  Direction: East or West

and so on.
Similarly,
Log1:
  X: integer
  Y: integer
  Direction: East or West

and so on.
In the same manner, turtles can be modeled as,
Turtle1:
  X: integer
  Y: integer
  Direction: East or West

Thus the initial state describes the frog at its initial position i.e. its Y-Position is at the bottom of the screen. The goal state is given by the frog reaching its home i.e. its Y-Position is at the top of the screen. The intermediate states describe the changing location of the frog and also the various entities present in the screen at that point of time. The transition from the initial state to the goal state can be achieved by applying operators as shown below.

Operators: Left (L), Right(R), Up(U), Down (D), Stay (S)

The various constraints that define legal states are as listed below. The player loses a turn when

- The frog is hit by a vehicle.
- The frog falls into water.
- The frog runs off the screen while on a turtle or a log.
- The player runs out of time. Also, there are restrictions on the movement of the frog such as, it is allowed to leap onto a log or a turtle and not onto a vehicle.

The knowledge gained by the user during game-play includes:
- The vehicles do not change lanes. This applies to logs and turtles too.
- The vehicles travel at a constant speed i.e. there is a fixed distance maintained in-between vehicles. This is the same in case of logs and turtles.
- When a frog leaps onto a log or a turtle, the frog’s speed and direction is changed to that of the log or turtle.

Our attempt at modeling Frogger in terms of problem space led us to observe a lack of support for the representation of the various interactions that occur during game-play in this mode of representation.

**Abstraction of games based on qualitative physics**

Qualitative physics is concerned with representing and reasoning about the physical world [15]. Qualitative physics basically deals with identifying the relevant objects, variables and parameters in the domain, identifying the relations among them and describing the behavior of the system in terms of qualitative characteristics (such as increasing, decreasing, oscillating, etc.) of the changes in the variable values over time [16]. Forbus’s program, Qualitative Process Engine (QPE) is a high performance implementation of the Qualitative Process Theory.

Ken Forbus developed the Qualitative Process Theory (QPT) as described in [17]. Qualitative Process Theory aids in understanding commonsense reasoning about physical processes [16] and provides reasoning about complex physical systems [17]. In QPT, a physical situation is described in terms of sets of objects and their relations. The representation of the attributes of an object is known as quantity. Objects in the physical world exhibit dramatically different characteristics depending on conditions applied on them [16]. QPT uses *Individual views* to model these changes. An individual view consists of individuals
(objects involved), quantity conditions (statements of inequalities between quantities of the
individuals), preconditions and relations (statements that are true when the view is valid).
The effect of processes on objects can be expressed in terms of changes in the parameters
of objects.

A physical process is associated with change in the value of parameters of objects
over a period of time. In QPT, a process is represented in terms of individuals, quantity
conditions, preconditions, relations and influences that the process has on the variables of
the objects.

Individual views and processes impose qualitative functional relations on param-
eters of objects, and active processes describe the changes variables undergo [16]. Using such
information, Qualitative Process Engine (QPE) predicts the future course of events, using
the notion of history which is in turn described through episodes (non-instantaneous) and
events (instantaneous) of individual views and processes. In addition, the set of variable
values is described using a finite number of distinguished values and the ordinal relations
between them in a quantity space. The process of identifying distinguished values toward
which variables evolve is called limit analysis.

Qualitative Process Engine (QPE) uses general knowledge about physical processes
and objects to draw predictions about the behavior and evolution of dynamic systems. The
QPE functions in four phases:

1. Identify the objects that exist in a given situation
2. Identify the process instances that are active by examining conditions that are satisfied
3. Determine the changes that will be caused by active processes
4. Predict behavior over time by producing an account of the activity of processes over
time [16]

The modeling of Frogger and Pinball in terms of Qualitative Process Theory has
been described in great detail in Chapter 3.

2.2 Merging virtual and real worlds

We now move on to looking at VGVs from the perspective of fusing virtual and
real entities.
2.2.1 Advantage of using a real-world environment in a game

Starner et al. [18] observe in that one of the major costs of creating current generation computer games is the development of content. Developing an entire 3D world for the player to explore is time consuming and labor intensive. The complexity of world-building to create a compelling experience is directly dependent on the dimensions of movements allowed to a player’s character [18]. By using a real world environment, the game designer already has a pre-made environment for his game. Thus, the designer can concentrate on creating just the virtual artifacts that deviate from the physical world in the game and on implementing the interactions between the virtual entities and the real environment.

2.2.2 Challenges

Ohta points out three major aspects which have to be taken into account while superimposing virtual objects onto a real scene appropriately: geometry, illumination and time [19]. Consistency of geometry and time are relevant to VGVs. Consistency of geometry deals with the requirement that the virtual object must possess a position within the real scene and has to appear at that correct location in the real scene. Error in this prevents the user from seeing real and virtual entities as fused [20]. This involves mapping both entities into common space. Consistency of time refers to the requirement that the coordinated motion of virtual and real objects is vital to maintain realism. Bimber and Raskar, in Spatial Augmented Reality [21] observe that while superimposing the real environment with graphical elements, fast and realistic rendering methods play an important role. The ultimate goal is to integrate graphical objects into the real environment in such a manner that the observer is unable to distinguish the virtual and real objects.

2.2.3 Is VGV an Augmented Reality system?

A Mixed Reality environment refers to any visual display where a completely real environment and a completely virtual one are presented i.e. anywhere between the extrema of Milgram’s Virtuality continuum [22]. It has been quite unclear where VGV falls in this continuum. We look at a couple of the widely accepted approaches which are used to define Mixed Reality systems or environments.

According to Milgram’s classification of Mixed Reality environments based on their visual displays, VGV’s fall into the category of Class 1 - Mixed Reality using Monitor
based video displays. These are Window on World displays upon which computer generated images are electronically or digitally overlaid. Looking at the continuum (refer fig.2.1), we observe that VGV falls somewhere beyond the real environment and towards Augmented Reality. The question is whether it falls into the Augmented Reality systems.

According to Milgram [22], the term Augmented Reality is appropriate for describing the essence of computer graphic enhancement of video images if real scenes. Thus by this definition, a VGV falls into an AR system.

Azuma in his paper *A survey of Augmented Reality*[23], defines AR as systems that have the following three characteristics:

1. Combines real and virtual
2. Interactive in real time
3. Registered in 3-D [23]

This definition ignores any system which contains 2D overlays on a video of real world scenes, including the VGV, from being an AR system.

2.2.4 Related Work

Early systems like Movie Maps [5] employed graphical objects as reference cues in a video. Thus the interaction between the graphical entity and objects in the video or real-world is minimal. *VIDEOPLACE* [4] is one among those early systems where interaction between real and virtual entities was a central part of the work, however interaction was limited to simple gesturing. *VIDEOPLACE* is a computer graphics environment where the user’s physical participation is coordinated with that of graphical objects, such that they

![Virtuality Continuum](image-url)
appear to react to the user’s movements in real time. For instance, when the user moves his fingers apart, one of the virtual elements expands in size as a result.

*Object Oriented Video* [1] introduced the concept of using various direct manipulation interactive techniques on a live video. This was achieved by generating models of the objects in a live video thus facilitating interaction between the objects and the user via graphic overlays. *Object Oriented Video* describes the mapping of a 3D video scene onto a two dimensional plane on the screen. This mapping is modeled as a perspective transformation from the world coordinate system to the display coordinate system (refer to fig.2.2) [1].

![Diagram of world coordinate system and display coordinate system](image)

**Figure 2.2:** A camera maps objects in the world coordinate system onto the display coordinate system by perspective mapping [1]

*Object Oriented Video* uses 2D modeling of the objects identified in the video. In 2D modeling, the camera maps 2D shapes of objects onto the screen. The 2D shape of an object is described as a set of 2D graphic primitives, such as rectangles, polygons or circles. Once the 2D model is defined, identifying the object at a specific point in a video image is transformed into that of identifying the graphic primitive at that point, which is supported by most graphic libraries. A similar procedure is employed in VGVs.

Gradually, there came applications which emphasized human - real world interaction. This was introduced by Rekimoto et al. [24] and they termed human - real world interactions as Augmented Interaction. They developed a prototype named *NaviCam*, which is a portable device that enhances the user’s ability to interact with the real world by recognizing the user’s situation, thus allowing the user to see the world with superimposed
graphic entities, which contain information about the situation, on a real video.

Further, Augmented Reality games such as *PingPongPlus* [3], suggested new directions to integrating athletic recreation and social interaction with engaging digital enhancements [3]. *PingPongPlus* transformed a game of ping-pong by augmenting it with dynamic graphics such as creating a water ripple effect whenever the ball hits the ping pong table and sound effects. *PingPongPlus* also attempted to modify the nature of playing ping-pong from a competitive one to one which encourages coordination, such as, in the Painting mode, where the users play to create an image while playing ping-pong.

*ARQuake* [7] presented a new entertainment application that is an AR extension of the desktop game, Quake. *ARQuake* allows users to play the game of Quake outdoors i.e. in the real world, against virtual monsters. In order to adapt the game of Quake to create an AR version, the authors had to make some modifications such as avoiding the usage of Quake worlds where the user was required to swim or fly, reduce some of the powers of the user during game-play and exclude certain kinds of monsters. Preprocessing in *ARQuake* involved generating a complete model of the environment where the game is to be played at and is imported as a Quake Map. Users of *ARQuake* found playing the game more natural mainly due to the free movement and haptic feedback during the usage of virtual weapons. One of the major problems with *ARQuake* is that of position inaccuracies; The alignment between the monsters and the buildings is not perfect and as a result monsters appear to walk through walls or pop out of thin air.

Similarly, Human Pacman [6] is another interactive, collaborative and role playing game, which extends the arcade game of Pacman and uses the real world as the playground. *Human Pacman* allows immersive role playing of characters such as Ghosts and Pacmen, unrestricted outdoor and indoor movement, tangible aspects of movement, perception, interaction and allows seamless transition between AR and virtual view of the Pac-world. *Human Pacman* deviates from Pacman, in that, it introduces a new character - Helper (who is always seeing the Pac-world in the Virtual Reality mode), who advises her partner i.e. Ghost or Pacman, about the position of the enemy and other attributes which are not visible in the AR mode. Players of *Human Pacman* had to deal with several problems due to the usage of HMDs and limitations in powering of the mobile computing device sets the duration of game-play to ten minutes.

Most of the Augmented Reality Games require the usage of 3D display devices such as Head Mounted Displays(HMDs) or Head Worn Displays(HWDs). These devices
suffer from several drawbacks including high cost. The helmets or goggles worn in MR and VR systems allow 3D display of the world to each eye and track the user’s head position. In order to create real life illusion of changing scenes as we move our head, VR/MR systems use sensors situated in goggles or helmets. But a system which is slow in producing changing images leads to a lag between the user moving his head and scene changing. If this delay is more than a hundred milliseconds or so, the user feels motion sickness. In addition, users of HMDs also deal with drawbacks such as high cost, insufficient resolution of display, difficulty in focusing on virtual objects and fatigue when used for long durations [25].

To overcome the disadvantages of using HMDs and HWDs, Wagner et al. [26] developed Invisible Train, which is an indoor, multi-player game, where players steer virtual trains over a real miniature train track. These virtual trains are visible to the players via their PDA’s video see through display. The game of Invisible Train suffered from problems of camera blur which led to the loss of registration during rapid movements and the devices’ short battery life of two hours.

An early prototype of AugFrog was implemented in our lab by Anna Ferencova in JAVA with the help of Thomas Horton and Lloyd Williams. The system had to be discarded, due to the computational complexity of the image processing algorithms that were used to implement the game, which had an adverse effect on the speed of the game.
Chapter 3

Modeling approach based on QPT

Modeling is essential in the work on VGVs to identify specific properties in a video game. These properties can then be used to determine the compatibility of a video with the video game, to create a VGV. Thus, an effective modeling approach must be able to obtain an accurate mapping between a video game and a video, identify the various interactions involved in the video game and aid in automatic generation of code for VGVs.

In order to study the dynamic behavior of video games such as Frogger and Pinball, we have modeled these two games based on Ken Forbus’s Qualitative Process Theory (QPT) [17]. The concepts associated with this theory are dealt with in detail in section 2.1.3. Any physical system can be described in terms of objects and the processes acting on them. Thus, a game can be represented via the various objects involved in; their changing behavior due to conditions (individual views), variables which represent the attributes associated with these objects in terms of values, and the relations between the various objects identified in the game. Using the concepts of Qualitative Process Theory, it has been possible to predict the behavior of the parameters in Frogger and Pinball via limit analysis and visualize their evolution via the changing values of variables, processes being active or terminated and individual views of objects becoming valid or invalid over time. The complete representation of Frogger and Pinball based on QPT and prediction of behavior of these games is presented later in this section. It must be noted that the purpose of the representation we have presented is only as a means of description of a game and not for usage by a Qualitative Process Engine.

Using the representation as described for physical systems in the paper [17], we have been able to model the physics of 2D trajectory-based video games, which include
Frogger and Pinball. It must be noted that the representation has been simplified for easier understanding and hence there are several cases where certain conditions are explained in a sentence format rather than using predicates or functions as in Forbus’s representation in his paper [17]. Also, in order to easily visualize the behavior of the games, the prediction and histories of the games are presented in a format that is described by Barr et al. [16].

Informally, an environment defines a two-dimensional space that contains regions and entities. Regions have spatial extent (i.e., bounds) and game-specific properties. Regions contain entities, which correspond to physical bodies. These entities can enter or leave regions and have the general properties of location, shape, orientation, and size. Moving entities also have a trajectory, taken from a small set of pre-defined functions: linear, parabolic, and hyperbolic, with parameters determined by the environment. Agents are entities under the control of the user; in addition, agents have a set of operators that can be executed by the user [14]. Critical to a model are object/object and object/region interactions: rules that govern how the properties of objects change based on their interaction. Here, we limit discussion to interaction via contact or overlap. The effects of such interactions largely determine the play of a given game. Using QPT, it is possible to represent interaction via the intersection of the histories i.e. events and episodes of the various objects present in the physical system. Thus, we are able to draw a mapping between the various interactions that occur during game-play and their corresponding processes as identified using QPT.

Here is a brief description to the notation used in the representation. The predicate \textit{Quantity-Type} is used to map a quantity to an object and the predicate \textit{Has-Quantity} is used to map a particular type of quantity to the object. While expressing the influence of any process on the rest of the system we use the notation I+(a, b). This indicates an increase in the value of ‘a’ due to the influence of ‘b’. Similarly, I-(a, b) denotes a decrease in the value of ‘a’ because of ‘b’.

\subsection{Modeling of Frogger}

In Frogger, the environment consists of a vertical stack of regions, each of which has a type: shoulder/median, road and river. The bottommost shoulder/median region is the starting point for the game, the uppermost the goal. The agent is the frog; objects include the vehicles on the road and the logs and turtles in the river (We neglect rewards associated
with their behaviors in this discussion). Vehicles, logs, and turtles have a constant linear trajectory, each contained within a river or road region; the creation and destruction of turtles in the environment is governed by a timing function. The frog can move in any of the cardinal directions; the size of a step in the vertical direction is determined by the width of the regions in the environment. The frog may also remain in its current location. The frog initially has a null trajectory; execution of operators results in instantaneous changes in its location.

Interactions are as follows. Contact between the frog and a vehicle results the destruction of the frog (i.e. an end of turn). If the frog enters a road region but does not contact a vehicle, there is no effect other than the frogs change of location (such interactions can be left implicit). Contact between the frog and a log or turtle in a river region results in the frog acquiring the trajectory of the object. If the frog enters a river region but does not contact a log or turtle, the frog is destroyed. If the frog contacts the horizontal bounds of the environment (i.e., the left and right edges of the screen) it is destroyed.

### 3.1.1 Representation of Frogger based on QPT

**Representation of the various objects identified in Frogger**

**frog**

- Quantity-Type(frog-x-position)
- Quantity-Type(frog-y-position)
- Quantity-Type(frog-state)
- Quantity-Type(frog-lane)
- Quantity-Type(frog-speed)
- Quantity-Type(frog-direction)
- Quantity-Type(frog-size)
- Quantity-Type(frog-shape)
- Quantity-Type(frog-on)

- Has-Quantity(L, frog-direction)
- Has-Quantity(R, frog-direction)
- Has-Quantity(U, frog-direction)
- Has-Quantity(D, frog-direction)
Has-Quantity(Alive, frog-direction)
Has-Quantity(Dead, frog-direction)
Has-Quantity(log, frog-on)
Has-Quantity(turtle, frog-on)
Has-Quantity(road, frog-on)
Has-Quantity(median, frog-on)
Has-Quantity(water, frog-on)

playing-field
Quantity-Type(playing-field-medium)
Quantity-Type(playing-field-bounding-area)
Quantity-Type(playing-field-region)

Has-Quantity(road, playing-field-medium)
Has-Quantity(water, playing-field-medium)
Has-Quantity(lanes, playing-field-region)
Has-Quantity(median, playing-field-region)
Has-Quantity(home, playing-field-region)

turtle
Quantity-Type(turtle-x-position)
Quantity-Type(turtle-y-position)
Quantity-Type(turtle-speed)
Quantity-Type(turtle-direction)
Quantity-Type(turtle-shape)
Quantity-Type(turtle-size)
Quantity-Type(water-level)

Has-Quantity(L, turtle-direction)
Has-Quantity(above, water-level)
Has-Quantity(below, water-level)
turtle-speed > ZERO

log
  Quantity-Type(log-x-position)
  Quantity-Type(log-y-position)
  Quantity-Type(log-speed)
  Quantity-Type(log-direction)
  Quantity-Type(log-shape)
  Quantity-Type(log-size)

Has-Quantity(R, log-direction)
log-speed > ZERO

vehicle
  Quantity-Type(vehicle-x-position)
  Quantity-Type(vehicle-y-position)
  Quantity-Type(vehicle-speed)
  Quantity-Type(vehicle-direction)
  Quantity-Type(vehicle-lane)
  Quantity-Type(vehicle-bounding-area)
  Quantity-Type(vehicle-shape)
  Quantity-Type(vehicle-size)

Has-Quantity(R, vehicle-direction)
Has-Quantity(L, vehicle-direction)
vehicle-lane > ZERO
vehicle-speed > ZERO
Representation of Various views of Frogger depicting different objects and their relations

Individual views of the frog object

**Individual view Frog-Alive**

Individuals:
- frog: agent controlled by the player

Preconditions:
- frog is within the bounding-area of the environment

Quantity Conditions:
- frog-state = Alive

Relations:
- The frog has the capability to move
- The game is still in progress

**Individual view Frog-Dead**

Individuals:
- frog: agent controlled by the player

Preconditions:
- frog-state = Alive

Quantity Conditions:
- frog-state = Dead

Relations:
- frog vanishes from the screen
- Player loses a turn
Individual views of the playing-field object

Individual view Start-Game

Individuals:

frog : agent controlled by the player
playing-field: the environment within which the game takes place

Preconditions:

The screen contains the frog

Quantity conditions:

The frog is at initial position

Relations:

The user has not moved the frog

Individual view Goal-Game

Individuals:

frog : agent controlled by the player
playing-field: the environment within which the game takes place

Preconditions:

frog-state = Alive

Quantity conditions:

playing-field-region = home

Relations:

Player wins the game

Influences:

frog-on = home

Individual view Game-In-Progress

Individuals:

frog: agent controlled by the user
playing-field: the environment within which the game takes place

Preconditions:
frog-state = Alive

Quantity Conditions:
frog-y-position is not equal to initial position

Also,

Let frog-y-position<sub>old</sub> be the Y Coordinate of the frog and frog-x-position<sub>old</sub> be the X Coordinate before movement,

let frog-y-position<sub>new</sub> be the Y Coordinate of the frog and frog-x-position<sub>new</sub> be the X Coordinate after movement,

then,

\[ |frog-y-position_{new} - frog-y-position_{old}| \geq ZERO \]

\[ |frog-x-position_{new} - frog-x-position_{old}| \geq ZERO \]

Relations:
frog is moving among the objects which are not controlled by the user or is on the median

Influences:
frog-on = road or frog-on = log or frog-on = turtle

**Process Representation in Frogger**

**process forward-movement**

Indivduals:
frog: agent under the control of the player

Preconditions:
frog-state = Alive
User presses 'Up' arrow key

Quantity conditions:

Let frog-y-position<sub>old</sub> be the Y Coordinate of the frog before movement,

let frog-y-position<sub>new</sub> be the Y Coordinate of the frog after movement,

\( (frog-y-position_{new} - frog-y-position_{old}) > ZERO \)

Relations:

frog-hop > ZERO

frog-hop = frog-y-position<sub>new</sub> - frog-y-position<sub>old</sub>
Influences:
There is a movement which changes the frog’s position such that
I+ (frog-y-position, movement)
I+ (frog-lane, movement)

process backward-movement

Individuals:
frog: agent under the control of the player

Preconditions:
frog-state = Alive
User presses 'Down' arrow key

Quantity conditions:
Let frog-y-position\_old be the Y Coordinate of the frog before movement,
let frog-y-position\_new be the Y Coordinate of the frog after movement,
then (frog-y-position\_old - frog-y-position\_new) > ZERO

Relations:
frog-hop > ZERO
frog-hop = frog-y-position\_old - frog-y-position\_new

Influences:
There is a movement which changes the frog’s position such that
I-(frog-y-position, movement)
I-(frog-lane, movement)

process left-movement

Individuals:
frog: agent under the control of the player

Preconditions:
frog-state = Alive
User presses 'left' arrow key

Quantity conditions:
Let frog-x-position\textsubscript{old} be the X Coordinate of the frog before movement, let frog-x-position\textsubscript{new} be the X Coordinate of the frog after movement, then $(\text{frog-x-position}_{\text{old}} - \text{frog-x-position}_{\text{new}}) > \text{ZERO}$

Relations:
- frog-Hop > ZERO
- frog-hop = frog-x-position\textsubscript{old} - frog-x-position\textsubscript{new}

Influences:
There is a movement which changes the frog’s position such that
- I- (frog-x-position, movement)

**process right-movement**

Individuals:
- frog: agent under the control of the player

Preconditions:
- frog-state = Alive
- User presses ‘right’ arrow key

Quantity conditions:
Let frog-x-position\textsubscript{old} be the X Coordinate of the frog before movement, let frog-x-position\textsubscript{new} be the X Coordinate of the frog after movement, then $(\text{frog-x-position}_{\text{new}} - \text{frog-x-position}_{\text{old}}) > \text{ZERO}$

Relations:
- frog-hop > ZERO
- frog-hop = frog-x-position\textsubscript{new} - frog-x-position\textsubscript{old}

Influences:
There is a movement which changes the frog’s position such that
- I+ (frog-x-position, movement)

**process collision**

Individuals:
- frog: agent under the control of the player
vehicle: Objects that not controlled by the player

Preconditions:
  frog-state = Alive
  frog-on = road
  frog-lane > ZERO
  frog-lane = vehicle-lane

Quantity conditions:
  frog is in contact with the vehicle

Relations:
  Player loses a turn

Influences:
  frog-state = Dead

**process leap-onto-turtle**

Individuals:
  frog: agent under the control of the player
  turtle: Objects not controlled by the player

Preconditions:
  frog-state = Alive
  frog is at the median or it is on a log i.e. frog-on = log
  Player presses ’Up’ arrow key

Quantity Conditions:
  turtle-y-position - frog-y-position = frog-hop

Influences:
  frog undergoes a jump such that
  I+(frog-y-position, jump)
  I-(frog-x-position, jump)
  I+(frog-speed, jump) i.e. frog-speed = turtle-speed
  frog-direction = turtle-direction
  frog-on = turtle
**process leap-onto-log**

Individuals:
- frog: agent controlled by the player
- log: Objects not controlled by the player

Preconditions:
- frog-State = Alive
- frog is on a turtle's back i.e. frog-on = turtle or it is on another log i.e. frog-on = log
- Player presses 'Up' arrow key

Quantity Conditions:

Influences:
- frog undergoes a jump such that
  - I+(frog-y-position, jump)
  - I+(frog-x-position, jump)
  - I+(frog-speed, jump) i.e. frog-speed = log-speed
- frog-direction = log-direction
- frog-on = log

**process sink**

Individuals:
- frog: agent controlled by the player
- playing-field: the environment of the game

Preconditions:
- frog-state = Alive
- playing-field-medium = water
- frog has successfully crossed the road
- Player has pressed the 'Up' arrow key

Quantity Conditions:
- frog-on = water

Relations:
frog is in contact with water

Influences:

frog-state = Dead
Game end
Player loses

Behavior prediction of Frogger based on the variable frog-y-position

The variables in Frogger are the various parameters or quantities of the frog object especially the position coordinates of the frog i.e. frog-x-position and frog-y-position, the lane in which the frog is at a given point of time and the state of the frog (dead or alive). We use frog-y-position to predict the different behaviors of the system.

Quantity Space for frog-y-position The quantity space for frog-y-position consists of the following distinguished values and relations.

frog-y-position$_{init}$ Initial Y coordinate of the frog
vehicle-y-position The Y coordinate at which a vehicle exists
median-y-position The Y coordinate at which the median exists
turtle-y-position The Y coordinate at which a turtle exists
log-y-position The Y coordinate at which a log exists
goal-y-position The Y coordinate of frog’s home.

Once the game begins, we see that frog-y-position is always increasing. Thus an episode EP0 is defined as,

\[ |dfrog-y-position| = + \]

Limit Analysis Whenever it is said that frog-y-position is equal to the y-coordinate of the vehicle, log, turtle or home, it should be noted that the x coordinate of the frog also falls within the bounding area of the vehicle, log, turtle or home of the frog.
frog-y-position = 0

An individual view of Start-Game and Frog-Alive is valid.

0 < frog-y-position < goal-y-position

Individual view of Game-In-Progress is valid. The Start-Game view terminates. An instance of forward-movement process is active.

frog-y-position = vehicle-y-position

An instance of collision process becomes active. The Game-In-Progress view becomes invalid. Eventually, Frog-Dead view becomes valid, thereby terminating all other processes and views.

frog-y-position = turtle-y-position

An instance of leap-onto-turtle process is active. Frog-Alive view and Game-In-Progress views are valid. An instance of forward-movement process is active.

frog-y-position = log-y-position

An instance of leap-onto-log process is active. Frog-Alive view and Game-In-Progress views are valid. An instance of forward-movement process is active.

frog-y-position = goal-y-position

An individual view of Goal-Game becomes valid. All other processes and views are terminated. An instance of forward-movement process is active.

Thus, we have modeled all the entities identified in the game of Frogger as objects and their various attributes as quantities. It is possible to map the different interactions that occur during the game of Frogger to the processes as modeled using Qualitative Process Theory. This mapping is listed in the table 3.1.

Table 3.1: Mapping between interactions and processes that are based on QPT in Frogger

<table>
<thead>
<tr>
<th>Interactions</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>frog / vehicle</td>
<td>collision process</td>
</tr>
<tr>
<td>frog / water</td>
<td>sink process</td>
</tr>
<tr>
<td>frog / log</td>
<td>leap-onto-log process</td>
</tr>
<tr>
<td>frog / turtle</td>
<td>leap-onto-turtle process</td>
</tr>
</tbody>
</table>
Figure 3.1: Prediction of Player-lose behavior based on frog-y-position
Figure 3.2: Prediction of Player-win behavior based on frog-y-position
The initial state of the game of Frogger is represented by the individual view Start-Game and the goal state of the game is depicted by the individual view Goal-Game.

**Representation of the behavior of variables in Frogger over time**

The prediction of behavior of Player-lose and Player-win scenarios in Frogger via the histories i.e. episodes and events, of individual views and processes and evolution of the variable frog-y-position, is depicted in figure 3.1 and 3.2 respectively.
3.2 Modeling of Pinball

The environment of Pinball consists of a play-field which is usually a table. This environment is bound on three sides by walls which specify the boundaries of the game space. The agent is the left and right flipper. The agent is present at the lower end of the play-field. The flippers keep the ball within the boundaries of the game area by deflecting it into the play-field on contact. Thus, the ball is the object which is under the control of the agent. The various targets contained in the environment such as bumpers, lanes, ramps etc. form the set of objects which are not under the control of the agent. The play-field is tilted in such a way that the ball always rolls down towards the flippers.

The agent possesses properties such as a fixed location, size, and shape, and when moved, it undergoes motion along a circular arc. The object under the control of the agent i.e. the ball, has attributes of location, size, shape, speed and undergoes a parabolic trajectory. The ball can move in all possible directions. The targets have properties of location, size and shape. The start and finishing point of the game is represented as states of the environment. The game starts when the ball is launched for the first time. The end of the game is defined by the state of the environment, when the ball’s position is outside that of the lowermost vertical end of the screen. In addition, the play-field in conventional Pinball games is an inclined surface, as a result of which, the ball always tend to roll towards the lower-end of the play-field.

Various interactions are possible. The contact between the flippers and the ball can lead to two different behaviors based on flipper motion or the lack of it. One, when the flipper is in motion, it leads to a change in the position, direction and speed of the ball. Secondly, if the flipper is stationary, there is only a change in the ball’s motion to a linear trajectory. The interaction between the ball and the targets such as bumper or ramps leads to a change in the ball’s location and direction. When the ball interacts with lanes, a change in the direction of the ball is observed due to the conversion of the ball’s parabolic movement into a linear one. The end of a Pinball game is indicated by the ball rolling off the play-field.

It should be clear from this discussion that not all aspects of the game are (or can be) modeled in this language. We discuss modeling limitations in the conclusion of this chapter.
3.2.1 Representation of Pinball based on QPT

Representation of objects identified in Pinball

**flipper**
- Quantity-Type(flipper-size)
- Quantity-Type(flipper-shape)
- Quantity-Type(flipper-type)
- Quantity-Type(flipper-speed)
- Quantity-Type(flipper-direction)
- Quantity-Type(flipper-x-position)
- Quantity-Type(flipper-y-position)
- Quantity-Type(flipper-bounding-area)

  Has-Quantity(left, flipper-type)
  Has-Quantity(right, flipper-type)

**ball**
- Quantity-Type(ball-x-position)
- Quantity-Type(ball-y-position)
- Quantity-Type(ball-size)
- Quantity-Type(ball-shape)
- Quantity-Type(ball-speed)
- Quantity-Type(ball-direction)

**target**
- Quantity-Type(target-x-position)
- Quantity-Type(target-y-position)
- Quantity-Type(target-bounding-area)
- Quantity-Type(target-shape)
- Quantity-Type(target-size)
ramp
  Quantity-Type(ramp-x-position)
  Quantity-Type(ramp-y-position)
  Quantity-Type(ramp-slope)
  Quantity-Type(ramp-size)
  Quantity-Type(ramp-shape)

plunger
  Quantity-Type(plunger-x-position)
  Quantity-Type(plunger-y-position)
  Quantity-Type(plunger-force)

playing-field
  Quantity-Type(playing-field-bounding-area)
  Quantity-Type(playing-field-height)
  Quantity-Type(playing-field-width)
  Quantity-Type(playing-field-slope)

Representation of Various views of Pinball depicting different objects and their relations

Individual views of the playing-field object

Individual view Start-Game
Individuals:
  ball: agent controlled indirectly by the player
  playing-field: the environment within which the game takes place
Preconditions:
The ball is located within the bounding-area of the background playing field.

Quantity Conditions:
- The ball is in its initial position

Relations:
- The ball is not yet in motion.

**Individual view End-Game**

Individuals:
- ball: agent controlled indirectly by the player
- playing-field: the environment within which the game takes place

Preconditions:
- score > ZERO

Quantity conditions:
- ball-y-position <= ZERO

Relations:
- The ball has rolled off the play-field

**Individual view Game-In-Progress**

Individuals:
- ball: agent controlled indirectly by the player
- playing-field: the environment within which the game takes place

Preconditions:
- ball is not in its initial position

Quantity conditions:
- The ball has already been launched i.e. post ball-launch

Relations:
- The ball moves around the play-field
Process Representation in Pinball

**process ball-launch**

**Individuals:**
- ball: agent controlled indirectly by the player
- plunger: object used to initially launch the ball into the play-field

**Preconditions:**
- The ball is still in its initial position
- The ball is in contact with the plunger

**Quantity Conditions:**
- User presses the key to launch the ball

**Relations:**
- The ball begins to move around the play-field.

**Influences:**
- plunger-force is the rate at which the plunger launches the ball, such that
  \[ I+(\text{ball-momentum}, \text{plunger-force}) \]
  \[ I+(\text{ball-y-position}, \text{plunger-force}) \]

**process ball-movement**

**Individuals:**
- ball: agent indirectly controlled by the player

**Preconditions:**
- ball has already been launched

**Quality Conditions:**
Let \( \text{ball-x-position}_{old} \) and \( \text{ball-y-position}_{old} \) be the coordinates of the ball before movement, let \( \text{ball-x-position}_{new} \) and \( \text{ball-y-position}_{new} \) be the coordinates of the ball after movement, then,
\[ | \text{ball-x-position}_{new} - \text{ball-x-position}_{old} | > \text{ZERO} \]
\[ | \text{ball-y-position}_{new} - \text{ball-y-position}_{old} | > \text{ZERO} \]

**Relations:**
- ball moves around the play-field.

**Influences:**
- Change in ball’s location.
**process collision-target**

Individuals:
- ball: agent indirectly controlled by the player
- target: Stationary object present in the play-field which are not controlled by the player

Preconditions:
- ball has already been launched

Quality Conditions:
- There is at least one point of contact between the ball and a target. i.e the ball’s circumference touches at least one point on the target-bounding-area

Relations:
- The ball bounces off the target

Influences:
- There is a collision-rate > ZERO, such that
  - I-(ball-momentum, collision-rate)
  - Change in the ball’s direction,
  - Increase in the player’s score

**process movement-on-ramp**

Individuals:
- ball: agent indirectly controlled by the player
- ramp: stationary inclined plane present in the play-field

Preconditions:
- The ball is in motion

Quality Conditions:
- The ball is on the ramp

Relations:
- The ball ends up near the flipper

Influences:
- I+(ball-momentum, ramp-slope)
- Change in the ball movement from a parabolic trajectory to linear
Change in the ball’s position
Increase in the player’s score

**process flipper-action**

Individuals:
- ball: agent indirectly controlled by the player
- flipper: agent controlled by the player

Preconditions:
- User presses the key(s) for flipper movement

Quality Conditions:
- ball is within the range of contact of flipper

Relations:
- The player keeps the ball within the play-field by deflecting the ball using the flippers

Influences:
- flipper-speed > ZERO
- \( I^+\) (ball-y-position, flipper-speed)
- Change in ball direction
- \( I^+\) (ball-momentum, flipper-speed)

Thus, we have modeled all the entities identified in the game of Pinball as objects and their various attributes as quantities. The mapping between the various interactions that occur during the game of Pinball with that of the processes as presented in the representation based on Qualitative Process Theory is as listed in the table 3.2.

<table>
<thead>
<tr>
<th>Interactions</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>flipper / ball</td>
<td>flipper-action process</td>
</tr>
<tr>
<td>ball / target</td>
<td>collision-target process</td>
</tr>
<tr>
<td>ball / ramp</td>
<td>movement-on-ramp process</td>
</tr>
<tr>
<td>ball / plunger</td>
<td>ball-launch process</td>
</tr>
</tbody>
</table>

The starting state of the game of Pinball is represented by the individual view Start-Game and the finish state of the game is depicted by the individual view End-Game.
Prediction of behavior of Pinball over time

The variables in the game of Pinball are the various properties associated with the ball - location coordinates of the ball, the direction of the ball and the momentum possessed by the ball.

Quantity Space of ball-y-position  The quantity space of ball-y-position varies between the interval of 0 to play-field-height with distinguished values such as 0, ball-y-position_{init}, flipper-y-position, collision-y-position, play-field-height.

Limit Analysis  Whenever it is said that ball-y-position is equal to the y-coordinate of the target or flipper, it should be noted that the x coordinate of the ball also falls within the bounding area of the target or flipper.

\[
\begin{align*}
\text{ball-y-position} &= \text{initial position} \\
&\text{and ball has not yet been launched} & \text{An individual view of Start-Game is valid.}
\end{align*}
\]

\[
\begin{align*}
0 < \text{ball-y-position} < \text{height of the play-field} \\
&\text{and ball is being launched} & \text{An instance of ball-launch process is active.}
\end{align*}
\]

\[
\begin{align*}
0 < \text{ball-y-position} < \text{height of the play-field} \\
&\text{post-launch} & \text{An individual view of Game-In-Progress is valid and an instance of ball movement process is active.}
\end{align*}
\]

\[
\begin{align*}
\text{ball-y-position} &= \text{flipper-y-position} & \text{An instance of the flipper-movement process becomes active.}
\end{align*}
\]

\[
\begin{align*}
\text{ball-y-position} &= 0 & \text{An individual view of End-Game is valid. All other views and instances of other processes are terminated.}
\end{align*}
\]

Representation of the behavior of variables in Pinball over time

The prediction of Game behavior in Pinball via the histories i.e. episodes and events of individual views and processes and evolution of the variable ball-Y-Position, is depicted in figure 3.3.

In the figure 3.3, the peaks indicate the collision of the ball with a target. It must be also noted that the ball is always under the influence of gravity. In addition, the curved peak in the figure 3.3 indicates the movement of the ball under the influence of gravity.
Figure 3.3: Histories and evolution of ball-y-position
without undergoing collisions with targets. Also, the decreasing portion of curve indicates that the ball has a tendency to move towards the lower end of the pinball table, due to the inclined surface of the table.

### 3.3 Advantages of the modeling approach

The advantages of this approach of modeling games are several. The most important of them, is its use in identifying an accurate mapping between a video game and a video. We are able to identify the various game elements involved in the video game in terms of objects in this approach. Similarly, the interactions involved in a video game are represented by processes. The mapping between a video game and a video, starts with identifying the game elements which are controlled by the player in the video game. These are virtual entities in VGV, due to the assumption that agents under the control of the user are virtual entities in a VGV. Next we need to identify the critical game elements in the video game which has a huge impact on the game-play i.e. defines the tempo of the game. These are identified via the behavior prediction of the system in our modeling approach. These vital game elements possess certain characteristics. These have to be present in the video. Thus these are the real entities in the VGV. To complete the mapping between the video game and the video in hand, any remaining game elements can be added as virtual entities.

This modeling approach enables the representation of a gaming situation as a physical system. It also allows prediction of the behavior of a game in terms of attributes of the various game-elements involved and aids in the identification of the range of values (along with distinguished values) that these attributes assume, over time. In addition, an interaction is defined in terms of a process; as a result there is a clear description of the various entities involved, the conditions which have to be satisfied for the interaction to occur and the effect the interaction can have on the rest of the system. Thus it is possible to visualize the various entities and their states that are active over a certain duration.

### 3.4 Limitation of the modeling approach

Using the modeling approach, it is not possible to predict the value of a variable at every given point of time accurately. This is because only the values that a variable can
take between each interval of time is defined, thus providing a lower and upper limit for the value that variable can hold within that time interval. But what happens within that time interval is not specified.
Chapter 4

System Design

4.1 Design

The goal for software design in the VGV can be thought of as mapping the properties of the 2D game model i.e. Frogger or Pinball to the structure and behavior of the video, to produce an integrated and playable game.

A high level overview of the design process is as shown in the figure 4.1.

Figure 4.1: System Design

The design process of a VGV begins with capturing scenes from the real world using a camera. The Graphics system is responsible for the generation of virtual entities which are then fused with a selected video from storage. The rules of the game under
consideration is introduced into the augmented video to produce a VGV.

In a VGV, virtual entities are computer generated objects. The graphics system consists of a graphics library such as OpenGL. In AugFrog, the frog is the virtual entity while in Human Pinball it is the left flipper, right flipper and the ball. In every game, at least one of these entities is controlled by the user of the system.

Video merging involves the identification and separation of the various entities found in the video and fusing these objects with the virtual objects created by the graphics system. One of the vital steps at this stage is distinguishing between background and foreground elements in the video. This is followed by environment extraction, which deals with identifying entities in the background which divide the environment into different regions of interest. This could be lanes in Frogger or the bounding walls of the play-field in Pinball. This is achieved using the background model obtained from the previous step. Similarly, entities of interest which usually constitute the foreground, have to be extracted from the video and modeled.

The system is generated by integrating the environment, entities of interest and virtual objects along with the model of the 2D game which specifies the rules and constraints of the game. This integration involves dealing with various interactions that arise between the virtual entities and the video.

4.2 Implementation

The critical aspects of software design for AugFrog and Human Pinball, and comparable VGVs, lie in two areas of image processing: scene analysis and object management. Scene analysis establishes the mapping between a 2D video game and a 3D video environment; for example, in AugFrog the point of view is from a roadside camera rather than the birds eye view of Frogger. Identification and management of objects in the video also includes their interaction with virtual agents and objects relevant to the original video game.

We aimed to use the simplest, computationally least expensive off-the-shelf techniques for image processing. AugFrog has been implemented using publicly available implementations of image processing algorithms and Human Pinball has been implemented using Intel Corporation’s open source library for Computer Vision, OpenCV. Graphics and window management are handled by OpenGL.

We use the term agent to refer to the virtual entity(s) that is controlled by the
user. There may exist other virtual objects that are controlled by the agent. For example, in Human Pinball, the virtual ball is controlled by the flippers, as the ball is deflected back into the play-field via flipper motion. All other entities identified in the game are termed as objects.

4.2.1 AugFrog

The main goal in mapping the video onto the problem space is to model several real-world interactions that could occur when a frog attempts to cross a road with heavy traffic. To achieve this, we need to extract the required information from the video traffic stream while identifying and recording attributes that can be used to map the virtual entity with objects of interest obtained from the video into a common space. There have been several situations during this phase, where the domain knowledge has been used as a heuristic to solve issues that arise from dealing with the real-world traffic video. An overview of the implementation of AugFrog is as shown in the figure 4.2

The preprocessing in AugFrog involves obtaining a frame from the recorded video, which consists of only the background, i.e. the foliage, median and the lanes, in this case. In video processing, background usually refers to a set of objects that remain passive in the scene. One of the approaches to performing background subtraction is by segmenting a frame into objects and identifying the background by choosing a threshold. Any region whose brightness is above the threshold is considered to be a foreground object and all those below the threshold as background entities [27]. Thus, we obtain a frame where there are no foreground objects. This image serves as a reference image for the background. All the frames that follow in the video stream are then subtracted from this background image.

Scene Analysis in AugFrog

Scene Analysis ensures appropriate mapping of the traffic video onto the model of the Frogger game. This occurs in several stages.

Separation of background from foreground: The first step is to separate the background consisting of foliage and lanes from the moving vehicles. The only objects in the video feed with which the frog can interact are those that are moving (we ignore, for example, the case of a car pulled over with a flat). A simple way to isolate those parts of an
Figure 4.2: Design of AugFrog
image that are in motion is via background subtraction. By constructing and maintaining a composite version of the scene with all the traffic removed, which is done during the preprocessing stage, foreground objects (i.e. moving vehicles) can be identified easily.

**Detecting lanes / median:** The next step involves detecting the lanes and the median. The lanes on the road surface then correspond to the regions in the Frogger model.

An edge detection algorithm (Canny) is applied on the composite version of the scene with all the traffic removed (this was also used during the background subtraction stage) which was obtained during the preprocessing stage. Looking at the edges that are detected by the algorithm, we see that the strong lines found close to the foliage represent the boundaries and the median (which is seen towards the central portion of the height of the image); the lanes are seen as broken lines. Scene analysis in AugFrog focuses on identifying the perspective associated with the roadside camera position. The difficulty in identifying lanes arises from the fact that lanes (i.e. broken lines in the edge detection result) are clearer on the nearer side than the farther side of the image due to the road-side view of the video used in our prototype. In order to solve this problem, we make the assumption that the lanes are correctly identified on the nearer side.

Scene analysis serves another role aside from establishing perspective: the rules for interaction in the Frogger model entail that interactions between the frog agent and vehicle objects in road regions need to be handled only on lanes; further, the interaction occurs only within the single lane occupied by the agent. For instance, at every point of time, the frog maintains not only its position and direction but also the lane in which it is present in. Consequently, when checking for object collision, we can test to make sure that the frog and the vehicle do in fact occupy the same lane. Similarly, we do not need to detect collision when the frog is on the median.

**Object Management in AugFrog**

Object management has several facets. One general concern is approximating 3D realism: a virtual character simply superimposed on a video of a 3D environment does not provide sufficient cues about position and potential interactions with objects in the video. AugFrog relies on two depth cues to address this issue, reduction of the size of the virtual agent due to perspective, and appropriate occlusion when the agent and objects overlap. Other aspects of object management deal with interactions between the virtual agent and
vehicles.

**Object (vehicle) Identification:** The main issue with identifying vehicles is in differentiating between overlapping vehicles (for instance, two vehicles that are traveling in nearby lanes) and a vehicle whose height extends over two lanes (say, a huge truck). To eliminate this problem, we identify vehicles in the same lane as that of the frog. In order to do so, we run a Canny edge detection algorithm on the foreground objects that we extract for every frame via background subtraction and then attempt feature extraction by identifying the wheels of the vehicle using Hough Transform for circles. To reduce the rate of false detection during the circular object extraction, a different range in radius is considered for the nearer and farther lanes, since the range of the sizes of wheels are different in the nearer and farther lanes. The Hough Transform for circles used in AugFrog is a modified C++ version of the implementation in JAVA available at [28].

Additionally, we also need to deal with the issue of the height of the vehicles. That is, while extracting vehicles within a lane, we see the top portion of vehicles that are in a lane closer to the camera than the lane under consideration. By marking only the wheels and the area of vehicles within the current lane, the area of the vehicles appearing beyond the current lane (due to the height of the vehicle) is discarded for further calculations. The only problem we run into with this approach is the issue when one vehicle (especially both its wheels) is completely hidden by another car in a given frame. In this situation, the vehicle that is hidden is not identified in that particular frame.

All necessary information that has been extracted from the traffic video can then be used to produce the interactions dictated by the model.

**Agent (frog) /object (vehicle)/region (road) interactions: contact**

The simplest method for detecting collisions between the frog and vehicles would be to check if the areas of the foreground objects overlap with that of the bounding area of the frog. But due to the camera perspective, we need to take into account the height of the image of the frog as well as that of the vehicles. The amount of overlap between the vehicles and the bounding rectangle of the frog is determined. We set a threshold for overlap, and any overlap count greater than or equal to the threshold is considered a collision. Due to the camera perspective, vehicles in the nearer lane have a greater area than the ones in the farther lane. Hence different thresholds for overlap are used for the nearer and the farther
Agent (frog) /object (vehicle)/region (road) interactions: no contact.

An overlap between the agent and vehicles can indicate a collision; alternatively, the frog might actually be in front of or behind the vehicle, if the count of overlap is below the specified threshold. In the latter cases, occlusion is appropriate. In addition to the thresholds described above, we also use knowledge of the physics of the domain: there will be an overlap at the feet of the frog only when it is behind the vehicle; if not, the frog is in front of the vehicle. Thus, the frog is hidden if there is an overlap of the vehicle area in the foreground with that of the feet of the frog. The virtual frog is introduced into the video frame via displacing the area of the frog in that frame. This overwritten portion of the original frame prior to the insertion of the virtual frog is stored in memory. When occlusion of the frog is required, this stored image is copied into the frame over the frog, thus producing a masking effect.

Agent (frog) /object (log)/region (river) interactions: contact

The Frogger model specifies an interaction for the frog hopping onto a log, such that it acquires the trajectory of the log. There are no logs in the traffic video, but we can merge the two classes of model objects (and the regions they operate in) to provide a new interaction. If a vehicle is within a specific threshold of the frog’s position but does not overlap on the horizontal axis, the user may execute an operator (via pressing a key) that causes the frog to jump onto the vehicle and travel on top for some distance until another operator is executed (via hitting another key). The frog returns to the road at a new location. If the frog travels on a vehicle to the edge of the screen, it is destroyed.

Implementation of this feature requires identification of individual vehicles and gauging their speed. This is handled by identifying the top of the vehicle via a recursive procedure applied on the foreground information and by having a fixed speed of frog movement relative to the motion of the speed of the vehicles in the video, relying on the assumption that the vehicles do not change lanes. The height jumped by the frog is recorded which is later used to get the frog back on the road when the user chooses to by pressing a key i.e. Page Down. A better approach to implement this would be to extract the vehicle under consideration and match it in subsequent frames via a template matching algorithm. But a template matching algorithm using a M*M template on a N*N image would result
in a computational complexity of the order of \( O(M^2N^2) \).

### 4.2.2 Human Pinball

In Human Pinball, we aim to map the video of people walking in a shopping mall into the problem space of the arcade game, Pinball. This is achieved using Intel’s open source library for computer vision, OpenCV, and for the graphics part, OpenGL is used. An overview of the implementation of Human Pinball is as shown in figure 4.3.

Preprocessing in Human Pinball involves identifying and extracting a frame from the video in hand which comprises the corridor of the shopping mall along with the surrounding walls and pillars.

**Scene Analysis in Human Pinball**

Scene Analysis in Human Pinball involves the identification of the environment i.e. play-field of the game.

**Environment Identification:** The first step in the scene analysis of Human Pinball, is to separate the corridor i.e. walls and the floor from the people walking around. This is done via background subtraction. Background subtraction involves calculating a reference image i.e. background obtained during preprocessing, from which other frames are subtracted and subjecting the result to a threshold in order to obtain an image which highlights the moving objects. The resulting image is a binary image of the foreground.

The next step is to identify the walls of the corridor. This sets the boundaries for the ball. This is achieved by running Canny edge detection on the background image using OpenCV’s `cvCanny` and then extracting the perimeter of the walls using Hough Transform for lines via OpenCV’s `cvHoughLines`.

**Object Management in Human Pinball**

Object management in Human Pinball deals with modeling entities of interest from the video, fusing virtual entities and the interaction between the real and virtual objects.

**Object (people) extraction:** This is a vital part of the system under implementation and hence it has to be done in an efficient and robust manner. Connected component labeling is
Figure 4.3: Design of Human Pinball
performed on the binary image of the foreground. This is done by a blob detection library in OpenCV named `cvblobslib`. This library includes methods to identify the bounding rectangle and position of each individual region in a binary image. To reduce noise, the library also has a method which filters blobs below a certain specified area.

The problem with `cvblobslib` is that it maintains the position of identified blobs in 2D co-ordinates. This was the main reason for foregoing the adoption of 3D graphics and modeling in Human Pinball. Also, modeling people in 2D introduced the problem of distinguishing between a person who is upright versus a person who is sleeping on the corridor. This is because the Y axis is used to move the ball along the corridor from the farthest end to the nearest end with respect to the user and the height of humans identified by `cvblobslib` is also measured along the Y-axis.

**Virtual entities:** The left and right flippers are placed at the lower vertical end of the play field. The flipper motion is controlled by the user. The object which is under the control of the agent i.e. the ball has attributes such as its location co-ordinates. It also possesses a time step in which it travels. To emulate realism, the ball changes its direction whenever it collides with a target.

**Interactions:** In Human Pinball, interactions occur between the object under the control of the agent i.e. the ball and objects, the environment and the agent i.e. flippers. All these interactions are identified with collision detection and an appropriate collision response is chosen based on the result of the collision detection algorithm. Collision Detection is a crucial aspect of any real-time simulation in which objects are not supposed to be able to penetrate each other [29]. On collision, the ball’s direction is modified along with its position. In cases where there is no collision, the ball is set to a new position based on its default time step.

Similar to the virtual frog in AugFrog, the ball undergoes scaling and occlusion in order to introduce 3D realism. For instance, when the ball is traveling from the farther end of the corridor towards the user, the ball is occluded whenever there is an overlap between the ball and one of the identified humans.

**Object under the control of the agent (ball)/ objects (human) interactions:** It is vital that the ball must not penetrate through the human. In Human Pinball, the collision
detection between the ball and the blobs is achieved by identifying the overlap between the ball and the bounding rectangle of the blob obtained from the results obtained from the blob detection. The ball then bounces off the humans as per the collision response.

**Object under the control of the agent (ball) / environment (surrounding walls) interactions:** Collision detection between the ball and the walls is identified using pixel overlap between the ball and the walls. The ball is then deflected into the scene based on whether it hits the left or right wall.

**Object under the control of the agent (Ball)/ Agent (flippers) interactions:** For the flippers and ball, collision detection is performed by the pixel overlap at the boundary of the ball with that of the flipper. The collision response differs depending on whether the user moved the flipper while the ball was near it or not. If there is no movement of the flippers, the ball moves along the flipper and finally reaches the end of the screen i.e. end of game. In case there is a movement of a flipper, then the collision response deflects the ball into the play-field, away from the user.
Chapter 5

Analysis

In the previous chapter, we saw an overview of the implementation of VGVs, we now delve into the same in greater detail. In the following section, we look at the technical aspects which have been used in AugFrog and Human Pinball. We discuss the algorithms which have are currently used in the implementation of the games as well as the failed attempts. The second section deals with the formative evaluation of these games.

5.1 Technical Aspects

As described in the software design, section 4.1, the critical aspects of software design for AugFrog and Human Pinball lie in Scene Analysis and Object Management.

5.1.1 Algorithms used for scene analysis in AugFrog

Background Subtraction

Background subtraction is a commonly used class of techniques for segmenting out objects of interest in a scene for applications such as surveillance. It involves comparing an observed image with an estimate of the image if it contained no objects of interest. The areas of the image plane where there is a significant difference between the observed and estimated images indicate the location of the objects of interest.[30]

Since the video is shot by a stationary camera and as it has been possible to obtain a frame from the video which does not have any of the objects of interest, we can use naive approach to background subtraction. We use a frame from the video where there
are no foreground objects i.e. vehicles, which will model the background. An absolute difference is determined between this background frame and every consecutive frame that is extracted as the video progresses. A foreground image is obtained by setting a threshold and classifying any pixel above the threshold as foreground and below the threshold as background. Therefore every pixel in the foreground satisfies this conditional check.

\[ |frame_i - background| > \text{Threshold} \quad (5.1) \]

**Canny edge detection algorithm**

The Canny edge detection algorithm is used in AugFrog to identify the lanes. The algorithm first filters out any noise and performs Gaussian smoothing of the image. Edge strength is then determined by 2D spatial gradient measurement using a Sobel operator. Once edge strength is determined, edge direction is calculated by finding the inverse tangent of the ratio of the gradient in the y direction to that in the x direction. This is followed by rounding off the edge direction angle to either horizontal, vertical or left or right diagonal and a non-maximum suppression is performed to suppress pixels belonging to non-edges. This is followed by thresholding with hysteresis to eliminate any broken edge contours.

The Canny edge detection algorithm has a low error rate, localized edge points and single response to an edge and thus provides a better edge detection even in noisy conditions. But this algorithm is time consuming.

The Canny edge detection algorithm provides a clear identification of lane markings in the lower part of the video. Unfortunately, the farther lanes are not as easily captured by the same processing; the lane markings are much fainter and more closely spaced. For correct lane identification, processing requires information that is not explicit in our Frogger model: that the same number of lanes is present on both sides of the median. To identify lanes missed by the edge detection algorithm, the distance between lanes is measured and external knowledge is applied that traffic lanes are approximately of equal width and that perspective causes their apparent width to shrink with distance. This additional information allows us to generate a correct mapping between the video environment and the road regions of the model.
5.1.2 Algorithms used in Object Management in AugFrog

Object (vehicle) identification using the Canny Edge Detection Algorithm and the Hough Transform for Circles

Initially, a recursive procedure on a binary image of the foreground, was used to identify the moving objects in the foreground. But this led to problems in distinguishing between vehicles that overlap and vehicles with a tall body i.e. the height of the vehicle which extends beyond one lane. To overcome this problem, and since we are interested in the interaction between the frog and the vehicles traveling in that lane, we extract the portion of the scene corresponding to the lane currently occupied by the frog. This information is maintained as an attribute of the frog object. In order to identify the vehicles present in the current lane, we identify the wheels of the vehicles in the lane. This way we can eliminate any vehicle present in the section of the lane due to its height. This is achieved by conducting an edge detection on the extracted section which is then followed by a hough transform for circles to identify the wheels.

The Hough transform is a procedure used to extract shapes especially lines and circles. The Hough transform uses a voting procedure in a transform space or an accumulator whose dimension is equal to the number of unknowns. For a given radius, the locus of possible circle centers from an image point (x,y) gives rise to a circle. Thus, each image point (x,y) in the co-ordinate space maps to a circle of radius r in the transform space. The circles formed by points on the perimeter of a circle in the co-ordinate space intersect at a point which is the center of the circle (a,b) in co-ordinate space. A circle can be represented in polar coordinates as

\[ x = a + r \cos \Theta \]  
\[ y = b + r \sin \Theta \]

where, (x,y) are image points,
(a,b) represents the center of the circle,
r is the radius of the circle,
\( \Theta \) is the gradient angle which varies from 0 to 360°

From the above equations we now end up with three parameters (a,b,r). Hence we use a 3D accumulator to represent the transform space. The values of \( \sin \Theta \) and \( \cos \Theta \)
can be computed in advance, and thus for a given radius, a and b can be easily determined. This procedure can be repeated to find circles within a specified range of radii.

A summary of the algorithms used for implementing AugFrog is as listed in table 5.1.

Table 5.1: Summary of algorithms used in the implementation of AugFrog.

<table>
<thead>
<tr>
<th>#</th>
<th>Aim</th>
<th>Algorithm Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Separation of background and foreground</td>
<td>Background Subtraction via thresholding</td>
</tr>
<tr>
<td>2</td>
<td>Identification of Lanes</td>
<td>Canny edge detection</td>
</tr>
<tr>
<td>3</td>
<td>Identification of vehicles in the current frog-lane</td>
<td>Canny edge detection and Hough transform for circles</td>
</tr>
<tr>
<td>4</td>
<td>Interaction (Collision and frog occlusion)</td>
<td>Pixel Overlap</td>
</tr>
</tbody>
</table>

5.1.3 Algorithm used for performing scene analysis in Human Pinball

Scene analysis in Human Pinball performs mapping of the video in hand onto the model of Pinball. This involves the detection of humans in each frame and determining their bounding rectangles. Also, we need to identify the walls which define the boundaries for the ball on the screen. Since OpenCV consists of numerous functions for image processing, there are usually more than one way to achieve a particular result. Hence we present our attempts at using various algorithms in this section.

Background Subtraction and Foreground Detection

OpenCV contains two models for background subtraction – *cvGaussianBGM*odel and *cvFGDS*at*Model*.

First, we tried to achieve background subtraction using the gaussian mixture model i.e *cvGaussianBGM*odel. This is an adaptive background mixture model which uses a multi-color background model per pixel, and also possesses the ability to detect shadows. This model is based on an algorithm described in [31]. Using this method on the video in hand resulted in obtaining a clear background, but the binary foreground image consisted of a lot of noise in addition to the foreground objects. This is perhaps due to the fact that this adaptive model requires a few frames to adapt.
Next, we used the `cvFGDStatModel` to achieve background subtraction. This is based on an algorithm described in the paper by Li et al [32]. This algorithm performs foreground object segmentation from real-time video using the Bayes decision framework, which can integrate multiple features to model the background for foreground object detection. Applying this method on our video, resulted in obtaining a foreground which was lesser noisy than that obtained using `cvGaussianBGModel`. But this algorithm seems to be prone to absorbing foreground objects if they are motionless for a long time [32]. Thus, there were portions of the humans which were visible in the background image which led to foreground objects with missing parts.

On studying the video in hand, we realized that the background consists of motionless objects i.e. walls and floor. Also, there were no significant illumination changes and the video was captured by a stationary camera. Hence, we require an approach that would separate the static background from the non stationary foreground objects. The only criteria were that the foreground had to be obtained with as little noise as possible and the entire process had to be fast enough so as not to slow down the game. Thus, we resorted to the naïve approach of background subtraction via thresholding as described in section 5.1.1. The absolute difference between images is achieved using `cvAbsDiff` method of OpenCV. A threshold is then set to create a binary image which consists only of the foreground objects. This is achieved using the `cvThreshold` method in OpenCV.

**Hough Transform for Line Detection**

The environment of Human Pinball consists of a corridor of a shopping mall. The perimeter of the bounding walls is identified using the Canny edge detection algorithm, which is achieved by OpenCV’s `cvCanny` method and the Hough transform for lines, using `cvHoughLines`, to detect the appropriate lines that define the boundaries of each wall.

The Hough transform for lines uses the normal parametric notion to represent straight lines as described in the equation below:

\[ p_j = x_i \cos \Theta_j + y_i \sin \Theta_j \]

(5.4)

where, \((x_i, y_i)\) represents a pixel in the image whose value is one ,
\n\n\n\n\n\Theta_j \text{ varies from 0 to } 180^0,\n\n\n\n\np_j \text{ varies according to the stepwise increments of } \Theta_j \]
A transform space is used, where every calculated \((p_j, \Theta_j)\) for a particular \((x_i, y_i)\) is assigned a value of one. Thus, plotting all possible \((p_j, \Theta_j)\) for a particular \((x_i, y_i)\) maps to a cosine curve in the transform space. This point-to-curve transformation is the Hough transformation for straight lines. When viewed in Hough parameter space, points which are collinear in the cartesian image space become readily apparent as they yield curves which intersect at a common point [33]. The Hough transform space is quantized in the algorithm into accumulator cells. For every \((x_i, y_i)\), the value of the accumulator cells along the \((p, \Theta)\) curve is incremented. The resulting peaks in the accumulator array are indicative of the presence of straight lines in the image.

5.1.4 Algorithms used for Object Management in Human Pinball

Object Identification

In the video used for Human Pinball, the moving objects in the foreground are people. Although the foreground that is extracted as described in the previous section 5.1.3, gives us the non-stationary objects, we need to determine their positions and hence their bounding rectangles. This information will aid in mapping the humans and the virtual entities into a common space for interaction.

Our initial attempt was to use the face detection module available in OpenCV using haar classifier. The \texttt{cvHaarDetectObjects} method performs rapid object detection with a cascade of boosted classifiers based on haar-like features. This method can be used to detect faces (both frontal and profile view), lower body, upper body and complete human body. The algorithm implemented by this method is based on the work by Lienhart and Maydt [34] which is a combination of the rapid object detection framework based in a boosted cascade of simple features by Viola et al.[35] along with a set of rotated haar-like features.

When this method was used on our video, it failed to detect some of the faces. This is due to the limitation that the algorithm works on detecting faces which have a dimension of at least 40x40 pixels, which was not the case in our video.

Our next option was to identify the connected components or blobs in the extracted foreground. OpenCV provides two options to achieve this. One is an automated blob tracker module - \texttt{cvblobTrackerAuto} and the other is a blob detection library namely \texttt{cvblobslib}. 

**cvblobTrackerAuto:** This is a five stage blob tracker and detection system as shown in the figure (See Fig. 5.1)

![Diagram of Blob Tracker Auto](image)

**Figure 5.1: Blob Tracker Auto**

The FG/BG Detection module performs foreground/background segmentation for each pixel. The Blob Entering Detection module uses the result (FG/BG mask) of FG/BG Detection module to detect new blob objects entered to a scene on each frame. The Blob Tracking module tracks each new entered blob. The Trajectory Generation module collects all blob positions. The Trajectory PostProcessing module performs a blob trajectory smoothing function using Kalman filtering.

When this was applied to the video in hand, it resulted in a very slow and inefficient tracking because certain blobs were not detected or since they were lost during tracking. This is mainly due to the drawbacks of the FG/BG module as it is uses the `cvGaussianBGModel` and `cvFGDStatModel` as explained in the previous section 5.1.3.

**cvblobslib:** This is a blob extraction library which performs connected component labeling on binary images. Mapping the concept of connected components in Graph Theory to image analysis, foreground pixels are considered as vertices in a graph and the adjacency or connectedness is determined using the edges or the boundaries in the image. `cvblobslib` is a based on an implementation by Dave Grossman [36]. The basic idea is to raster scan the image, numbering any new regions that are encountered during the process and merging regions identified previously if they are found to be connected on a lower row. The algorithm calculates the area, bounding box and perimeter. The perimeter calculation is done in two passes. During the first pass, the image is converted into run code. During the second pass, the run codes are analyzed for two consecutive rows. The areas are then merged
or split based on selective connectivity. Different cases exist based on the starting and ending columns of each region in the run code of each of the two rows under consideration. Also, criteria such as the colors i.e. black or white, whether or not they were previously encountered and whether there is a bridge between two old regions, are also used while identifying connected regions [36].

The execution time of the algorithm used to detect blobs in cvblobslib is to the order of \( \text{rows} \times \text{columns} + \text{rows} \times \text{number} - \text{of} - \text{blobs} - \text{detected} \). As a result, the computational time of this algorithm is directly proportional to the number of rows and columns. This imposes a limitation on enabling the full screen mode for the game.

Agent Interactions

Collision Response:

**Attempt at modeling the collision detection and collision response in 3D space**

Initially, we made an attempt at modeling the entire scene in 3D co-ordinate space using OpenGL’s extensive support for graphics in 3D. Although, we were able to implement ball movement, collision detection and collision response successfully in 3D, we ran into problems, due to which, we had to abandon this approach. Firstly, OpenCV’s blob detection identifies blobs only in 2D coordinates. Secondly, we had to recreate each frame of the video into 3D space. i.e, extract each frame using OpenCV, then using OpenGL’s texture mapping, map the frame into 3D space. Gradually, it became necessary to divide each frame into parts such as left wall, right wall, floor etc. and render them as sub textures. But this also meant that the moving objects also had to be rendered as a sub texture which made this problem a lot more complicated.

**Attempt at modeling the collision detection and collision response in 2D space**

Since we are modeling the real world, the object under the control of the agent i.e. the ball possesses attributes such as position and a velocity vector. Also, the ball travels at a particular time step which changes when it collides with entities. The ball is subject to acceleration due to gravity and undergoes a loss of inertia on collision.

Once a collision is detected between the ball and agent, objects or the environment, the ball’s post collision velocity vector is computed and its new position using the new time
step (that was calculated during collision detection) is calculated. The collision detection algorithm gives us the collision point. To calculate the resultant vector, $R$, of the ball after collision, laws of physics are applied. We know the angle made by the initial vector, $I$, with the normal to the plane, $N$, is equal to the angle made by the normal to the plane, $N$, with the resultant vector, $R$, at the collision point. This is shown in the figure below

![Figure 5.2: Collision Response](image)

This is represented by the equation

$$R = 2(-I \cdot N) \cdot \hat{N} + \hat{I}$$  \hspace{1cm} (5.5)

where, $R$ is the new direction (after collision), $I$ is the initial vector (before collision), $N$ is the normal to the plane (this is any entity with which the collision occurred).

But this led to considering planes as a straight line and not as rectangles. Therefore we switched to a simpler approach of employing simple pixel overlap for collision detection and changing directions of the ball and employing a time step for the ball as collision response. The ball moves along the length of the corridor.

A summary of the algorithms used for implementing Human Pinball is as listed in the table 5.2.
Table 5.2: Summary of algorithms used in the implementation of Human Pinball.

<table>
<thead>
<tr>
<th>#</th>
<th>Aim</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Separation of background and foreground</td>
<td>Background Subtraction via thresholding</td>
</tr>
<tr>
<td>2</td>
<td>Identification of walls</td>
<td>Canny edge detection and Hough Transform for lines (pre-processing)</td>
</tr>
<tr>
<td>3</td>
<td>People Identification</td>
<td>Blob detection library - <em>cvblobslib</em></td>
</tr>
<tr>
<td>4</td>
<td>Ball Movement</td>
<td>Set position based on time step</td>
</tr>
<tr>
<td>5</td>
<td>Collision Detection</td>
<td>Pixel Overlap</td>
</tr>
<tr>
<td>6</td>
<td>Collision Response</td>
<td>Modification of the direction of the ball along with position</td>
</tr>
</tbody>
</table>
Chapter 6

User Evaluation of AugFrog and Human Pinball

Computer Games provide a unique test bed for HCI, computer graphics, Computer Supported Cooperative Work and perception [18]. The entertaining nature of gaming interactions entices users who would usually not be interested in testing prototype systems.

We conducted a formative evaluation of the two games we developed as proof of concept, AugFrog and Human Pinball. The sample size used for the evaluation was five. The users were given a brief introduction to each of the games prior to their game-play. Users were informed about the objective of the game, the player controls, scoring and the goal of the game along with a short description about the game (rules and constraints). Each user had to complete a user feedback form at the end. The questions in the feedback form were mainly targeted at collecting the interesting and difficult aspects of each game, along with suggestions for improvement of the games.

6.1 User feedback

In this section, we present the feedback given by the users’, to the various questions asked in the feedback form, in verbatim.

The users’ views about the interesting aspects in AugFrog and Human Pinball are as given in table 6.1 and table 6.2.

The difficulties faced by the users in AugFrog and Human Pinball are as listed in the table 6.3 and table 6.4.
### Table 6.1: Users’ feedback on the interesting aspects in AugFrog

<table>
<thead>
<tr>
<th>User #</th>
<th>User Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The frog’s ability to jump on the car is pretty cool</td>
</tr>
<tr>
<td>2</td>
<td>The system can detect real object shapes</td>
</tr>
<tr>
<td>3</td>
<td>Interaction with real traffic</td>
</tr>
<tr>
<td>4</td>
<td>It was more entertaining since the obstacle was more disastrous</td>
</tr>
<tr>
<td>5</td>
<td>The frog jumps on and off vehicles! I never knew frogs can jump that high.</td>
</tr>
</tbody>
</table>

### Table 6.2: Users’ feedback on the interesting aspects in Human Pinball

<table>
<thead>
<tr>
<th>User #</th>
<th>User Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Integrating a game with an actual scene from real life</td>
</tr>
<tr>
<td>2</td>
<td>The ball bounces off human shapes</td>
</tr>
<tr>
<td>3</td>
<td>It adds an element of physical violence that I find missing in normal Pinball</td>
</tr>
<tr>
<td>4</td>
<td>Fun, entertaining, easy to follow and control the ball</td>
</tr>
<tr>
<td>5</td>
<td>Human Targets! Cool! I wish there were political personalities</td>
</tr>
</tbody>
</table>

### Table 6.3: Users’ feedback on the difficulties while playing AugFrog

<table>
<thead>
<tr>
<th>User #</th>
<th>User Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>When the frog is crossing the street from the farthest point, it was still facing forward which was counter intuitive. So jumping on a car was hard.</td>
</tr>
<tr>
<td>2</td>
<td>None</td>
</tr>
<tr>
<td>3</td>
<td>The frog moves slowly in screen space</td>
</tr>
<tr>
<td></td>
<td>Traffic is bursty; sometimes very easy to cross.</td>
</tr>
<tr>
<td></td>
<td>It's hard to tell what lane the frog is in when far away</td>
</tr>
<tr>
<td>4</td>
<td>None</td>
</tr>
<tr>
<td>5</td>
<td>Finding lanes in the farther lane is difficult.</td>
</tr>
</tbody>
</table>

### Table 6.4: Users’ feedback on the difficulties while playing Human Pinball

<table>
<thead>
<tr>
<th>User #</th>
<th>User Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sometimes it is hard to see where the ball is before you get used to the game</td>
</tr>
<tr>
<td>2</td>
<td>Sometimes the ball becomes too small to see</td>
</tr>
<tr>
<td>3</td>
<td>The ball was often hard to track when obscured (which often happened close to the camera) or in the distance and tiny.</td>
</tr>
<tr>
<td>4</td>
<td>There was sometimes a gap of a few seconds between pressing z or / and the flipper moving. But generally the game performed very well</td>
</tr>
<tr>
<td>5</td>
<td>None</td>
</tr>
</tbody>
</table>
Users suggestions for the improvement of AugFrog and Human Pinball are as listed in tables 6.5 and 6.6.

Table 6.5: Users’ suggestions for the improvement of AugFrog

<table>
<thead>
<tr>
<th>User #</th>
<th>User Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduce a frontal view for the frog to make the direction in which it is moving more intuitive</td>
</tr>
<tr>
<td>2</td>
<td>None</td>
</tr>
</tbody>
</table>
| 3      | Add something to do once you reach the other side such as start over or come back  
|        | Add in sound effects                                                        |
|        | Speed up the frog                                                           |
|        | Use more challenging traffic                                                |
|        | Use a perspective close to viewing from overhead                            |
| 4      | None                                                                         |
| 5      | Include more distractions or other frog-eating animals on the other side of the road |

Table 6.6: Users’ suggestions for the improvement of Human Pinball

<table>
<thead>
<tr>
<th>User #</th>
<th>User Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Input how hard you want to hit the ball with the flippers, sometimes it goes very fast</td>
</tr>
<tr>
<td>2</td>
<td>When the ball hits a person, the person could say &quot;ouch&quot;.</td>
</tr>
<tr>
<td>3</td>
<td>Add a visual effect to the ball to aid with tracking (glow or trail)</td>
</tr>
<tr>
<td></td>
<td>Add sound effects to make it &quot;cooler&quot;</td>
</tr>
<tr>
<td></td>
<td>Make the ball motion smoother</td>
</tr>
<tr>
<td></td>
<td>Use a perspective closer to viewing from above</td>
</tr>
<tr>
<td>4</td>
<td>It was a good game</td>
</tr>
<tr>
<td>5</td>
<td>Include open doors, possibly make it more 3D. Also human reaction</td>
</tr>
<tr>
<td></td>
<td>when the ball hits them can make it more interesting, Different worlds etc.</td>
</tr>
</tbody>
</table>

6.2 Summary

In this section, we summarize the feedback provided by the users.

While playing AugFrog and Human Pinball, the users found it interesting to play in an environment where a game is integrated with scenes from real life and thus providing the ability for the user to interact with entities from the real world. In AugFrog, the frog’s
ability to jump onto a nearby vehicle seemed to catch everyone’s attention.

The difficulties users faced in Human Pinball were mainly due to the shrinking of the ball to a size which is not easily visible to the user and its high speed.

While playing AugFrog, users found it difficult to judge the lane in which the frog is present on the father end of the screen. This aspect was already known to us and is due to the roadside perspective of the video. As a result, the lanes nearer to the user are clearer than those that are farther away. One of the users found the frog’s lack of a front view non-intuitive which in turn resulted in difficulties in jumping onto a car. This shows that the change in frog position or lane was not enough of a cue to judge the direction of movement to the user.

Several users suggested the addition of sound effects and visual effects when the ball hits a human being, in order to improve the game of Human Pinball. One of the user suggested adopting a top-down perspective for game play.

Users of AugFrog suggested including sound effects, using a video with busier traffic, the inclusion of more objects to distract the frog and the addition of features that enable further game-play once the frog has completed crossing the road successfully to improve the game.
Chapter 7

Discussion and Summary

7.1 Discussion

Through the formative evaluation of AugFrog and Human Pinball as described in section 4.2, it has been possible to gain some general insights from observations of people in our lab playing the game.

*Virtual versus real agents and objects* - One assumption made in our approach to VGVs is that agents under the control of the user are always virtual; this appears to be a reasonable assumption given that users control agents via operators. A more interesting question is whether virtual objects should be added to the video environment, pushing a VGV in the direction of pure video games. We have begun to consider this, as discussed below.

7.1.1 Video quality

Given our goal of making VGVs constructable by end users, it is important to point out that AugFrog and Human Pinball work on a small set of videos we have produced ourselves or downloaded from the internet; they are of relatively high quality, rely on a fixed camera, and involve recognizable objects. Some amount of additional work would be needed to generalize the game even for the obvious possibility of a Web traffic cam. Our expectation, however, is that once the necessary software has been developed to extract essential information from a set of videos, that the videos would be usable with the logic of other video games.
7.1.2 Image processing limitations

Even state-of-the-art image processing algorithms can fail unexpectedly, and this is especially true for algorithms that have not been tuned to a specific domain, as indicated above. How good does an image processing algorithm have to be for it to work for a given VGV? Perhaps surprisingly, the answer is "It depends," and perfect performance is not always necessary.

One clear limitation in AugFrog is that vehicles are sometimes spuriously detected and occlude the frog, which results in its blinking out of existence for a frame or two. This reduces the realism of the game. The implementation also sometimes fails to identify collisions between the frog and a vehicle in the farther set of lanes. Because collisions interrupt the flow of the game, however, an occasional miss turns out to be tolerableviewed as a near escape by users. The reason for this is the issues in the implementation of the Hough transform for circles.

In Human Pinball, the only issue that has been seen during implementation is the fact that since the game has been implemented in a 2D environment, the height of humans and ball movement along the corridor (away from the user) is along the same Y-axis. This is a result of the lack of 3D support in OpenCV’s blob detection capabilities. The only possible solution to eliminate this problem would have been to ignore the height of each human; thus representing each blob as a straight line in the 2D co-ordinate system. But this solution has not been adopted since it would run the risk of the ball penetrating a human which is impossible in reality.

7.1.3 Modeling limitations

While it is reasonable to focus on modeling the objects in a domain such as highway traffic, in that this is what we can expect end users to consider relevant, the game of Frogger depends on other aspects of the environment that are only implicit in the model. One is that spaces between vehicles are more important than the vehicles themselves. Information about between-vehicle spaces can be derived from the models of vehicles, but it may not be obvious to an end user that this is important to consider. The second is that the tempo of a real environment is out of our control (leaving aside the possibility of modifying video playback speed, which would make the use of live video problematic). Much of the engagement and difficulty of playing a video game comes from how fast things are happening; in AugFrog
and Human Pinball, this is fixed by the environment rather than being under programmatic control. This might be addressed by the addition of virtual vehicles or virtual people, as suggested above, but this is an avenue we have not pursued.

A different modeling issue is posed by the question of what counts as an appropriate match between a model and a given video. The traffic video is only a partial match even for the limited model we have provided for Frogger, and we have implemented one interaction that reflects a transformation of the model rather than its original form.

### 7.1.4 Departures from standard game play

As Ishii et al. have observed during their work on *PingPongPlus* [3], the augmentation of even a well-known game can give rise to player behaviors that are not generally considered part of the basic game, and may be unexpected to the designers. This is the case in AugFrog as well. Because the tempo of AugFrog in this video differs from that of Frogger, it turns out to be more interesting to play in traffic (e.g., jumping on and off vehicles) than simply to reach the far side of the highway. In Human Pinball, targets i.e. people, are non stationary unlike those in Pinball; also, they can become stationary or change their directions or even move out-of-sight. In addition, the number of targets in Human Pinball keeps varying, due to the movement of humans (into shops or out of sight) in comparison to the fixed number of targets in the conventional game of Pinball. Such unpredictable events make the games more interesting. This is at least partly an issue of exploration on the part of the user: What happens when I do X? We believe that such surprises will occur for video game play in any sufficiently detailed video environment, thus increasing the randomness in game-play.

### 7.2 Conclusion

We end with the observation that while AugFrog and Human Pinball along with its associated concepts are not completely novel, we believe that they provide an interesting direction for a merging of virtual and real interaction, focusing not just on the physical structure of the respective environments but on procedural and interactive aspects as well.

In this research work, we have been successful in proving that it is viable to create a system that creates a playable VGV by integrating the logic and interactions involved in a video game with a pre-recorded video environment.
Although, there are numerous options available for open source image processing libraries, it has been difficult to find a mapping between the video in hand and an implementation of the desired image processing algorithm, which works accurately on it. The reasons for this is due to the limitations of the implementations of the algorithms in the image processing libraries which impose the requirement of domain knowledge and high video quality.

In summary, we have been able to identify several problems and their causes, thus discovering several areas for future exploration.

7.3 Future Work

Extending this current work onto live video would be interesting. This should be easily achievable due to the extensive support for live video capturing and processing provided by OpenCV. The only issue would be to undergo a preprocessing stage in order to identify and extract a frame from the video which consists of the elements which are considered as the background and thus identify the boundaries of the field of play. Alternatively, an efficient implementation of a background subtraction algorithm which is able to separate the background and foreground without generating noise or is not computationally intensive which works on any video, irrespective of the video quality should be identified. Thus a preprocessing stage can be circumvented.

VGVs can also be extended to be used for education or training purposes. For instance, a user can learn how to cross the road via making the virtual frog cross a road if AugFrog is implemented with a video of traffic at an intersection. Thus providing a more cost efficient learning environment than the Augmented Reality versions of similar applications, due to the usage of off-the-shelf components and standard PC equipment.

The modeling approach that we discussed in chapter 3, can be used to automatically generate code to create video games. This would require building a parser that can break down the representation based on Qualitative Process Theory. This should be then fed to a software module, which can automatically generate code based on the definition specified by the model, such as, create various objects and their attributes, describe processes in terms of event handlers and so on.
Bibliography


[34] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection.