Implementing Games on Pinball Machines

Daniel Wong Darren Earl Fred Zyda
Ryan Zink Sven Koenig Allen Pan
Selby Shlosberg Jaspreet Singh
Computer Science Department
University of Southern California
Los Angeles, California, USA
{wongdani,fzyda,skoenig}usc.edu
{allenpan,shlosber,jasprees}@usc.edu
{earl.darren,ryanzink}@gmail.com

Nathan Sturtevant
Computer Science Department
University of Alberta
Edmonton, Alberta, Canada
nathanst@cs.ualberta.ca

ABSTRACT

Almost no research has been done on designing pinball games although much research has been done on designing video games. We are interested in designing pinball games on pinball machines to teach computer science students about how to interface to mechanical systems in a fun and motivating way. Thus, we have developed a pinball machine interface between a PC and a recent Lord of the Rings pinball machine. We demonstrate that it is easy to innovate pinball games by designing and implementing Pinhorse, a pinball game that avoids some of the design problems of existing pinball games. For example, it features a true multi-player mode where each player directly influences the game of the other player within a limited amount of play time. This paper describes both our innovative pinball game and the hardware and software of our pinball machine interface that enables game designers to develop such pinball games on real pinball machines.

Categories and Subject Descriptors

K.8.0 [**Personal Computing**]: General—*Games*

General Terms

Design

Keywords

Game Architecture, Game Development, Hardware Interface, Pinball Machine, Software Interface, Teaching Computer Science

1. INTRODUCTION

The faculty members of the Department of Computer Science at the University of Southern California (USC) believe that teaching computer science hands-on via the development of games helps students to learn computer science. We have therefore created a Bachelor's Program in Computer Science (Games) and a Master's Program in Computer Science (Game Development), which not only provide students with all the necessary computer science knowledge and skills for working anywhere in industry or pursuing advanced degrees but also enable them to be immediately productive in the game industry [13]. As part of this overall effort, different faculty members explore different ideas. Almost no research has been done on designing pinball games although much research has been done on designing video games. We are interested in designing pinball games for two educational reasons:

- The **first reason** is the same as the reason behind letting students design video games [12]. Designing games can be used to teach many computer science topics and many computer science skills, including computational thinking skills, software engineering skills and programming skills. Designing games can also be used to teach a variety of skills that are not taught in traditional computer science classes, including creativity, design skills, artistic skills, problem-solving skills and teamwork skills, such as collaboration skills with non-computer scientists. Students are motivated to learn how to program games because they find games fun to play and are thus curious about how to create them, because they can play their games themselves and because they are driven by an element of friendly competition due to their desire to create the best game. There is evidence that teaching computer science via game development can help to increase student enrollment and retention in computer science, and the US needs to double the number of science and technology graduates by 2015 according to the July 2005 report of the TAP Forum [3].

- The **second reason** is different from the reason behind letting students design video games. The standard computer science education tends to teach students only about software but not about interfacing it to mechanical systems, such as electronics, signal generation, embedded systems, communication protocols, interface programming and real-time programming. However, this skill is important, for example, in

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FDG 2010, June 19-21, Monterey, CA, USA
Copyright 2010 ACM 978-1-60558-937-4/10/06 ...$10.00.

the context of robotics. Designing pinball games can be used for this purpose since pinball machines interface to the physical world. They contain actuators (in the form of motors and solenoids), sensors (in the form of switches) and visual outputs (in the form of lights and the dot-matrix display). Their game is determined by the input-output behavior of the computer, that is, what outputs the computer activates and when it activates them in response to its input-output history. Pinball machines are essentially robots but have a variety of advantages over mobile robots, namely that they are easier to maintain and that their low-level control is simpler and more robust since they provide very controlled environments for pinballs to move in (resulting in a motivational experience).

We have developed a pinball machine interface between a PC and a recent Lord of the Rings pinball machine, which enables students to implement pinball games on pinball machines. Pinball games have not substantially changed for far more than a decade [10]. The students of the small pilot CS499 class "Designing and Implementing Games on Pinball Machines" at USC (and co-authors of this paper) demonstrated that it is easy to innovate pinball games by designing and implementing Pinhorse on our first-generation pinball machine interface, a pinball game that avoids some of the design problems of existing pinball games. For example, it features a true multi-player mode where each player directly influences the game of the other player within a limited amount of play time. We first give an overview of the hardware and software of our pinball machine interface and then describe Pinhorse. Our pinball machine interface allows game designers not affiliated with the manufacturers of pinball machines to create pinball games and might allow consumers in the future to buy a new (cheap) pinball game for their pinball machine rather than having to buy another (expensive) pinball machine.

2. PINBALL MACHINE

We decided to work with a used solid-state Lord of the Rings (LOTR) pinball machine, see Figure 1 (left), because the layout of its playfield is flexible and not particularly theme-specific, see Figure 1 (center). Stern Pinball produced about 5,100 of these pinball machines, starting in 2003. Used LOTR pinball machines cost about \$3,000-\$3,500. We control the pinball machine via its I/O Power Driver Board, which was used by all Sega and Stern pinball machines with a WhiteStar or WhiteStar II board system (roughly from 1995 to 2004). Thus, we expect the pinball machine interface to be usable with a variety of pinball machines from Stern. The LOTR pinball machine consists of the following input and output devices, see Figure 1 (right):

- The **input devices** of the LOTR pinball machine consist of the 58 playfield switches and the 7 dedicated switches, which correspond to switches that humans interact with (such as the left and right flipper buttons). The LOTR pinball machine supports up to 64 playfield switches, arranged in an 8x8 switch matrix. The controller reads the playfield switches by software polling. It strobes each column and then reads the row signal after it has gone through RC filters (for noise filtering) and a comparator (for signal buffering) to create a stronger steady signal. The LOTR pinball machine supports up to 8 dedicated switches. The controller reads the dedicated switches directly.

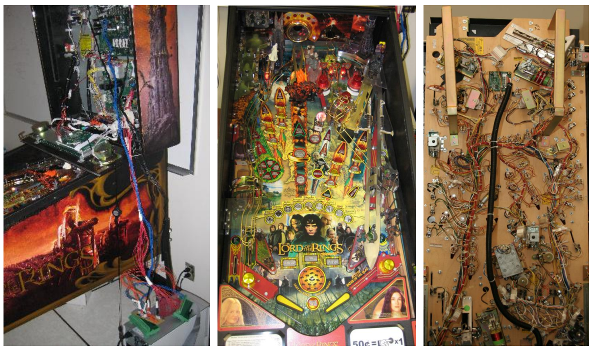
Figure 1: LOTR Pinball Machine

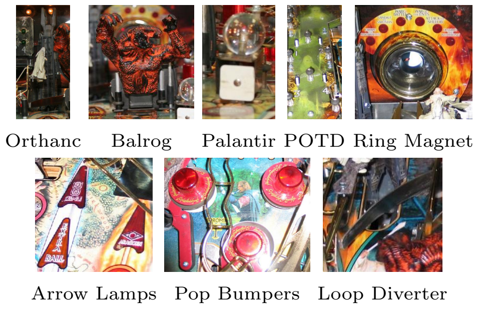

Figure 2: Important LOTR Playfield Parts

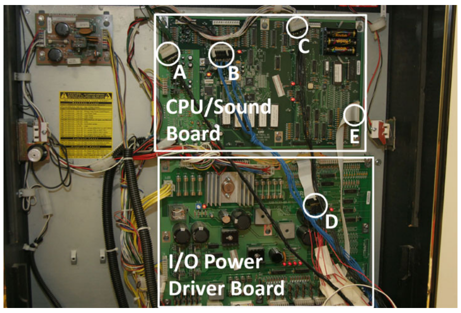

Figure 3: LOTR Backbox with Boards

- The **output devices** of a LOTR pinball machine consist of the lights (namely 80 lamps, 9 flash lamps and 19 LEDs) and 23 low and high current solenoids (including 2 slingshots, 3 vertical upkickers, 3 pop bumpers, 2 flippers, 1 loop diverter, 1 ring magnet and 1 Balrog motor), see Figure 2. They also consist of a pair of speakers and the dot-matrix display. The lamps are arranged in a 10x8 lamp matrix and need to be strobed, similar to the playfield switches, at approximately 1 ms to minimize flickering.

The LOTR pinball machine is controlled by the following components, see Figure 3:

- The **CPU/Sound Board** processes the switch signals and generates the control signals for the speakers, the

Display Controller Board and the I/O Power Driver Board. It includes the 8-bit 68B09E microprocessor, the CPU Game ROM and the Sound/Voice ROM.

- The **I/O Power Driver Board** drives the lights and solenoids. It includes registers that hold the status of each light and solenoid. The CPU/Sound Board sets the data of these registers. The I/O Power Driver Board then activates the lights and solenoids accordingly.

- The **Display Controller Board** controls the 128x32 plasma dot-matrix display (DMD). It includes the Image ROM. The CPU/Sound Board determines which image from the Image ROM to display. The Display Controller Board then retrieves the image and generates it on the DMD.

3. HARDWARE INTERFACE

One extreme of programming the LOTR pinball machine is to reprogram its ROMs. However, a reverse engineering effort is very difficult without any available documentation for the proprietary boards other than the LOTR manual. The other extreme of programming the LOTR pinball machine is to replace all boards. However, this is costly and time-consuming since there are more than 100 switches, lights and solenoids. We therefore decided to replace the CPU/Sound Board with a PC, which is used for running the software interface and pinball game. We use a Dell Outlet Inspiron 530 PC with an Intel Core 2 Quad Q6600 Kentsfield 2.4GHz processor and the default installation of OpenSuse 10.3 Linux. The PC costs $469, and Linux costs $0. We decided to interface the PC to the pinball machine via a small number of existing connectors, namely the connectors A-E in Figure 3, rather than soldering wires onto the existing boards, which would make it difficult to both make additional pinball machines programmable and transform them back into their original state (which means that they would lose retail value). We also decided to interface the PC to the DMD because we wanted to be able to generate images on the DMD on the fly rather than having to store them in the Image ROM. Finally, we decided to use the PC speakers for playing music tracks, sound effects and voice clips. Our first-generation hardware interface consisted of the following components:

- The **Intermediate Switch Detection Board** used passive analog circuits to filter and buffer the incoming switch signals similar to what the CPU/Sound Board did before. The parts of this custom-made board cost about $30.

- The **DMD Board** interfaced the PC to the DMD. It used a Parallax Propeller, an 8-core 32-bit microprocessor. The Parallax Propeller Demo board costs about $80.

- The **Digital I/O Board** interfaced the PC via the PCI bus to the I/O Power Driver Board and the Intermediate Switch Detection Board. It used a National Instruments PCI-6509, a high-current 96-channel 5V TTL/CMOS Digital I/O card. The PCI-6509 costs $299, and its cabling kit costs $259.

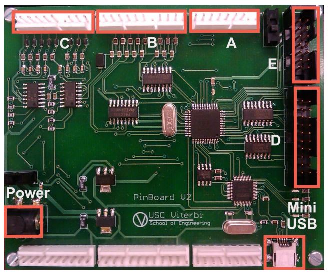

Figure 4: PinBoard

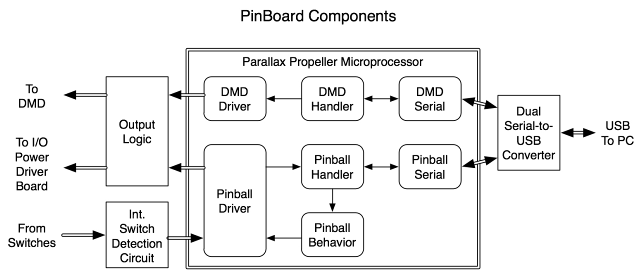

Figure 5: Hardware Interface

This first-generation hardware interface had some disadvantages: First, it cost about $670, which made it expensive to replicate. Second, it consisted of three boards, which made it difficult to replicate due to the complicated wiring. Third, the PC was responsible for polling the playfield switches and strobing the lights and solenoids but had trouble generating its signals with sufficient speed and accuracy, resulting in the lights flickering slightly and the strength of the solenoids not being completely consistent. The solenoids are controlled via pulse-width modulation. Their strength is directly proportional to the duty cycle of the signal. In order to accurately drive a solenoid, the PC needs to accurately generate a signal at the desired duty cycle, which is impossible with Linux due to its scheduling policies.

The second-generation hardware interface consists of only one custom-made printed circuit board, called PinBoard, see Figure 4. PinBoard communicates with the PC via a dual serial-to-USB converter that maintains two virtual serial ports that share a standard USB connection, enabling even laptops and mobile devices to control the pinball machine. Connectors A, B and C in Figure 4 connect PinBoard via custom-made cables to the switches, as shown in Figure 3, and are used to read their statuses. Connector D connects PinBoard via a ribbon cable to the I/O Power Driver Board and is used to control both the lights and solenoids. Connector E connects PinBoard via a ribbon cable to the DMD to control it. The total cost of the hardware interface (including the custom-made Molex connectors and header pins) is only about $150 plus the cost of the PC. Figure 5 shows the architecture of PinBoard, which uses a Parallax Propeller with eight independent cores. The Parallax Propeller runs the following seven tasks (one on each core) without the need for interrupts or complicated task scheduling, which allows it to generate its signals with sufficient speed and accuracy:

- The **DMD Serial Task** handles the low-level serial communication protocol between PinBoard and the

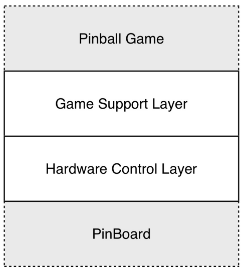

Figure 6: Software Interface Layers

PC with respect to the DMD.

- The **DMD Handler** processes the incoming commands and data from the PC.
- The **DMD Driver** generates the control signals for the DMD.
- The **Pinball Serial Task** handles the low-level serial communication protocol between PinBoard and the PC with respect to the switches, lights and solenoids.
- The **Pinball Handler** processes the incoming commands from the PC. The PC can read the statuses of the switches, change the statuses of the lights and solenoids and program the type, behavior and properties of the lights and solenoids.
- The **Pinball Behavior Task** determines the desired statuses of the lights and solenoids. Each light and solenoid has an associated type that determines its behavior. For example, the flippers require a longer high-power signal to kick them up and then a longer low-power signal to keep them up, while the pop bumpers require a short pulse of a high-power signal. The Pinball Behavior Task also performs safety checks to prevent fuses and transistors from blowing. It uses conservative timeouts for all solenoids. For example, the loop diverter automatically turns off after one minute, and the Balrog motor turns off immediately when Balrog is completely open or closed. It uses timeouts of half a second for the flash lamps and solenoids since they need to be active only for fractions of a second.
- The **Pinball Driver** generates the control signals with respect to the switches, lights and solenoids. For example, to control the lights and solenoids, it sets the data of the registers of the I/O Power Driver Board. To read the levels of the playfield switches, it strobes the columns of the switch matrix and reads the row signals via the intermediate switch detection circuit on PinBoard. It does not only read the levels of the switches (to determine which switches are active) but also uses edge detection to recognize changes in their levels (to determine which switches just became active).

4. SOFTWARE INTERFACE

The software interface allows a pinball game to control all aspects of the pinball machine. The software interface allows game designers and students to prototype pinball games rapidly by providing an almost complete abstraction of the hardware. All functionality of the software interface can be provided to the students to shield them from the hardware. Alternatively, some functionality can be omitted to expose them to the hardware by letting them implement the functionality themselves. The software interface consists of the following layers, see Figure 6:

- The **Hardware Control Layer** interfaces to PinBoard. A pinball game has to know the statuses of the switches, lights and solenoids, for example, whether a switch is closed or a lamp is on. The Hardware Control Layer thus maintains local representations of the statuses of the switches, lights and solenoids, both by reading the statuses of the switches from PinBoard hundreds of times per second and by remembering how the Game Support Layer changed the statuses of the lights and solenoids. It allows the Game Support Layer to read the local representations of the statuses of the switches, lights and solenoids and change the actual statuses of the lights and solenoids, which also updates their local representations.
- The **Game Support Layer** interfaces to the Hardware Control Layer. The Game Support Layer provides high-level API calls and additional functionality that allow game designers to develop pinball games easily. It is easy to use and extend.

  The **API calls** execute tested procedures that simplify the amount of knowledge and code required to program common input/output tasks needed in pinball games. Initially, the game designers had to program all of these tasks directly. The API calls now shield them from details of interfacing to the Hardware Control Layer and from having to write many lines of code to accomplish common tasks needed in pinball games, such as reading the current statuses of switches, setting the current statuses of the lights and solenoids, playing audio and generating and displaying DMD graphics. Table 7 shows examples of the API calls provided by the Game Support Layer.[1] For example, the initial game code (without API calls) for activating the right slingshot when a pinball triggers its playfield switch looked as follows:

```
double lastT = elapsedT;
int iteration = -1;
pb.switches.set(1);
while (pb.switches.get_status13()
       || pb.switches.get_status12())
{
  pb.sollmp.execute();
  pb.switches.execute();
  pb.dswitch.execute();
  gettimeofday(&current, NULL);
  elapsedT = difftime(current,start);
  if(elapsedT - lastT < 0.5)
    continue;
  iteration++;
  lastT = elapsedT;
  switch(iteration)
  {
```

[1]There are different API calls for different audio files since music tracks occupy their own channel and are thus treated differently from sounds effects and voice clips, which occupy the remaining 15 audio channels of SDL and can occur simultaneously.

Switches	Lamps	Solenoids	Audio	DMD
Is switch active?	Turn on lamp.	Turn on solenoid.	Play music track.	Clear buffer.
Is switch becoming active?	Turn off lamp.	Turn off solenoid.	Play sound effect.	Display buffer.
Is switch being released?	Set lamp status.	Set solenoid status.	Play voice clip.	Print text.
	Toggle lamp status.	Toggle solenoid status.	Stop music tracks.	
	Turn on all lamps.	Set maximum solenoid power.	Stop sound effects.	
	Turn off all lamps.		Stop all audio.	

Figure 7: Examples of Provided API Calls

```
    case 0:
      pb.sollmp.sola.setActivate5(1);
      pb.sollmp.solc.setActivate3(1);
      break;
    case 1:
      pb.sollmp.sola.sola5.setTimeOff(2,0);
      pb.sollmp.solb.setActivate3(1);
      pb.sollmp.solc.setActivate5(1);
      break;
    case 2:
      pb.sollmp.solb.setActivate4(1);
      break;
    case 3:
      pb.sollmp.solb.setActivate5(1);
      pb.sollmp.solc.setActivate4(1);
      pb.sollmp.solc.setActivate6(1);
    default:
      iteration = -1;
  }
}
```

The same game code with API calls is shorter and easier to understand:

```
ballStuck = true;
while (ballStuck)
{
  setSolenoid("Trough Up-Kicker", 1);
  setSolenoid("Left VUK", 1);
  setSolenoid("Top VUK", 1);
  setSolenoid("Right VUK", 1);
  setSolenoid("Sword Lock Release", 1);
  if (!(switchActive("Left VUK")
        && switchActive("Top VUK")
        && switchActive("Right VUK")
        && switchActive("Sword Lock Low")))
    ballStuck=false;
  usleep(3000000);
}
```

DMD operations benefit the most from API calls since SDL graphics operations can be unwieldy. For example, it takes more than ten lines of code to print a single character to the DMD without API calls. The same operation with API calls takes only two lines of code, namely one for printing the text to a buffer (for double buffering) and one for displaying the buffer on the DMD.

The **additional functionality** includes the delayed execution of functions. For example, a vertical up-kicker should be activated fractions of a second after a pinball triggers its playfield switch to allow the pinball to settle first. The additional functionality also includes the tracking of the pinball with a naive tracking algorithm that updates the position of the pinball based on the position of the most recently activated playfield switch.

The software interface is written in C/C++ and uses only a small number of external libraries (namely libserial for serial communication with PinBoard, SDL for audio and

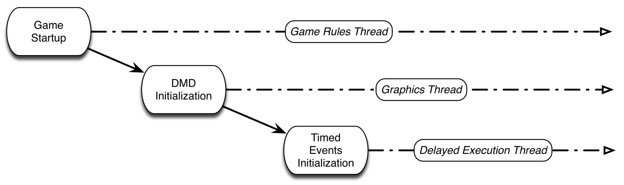

Figure 8: Thread Initialization and Execution

graphics and a lightweight XML parser) to achieve portability. Initially, information about the pinball machine and pinball game, such as the locations of switches and lamps on the playfield and the locations of audio files on the hard drive, had to be hard coded in the relevant code sections of the pinball game. It is now stored in an XML file, whose structure generally reflects the hierarchy and contents of the objects used in the software interface. The XML file allows game designers to configure the software interface easily since it makes it easy for them to store, find and change information about the pinball machine and pinball game and to maintain different sets of information for different pinball machines and games. For example, the initial game code (with hard coded information) looked as follows:

```
sfx[1]=Mix_LoadWAV("Sfx/star_destroy.ogg");
sfx[3]=Mix_LoadWAV("Sfx/grain_bonus.ogg");
voice[MADE]=Mix_LoadWAV("Sfx/voice/centerRamp.wav");
```

The same information in an XML file is easier to change:

```
<PinballBoard>
  <Resources>
    <SoundEffects>
      <SoundEffect path="Media/Sfx/grain_bonus.ogg" duration="1">
         GrainBonus
      </SoundEffect>
      <SoundEffect path="Media/Sfx/star_destroy.ogg" duration="1">
         StarDestroy
      </SoundEffect>
    </SoundEffects>
    <VoiceClips>
      <VoiceClip path="Media/Voice/centerRamp.ogg" duration="1">
         CenterRamp
      </VoiceClip>
    </VoiceClips>
  </Resources>
</PinballBoard>
```

The software interface is multi-threaded, see Figure 8. The first thread (called game-rules thread) runs the pinball game. The second thread (called graphics thread) generates the DMD graphics using double buffering and transfers it to PinBoard for display on the DMD, which guarantees a high refresh rate despite the large amount of processing required when drawing graphics, such as converting text to pixels. The third thread (called delayed execution thread) handles the delayed execution of functions (called timed events). It maintains a priority queue of functions and their event times

Mode	Property	Issue	Solution
single-player mode	predefined shots	unexciting for experts	use dynamic shots
multi-player mode	same as turn-based single-player mode	unexciting for experts	allow player interaction
	different play times for experts and novices	unexciting for notices	limit play time per player

Figure 9: Issues of Current Pinball Games

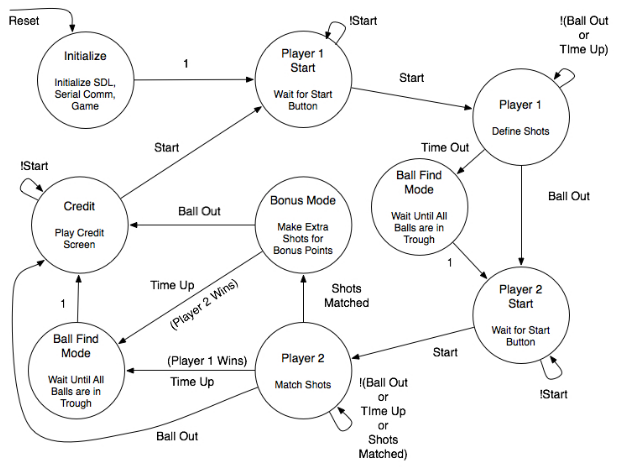

Figure 10: State Diagram of Pinhorse

and calls each function at its event time. Future versions of the software interface will likely use even more threads to guarantee small reaction times.

5. DESIGNING A NEW PINBALL GAME

Pinball machines provide strong design constraints on pinball games and need to be used in creative ways to create pinball games since pinball games are partly determined by the playfield of the pinball machine, which we do not want to alter physically. When we set off to design a new pinball game, we first had to understand what makes pinball games fun. We attended a meeting of the Orange County Pinball League, where we were able to play 15 different pinball machines. We also observed expert pinball players and asked them questions about their strategies. We learned that pinball games seem to follow a fairly strict formula where the players need to make a series of predefined shots to advance the game. This can make them unexciting for expert players since they quickly become monotone. In multi-player mode, pinball games keep a score for each player, with little to no change in game play. This can make them unexciting for nonexpert players since the expert players in the group can keep the pinball in play for long periods of time and the nonexpert players are then idle most of the time, see Table 9 for a summary.

This experience made us settle on three design goals: First, we wanted to design a pinball game where the sequence of shots required to win the game is determined during the game and can thus be different each time the pinball game is played. Second, we wanted to design a multi-player game where each player directly influences the game of the other player. Finally, we wanted to design a pinball game that limits the play time of each player.

To satisfy the three design goals and ensure that our pinball game would be different from existing pinball games, we designed a pinball game based on the concept behind Horse. Horse is a game played on a basketball court where the first play makes a shot that the second player must duplicate from the same position on the court. We adapted this concept to pinball by dividing the play into two roles: a shot establisher and a shot matcher. The first player completes a series of shots within a fixed amount of time with no restrictions on his play. His goal is to create a sequence of shots that is difficult to match either due to its length or the skill required to make the individual shots. Once his time is up, the flippers become disabled and the pinball drains. The second player then attempts to replicate this sequence of shots within the allotted time. If he is successful, he is allowed to complete additional shots with no restrictions on his play to improve his score, see Figure 10. A full game would eventually consist of several iterations with players switching roles and keeping running totals of their performance.

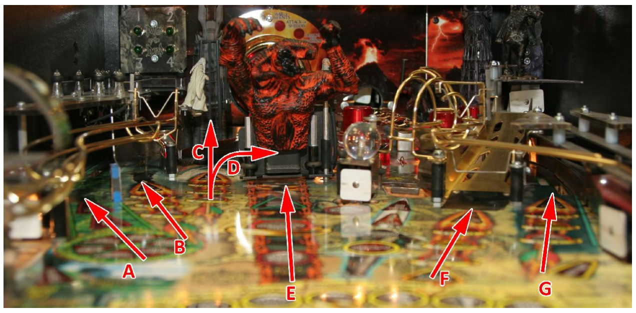

Figure 11: Shotmap of Pinhorse

We defined seven possible shots based on the physical layout of the playfield, namely the left orbit (A), middle orbit (D), right orbit (G), left ramp (B), center ramp (E), right ramp (F) and the Orthanc tower (C), see Figure 11. These are the shots that skillful players can make reliably. If the player hits the Palantir (chosen for its unique lighting characteristics and visibility), the loop diverter and Balrog open for a short amount of time. The loop diverter opens to make the Orthanc tower shot easier, and the Balrog opens to enable the center ramp shot. These dynamics allow skilled players to make substantially more difficult shot sequences. We quickly learned that replicating a sequence of shots is almost impossible for regular players without intervening shots and thus relaxed the rules to allow intermediate shots by the second player. For example, if the first player defines the shot sequence “left orbit” and “right ramp,” then the second player can replicate it with the shot sequence “left orbit,” “right orbit” and “right ramp.” We call the resulting pinball game Pinhorse.

6. IMPLEMENTING THE GAME

Shots consist of sequences (usually of length two) of playfield switch edges in quick succession, which allows Pinhorse to detect whether the shot was sufficiently strong to complete successfully and what the direction of the shot was. Some shots consist of more than one sequence of playfield switch edges, which allows the player to complete it in different ways. Pinhorse adds every playfield switch edge to a small buffer and then scans the buffer in an attempt to match a shot (shot recognition). For the first player, recognized shots are added to the end of a queue. For the second player, recognized shots are compared to the first shot in the queue. If they match, the first shot in the queue is removed (shot matching). Pinhorse knows that the second player has matched all shots of the first player once the queue is empty and then starts a bonus mode where the second player can make extra shots to increase his score.

Pinball games require both visual and audio cues to communicate the current game state to the players and create the right mood. Pinhorse uses the DMD to display the current player, the number of shots made by the first player or the number of shots still to be made by the second player, the next shot to be made by the second player and the remaining time. However, since the pinball action is fast-paced, Pinhorse makes extensive use of the lights to give visual cues to the players.

Pinhorse uses a variety of light effects. The software interface addresses the lights according to the architecture imposed by the I/O Power Driver Board. We therefore implemented a wrapper that allows Pinhorse to reason about lights in Cartesian coordinates. These coordinates are derived as the projections of the lights onto the playfield glass from the point of view of the player and thus loosely reflect the absolute positions of the lights. The lights are grouped hierarchically, for example, into circles, arcs and arrows. Lighting functions operate on these groups and can access the system time and maintain state to realize complex light patterns, such as rotating half circles, expanding rings, balls with time-dependent radii and strobing lights. Each light pattern has several configurable parameters, such as the velocity of progression, acceleration, start coordinates and duration. More than one light pattern can be active at any time.

Pinhorse defines about 20 groups of lights and uses light patterns in several ways. For example, Pinhorse strobes Palantir at an increasing rate after a Palantir hit to indicate the amount of time left before the loop diverter and Balrog close again. For the first player, Pinhorse confirms that a shot was established with a quick flash of light. The speed of a rotating half circle across the entire playfield indicates the amount of remaining time. For the second player, Pinhorse indicates the next shot to be made with a ball of light with time-dependent radius centered on the corresponding blinking arrow lamp to catch his attention. As with the first player, the speed of a rotating half circle of lights indicates the remaining time. However, the animation is restricted to a small group of lights near the flippers so as to not interfere with the indication of shots. In the bonus mode, all lights on the board are strobed to signify the accomplishment of the second player.

Pinhorse uses sound effects to inform the players of the outcomes of their actions. For example, Pinhorse plays voice recordings to reinforce which shot the second player is expected to make next, which greatly improved the player experience. Pinhorse also uses sound effects to encourage the second player to act quickly when his time runs out. Finally, Pinhorse plays background music during game play.

7. CURRENT STATUS / FUTURE WORK

Pinhorse is a proof of concept game that demonstrates what can be accomplished with the pinball machine interface. Pinhorse was intended to help us improve the hardware and software interface and demonstrate the feasibility of writing a pinball game within a semester. Pinhorse consists of about 680 lines of code, the helper classes for shot recognition, shot matching and light patterns consist of about 1,600 lines of code, and the software interface consists of about 2,300-25,00 lines of code. Almost all of this code can be reused to speed up the development of future pinball games and to increase their complexity. Pinhorse incorporates almost all of the playfield parts as well as the DMD and audio but needs to be developed further to reach the sophistication of existing pinball games. For example, the first player can define as many as 10 to 12 shots in the 60 seconds that we allotted for each round, which can be difficult for the second player to replicate. Future versions of Pinhorse could be made easier with a system of power-ups that the second player can earn with particular shots and that then increase his remaining time or turn off features of the pinball machine that randomize the movement of the pinball, such as the slingshots or pop bumpers.

8. COLLABORATIONS

We have recently made our pinball machine interface available to the University of Alberta (Canada) to enable their students to implement pinball games as well, for example as part of their award-winning game class [11], and allow for an unbiased evaluation. They have purchased their own LOTR pinball machine and currently are undertaking several projects that are expected to result in a close collaboration between USC and the University of Alberta:

- The **first project** was to build an event-based layer on top of the existing software interface and to structure the code to simplify the addition of new projects. The event-based layer reduces the need for pinball games to continually poll the software interface for state changes. Pinball games now install event handlers, and the software interface calls these event handlers when state changes occur. The event-based layer also provides the functionality to switch solenoids and lamps on and off after given periods of time. This project was successfully completed and simplifies the creation of pinball games greatly. For example, the base functionality of Pinhorse was re-implemented with fewer than 200 lines of code, most of which were dedicated to DMD operations.
- The **second project** is to adapt ScriptEase [7] to generate pinball games automatically from high-level game specification. ScriptEase is software whose original purpose was to generate stories for the role-playing game Neverwinter Nights. ScriptEase could provide an intuitive interface for non-programmers to design pinball games using common design patterns. This project is expected to start in Summer 2010.

9. RELATED WORK

There are only a small number of teaching and research efforts that use actual pinball machines. Actual pinball machines have, in research, been used to study hybrid system control [9] and develop and evaluate machine learning algorithms by an undergraduate student of Sven Koenig (one of the authors of this paper) at Georgia Institute of Technology (USA) in 1999. In teaching, they have been used to teach real-time and embedded systems as part of "Introduction to Embedded and Real-Time Programming" (CS160) at Brown University (USA) in Spring 2007, "Smart Product Design Laboratory" (ME218a) at Stanford University (USA) in 2007, "Designing with Microcontrollers" (EE476) at Cornell University (USA) in Spring 2007 and "Special Topics in Electrical Engineering: Pinball Machine Project" (ENEE 488Q) at the University of Maryland at College Park (USA) in Spring 1997. A similar effort existed at Brooklyn College (USA) around 1997 [5]. Pinball machines have also been used to teach signal and image processing as part of "Project Course in Signal Processing and Digital Communication" (EQ2430/EQ2440) at Kungliga Tekniska Högskolan (Sweden) in Spring 2004 and "Project: Pinball" (EOH 2004) of the ACM Special Interest Group for Computer Architecture at the University of Illinois at Urbana Champaign (USA) in Spring 2004. One of these efforts attempted to build a pinball machine from scratch, which allows them to customize the hardware for easier control. Other efforts used old electro-mechanical pinball machines, which are easier to control than newer solid-state pinball machines. The state of the art in controlling a pinball machine was as follows: Some efforts controlled the flippers by modifying the hardware [4]. Some tracked the ball via input from the playfield switches [9] or an overhead camera [8] [6]. The most sophisticated project so far mimicked the microcomputer-based control unit to read the switches and control the solenoids [2] [1]. We, on the other hand, provide a hardware and software interface that controls all aspects of an existing solid-state pinball machine (including the lights, solenoids, DMD and speaker) without needing to modify its hardware. Furthermore, we have used the hardware and software interface to program a complete (but simple) pinball game that makes use of all of these aspects.

10. CONCLUSIONS

The development of the pinball machine interface and Pinhorse continues. The pilot CS499 class used to build Pinhorse was different from the kind of classes that we envision to offer in the future since it was a hands-on project class used to debug the existing hardware interface, develop a first version of the software interface and a first pinball game from scratch, all in one semester without worked out class material. Thus, we were not able to formulate the learning objectives up front or evaluate the class afterwards. Therefore, it is future work to offer a more systematic class and evaluate it rigorously. Similarly, Pinhorse is only a proof of concept game and does not reach the level of sophistication of commercial games in terms of complexity or engagingness. Therefore, it is future work to develop more sophisticated games and evaluate them rigorously. However, we hope that this paper will raise awareness of the availability of our pinball machine interface as a testbed for creating games different from video games. More information can be found on our pinball project webpages at

idm-lab.org/pinball

together with an 11-minute YouTube video that demonstrates the features of Pinhorse and was viewed more than 3,000 times in its first three months.

11. ACKNOWLEDGMENTS

Figure 2 uses parts of images by Cassidy Thomas, obtained from the Internet Pinball Database at www.ipdb.org. The game music of Pinhorse is a mix of "Fearless Flight" from Nullsleep, "March of the Nucleotides" from Bit Shifter and other pieces from Castlevania. The pinball project was supported in part by a grant from the USC Fund for Innovative Undergraduate Teaching and the National Science Foundation under Grant No. 0113881. Daniel Wong was supported by the USC Undergraduate Merit Research Program and the Rose Hills Foundation. We thank the Orange County Pinball League for their hospitality.

12. REFERENCES

[1] J. Bork. Controlling a pinball machine using Linux. *Linux Journal*, 139, 2005.

[2] J. Bork. Reverse engineering a microcomputer-based control unit. Master's thesis, Industrial Technology, Bowling Green State University, Bowling Green (Ohio), 2005.

[3] Business Roundtable. Tapping America's potential: The education for innovation initiative, 7 2005.

[4] D. Clark. An inexpensive realtime testbed - the pinball player project. In *Proceedings of the IEEE Workshop on Real-Time Applications*, pages 86–88, 1994.

[5] D. Clark. Progress toward an inexpensive real-time testbed: The pinball player project. In *Proceedings of the Real-Time Educational: Second Workshop*, pages 72–79, 1997.

[6] R. Cohen. Designing an experimental pinball wizard. *The Electronic System Design Magazine*, 19, 1989.

[7] M. Cutumisu, C. Onuczko, M. McNaughton, T. Roy, J. Schaeffer, A. Schumacher, J. Siegel, D. Szafron, K. Waugh, M. Carbonaro, H. Duff, and S. Gillis. ScriptEase: A generative/adaptive programming paradigm for game scripting. *Science of Computer Programming*, 67(1):32–55, 2007.

[8] S. Gustafsson, J. Munoz, S. Norell, D. Real, and Y. Xiao. Smart pinball project - final report. Technical report, Skolan för Elektro- och Systemteknik, Royal Institute of Technology, Stockholm (Sweden), 2004.

[9] G. Lichtenberg and J. Neidig. An example of hybrid systems control: The pinball machine. Technical Report 2003.13, Lehrstuhl für Automatisierungstechnik and Prozessinformatik, Ruhr-Universität Bochum, Bochum (Germany), 2003.

[10] M. Rossignoli. *The Complete Pinball Book*. Schiffer, 1999.

[11] N. Sturtevant, H. Hoover, J. Schaeffer, S. Gouglas, M. Bowling, F. Southey, M. Bouchard, and G. Zabaneh. Multidisciplinary students and instructors: A second-year games course. In *ACM Technical Symposium on Computer Science Education*, pages 383–387, 2008.

[12] M. Zyda. Creating a science of games. *Communications of the ACM*, 50(7):26–29, 2007.

[13] M. Zyda and S. Koenig. Teaching artificial intelligence playfully. In *Proceedings of the AAAI-08 Education Colloquium*, pages 90–95, 2008.