# Programming Pinball Machines for Fun and Education*

Daniel Wong Darren Earl Fred Zyda Sven Koenig
University of Southern California
Computer Science Department
wongdani@usc.edu earl.darren@gmail.com fzyda@usc.edu skoenig@usc.edu

## ABSTRACT

The University of Southern California has recently created a Bachelor's Program in Computer Science (Games) and a Master's Program in Computer Science (Game Development). As part of this effort, we are currently working on creating a motivational project class on programming pinball machines, where the students interface a PC to an existing pinball machine and then re-program the pinball machine with a pinball game developed by them. In Summer and Fall 2008, we performed a feasibility study with the objective to develop the computer interface between a PC and a recent Lord of the Rings pinball machine, the software to drive the interface, libraries that provide abstractions of this interface, and a program that uses these libraries to implement an engaging pinball game. In this paper, we describe the results of this successful feasibility study. As far as we know, this is the first time that anyone has managed to control an existing pinball machine completely and re-program it with a new complete (but simple) pinball game.

## Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education, Curriculum*; K.8.0 [**Personal Computing**]: General—*Games*; B.4.m [**Hardware**]: Input/Output and Data Communications—*Miscellaneous*

## General Terms

*Figure 3 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 Null Space, "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. Daniel Wong did his research in Summer 2008 as a summer scholar in the undergraduate merit research program, funded by the Viterbi School of Engineering, and received a scholarship from the Rose Hill Foundation in Fall 2008. We thank the Orange County Pinball League for their hospitality.

**Figure 1: Lord of the Rings Pinball Machine**

Computer Interface, Game Development, Pinball Class, Pinball Machines, Teaching Computer Science

## 1. INTRODUCTION

The faculty members of the Department of Computer Science at the University of Southern California believe that teaching computer science hands on via the development of games helps them to motivate students to learn computer science. They 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-development industry [19]. In this paper, we describe our experience with a feasibility study for a motivational project class on programming pinball machines, where the students interface a PC to an existing pinball machine and then re-program the pinball machine with a pinball game developed by them. Solid-state pinball machines basically consist of a computer that reads the switches and controls the lights, solenoids, speakers and the dot-matrix display [13]. A pinball 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. The pinball class will cover how to build a hardware interface to the pinball machine, how to design and implement the software interface for the low and high level real-time control of the pinball machine (including how to best represent the rules of the pinball game), and how to write the rules of the pinball game.

In Summer and Fall 2008, we performed a feasibility study

with a Lord of the Rings pinball machine, see Figure 1 (left). Our objective was to develop the hardware interface between a PC and the pinball machine, the software interface to drive the hardware interface, libraries that provide abstractions of this software interface, and a program that uses these libraries to implement an engaging pinball game. In this paper, we describe the results of this successful feasibility study. As far as we know, this is the first time that anyone has managed to control an existing pinball machine completely and re-program it with a new complete (but simple) pinball game.

## 2. THE PINBALL CLASS

The pinball class will be a semester-long capstone class for students that have already taken some traditional computer science classes and now have to use a variety of knowledge and skills that they have acquired in these classes to solve a larger problem, namely to develop a way of programming a pinball machine easily and to create a pinball game that is playable, engaging and entertaining. Fundamentally, the pinball class will be a design class. Some of the design problems are from computer science and thus look familiar to computer science students, for example, designing software for writing the rules of the pinball game. Other design problems are not from computer science and thus unfamiliar to computer science students, for example, how to design an engaging pinball game. The students need to solve the following technical problems as part of the pinball class:

**How to build and program a hardware interface to the pinball machine?** We have decided that we will provide students with the hardware interface for the initial offerings of the pinball class since many computer science majors might not have practical experience with TTL/CMOS logic and simple electronics.

**How to implement software for the low and high level real-time control of the pinball machine?** The students need to let lights blink, that is, switch them on and off at the correct times. They need to form complex light patterns of light with the many available lights. They need to detect patterns of switch closures and determine the mode based on them. They need to synchronize the lights, sounds, speech, music and information displayed on the dot-matrix display, which is important because the various output modalities take different amounts of time and compete for the available output resources. For example, a sound effect and an animation on the dot-matrix display take different amounts of time and are long enough that the situation might change, which might require a different sound effect and a different animation. On one hand, it is not satisfactory to stop the old sound effect and animation before they are finished. On the other hand, it is not satisfactory either to delay the new sound effect and animation long beyond the event that prompted them. There are different ways how the necessary hierarchical control structure can be implemented, for example, with a blackboard structure on which concurrent processes operate. Thus, the students need to experiment with different ways of implementing the hierarchical control structure to achieve real-time control, assign scarce resources, manage the complexity of the software and make it easily adaptable to changing the specifications of the pinball game (including how to best represent the rules of the pinball game), as will frequently arise during game development. These tasks require understanding of concepts from operating systems, concurrent processes and rule-based expert systems (such as black-board systems) as well as software engineering. Furthermore, many computer science majors might not have practical experience with playing sounds, speech and music.

**How to write the rules of the pinball game?** The students need to have a good understanding of what makes pinball games engaging. For example, pinball players look for variability in game play via different modes but also the opportunity to make multiple shots and work on multiple objectives, switching among them as opportunities present themselves. Thus, the rules of pinball games need to get designed carefully, including the shots, modes and scoring. The user interface also needs to get designed carefully, including the lights, sounds, speech, music and information displayed on the dot-matrix display, both to communicate the state of the pinball game to the user (which requires user-interface skills that some students will be familiar with) and set an appropriate mood for the current state (which requires artistic skills typically not taught to computer science students).

The students will form teams that include students with majors different from computer science, such as electrical engineering and arts, since the task requires a variety of expertise and talent. Each team will determine its own path to develop and program a pinball game. The pinball class will not prescribe any path to a good solution. In fact, there is no material available on the internet about how to program pinball machines and create engaging pinball games. Thus, the teams can try out ideas, make mistakes and learn from them, similar to problem-based learning [15, 1] and learning by design [11]. They will be assisted in two ways. First, the students will read papers on related topics, such as how to form project teams, how to design operating systems for embedded applications, how to design black-board systems, how to program concurrent processes, what makes pinball games engaging and how use lights, sounds, speech, and music to set an appropriate mood. Thus, the readings will cover not only topics from computer science but a variety of other disciplines as well. Second, the students will use pin-up sessions to foster ideas by bringing groups together for brainstorming and to share their work with others and hear their feedback [9]. In several iterations of research and reflection cycles, the students will brainstorm about ideas, try them out, and report on their experience, which forces them to explain their ideas to other students and justify them [11].

## 3. ADVANTAGES OF A PINBALL CLASS

Traditional computer science classes usually cover individual topics in hardware and software, such as computer architecture, operating systems, artificial intelligence and software engineering. Traditional computer science classes typically emphasize individual work, resulting in disjoint knowledge with an insufficient ability to solve complex problems. Finally, traditional computer science classes often present the students with homework that requires them to use the material that was taught immediately preceeding the homework, which means that the general approach to the solution is

known in advance. The pinball class will reinforce the material covered in traditional computer science classes but will also give students a more holistic view of computer science. It will teach or reinforce a variety of skills, including hardware skills, computational thinking skills, software engineering skills and programming skills. It will also 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 (including collaboration and communication skills). The students, for example, will learn to work as part of heterogeneous teams and respect students with different expertise. All of these skills are important job skills that are seldomly taught at universities.

A pinball machine is a game. Traditional game development classes have advantages similar to the pinball class [6]. However, pinball machines have the advantage over other games that they cover aspects of hardware and robotics in addition to aspects of software. A pinball machine is essentially also a robot, given that it interfaces to the physical world. Traditional robotics classes cover software and hardware. However, pinball machines have the advantage over robots that they are cheaper, much easier to maintain and, if necessary, can even be serviced by readily available pinball technicians for a small fee. Their low level control is simple and the resulting behavior thus robust, resulting in a motivational experience. Also, the students can concentrate on developing the high-level behavior, which will allow them to write the software from scratch in one semester. At the same time, programming pinball machines is not trivial since it requires knowledge of different areas of computer science and the size of the resulting code is substantial and thus needs to get managed carefully.

## 4. THE PINBALL PROJECT

There is a substantial overhead in creating a pinball class since it requires both hardware and software development. In Summer and Fall 2008, we performed a feasibility study with one faculty advisor (Sven Koenig) and three students of very different background who were interested in programming pinball machines: Daniel Wong was an undergraduate student in electrical engineering and computer science and responsible for the computer and software interface; Darren Earl was a Master's student in computer science and responsible for game programming; and Fred Zyda was a Ph.d. student in computer science and also responsible for game programming. Our objective was to develop the hardware interface between a PC and a pinball game, the software interface to drive the hardware interface, libraries that provide abstractions of this software interface, and a program that uses these libraries to implement a engaging pinball game. Daniel created the hardware and software interface in Summer 2008. Darren and Fred designed and implemented the pinball game as part of "Designing and Implementing Games on Pinball Machines" (CSCI499) at the University of Southern California (USC) in Fall 2008, while Daniel refined the hardware and software interface. Daniel worked on the hardware and software interface for approximately 200 hours in Summer 2008 and 75 hours in Fall 2008, split roughly half and half between the hardware and software interface. Darren and Fred worked on the pinball game for approximately 3 hours a week during Fall 2008. The completed pinball game was demonstrated at the biannual demo day of the GamePipe Laboratory in December 2008 to students, faculty members and representatives from the game industry.

**Figure 2: Playfield and Underside**


**Figure 3: Important Playfield Parts**

## 5. THE PINBALL MACHINE

We decided to work with a used solid-state Lord of the Rings (LOTR) pinball machine from Stern Pinball of Chicago, currently the only manufacturer of pinball machines. Stern Pinball produced about 5,100 of these pinball machines, starting in 2003. Used LOTR pinball machines cost about $3000-$3500. The layout of their playfield is flexible and not particularly theme-specific, see Figure 2 (left) for the layout and Figure 3 for some of the important playfield parts. LOTR pinball machines are highly rated for playability and need to be used in creative ways to create engaging pinball games since we do not want to alter the playfield physically. 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 hardware interface to the pinball machine and software to be usable with a variety of pinball machines from Stern.

The **input devices** of the LOTR pinball machine consist of the 58 playfield switches, see Figure 2 (right), and the 7 ded-

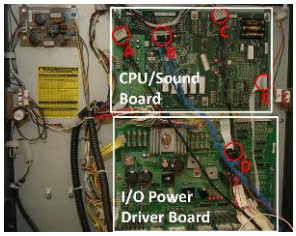


**Figure 4: CPU/Sound and I/O Power Driver Boards**

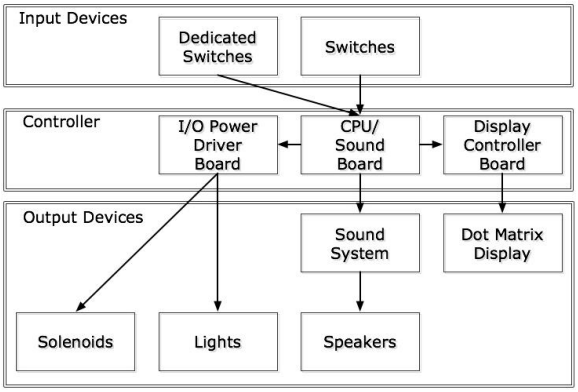


**Figure 5: Original Hardware Architecture**

icated 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 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. 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 (right). They also consist of a pair of speakers and the dot-matrix display. The lamps are arranged in a 10x8 matrix and need to be strobed, similar to the switches, at approximately 1ms to minimize flickering. The LOTR pinball machine is controlled by the following components, see Figures 4 and 5:

The **CPU/Sound Board** processes the input 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 output devices. It includes registers that hold the status of each output device. The CPU/Sound Board sets the data of these registers. The I/O Power Driver Board then activates the output devices accordingly.

The **Display Controller Board** controls the 128x32

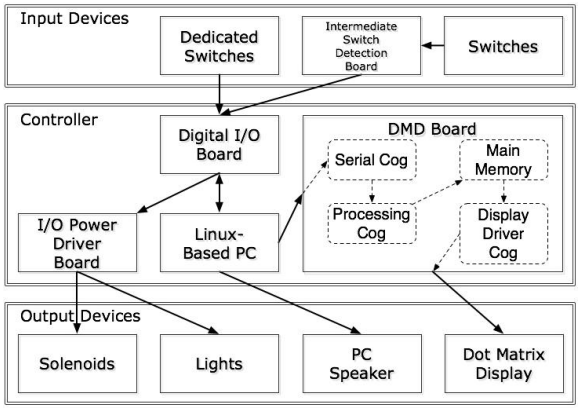


**Figure 6: New Hardware Architecture**

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.

## 6. THE 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 more than 100 input and output devices. We therefore decided to replace the CPU/Sound Board with a PC. We decided to interface the PC to the pinball machine via a small number of existing connectors, namely the connectors A-E in Figure 4, 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 our own Display Controller Board 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 sounds, speech and music. Thus, the pinball game can play arbitrary sound files with the high-level commands provided by SDL. The LOTR pinball machine is thus controlled by the following new components, see Figures 1 (right) and 6:

The **PC** is used for programming and running the game code. 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, except that we recompiled the kernel with support for a high-resolution timer and installed libraries for the Digital I/O Board (NIDAQmx driver) and serial communication (libserial). The PC costs $469, and Linux costs $0.

The **DMD Board** interfaces the PC to the DMD. We use a Parallax Propeller, an 8-core 32-bit microprocessor. Each of its cores (called cogs) is capable of running independently of the others. We use one cog to drive the DMD at about 70Hz, the recommended refresh rate. We use one cog to

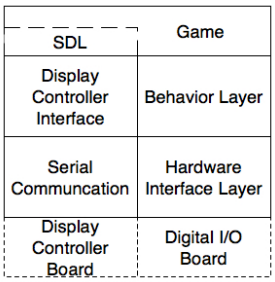


**Figure 7: New Software Architecture**

handle the serial connection with the PC at about 115.2k Baud. Finally, we use one cog to process the data received from the PC. When data is received from the PC, the Serial Cog hands it over to the Processing Cog, which writes the data to memory, forming an image. The Display Driver Cog then reads the image and drives the DMD to display it. The images are double buffered to create a steady image on the DMD. The Parallax Propeller Demo board costs about $80.

The **Intermediate Switch Detection Board** filters and buffers the incoming switch signals similar to what the CPU/Sound Board did before. The parts of this custom-made board cost about $30.

The **Digital I/O Board** interfaces the PC to the I/O Power Driver Board and the Intermediate Switch Detection Board. We use a National Instruments PCI-6509, a high-current 96-channel 5V TTL/CMOS Digital I/O card. We chose the PCI-6509 for its ample ports, its support for 5V TTL/CMOS levels which makes it compatible with components on the I/O Power Driver Board, and its ability to source or sink up to 24mA of current which enables it to drive the data bus of the I/O Power Driver Board. Its 96 channels are organized into 12 ports of 8 channels each. To read the playfield switches from the Intermediate Switch Detection Board, we use one port as output to strobe the column and one port as input to read the row signal. To read the dedicated switches from the Intermediate Switch Detection Board, we use one port as input. To set the data of the registers of the I/O Power Driver Board, we use two ports as outputs, one for the address of a register and one for the data to be written into that register. The PCI-6509 costs $299, and its cabling kit costs $259.

All cables connect to the Intermediate Switch Detection Board to keep the cabling organized. The custom-made Molex connectors and header pins cost about $40. The total cost of all additions is approximately $708 plus the cost of the PC. We expect to be able to reduce this cost to approximately $300 plus the cost of the PC with an embedded system currently in development.

## 7. THE SOFTWARE INTERFACE

The software interface allows a pinball game to control all aspects of the pinball machine via function calls. It is written in C/C++ and provides the interface between the pinball game and the Digital I/O Board via the NIDAQmx driver and the pinball game and the DMD Board via serial communication (a USB connector as a virtual COM port), see Figure 7:

The **Display Controller Interface** interfaces the pinball game to the DMD Board. It repeatedly grabs the content of the screen and transfers it to the DMD Board. Thus, the pinball game can create arbitrary graphics on the fly with the high-level commands provided by SDL. The Display Controller Interface can display approximately 15 frames per second on the DMD, which is sufficient for full motion animation.

The **Digital I/O Board Interface** interfaces the pinball game to the Digital I/O Board. It consists of two layers:

The **Hardware Interface Layer** communicates with the Digital I/O Board via the NIDAQmx driver to read the switches and drive the lights and solenoids. For example, it converts the desired status of the lights and solenoids into byte representation, places the address of the register onto the address bus, the data of the register onto the data bus and finally clocks the register to save the data.

The **Behavior Layer** uses the Hardware Interface Layer to implement behaviors for every output device. Pinball games need to contain a main loop that repeatedly calls the Main Loop Function of the Behavior Layer to read the current status of the switches and set the current status of the lights and solenoids. The Main Loop Function detects both levels and edges of switches by monitoring the history of the switch status. It activates the lights and solenoids according to behaviors that the pinball game can set via function calls.

The Behavior Layer implements three different behaviors: The standard behavior for lights and solenoids is to turn on (= activate) immediately and then turn off (= deactivate) after a set period of time. The second behavior is for those solenoids that need to stay on for extended periods of time and thus are typically individually fused, namely the flippers, the ring magnet, the loop diverter and the Balrog motor. Their behavior is identical to the standard behavior but they require pulse-width modulation to limit the amount of current going through them, which reduces the amount of heat and allows them to stay on longer. Thus, the behavior needs to activate and deactivate them repeated while they are turned on. The third behavior is for the Balrog, which acts as a door that opens and closes. Its behavior needs to set the relay to the desired turning direction and then turn on the Balrog motor.

The Behavior Layer performs safety checks to prevent fuses and transistors from blowing. These safety checks are the result of trial and error. They implement conservative timeouts for all solenoids driven by pulse-width modulation. 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. The safety checks implement timeouts of half a second for the lights and all other solenoids since they need to be active only for fractions of a second.

## 8. CURRENT STATUS / FUTURE WORK

We encountered the following difficulties during the project: First, we had the LOTR manual available but not much further documentation. Thus, we had to develop parts of the hardware and software interface via trial and error, which was especially problematic for the I/O Power Driver Board, DMD and LEDs. Second, we had to develop the safety checks via trial and error, blowing many fuses and some transistors in the process. Third, the hardware and software interface were more complicated than necessary due to our design decision to interface the PC to the I/O Power Driver Board via a small number of existing connectors. For example, we needed to design the Intermediate Switch Detection Board to replicate part of the original CPU/Sound Board, and we needed to strobe the playfield switches and lamps. The playfield switches cannot be strobed too quickly (otherwise the data will not have time to propagate through the wires), and the lamps cannot be strobed too slowly (otherwise a watchdog timer on the I/O Power Driver Board resets the system). Fourth, the PC needs to run fast both to strobe the lamps sufficiently fast (to prevent them from flickering and the watchdog timer from triggering) and to control the solenoids with precision. Fifth, the software interface suffers from slight inconsistencies in timing due to our design decision not to use an operating system with a real-time kernel since the system load then determines how often the software interface gets called in the main loop, which determines the flickering of the lamps and the precision of control of the solenoids. We mitigated this effect by recompiling the Linux kernel with support for a high-resolution timer with an accuracy of 1ns and gave the pinball game high priority so that other processes would interrupt it less.

The software interface is now fully functional, with three minor issues: First, the software interface is currently able to display only two colors on the DMD but is already designed to use software modulation to produce three levels of grayscale. Second, the software interface does not perform all necessary safety checks to guard against unreasonable game programmers. For example, the flow of current that results from activating all solenoids at once can blow fuses and potentially damage the I/O Power Board. Third, the software interface still suffers from minor inconsistencies in timing: The lamps flicker subtly (although this is barely noticeable); the ring magnet is not always able to catch the pinball and throw it backwards; and the force applied by the vertical upkickers is not completely consistent. We are now working on creating an embedded system to produce the control signals for the pinball machine, which will solve the inconsistencies in timing, lower the total cost by replacing the Digital I/O Board and reduce the number of cables required by handling all data transfer through serial communication.

## 9. DESIGNING A NEW PINBALL GAME

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 (at least one of whose players was ranked in the top 20 in 2008 according to pinballrankings.com), 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

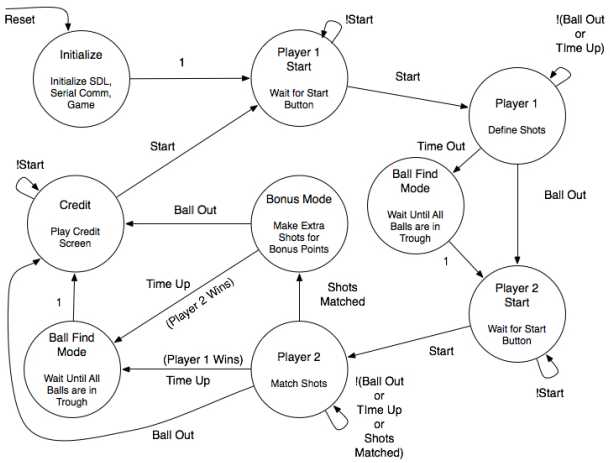


**Figure 8: State Diagram of Pinhorse**

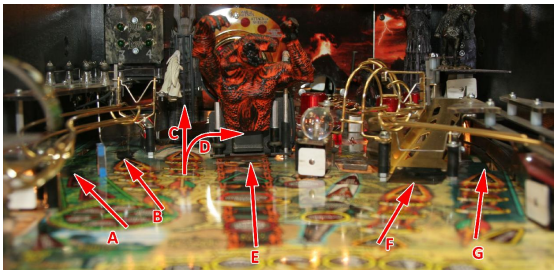


**Figure 9: Shotmap of Pinhorse**

shots to advance the game. This can make them unexciting for expert players since they quickly become monotone. In multiplayer 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.

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 multiplayer game where each player directly influences the game of the other player. Finally, we wanted to design a pinball game that limits the playtime of each player.

To satisfy the three design goals and ensure that our pinball game would be different from the original pinball game of LOTR pinball machines, 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 8. A full game would eventually consist of several iterations with players switching roles and keeping running totals of their performance.

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 9. 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.

## 10. IMPLEMENTING THE GAME

Shots consist of sequences (usually of length two) of 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 switch edges, which allows the player to complete it in different ways. Pinhorse adds every 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 their score.

Pinball games require both visual and audio cues to communicate the current state of the game 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 player.

Pinhorse uses a variety of lighting 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 by 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 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.

## 11. CURRENT STATUS / FUTURE WORK

Pinhorse was intended to help us improve the computer and software interface and demonstrate the feasibility of writing a pinball game within a semester. The Pinhorse program consists of about 680 lines of code, the helper classes for shot recognition, shot matching and light patterns consist of about 1600 lines of code, and the software interface consists of about 2500 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 works well, with three minor issues: First, Pinhorse occasionally fails to detect shots due to switch edge-detection errors and adjustment problems. For example, the pinball can sometimes squeeze by the upper switch of the right orbit without triggering it, which could be prevented only by adjusting the switch position. Second, pinball games enter a ball-find mode when the pinball gets stuck somewhere and no switches trigger. Pinhorse, however, has very short rounds and thus enters the ball-find mode only when the pinball is missing at the end of the round. A stuck pinball thus ends the turn of a player. Finally, 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. Pinball Horse could be made easier

with a system of power-ups that the second player can earn with particular shots and that then increase their remaining time or turn off features of the pinball machine that randomize the movement of the pinball, such as the slingshots or pop bumpers.

## 12. RELATED WORK / CONCLUSIONS

Several universities have recently introduced game development classes, see for example the issue of the Communications of the ACM on "Creating a Science of Games" [18], the proceedings of the First to Third Annual Microsoft Academic Days Conference on Game Development in Computer Science Education and the proceedings of the ACM Technical Symposium on Computer Science Education, including [14]. Game development classes typically develop only software that does not interface to the physical world, such as real-time strategy games.

**Simulations of pinball machines** have, in research, been used to develop and evaluate machine learning algorithms [10, 17, 16]. In teaching, they have been used to teach artificial intelligence as part of "Artificial Intelligence" (CS6361) at Georgia Institute of Technology (USA) in Winter 1999 taught by Sven Koenig (one of the authors of this paper).

**Actual pinball machines** have, in research, been used to study hybrid system control [12] and to 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 [12] or an overhead camera [8, 7]. The most sophisticated project so far mimicked the microcomputer-based control unit to read the switches and control the solenoids [3, 2]. 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. Thus, our project extends the start of the art substantially. Additional information and a video of Pinhorse, our pinball game, in action can be found on our webpages at **idm-lab.org/pinball**.

## 13. REFERENCES


[1] H. Barrows. *How to Design a Problem-Based Curriculum for the Preclinical Years*. Springer, 1985.

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

[3] J. Bork. Reverse engineering a microcomputer-based control unit. Master's thesis, Industrial Technology, Bowling Green State University, Bowling Green (Ohio), 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] D. Cliburn. Games across the curriculum: Can we quantify their effectiveness? (birds of a feather session). In *ACM Technical Symposium on Computer Science Education*, 2007.

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

[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] J. Holbrook, B. Fasse, and J. Gray. Creating a classroom culture and promoting transfer with 'launcher units'. In *American Educational Research Association*, 2001.

[10] M. Johnson. Algorithms for pinball simulation, ball tracking and learning flipper control. Master's thesis, Department of Electrical Engineering and Computer Science, Massachussetts Institute of Technology, Boston (Massachussetts), 1993.

[11] J. Kolodner, D. Crismond, J. Gray, J. Holbrook, and S. Puntambekar. Learning by design: From theory to practice. In *Proceedings of the International Conference of the Learning Sciences*, pages 223–229, 1998.

[12] 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.

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

[14] 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*, 2008.

[15] S. Williams. Putting case-based instruction into context: Examples from legal and medical education. *Journal of the Learning Sciences*, 2(4):367–427, 1992.

[16] N. Winstead. Some explorations in reinforcement learning techniques applied to the problem of learning to play pinball. In *Proceedings of the AAAI-03 Workshop on Entertainment and AI/A-Life*, pages 1–5, 1996.

[17] N. Winstead and A. Christiansen. Pinball: Planning and learning in a dynamic real-time environment. In *AAAI-94 Fall Symposium on Control of the Physical World by Intelligent Agents*, pages 153–157, 1994.

[18] M. Zyda. Creating a science of games. *Communications of the ACM*, 50(7), 2007.

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