ETC PRESS

Everything You Wanted to Know

(and more) About the Jam-O-Drum

A Study in Project Management

Cycle 1 Fall 2001

FOREWORD

This manual was created to serve as the primary source for Jam-O-Drum documentation. However, it has a greater purpose in that it may be used as a case study and guide for the creation and development of other software-based entertainment experiences. The wide range of disciplines and materials used in the creation of a Jam-O-Drum experience afford a sort of diverse benchmark that may supplement many production ventures. The manual is structured around a particular Jam-O-Drum project known as ***Musica***. The strategies and tips contained within are the amalgamation of several years of project management study and experience, coupled with the lessons learned in this project cycle. I hope this will save the reader countless man-hours of pedestrian troubleshooting and allow the most important aspect of the project, the experience, to flourish.

- DV

ACKNOWLEDGEMENTS

The ***Musica*** team would like to thank the many people who helped us stumble through the project step-by-step. Much ado to Wil Paredes and Cliff Forlines for helping to set up the software environment; Dennis Cosgrove, Jason Pratt and the rest of Stage 3 for their continual support of Alice; Ben Carter for helping to assemble the Jam-O-Drum base; Randy Hsiao and the Jam-O-World team for contacts; Donnie Antonini for his assistance with the finer points of Alice; Mike Rankin for endless 3D Studio guruship; and of course Frank Garvey, our advisor, for helping us etch out our concept, keeping us on track, and assorted electrical wizardry. Last, but not least, we would like to thank Janeen, Don, and Randy for their unrelenting support and feedback.

-- The ***Musica*** Project Group

HOW TO USE THIS DOCUMENT

This document is divided into three parts and a series of appendices, each with a purpose and chronological placement to guide the reader through a Jam-O-Drum cycle.

Part one deals presents the cycle from a project management viewpoint. The Jam-O-Drum is introduced and the experience concept is established. Step-by-step techniques guide the reader through the critical first days of the cycle and into the development of the project. Concepts such as the demographic, division of responsibility, and software engineering form the skeleton of the development process. Next the option of legacy development is discussed followed by an inventory list for the experience. At the end of part one the reader is challenged to adopt a unique perspective during the project cycle.

Part two delves into the technical details of the perennial committees. Practice and theory are covered for hardware, software, modeling/painting, animation, music and user testing. Each chapter offers both a look at Jam-O-Drum experience fabrication in general, and also the particulars of the example experience, ***Musica***.

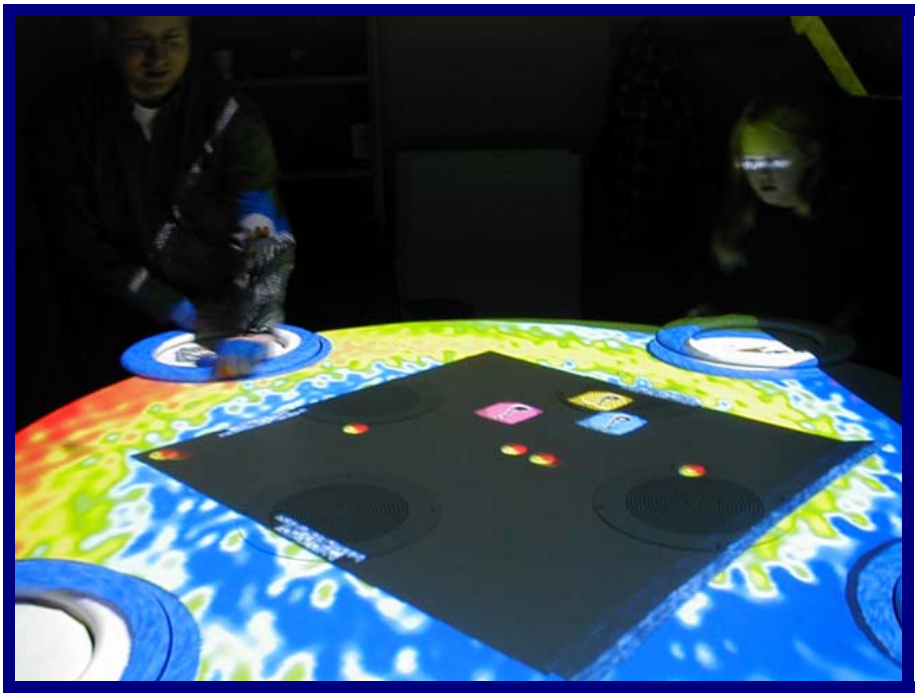
Part three encourages the reader to look beyond the cycle's duration and campaigns for a further dedication to project continuity in the form of documentation and accountability. The Jam-O-Drum as a commercial product is also examined in depth, providing an exhaustive list of the materials and labor that go into producing a Jam-O-Drum experience. With this information it is easily possible to pitch the Jam-O-Drum for sale to potential vendors.

The appendices cover a selection of the digital artifacts from the accompanying CD including screenshots, contact information, recommended reading, troubleshooting and Jam-O-Drum diagrams and schematics.

This document is intended to be all encompassing and touch on virtually every facet for the Jam-O-Drum phenomenon. The editor welcomes any comments or corrections.

TABLE OF CONTENTS

PART I: ELEMENTARY PROJECT MANAGEMENT



1. SO YOU'VE DECIDED TO CREATE A JAM-O-DRUM EXPERIENCE

1.1. What is the Jam-O-Drum?

The Jam-O-Drum (JOD) is a large, multiuser, input/output device designed to encourage collaboration. The JOD is a round table 6'4" in diameter divided into four quadrants by user stations. At each station is a MIDI drumpad, a turntable-like ring, and a speaker. Guests provide input to the Jam-O-Drum by hitting the drum pads with their hands, and spinning the turntables to either the left or the right. A large mirror is hung over the table and receives a signal from a high contrast/resolution projector. The software application is displayed on the surface of the Jam-O-Drum from the light reflected off of the mirror. Guests may play the game on the table together while making music emanate from the four speakers.

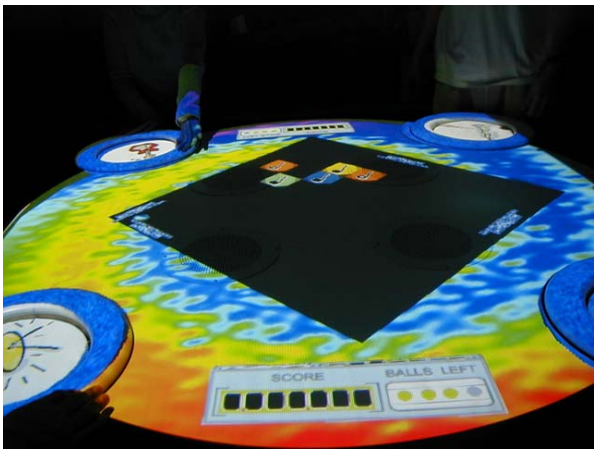


Figure 1. Jam-O-Drum 3.0.

1.2. The Experience Concept

The purpose of the Jam-O-Drum is to provide a collaborative multimodal form of communication and expression. The common demographic for Jam-O-Drum experiences is school children, though frequently the visceral appeal in the Jam-O-Drum attracts people of all ages. Guests may collaborate to play a simple video game while enabling them with the direct power to create music, sounds, and colorful graphic animations. It is imperative that all Jam-O-Drum teams keep this in mind when

designing an experience. The rich, visual and aural cues received from the hitting and rotating of the drums parts provide a vibrant, powerful sense of stimulation.

1.3. What Is Musica?

Musica is a collaborative experience for the Jam-O-Drum targeted at children of age seven or older and centered around basic musical notation. One to four guests play a musical game inspired by the arcade classic Arkanoid. Each player begins with a paddle in the shape of a ½ rest symbol. Players deflect the brightly-colored balls with the paddles via rotation of the turntables in an attempt to break patterned blocks containing musical notes and symbols. Each block when broken will add its note to a growing musical sequence that may be played when a player strikes a drumpad. Upon successful completion of the level, the players will be rewarded with a new set of balls and a prerecorded musical sequence.

Figure 2. Musica, a Jam-O-Drum Experience.

Musica conveniently entails just about every major tenet of software engineering and project management. The experience is ultimately intended for installation at the Children's Museum in Pittsburgh, but a final product couldn't be developed in the first cycle. Fortunately, though, the project will present a playable demo ready for user-testing and revision, and leaves off at a good stopping point for handoff to the next team. Throughout the remainder of this text all examples and references will be serviced by the ***Musica*** project and its first cycle.

1.4. Seven Weeks or Fourteen?

Probably the most fundamental factor in deciding what kind of experience you're going to create for the Jam-O-Drum is your cycle length. Creating a professional, ready-for-

sale application in seven weeks is daunting at best, but hopefully with this document (and all the hardware working before you start), you can pull it off. For a fourteen week cycle, this becomes much easier as a fully-functional game can be created from concept to testing in seven weeks, and the second half of the cycle can involve scores of neat extras for setting up the environment and adding to the experience. This way when it's complete you will have a robust and user-friendly production. Also, if you are considering rewriting the rendering engine and/or input code for the JOD, this would most likely necessitate a fourteen-week cycle (see Chapter 4, Legacy Code: Pro and Con).

2. PROJECT MANAGEMENT

2.1. Yes, This Should Come First

Entertainment Technology ***is*** exciting. The feeling of getting a new project, a new team, a new goal, a new product…it's breathtaking. Picture yourself at your first project meeting. One of the first things you (and everyone else) are feeling is that tempting chorus of "Concept, Creation, Celebration." It says, "Let's get going!" But it's a siren song. It's so easy to sit around and brainstorm of great things to do. "Maybe it could be like this. Oh, yea, then we could add this to go along with so and so…". The conversation dances on and on in inspired, ethereal circles. After about twenty or thirty minutes of this scattered bliss, the joyful mood is broken. The one guy in the back sitting back in his chair clears his throat and says, "That's all well and good, but is it feasible given the resources we have?" And there it is, the honeymoon is over, and it's all downhill from there.

But it doesn't have to be like that! The unhindered delight of creative exploration ***can*** find harmony with the cold, unfeeling regime of pragmatism. And you don't even need to pick a token bad guy to drain the life out of every idea you have (most of the time).

Quite frequently managers are portrayed in the media as pointy-haired idiots that don't know a keyboard from a two-by-four. That's personnel management, however. Project management, on the other hand, is mainly good planning and recognizing your resources. Every member on the team should be self-managed and assertive, yet there is a need for a hierarchy of responsibility and division of labor across the board. By simply being candid, questioning everything, and always expecting a setback, you can maximize your worth to your project and your group. As a wise man once said, C.S. stands for nothing more than Common Sense.

Throughout the rest of this chapter, we'll go over the basics of project management that every person should know. This includes some of the methods team leaders need, and the tools individuals can use to survey and map out how the time will be spent during the cycle.

2.2. The Five Things No One Thinks Will Happen (But Always Do)

1. No one finishes early.

People will unconsciously alter their level of effort and production to expand the task duration to its limit. Whether it's from procrastinating, nitpicking, or stalling during the learning curve, all but the most motivated of zealots will always finish assigned tasks on time or late.

2. Everything takes longer than it should.

No matter how terrific a team of hackers you may have, things will run over. Projects like the Jam-O-Drum have many critical points of failure because they involve so many systems. Electrical components break, hardware doesn't play with software, basic image manipulation is finicky, and the experience may just turn out to be no fun. Whether the Jam-O-Drum is physically complete and set up when you receive it or not, there is a lot of conversion going on between a guest striking a drum pad and an unstable Windows PC trying to keep track of game physics and about six sounds at once. One of the very first milestones that must be set, and met with total success, is mastery of the tools. As well as one thinks they have a grasp on how the system operates, the learning won't stop until the project is over. For this reason, it's imperative to provide more time than one would think is necessary when planning the acclimation of the materials; and tests and demonstrations of the mastery are needed to back up these claims. Otherwise a lot of time will be wasted in development still trying to learn the system.

3. Assumptions make for a fast trip to nowhere.

This principle is linked with the last, because it's always easy to believe one is in control of their domain. The hard truth is you're never totally in control, and to believe otherwise is what burns time and produces rejected artifacts. Never underestimate how foreign the system can seem. Remember, it was created by a group of people who have an entirely different way of thinking than you, in addition to a different skill set. It will take a lot of documentation, a lot of talking and a lot of questioning even the most mundane of previous decisions.

4. Requirements will change.

The strongest canon in software engineering is that things will go wrong, and requirements will change. The client may alter his idea of where the project is going, the venue/environment may undergo some drastic modification, or things may just not work, prompting a stern paring down of the objectives and direction. Hardware is discontinued, software is unsupported, and new versions just don't mesh like the old ones. For all these reasons, perform risk analysis and rapid prototype.

5. Anything less than a ridiculous level of communication will lead to ruin.

This is the most crucial of all things taken for granted. In times where development and revision is on an accelerated pace (usually at the end of the cycle), the project is most prone to fall apart. Some people are making revisions almost constantly, others may be lost as to their role in the production during crunch-time, and of course everything either gets done twice or not at all. Anytime a change to the project is made, remember the five W's: who made the change, what was done, where was it made, when did it occur/will it be finished, and why was it performed. This sort of email should be sent out, in bulleted list format, every time someone closes up shop for a session. And check in the files!

It's also critical that tradeoffs are agreed on by a majority of the group, and that everyone is aware of them. Features will be cut, sample quality may be sacrificed in the interest of space, and personal testing may reveal a fundamental change is needed in the way routine is handled. These are all things everyone has to know, regardless of title and background. Appreciating your teammates' skills and their viewpoint will undoubtedly produce higher quality work.

2.3. Your Concept, Your Demographic

Creating an entertainment experience is a lot like having a dinner party. The way you present yourself, your house, and your meal is tightly interwoven with who's coming over. This can start from either end. Perhaps you feel like having a swanky, highbrow European dinner that lasts for hours, rife with all the accoutrements of style. Or maybe it's just chili, tacos, and football. On the other hand, you may have in mind to invite all of your prominent business partners, or just the guys at the bowling alley. Either way, either the concept or the demographic gives you a kick-start on how to define the other.

For ***Musica***, we decided early on we wanted to make an experience for children. This was motivated heavily by the fact that the Jam-O-Drum had been installed in a number of museums prior to our cycle. This need not be an overbearing precedent. The Jam-O-Drum could be used in high school to motivate interest and participation in the arts, for collaborative public performance, or even music theory instruction. In our case, we knew that since our demographic was elementary and middle school children, we had to provide a concept that was visually and aurally rich, as well as easy to pick-up, with very a visceral entertainment appeal. Also we decided eventually we'd like to set up the drum in a children's museum that would have a high turnover and a steady flow of traffic, the experience would have to allow anyone to start playing at any time, and the entire experience shouldn't consist of more than a few simple stages.

Since the Jam-O-Drum appeals to such a wide audience, it matters less about which area you choose to start with. If you have a great concept, then it's not hard to fit a demographic to it and thusly define your expectations as to the level of detail and depth the experience will go into. Conversely, it's quite easy to create a concept that embodies an underlying theme and goal for a particular audience. The key point in this step is to pick one or the other, immerse yourself in it, and then justify both your concept and your demographic by a multitude of relationships. (Kids like playing, kids like learning. Making music is playing, teaching music is learning.)

2.4. The Mission Statement

The crux of an arched bridge is the keystone. The keystone absorbs the brunt of the stress in the structure, and holds everything else in place. Just like bridges, projects need keystones. The experience mission statement gives strength and resolve to the project. Whenever there is doubt in the project, repeat the mission statement.

Actors define their objectives with an infinitive. Corporations often adopt the same strategy for their mission statements. It should be strong, compelling, and include the demographic and the goal. [Our mission statement is] To provide a rich, colorful, entertaining experience for schoolchildren while familiarizing them with the basic symbols and concepts of musical notation and theory. Pick a battle cry you can rally around, you'll need it later if morale sinks.

2.5. Committee Structure

After the group has decided on the concept and the demographic, the next pivotal task is getting to know your project team. For the most part people will have an idea of what areas they'd like to work with. This desire will stem from the skills one already has and those looking to be developed. For the project to succeed, and people to have fun (the former is actually quite dependent on the latter), there needs to be a healthy balance on all required committees of rookies and veterans. A committee of expert programmers will most likely produce a tight and efficient solution to the software problem, but a mix of masters and amateurs will enable both to grow from the experience, in communication and problem-solving skills, but also in devotion to the project. People by nature like to be challenged, but not smothered or overwhelmed in the face of rival skill. Most importantly, every facet of the experience needs to be serviced by a committee with a well-equipped team. If no one in the project team has the necessary skills to develop a critical aspect of the experience, then perhaps it is in the best interests of the project and the group to reexamine the scope of the cycle.

Distribution of labor and responsibility is quintessential to giving individuals enough breathing room to concentrate on their tasks. Ultimately, one person is responsible for a particular part of the experience, and the leadership at that level must be unfaltering.

The Jam-O-Drum is much like a video-game experience, but with custom hardware and collaborative intent. With this in mind, the ***Musica*** team of five people developed the following hierarchy:

Committee	Members	Role
Hardware	5	Purchasing, assembly, integration, support of all tangible materials: the JOD, drum pads, drum module, amplifier, mixer, computer, reflecting mirror, etc.
Software	3	Mastery, development, documentation and support of legacy and new code.
Modeling and Painting	3	Creation and support of all models, textures, backgrounds, static game graphics.
Animation	3	Creation and support of all animations and visual effects.

Music	3	Creation and support of all musical sequences, samples, and sound effects.
User Testing	5	Development of user test scenarios, obtaining test subjects, recording and analysis of test data.

Table 1. Musica Committee Breakdown.

Of course, each committee was also responsible for the mastery of all relevant tools, both hardware and software. Concept was engineered by the entire group during the first few weeks of the cycle. Storyboards and concept art were produced by the majority of the team to supplement this. The User Testing committee was not actually utilized during the first ***Musica*** cycle because of time constraints, but it is one of the most important aspects of developing the experience, and was left as such for the next cycle.

2.6. You've Been Elected

2.6.1. Many Hats

Entertainment projects require a lot of interdisciplinary work. Teams are going to be small and the parallelism in the work is relatively thick. As such, you're going to wear a lot of hats during your cycle. You'll probably serve on half the committees in one way or another, and for one of them chances are you'll be a chair. When volunteering (or being selected) for chair, it's best that your leadership role be in your strongest suit. If no one is adroit in one particular area, it will take the person with the most background or most willingness to learn it quickly. Being a committee chair is not something to be taken lightly. Responsibility for the success of the committee's entire aspect lies on the chair's shoulders. Being said, chair is not for the weak of heart or introverted.

2.6.2. Responsibilities

Besides preferably being one of the most skilled members of the committee, the chair has to carry on a number of organizational tasks to keep the committee on task, efficient, and operating at its peak level. It's in the chair's duties to:

- **schedule** all committee meetings
- **plan** the development cycle for the committee

- **allocate** tasks and distribute the workload evenly to the committee members
- **monitor** that the committee's progress remains on schedule and true to the mission statement
- **appoint** a committee scribe to take notes at all committee meetings
- **ensure** documentation is maintained and all committee productions are archived properly

2.6.3. Intangible Requirements

Aside from the executive tasks of a chair, there are more subtle obligations as well. A chair must be a leader, and act as such. Delegation and policy must be dispatched with authority, and deadlines must be enforced. The committee members must be driven, motivated and inspired by the chair. It's the chair's responsibility to push the team to excel and demand performance. If there is not strong leadership, a committee will stall and wander. It's imperative that the committee's goals, the timeline, and contribution are all clearly drawn out and reaffirmed periodically by the chair. Exceptional people only do exceptional things when motivated to do so.

2.7. Planning Your Cycle / Software Engineering

2.7.1. The Need for Engineering

Writing software is a science, designing software is engineering. Just like civil engineering, the process may be implemented by anyone with some technical know-how, but without proper training is a recipe for disaster. You wouldn't drive across a bridge built by someone who simply knew how to mix concrete and use a blowtorch, would you?

Software development is a field harder than any other of its kind, namely because there aren't any tried and true methods for its process. There is no underlying physical science for programming computers other than it's only a matter of time until they (and their programs) fail, and most fail a lot sooner than you'd like. To make matters worse, your project isn't simply an exercise in software development. It's a debacle-in-waiting of project management, and interdisciplinary project management at that. All at the same time, you have people constructing and assembling hardware, spending hundreds of dollars on materials and services, crafting models and animations, writing music, creating a theme and concept, and of course writing hundreds of lines of fragmented,

inefficient, and unintelligible code. Oh, and you have seven weeks to do it. There never existed a project that cried louder for software engineering process.

Now that your palms have grown sweaty and the dark, ominous thunderhead of failure is rumbling in the distance, you are prepared to listen to reason. You **will** use strict method to engineer your experience, because wandering from procedure for just one day can set you back weeks. Fortunately for you, the path to successful project management involves very simple concepts and actions. Unfortunately for you, it's so numbingly simple that you're likely to forget it, or brush it aside in the name of short-term brute strength.

Leadership and authority must be bold and unflagging. Deadlines must be met constantly, with justification reinforced at every stage. Communication has to occur like breathing, not a decision may go unannounced or unjustified to the team. Risks must be analyzed constantly, and a backup plan must exist for any possible threat. Prototypes will be generated frequently, and tested often internally and externally. Every step of the cycle must be mapped out and responsibility allocated so no one person may sit idle waiting on something else. But most importantly, every microscopic interaction with the project must be viewed objectively, in perspective with regard to the rest of the cycle, and handled with the utmost common sense. These are the commandments of engineering your experience.

2.7.2. Getting Status

There needs to be bracketing in all areas of your project. Committee chairs report to advisors, committee members report to chairs. The committee chair must be totally aware of each of his members' status at any one time: what they have done, what they're currently working on, what they have left to do; also, what resources are being utilized, what the critical elements are for the tasks' completion, and what obstacles the members' may encounter. There are a lot of balls to juggle, but with constant affirmation and electronic record of the committee's progress, it barely becomes manageable. Every director/cast member relationship like these must adhere to an unyielding process and be able to present all the incident effects at a moment's notice. Only through rigorous bi-directional communication can failure and waste be weeded out.

It may also be helpful to have executive committee chair meetings weekly as well. By exchanging status from an aspect-oriented point-of-view, the chairs may better decide where to allocate resources and which team-members may have their duties' priorities downgraded (or upgraded) in one particular area or another to keep all of the project's development on the same page. This also serves as a forum for chairs to analyze in what areas team members (including themselves) are most productive and where a rebalancing of responsibility may increase overall productivity.

2.7.3. Risk Management

Risk analysis is the art of brainstorming failure. By studying all the possible (and virtually impossible) pitfalls your committee may encounter, the team greatly increases its chances for success. A good way to start is to write a rough timeline of your project's evolution along the top of a whiteboard. Concentrate on the elements needed for the experience's completion for a moment and then brainstorm as many possible things going wrong, epic or infinitesimal, and write them on the board, with arrows tracing the disaster to the point(s) on the timeline where it could occur. A half an hour of this should produce several whiteboards of tragedy. After assessing each of these risks individually, the processes of risk reduction and risk management may begin. Pretend each potential problem is a time bomb. Even if you can't think of a direct solution to diffuse it, there are a myriad of paths to take to lessening its likeliness of happening or impact.

With the density of this material being generated, it would be worthwhile to record and catalog all of your risks and prevention methods in tabular form. Also, your risks must be reexamined periodically, for new problems can spring up as the project evolves and begins to branch down different paths. Each week's worth of work will produce new problems and hopefully solutions, the key is to get to them before they cause your project to stall.

2.7.4. The Spiral Process

The Spiral Process is a risk-based software development method for rapidly producing prototypes and performing analysis of the production in a cyclic manner. Cycles move from risk analysis to requirements to design to implementation/testing. As the process moves onward, the phases get longer and the materials produced become more and more complicated.

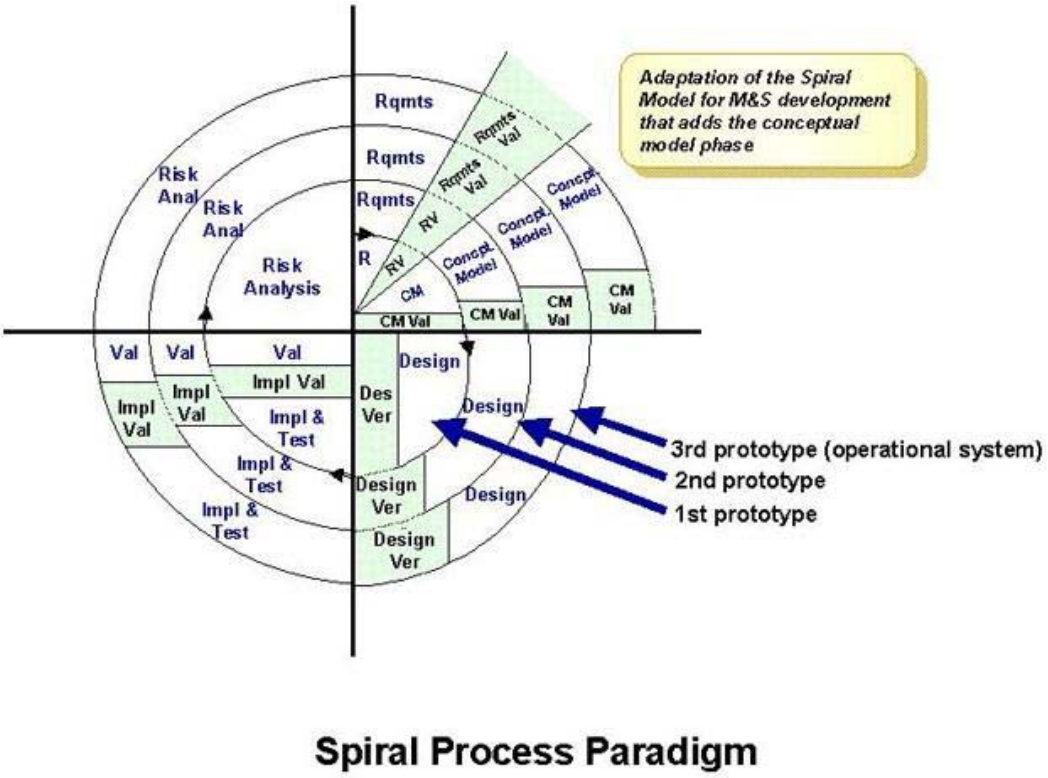


Figure 3. Spiral Process Paradigm with Conceptual Model.

By following this path of recursive, risk-based development for your cycle technical and conceptual anomalies are more likely to be found and dispatched earlier, producing more reliable and robust artifacts.

2.7.5. The Beauty of MS Project

MS Project is an electronic godsend and you using it to your advantage will manufacture statistical wizardry not even the staunchest autistic could produce. MS Project is a tool for production management. It keeps track of resources, team members, schedules, tasks, critical paths, and that's just for starters. The way it works is your project is centered around a timeline, a beginning and an end. In between you populate the calendar with action items for events as broad as user testing to tasks as small as assembling a document. Subtasks may be nested within super tasks and phases of the project. All of these may then be assigned to team members or committees. Dependencies may also be built into your burgeoning time-pool of man-hours thus creating critical paths for risk management.

One of the most effective and easy features to use is the PERT chart. This graphical representation of your cycle extends tasks into horizontal bars as time marches on. The dependencies and evolutions in your project will be made visible, and individual team members' names can be tagged for easy reference. To wrap it all up you can print out a long table view of your timeline and tack it up on your wall, so all may see of your march towards success.

MS Project is available for installation on *randon* and strongly advised to be a tool in each team member's closet for self-management and communication.

2.7.6. Scheduling Milestones

Milestones are a way of defining the battles you will have to succeed in before winning the war of your cycle. It is prudent to lay them weekly or bi-weekly, depending on your meeting period with your advisor. They are ubertasks that will represent areas of your project that will be unequivocally completed to keep the project on track. These may be iconified in your MS Project file to help visualize the core victories of your experience's growth. Some examples include mastery of the tools, first functional prototype deployed for testing, and all modeling and painting complete.

Beneath the canopy of milestones, it is often beneficial to recursively define a set of sub-milestones and "inchpebbles". These are the small steps taken towards completion of your milestone. The more detailed and precise you can make your inchpebbles, the more likely you will meet your milestone on time because there is an understanding of the justification and process to translating between one phase of the cycle to the other.

2.7.7. Assigning Tasks

Assigning tasks is something that's done frequently but rarely properly. The secret is in documentation and accountability. Whenever a task is assigned, regardless of its importance, the designated scribe must record the action item in strong, inflexible words and attach the bullet-item to a name and a firm deadline. The relevant committee must also agree on the feasibility of the task so that excuses along the line of "It was too hard" will be nullified. In the aftermath of the assigned duration, the task must be completed without fail. If a task is not met, the problem must be analyzed by the group to see why there were complications and how to remedy them, be they practical or personal. Strict task assignment and completion is a terrific way to boost morale

because of its orderly sense of a definitive challenge and following completion. Without clear-cut task assignment, vital requirements of the project will fade and wither, eroding the constitution of the project.

2.8. A Career's Worth of Study

This chapter has not even begun to illuminate the subjects of project management, and more importantly software engineering. However, it has hopefully opened your eyes to the fact there is a lot of process that exists to help produce markedly better results with fewer resources and time. It is strongly recommended that any aspirant of project direction/creation, entertainment or otherwise, take at least one course in project management and another on software engineering. An excellent reference for the philosophy and research of software engineering may be found at the Carnegie Mellon Software Engineering Institute (http://www.sei.cmu.edu).

3. THE PLAYERS

3.1. The A-Team

In the early 1980s there was a television show about a group of ex-commandos who helped innocent people in the Los Angeles underground. Each had his role in the group. One was the leader, one a con man, another a builder. They all worked together to help carry out the mission. Your project team is like your own personal A-Team with the mission to protect the innocent and build a killer experience. Inside your group, you have resources of experience, background knowledge, and talent. The most effective way to employ the team is to maximize your individual skills and time, while consulting for outside help when needed. Sometimes the best way to build a project is not to build it at all, and to leverage the technology and efforts of those external to your group for your own purposes, but more on that later.

A great place to start in building relationships outside the group is to recognize the people most easily available to you and their role in the development of your project.

3.2. ETC Co-directors

The Co-directors of the ETC are the final word. They pick the projects for each cycle, make the group assignments, approve the majority of acquisitions, and ultimately are the powers that be to rate the success of your project and your personal contribution. They also monitor the educational and commercial value of the project on a local as well as global scale. The Co-directors are good people to talk to if conceptual advice or guidance is needed for the project, or if human factors outside the scope of the advisor need addressing.

3.3. Your Advisor

Your advisor is your first and most direct link to all project-related activities. He regulates the meetings, committee responsibilities, deadlines, and virtually every other task you'd expect the manager of a project to handle. He is also an excellent insight into past projects' strengths and weaknesses, particularly if they have advised on the project material before. Simply put, your advisor is the first stop for any project question.

3.4. The Program Coordinator

The Program Coordinator is essentially the head of non-technical operations for the ETC, and a good reference for any necessary purchases, scheduling issues, or room/resource requests.

3.5. The Technical Coordinator

The Technical Coordinator is your software and hardware guru. He is responsible for the mailing lists and project shares on randon, the fileserver. He also maintains user accounts and privileges for all ETC digital assets.

3.6. Stage 3 Research Group

The Stage 3 Research Group is a wholly owned subsidiary of the Entertainment Technology Center and the Human Computer Interaction Institute at Carnegie Mellon University. They are most famous in the ETC for their 3D graphical prototyping environment Alice. Alice is traditionally the most popular method in ETC Projects for creating interactive 3D worlds. Because of Stage 3's close relationship with the ETC, the developers are an excellent resource for Alice-related questions and techniques. Much of the integration that goes on between Alice and the rest of your project will require frequent trips to Stage 3.

3.7. Previous Jam-O-Drum Team Members

Probably the most important technical resource to your team will be previous Jam-O-Drum team members. They have the scars and experience of working with every aspect of a JOD project and are quite likely to be of help. ***Musica*** made extensive use of two previous ETC graduates in locating, retrieving, unpacking, and hacking together the code base that was needed for the software foundation. As a handy reference, a list of previous Jam-O-Drum team members is listed as an appendix to this document.

4. LEGACY DEVELOPMENT: PRO AND CON

4.1. You Are the Best Hacker in the World

Software development, especially for games, has always been an ego industry. "I have the smallest kernel, I have the most realistic AI, I have the engine everybody wants to develop for." Because games are so often measured by the level of realism in the graphics and the fastest play, it's only natural that swarms of hired code slingers live and die by their source. However, it's not always in the interests of the project to have the absolute best code around. Good artists copy, great artists steal.

You are of course the best hacker in the world. If you weren't you wouldn't be here building what's going to be the best multimodal collaborative game in the world. Code has been written for the hardware already, code that has seen several cycles of burgeoning size, complexity and redundancy. But it works…sort of. So now you're faced with the difficult decision of leveraging the legacy code for your application. Furthermore, if you do, how much? True, you could probably crank out a blazing hot graphics engine, a lightweight MIDI and WAV player, and a couple masquerading game controllers, but it would take a while. Also, you'd be on your own if you did, no one would be there to support you, and if you fell flat on your face you'd never live to hear the end of it. So you could, on the other hand, leverage 90% of the necessary source from the last cycle and save yourself weeks worth of time, agony, and trouble. The code worked last time, it should work again, right? But then again, the old code base is huge, and slow, and unstable. It involves half a dozen layers of indirection, passing system calls off to who knows where, and if you want to do anything other than play tic-tac-toe, chances are it will crash and burn. Nothing is ever easy.

So how do you make the decision? Well, there are two things you must do when making your choice: make it fast, and make it hard. Fast because this isn't an academic study in rendering methods; you need to produce a fully functional product in as few as seven weeks, and that is virtually a heartbeat. Hard because once you make your decision, there is no turning back. The time allotted for your cycle will not support any major changes in the direction of the project. If you make the wrong decision, crash, and burn, oh well. You're wiser and the group has learned several valuable lessons to

use in the future. In ***Musica*** the decision was made to use as much legacy code as possible. Though it was with little documentation, didn't work, and for the first week or so lost, it was clear there was no chance anything of tangible value would be produced if the choice was made to rebuild. This is not to say that a bold rebuilding of Jam-O-Drum code is not a feasible option. If a project cycle dedicates itself to simply producing a JOD experience *builder* with a custom rendering engine, stripped down media support, and little vertical depth linguistically, then this would be a fine option and probably save future cycles infinite time and resources – if it was done right.

4.2. Jam-O-Drum Code base History

The Jam-O-Drum code base is several years old, and crosses multiple language barriers (See Chapter 8, under System Architecture). Hardware talks to DirectX (C++), which talks to a Java wrapper through the JNI for select parts of DX, which talks to Python code for media manipulation, which eventually informs Alice (which incidentally is written in Java) what to render and when. Two JOD experiences prior to ***Musica***, Jam-O-World and CircleMaze, both run under this structure with various styles of media manipulation. CircleMaze is more lightweight and less Alice dependent, whereas Jam-O-World has a huge Python world script for Alice and handles audio directly through calls back to DirectX (bypassing Alice's conduit to the JMF). ***Musica*** is an evolution of two previous experiences in that all game logic is handled in Java, model imports and animations are dispatched by Alice, and the sound code is a mix of Alice and Python running through the JMF. The practicals of receiving input from the JOD were left untouched and remain at the hands of the DirectInput Java wrapper.

4.3. Alice v. OpenGL v. DirectX

One of the hottest areas of debate in implementing graphics for ETC projects has been the use of Alice. Originally, Alice was the sole option for creating the JOD experiences, but now as other ETC projects investigate the feasibility of other engines such as Unreal and Lithtech, it is becoming more feasible for a rewrite of the traditional JOD code to allow a much more specialized, easier to use interface for application building.

The centerpiece of the output battle is of course the graphics engine. Alice is of course free, easy to prototype with, and carries unparalleled support. On the other hand, a good graphics hacker could very easily write an OpenGL rendering engine for

polygonal meshes, textures, and lighting. This would probably produce the fastest of any rendering alternative as the code can be written in C and hand-tweaked for specialized performance, making use of the many hardware features in today's mainstream graphics cards. OpenGL also carries a dedicated (but slowly evaporating) support group by way of fans, corporations, and of course numerous graphics texts and courses. DirectX would be arguably as fast as an OpenGL implementation, and carry with it the ease of integration with other Microsoft products (i.e. Windows). There is also a good deal of support for DX development from Microsoft, but a number of people hardly consider that support. The one great advantage Alice and DX do have over OpenGL is that they provide a conduit to the other hardware in the computer. Alice can play MP3 and WAV files without much difficulty, and DirectX handles many sound formats, including direct access to input devices. Lastly as a rising alternative, one of the popular graphics engines may be leveraged for displaying JOD content, but for now this seems less useful as the drum uses a non-directional tabletop for a viewing surface. From an end-to-end viewpoint of continuity, a stripped down DirectX implementation across the board would be quite appealing.

Ultimately, though the choice will be yours, and all are appealing in one way or another. After the cycle length, concept, and demographic have been decided, this choice should be a much easier one to make.

5. WHAT YOU'LL NEED (INVENTORY)

5.1. A Stock List for Fun

This chapter acts as your garden-variety grocery list for cooking up Jam-O-Drum experiences. You'll probably want to print it out and carry it around in your pocket, crossing off items as you procure them. There are of course alternatives to the name brand components listed below, but with the name comes the peace of mind of knowing it's worked before, and ideally, will work again.

Most of the electrical components can be obtained from the physics store in Hammerschlag, though several of the input pieces will need to be custom ordered from retailers on the internet. Because of this, and problems with finding components in stock, it is *highly* recommended that all needed materials are obtained in the first week and shipped next-day delivery. The Program Coordinator is your one stop source for purchasing, so be sure to visit her early. Also, when submitting a request for purchasing it's important to follow correct protocol (see Appendix A, Materials Purchasing Procedure).

5.2. Jam-O-Drum Components

(1) Jam-O-Drum base and tabletop

(1) Alesis DM5 drum module

(1) MIDI cable (DIN) and converter to mini-DIN

(4) high quality 9" speakers

(1) sub-woofer

(1) stereo mixer

(1) amplifier

(4) analog turntable devices

(4) 10" Drumtech drum pads

(4) analog-to-digital encoders

(4) EDIVIDE encoder dividers from US Digital

(1) electrical project box app. 5” x 4” x 4”

(2) game port ends (male)

(1) 5V DC plug

various audio cables and adapters

5.3. Host Computer

A strong experience of course needs a strong computer to run it, especially if you choose to use the Alice rendering engine. Graphics, memory and bandwidth are the key elements of any Jam-O-Drum host computer. Below are the recommended settings (as of 10/2001) for your box. Update your expectations based on the cycle’s inception, accordingly. Take note that the computer needs to have two sound cards of different make, and one of them must have a MIDI input. This is require because the Jam-O-Drum needs two game ports to run the four turntables, and a MIDI in for the drum pads. Only the primary soundcard is used, however, to prevent conflicts (See Section 7.8, Host Computer Setup).

Computer Name: swissmiss.etc.cmu.edu

OS: Windows 2000 Professional

Processor: 1.5 Ghz Intel P4

RAM: 512 mb RDRAM

Graphics Card: 64 mb Visiontek GeForce 3

HD: 20 gb Maxtor

Primary Soundcard: SB Live! Value 5.1

Secondary Soundcard: Montego Aureal Audio

5.4. Hardware Tools

strain reducers	electrical tape	heat shrink
solder	soldering iron	very small allen wrench
power phillips-head screwdriver	multimeter (continuity tester)	wire stripper
wire cutter	scissors	alligator clips

5.5. Software Development Tools

MS Visual Studio 6.0 (available for license from the Technical Coordinator)

JBuilder (available on the Musica distribution CD)

Python 2.1 (obtainable from http://www.python.org)

JAlice 4.22.01 (available on the Musica distribution CD)

5.6. Content Creation Tools

3D Studio Max R4 (available for license from the Technical Coordinator)

DeepPaint v6 (available for license from the Technical Coordinator)

Cakewalk Pro Audio 9 (available on the PCs in the recording studio)

6. STANDING ON THE GORGE (10,000 FEET ABOVE)

6.1. Don't Sweat the Small Stuff…

A project can seem daunting at any stage. Thousands of lines of code, dozens of electrical wires, graphics, music, user testing, hardware…there are a million things to do, and there's only fifty to a hundred days to do it all in. It's incredibly easy to get mired in the drudgery of any minute aspect of the project. Code may not work for hours, output may never appear, and the machine may not even start. All of this, though, is trivial. The key point to keep in mind is any one forlorn moment in the cycle is hardly a drop of water in the bucket when it's all over. Tomorrow will be another day, the work will be the same, but your perspective will be radically different. When things are rough, take a break and do something as far from the task at hand as possible. Try not to work more than three or four hours at a time. Keep the basics in mind, you need to eat, sleep, and stretch often to keep yourself loose. The more impossible the scenario may seem, the more likely a decent eight hours away from it will prompt a quick remedy. Perspective is one of the most powerful assets you have in your arsenal to solve any problem, design or implementation.

6.2. The View from 10,000 Feet

It's good practice to realign your viewpoint of the project periodically. The human mind works in abstractions and associations, and you can maximize your effective by breaking the situation down in smaller and smaller granules. The best place to start is the "View From 10,000 Feet", or the highest possible outlook on the project. An example can be built from trying to write an appropriate musical sequence for a game.

> *You are building an experience for the entertainment and education of children. You are using the Jam-O-Drum to form a series of relationships between the players, the music, and the game. The music must be light-hearted and spirited to induce positive feelings and reassurance. Light-hearted music is constructed from ascending pitches and major keys…*

You get the idea. Though it may sound odd, it's actually quite helpful to meditate for several minutes on the project's mission statement or goal. Any problem can be solved if it's only taken to a high enough level and filtered down through the familiar until reaching the unfamiliar or foreign. Also, if you're distracted or finding it hard to focus,

try to realign your direction through a motivational sequence or mantra. A favorite game, movie clip or musical piece that you can relate to your project is helpful to have on hand. Saturating yourself with the pure essence of what you love and admire about your work can be a great ego-booster and allow you to return to your task with renewed vigor.

PART II:
DIVIDE AND CONQUER (COMMITTEES)

7. HARDWARE

7.1. Room Setup

The room must fulfill three requirements to house a working Jam-O-Drum: it must be large enough, it needs to have a ceiling solid enough to support the weight of the mirror, and it must be able to obscure light for daytime use.

The Jam-O-Drum is 6'4" in diameter, and the projector will need to be 4 to 9 feet away from table. This depends on the particular dimensions of the mirror and the zoom capabilities of the projector. Generally, having the projector further from the table provides a crisper image and less warping of the image if the mirror is not at an exact 45-degree angle. At least two to three feet, minimum, should be left on all sides of the table for participants and walking space. A room 15+ Feet long and 12+ Feet wide is recommended, although a smaller room may work.

The mirror for the Jam-O-Drum hangs from the ceiling. The ceiling must be able to support the weight of this mirror, the wood it is mounted on, and the chains used to hang it.

Lastly, a room with no windows is ideal for the Jam-O-Drum because the best projections on the table occur when the projector is the only light source in the area. Use blinds, thick towels, or cardboard to cover windows during the daytime—or whatever material is on-hand that best obscures light.

7.2. Soldering Basics

7.2.1. Preparation

You will need a soldering iron (or gun), wire strippers, about a dozen tubes of heat shrink of various sizes, a small heat sink or alligator clip, and an outlet in a well-ventilated location. A blow drier is optional for shrinking the heat shrink, one may use the soldering iron instead. Plug in the soldering iron and give it five minutes to heat up. Open any windows and turn on any fans, as prolonged inhalation of the fumes may cause one to twitch or require medical attention. It is also recommended to wear safety glasses as specs of hot rosin may spit a few inches while soldering.

For any wires that must be soldered together, strip about two inches of insulation off, so they can be twisted together with no problems. For wires that need to be soldered to a pin or surface, about half an inch of stripping should suffice.

7.2.2. Avoiding "Cold Solder Joints"

The proper method to solder a joint is to hold the soldering iron against one of the wires of the joint until it heats up enough to melt the solder itself. The solder should never need to touch the soldering iron. This produces the best channel for current in a circuit, and may actually prevent current if performed improperly, forming a "cold solder joint".

7.2.3. Using a Heat Sink

A heat sink's purpose is to heat as a normal sink's function is to water. A heat sink collects heat and disposes of it through dissipation. Clipping the heat sink on the stripped part of a wire to be soldered avoids agitating any heat-sensitive parts connected to the opposite end of the wire. Heat sinks should be used whenever possible to keep excess heat from reaching any circuitry. They can also be used so solder connections on the other end of the wire do not melt and come undone.

7.2.4. Heat Shrink v. Electrical Tape

Heat shrink and electrical tape both serve the same purpose: to insulate wires that need to carry an electrical signal. Electrical tape is more difficult to use when wires are soldered in proximity to one another, and it degrades quicker over time. Heat shrink is a better choice for most situations because it is cleaner and more durable. To use heat shrink, select a piece slightly larger than the wire(s) it will be covering, cut it to size, and slide it onto the wire before soldering (otherwise it won't fit). It is easy to forget to put the shrink on first, make it a point to remember.

Once *all* the soldering is complete for the component you are working on, and it has been *tested*, then it is time to heat the heat shrink. Move the heat shrink over the exposed wires and hold the soldering iron close (but do not touch) and it will start to contract and harden. This takes patience—about 60 seconds or so for a smaller piece. Use a blow drier if one is available in order to speed things up.

7.2.5. Finishing Up

When done soldering, one must treat the tools properly so as not to ruin them, or hurt oneself. First, melt some solder directly onto the tip of the iron until it has a silvery appearance. This is called "tinning" the soldering iron, and keeps it in a state that protects it and allows it to heat up more quickly next time someone uses it. Second, place it back in its sheath and unplug it. Make sure it's in a place where no one can burn himself. Lastly, wash your hands with soap and water to remove the lead of the solder from your skin.

7.2.6. Caveats

There are a couple things to look out for when soldering. Taking the time to go over these points could save much time and frustration.

When stripping wires, be careful not to cut them. It is easy to damage the wires and not know it until they break off after soldering. Apply minimum force on the wire strippers, one should almost feel he is pulling the plastic insulation apart rather than cutting it. Test all circuits for continuity before shrinking heat shrink, otherwise the shrink will have to be cut off a bad circuit and replaced (which is a pain and a waste of time).

7.3. Input Devices

7.3.1. Receiving Data

There are two input devices built into the Jam-O-Drum: turntables, and drum pads. Each of four Jam-O-Drum users has his own turntable and drum pad. The turntables can detect a turn left and a turn right, while the drum pads are capable of detecting a hit and its force. This limited set of inputs is a challenge, as it does not have all the possibilities of a game controller—just left, right, and hit. This does, however, lend itself to creativity of design and novel experiences.

7.3.2. Turntables

The turntables are flat donuts, 1' 3 ¾" in diameter, mounted on ball bearings. The computer can detect when a turntable is spinning left or right, and this is one of the two forms of input into a Jam-O-Drum experience. Since they are rather large and have a moderate amount of friction, it is demanding to have to turn the turntable more than a

quarter rotation very often, especially for a child. Keep this in mind while designing experiences; don't require large turns of turntables repetitively.

7.3.3. Drum Pads

The Drum Tech drum pads are flat 10" pads that connect to a drum module. The pads are able to detect when they are hit as well as how hard. The drum module sends this information as Midi signals to the computer. The drum trigger also has 20 different sets of sound samples that it can play with the strike of a pad. User testing has shown that this feature is one to include in a Jam-O-Drum experience. People love to hear immediate feedback when hitting a drum pad, and the drum trigger module does this without any work on the computer end.

7.4. Audio Equipment

The audio equipment wiring is not complicated for JOD-only installation. If house speakers or surround sound equipment is desired, the mixer must be used to reroute particular channels of the computer's sound output to the speakers located around the venue. The SB Live! card does support Dolby 5.1/EAX. For more information on this wiring, please see the Zeum venue audio schematic. For the ***Musica*** sound layout, consult the respective schematic.

Speakers – are connected to the subwoofer. The subwoofer has a left and right output. Each speaker was wired in mono by combining the stereo channels.

Subwoofer – the subwoofer is connected to the amplifier.

Amplifier – the amp is connected to the mixer.

Mixer – receives sound from both the drum module and the speaker output from the computer.

Drum Module – the drum module sends midi to the computer and has left and

right outputs that are sent to the mixer so the actual drum sounds are played also.

Cables – ¼" cable, RCA, 12g. speaker cable, headphone cable, and MIDI cable was used.

7.5. Electronics

7.5.1. The Black Box

The "Black Box" is the project box that encapsulates all that goes on between the four rotational encoders as inputs and the two game port connections. The four wires from each of the four rotational encoders under the turntables go into the project box along with a power cord, and two male 15-pin game port inputs come out of the box.

7.5.2. Gameports

7.5.2.1. Materials

- A project box roughly 6"x5"x4"
- 2 Male 15-Pin inputs for the game ports.
- +5V DC Power Supply with bare wires for soldering
- Strain reliefs to fit around 4 or more wires at a time
- Access to a drill or drill press
- Screwdrivers
- Soldering Equipment
- Ties or tape to bundle wires
- 4 of the CA-3133 Finger-latching cables, ideally 3 feet in length or greater
- The 4 EDIVIDES used in the rotational encoder to EDIVIDE wiring
- Extra wire
- Perf board – A piece the size of half a graham cracker (optional)

7.5.2.2. How to Proceed

This is one of the most complicated and difficult tasks for construction of the Jam-O-Drum, but is necessary to interface the turntables with the computer. For the specifics, refer to the "Joystick Encoder Wiring from EDIVIDE Encoder Resolution Dividers" diagram.

The diagram neatly depicts the wiring from the EDIVIDE out ports to the two game ports and the power supply. The EDIVIDEs are labeled 1 through 4, corresponding to their turntables. The turntables may be numbered any way one wishes, but it must be kept consistent with the encoder wiring and the wiring of the drum pads. Turntables 1 and 2 end up routing to "joystick 1" while 3 and 4 input to "joystick 2". Joystick is

synonymous with "game port" in the diagram, since the encoder box emulates two joystick connections with the game ports.

Note that in the diagram, only touching wires of the same color are wired together. For example, all the orange power wires are connected to one another from the EDIVIDEs and to the power supply, but they are not wired to anything else because the other wires they cross in the diagram are different colors. Also, make sure to read the notes on the diagram, as they are important. If wired improperly, the encoder may not be detected by the computer as a joystick, and it could even harm the computer.

Holes should be drilled in the project box for the input wires, power supply, and 15-pin game port connectors. Make sure the holes are consistent with the size of the strain reliefs. Either the wires need to be put through these holes before the soldering is done, or slits can be sawed down to them to insert the wires later, and then sealed off.

The perf board may be used to organize groups of wires. But in this case it has been found more effective to twist wires or groups of wires around each other, solder them together, and then heat shrink them. For the ground wires up to 9 wires need to be soldered together: one from each EDIVIDE, two from each 15-pin connector, and one from the ground of the power supply. To get these all touching each other on perf board is messy. One is better off having two or three inches of stripped wire on each, then twisting them all around each other like the smaller strings in a rope. A much better connection can be made this way.

A similar tactic should be used when wiring the 7 pins (connected in red in the diagram) on each 15-pin connector. For the connectors, it is best to do all other wiring first, move the heat shrink into place, *then* do the red wiring since it must be wired around the other connections.

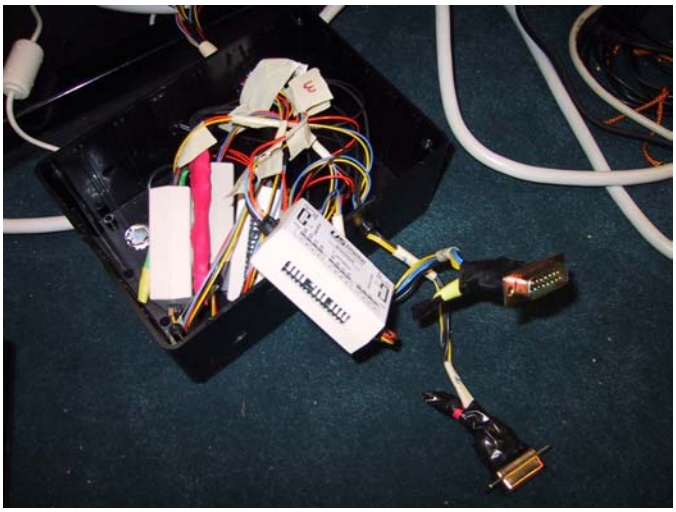


Figure 4. Switch settings on E-DIVIDE.

Once the soldering is complete, clamp the strain reliefs onto the wires and set them in the holes of the box. Place the connected EDIVIDES into the box with the other intermediate connections, and screw the top on. The Black Box is at this point complete! Plug the power supply in and connect the 15-pin connectors to the two game ports, and run the Windows test program to make sure things are working correctly (in Settings → Control Panels → Gaming Options (or Game Controllers, etc.).

7.5.2.3. Caveats

When soldering to metal encased in a material such as plastic, the plastic may melt. The 15-pin inputs for a game port are lodged in plastic, which can melt even before the solder. Use a heat sink if possible for this sort of problem, however, for the 15-pin input, it is not possible. In this case it is acceptable to drip solder onto the connection rather than heat the wires. If this approach is used, make sure to test continuity afterwards to catch any cold solder joints. If the plastic does melt and a pin moves around, melt the plastic and use pliers to reposition the pin.

The circuitry for the turntable encoders requires a power supply. Make sure if this is already part of the circuit that it is unplugged before working on the circuit further. Unplug as much as possible to isolate the circuit at hand so as not to damage peripheral equipment.

7.5.3. Encoder to EDIVIDE Wiring

7.5.3.1. Materials

The following items are needed to wire the turntable's rotational encoders to the EDIVIDE encoder resolution dividers:

- 4 Rotational Encoders
- 4 CA-3133 10-Foot (or longer) Finger latching mating connectors
- 4 CON-FC5-22 Finger latching connector shells
- 4 EDIVIDEs
- Soldering equipment (see soldering section above)

7.5.3.2. What is the EDIVIDE?

The EDIVIDE is an "encoder resolution divider". Its function is a relatively simple one, although imperative. The ratio of the large turntable to the wheel attached to the rotational encoder is rather large. A small turn of a turntable spins its rotational encoder's wheel many times, producing an excess of signals. The EDIVIDE can be configured to reduce the number of signals by up to 4096 times. Without the EDIVIDEs, the computer would be saturated with joystick input.

7.5.3.3. How to Proceed

The latching connectors have four wires, one each for the A signal, B signal, ground, and power. Pin 1 on the EDIVIDE is for ground, pin 2 is an index (unused for the Jam-O-Drum), pin 3 is the A channel, pin 4 for +5V DC power, and pin 5 is for the B channel. Strip the latching cable wires about an inch and arrange them in the finger-latching connector shell so they match up with the outputs of the rotational encoder. The outputs from the rotational encoder may be in a different order than the inputs of the EDIVIDE.

Optionally, the connector shells may be left out, and the wires can be soldered directly to the rotational encoders. Previous Jam-O-Drum projects have used this approach with no problems. However, it is cleaner and easier to use the connector shells.

Once the connector shell is in place, plug it onto the rotational encoder and plug the other end into the EDIVIDE. Do this for each EDIVIDE.

7.6. The Mirror

7.6.1. The Best Mirror

The ideal mirror is as wide as the Jam-O-Drum table surface (including the turntables and drum pads), and 1.414 times as high as the table is wide. Since the most recent Jam-O-Drum table is 6'4" in diameter, this means the best mirror will be about 9' tall. When the mirror is hanging at a 45-degree angle above the table, this aligns the front and back ends of the mirror with the edges of the table and allow for maximum flexibility with projector placement.

A 9' mirror may not be available or affordable, and a smaller size will work but require stricter positioning of the projector. ***Musica*** used a mirror about 5' feet high, and it produced an acceptable, although imperfect, projection. For this mirror the projector was angled upwards from the height of the table, rather than aim it horizontally at the mirror level with its center. Light rays traveling to the top of the mirror and hitting the near side of the table traveled more distance than the rays hitting the bottom of the mirror and the far side of the table. This allowed the rays on the closer portion of the table to spread out more and produce larger images at one end than the other, slightly warping the projection. Using a 9' mirror would avoid this problem. Also, children are more likely to hurt their eyes looking into the projector when it is positioned here, near child eye-level.

7.6.2. Mounting

The mirror needs to be attached to a large piece of wood for hanging. Use *at least* ¾ inch thick plywood, and leave 5 inches or so to stick out beyond the edges of the mirror (for screws). The objective here is to mount the mirror on a sturdy piece of wood that won't bow. A foreseeable possibility for extra sturdiness would be to construct a skeleton of two by fours to put the plywood on.

One can attach the mirror to the plywood with a series of screws and metal discs (like fat washers) around the circumference of the mirror. The screws will hold the metal discs tightly clasping the edge of the mirror to the wood. The hardware store where the mirror is purchased may be able to do the mounting itself, or at least suggest the best method. It is strongly suggest to inquire about this.

7.6.3. Hanging

To hang the mirror, begin by screwing eyehooks into the back corners of the plywood. Next, measure the distance from the top eyehook on one side of the mirror to the bottom eyehook on the same side. Multiply this distance by the factor 0.707. The result is the distance the hanging chains need to be apart from the front to the back. The chains of each side need to be as far apart lengthwise as the eyehooks on one side are from the other side.

Chains are suggested for hanging because they allow for quick adjustment of mirror height. Use s-hooks to attach the eyehooks on the plywood to the dangling chains. Use a level or similar device along the back of the plywood to make sure the mirror is hanging at a 45-degree angle. The back of the mirror should line up with the back of the table. Once one sets up the mirror and projector, two people can move the table to align with the mirror and the image by grasping its metal frame. The table should be positioned so that there are two turntables on either side of the projector's line of sight to avoid creating shadows that obscure the projection.

7.7. The Projector

7.7.1. Positioning

For a full-sized (9 foot) mirror, center the projector vertically and horizontally to the mirror. Place it about 6 feet from the front of the table. One can hang the projector in a similar fashion to the mirror using a platform, or place it on the surface of an object of correct height. This is just a starting point, as not all projectors are the same and have different zoom capabilities. Use trial and error to figure out the best location for optimum table coverage.

For smaller mirrors, a closer positioning will be necessary for better table coverage. Also, moving the mirror up the chains to a higher position will give more flexibility and coverage. One can also try moving the projector to a lower spot and angling it upwards slightly towards the mirror, although this configuration produces a distorted trapezoidal projection on the table rather than a uniform rectangle.

7.7.2. Flipping the Image

Since the image on the table will most likely need to be exactly as shown on the computer monitor, the projector needs to be set up to flip the image to counter the

mirror's effects. Make sure to use a projector that has this capability, which can usually be accessed through a menu on the projector's display.

7.8. Host Computer Setup

The host computer is fairly simple to set up if you are sure to configure the hardware to use the appropriate options. If you are buying a new computer, have the hardware and software installed by a professional. The extra dollars will save you time and agony in the long run.

- Enable the SB Live! card as the default wave/midi player and recorder in the Windows multimedia settings (if this is not set, you will not hear any sound)
- Enable the SB Live! MIDI UART as the default MIDI input device in the Windows multimedia settings
- Install two 2-axis, 4-button joysticks under the Windows game controllers settings and TEST THEM with two joysticks, one on each gameport on your two sound cards
- Make sure all the following software is installed and running properly.
    - JBuilder
    - MS Visual Studio 6.0
    - MS DirectX (latest version)
    - MS DirectX SDK 7.0a
    - MS Core SDK
    - MS Visual Source Safe
    - JAlice ver. 4.22.01
    - Winamp (for media testing)
- Be sure to register the computer on the SCS domain for testing so you have access to randon and other network resources. http://www.cs.cmu.edu/afs/cs.cmu.edu/help/www/admin/netregister.html

8. SOFTWARE I: THEORY

8.1. Legacy Code: A System Design Example

Jam-O-Drum experiences have all used a shared code base for the past several cycles. This is partially because the Alice rendering engine was a project requirement, and partially because leveraging the DirectX wrappers made input and output simpler. For the next two chapters, all of the Jam-O-Drum legacy code will be explained in design and implementation. Even if you decide to abandon much of the legacy structure, these chapters can be a boon in providing one example of how the software system could work, and the interplay the different modules will need.

This chapter will cover the philosophy of the multi-lingual hierarchy and the module interaction. The following chapter will provide a more detailed look at the source in practice.

8.2. System Architecture

8.2.1. The Wedding Cake (A Marriage of HLLs)

The Jam-O Drum library was designed to be extensible and flexible for future use. The idea was to offer a standardized framework to create new Jam-O Drum experiences quickly based on the input and output encapsulated in the Java classes. Hardware is handled by the host computer's DirectX, which in turn is wrapped by Java. In Java most of the game's logic and state is maintained. Python code handles most of the animations and effects that the Alice renderer produces. Sound may be handled either by the Java Native Interface (JNI) and its wrapper of DirectX, or by Alice in the form of Python.

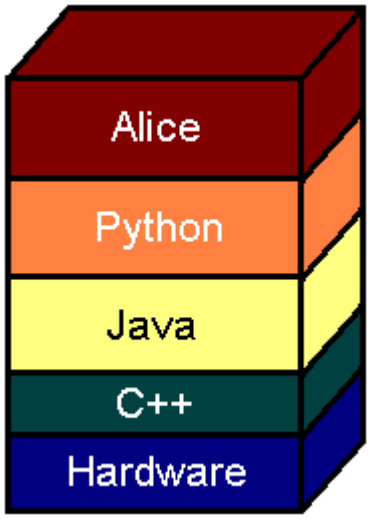


Figure 5. Jam-O-Drum Experience Builder Architecture

A detailed diagram depicting the communication between the software layers and their components is listed in the diagrams and schematics appendix.

8.2.2. C++

Microsoft's DirectX enables direct manipulation of the PC's input and output hardware. DirectX is one of the most popular methods for game developers use when creating new applications or engines. While DirectX is indeed powerful, it is also quite expansive and difficult to take in at first. To use it efficiently, a programmer must have a significant amount of experience with Microsoft development tools and their mindset. Because of the difficult interface to the DirectX library, the most important features have been wrapped in Java so the JOD experience developer need not be overly concerned about having a close relationship with it.

8.2.3. Java

The Java layer of the system is the second largest (DirectX is huge) and host to the Jam-O-Drum Control Panel, which is used to start JOD experiences. This is the main JOD library on top of which new experiences can be created. The philosophy of the JOD Experience Builder is to encapsulate all of the Jam-O Drum resources in a set of packages. The Java packages assume that the hardware is operating properly and maintains little knowledge of the underlying system's state.

New JOD experiences should begin by building the experience framework under the **Controllable** interface. This implements a Java control panel that can be used to launch JOD applications. There is also an **Input** class that is an abstraction of the physical drum pad and turntable devices. **Listeners** are used to look for input from the devices so the application can process the data accordingly.

The constructs in the experience that handle application launching and input processing allow a rapid prototyping of Java-based experiences. After the framework of the game is laid, output may be handled through the Python/Alice layer for rendering and effects.

8.2.4. Python and Alice

Alice is a rapid-prototyping environment for 3D worlds created by the Stage 3 Research Group. Alice has been used traditionally for Jam-O-Drum experiences because it allows quick production of environments involving models, animation, and sound. In the name of simplicity, Alice worlds can be quickly made to generate JOD interfaces for the tabletop. Sounds effects and music may be easily played using Alice's connect to Java and the JMF. Alice also handles all of the animations needed for game play. While Alice is not necessarily a requirement for a Jam-O-Drum experience, the legacy and support are present for this sort of development. In the next chapter the basics of Alice and its use for JOD experiences will be described.

9. SOFTWARE II: PRACTICE

9.1. High-Level Languages for Abstraction

This chapter covers the ***Musica***/Jam-O-Drum code from a functional perspective. What follows will cover the salient points of the classes and methods in the software model. Sections 9.2 covers the elementary Jam-O-Drum Experience Builder, on top of which other experiences are developed. Each language layer and its specifics will be addressed in turn. Section 9.3 involves source created specifically for ***Musica***, and is a good example of what is needed to be constructed for an experience on the builder.

9.2. The Jam-O-Drum Experience Builder

9.2.1. Why JODEB?

The Jam-O-Drum Experience Builder (JODEB) is a foundation for new rapid JOD prototyping. By adding a few simple Java classes and calls to run Python code in Alice, new experiences may be constructed and tested quickly. JODEB involves wrapping C++ DirectX functions for ease-of-use, and standard procedures like processing input and output are already handled. The JOD experience makes use of the existing basis to present the content for the new application.

9.2.2. C++ Wrappers and JNI

As discussed in the previous chapter, the Jam-O-Drum legacy code makes use of DirectX to handle all direct communication with the system hardware. DirectX is written in C++, and encapsulates a large system of input and output functions for the sound, game controllers, and display. Jam-O-Drum wraps features of the DirectSound and DirectInput modules, as well as Windows Multimedia for receiving MIDI input. JAlice wraps its own part of Direct3D internally, but that's of no concern to the programmer.

In addition to the JavaDocs produced from the ***Musica*** code, below is a brief summary of the DirectX behaviors wrapped by the JNI. The key classes are listed and described by function.

9.2.2.1. DirectX

The DirectX.java class defines an instance of the DirectX presence for Java to utilize. It holds a DX event queue that keeps track of occurrences necessitating communication with the hardware. It also encapsulates the initialization and shut down of DX manager. The HWND is a handle for the window the operations are being performed in.

9.2.2.2. DirectXEvents

9.2.2.2.1 DXEventQueue

The DXEventQueue is basically a thread that receives and dispatches events as they are triggered within the application.

9.2.2.2.2 Other DXEvents

The other DXEvent classes are declared wrappers for any instance of an object, listener or dispatcher of DirectX events. Both DirectSound and DirectInput also have listener objects to look for events to add to the queue.

9.2.2.3. DirectInput

9.2.2.3.1 DirectInput

DirectInput keeps track of all active input devices in the application (a combination of keyboard, joystick, mouse), and provides access to them much like DirectSound's sound devices.

9.2.2.3.2 Device and PolledDevice

Device is an instance of any DirectInput device that allows access to its state. It is based on the DXEventDispatcher. PolledDevice, which extends Device and implements Runnable, handles the starting and stopping of a thread to poll and generate events from the input device. The main difference between the two is the methods required for extracting data. Mouse and keyboard are Devices, whereas Joystick is a PolledDevice.

9.2.2.3.3 Joystick, Mouse and Keyboard (Three Great Friends)

Joystick extends the PolledDevice class with a smattering of specific operations for maintaining a joystick's state and listener threads. Mouse and Keyboard are derived from the Device class, their classes exist primarily to wrap their C++ equivalents with little change.

9.2.2.4. DirectSound

9.2.2.4.1 DirectSound

This class allows the creation of a new instance of a DirectSound manager. The manager is initialized and shutdown through the JNI much like that of the DirectX class. The class also maintains the number and access to the sound devices in use.

9.2.2.4.2 SoundDevice

SoundDevice allows a conduit for audio output. Each SoundDevice has an associated SoundBuffer (or set of buffers) and listener. Speaker configuration and mixer status may be set for a particular device.

9.2.2.4.3 SoundBuffer

The SoundBuffer class services the playback of a particular event request. The JNI can load, unload, play, stop, or precache files in the buffer. The priority, position, and velocity (volume) of the buffer may also be modified for advanced manipulation of the samples.

9.2.3. Java and the Experience Builder Core

The fundamental Java JOD library is a package called edu.cmu.etc.jamodrum. It contains JBuilder packages and classes. Every Jam-O-Drum experience is actually a sub package within this package. Understanding this package is the foundation necessary to write new experiences. The following is a description of the usage of the library, illustrated with some successful experiences as examples, including Hip-Hop (Dancer), CircleMaze and ***Musica***.

The figure on the next page provides an overview of classes in the builder.

USAGE OF THE BASIC JAVA JAM-O DRUM LIBRARY

JAVA CLASSES

Figure 6. Integral JODEB Classes.

9.2.3.1. Input

Much as you'd expect, the input class receives data from the Jam-O-Drum experience. It has one input for the turning disks and one for the drum pads.

9.2.3.2. ZeumControlPanel

The Basic JOD library has a main file called ZeumControlPanel. This is the main class of the library and constitutes the control panel of the Jam-O Drum from which every experience can be started. It appears as a window with a pull-down menu with which the operator can select a loaded experience. In order to add a new experience to the widget, the author must modify this class to include it.

There are three Java interfaces that must be implemented in order to create a new experience: **Controllable**, **PadListener** and **DiskListener**.

9.2.3.3. Controllable

The Controllable interface allows the control panel to start and stop the new experience. It has the following two methods that must be implemented:

Start(Input in, Frame frame)

This method is called by the control panel whenever a new experience needs to be started. All initialization for the experience should be done in this method.

The first parameter is an Input object that refers to the input of the Jam-O-Drum. This object is created by the control panel and is passed through this function, already initialized. The new experience can assume the input is working and ready to receive input. The second parameter is a reference to the control panel frame.

Stop()

This method is called by the control panel when the experience is stopped by the Jam-O-Drum operator. All shutdown for the experience should be performed in this method.

9.2.3.4. PadListener

The PadListener interface allows the experience to receive the input from the drum pads of the Jam-O-Drum. It allows the experience to handle the event raised when a guest strikes a drum pad. It has the following method that must be implemented:

padHit(PadInputData inData)

This method is called when a drum pad is struck. The parameter inData is an object that contains all the available information of the pad hit and allows one to know which drum pad has been hit and how hard (velocity).

9.2.3.5. DiskListener

The DiskListener interface allows the experience to receive the input from the turning wheels of the Jam-O Drum. It allows the experience to handle the event raised when a disk is turned. DiskListener has three methods that must be implemented. The most important is the following:

diskTurn(DiskInputData inData)

This method is called when a disk (turntable) is moved on the JOD. The parameter inData is an object that contains all the available information of the turn and allows to know which wheel has been turned and in which direction.

These three interfaces are enough to receive the input of the Jam-O Drum and to start and stop the experience. The experience author may concentrate on the application itself because the mundane details of its operation are already implemented.

9.2.4. The Java-Python-Alice Love Triangle

9.2.4.1. JAlice as a Media-Interaction Engine

For display, the video signal from the graphics card is routed to a high-fidelity projector and mirror to the Jam-O-Drum table. The experience author can choose how to handle the rendering of the experience, but hear we discuss the existing method. In order to illustrate briefly how the existent experiences handled the output, we are going to talk about the interaction between the actual experience and JAlice.

A JAlice world consists of a set of 3D objects (commonly imported from 3D Studio source) and JPython scripts that can be executed by the engine in order to move and animate objects. In order to use JAlice as the animation engine, the experience author must create a new JAlice world with all of the components already imported and arranged in their startup orientation.

To integrate the JAlice world with the JOD experience, the world must not be executed from the JAlice executable, but instead, it must be invoked by the Java code of the experience. From there, the Java JAlice library may be used to start the engine and load the world. This indirect way to run the JAlice engine is perpetuated by a configuration file.

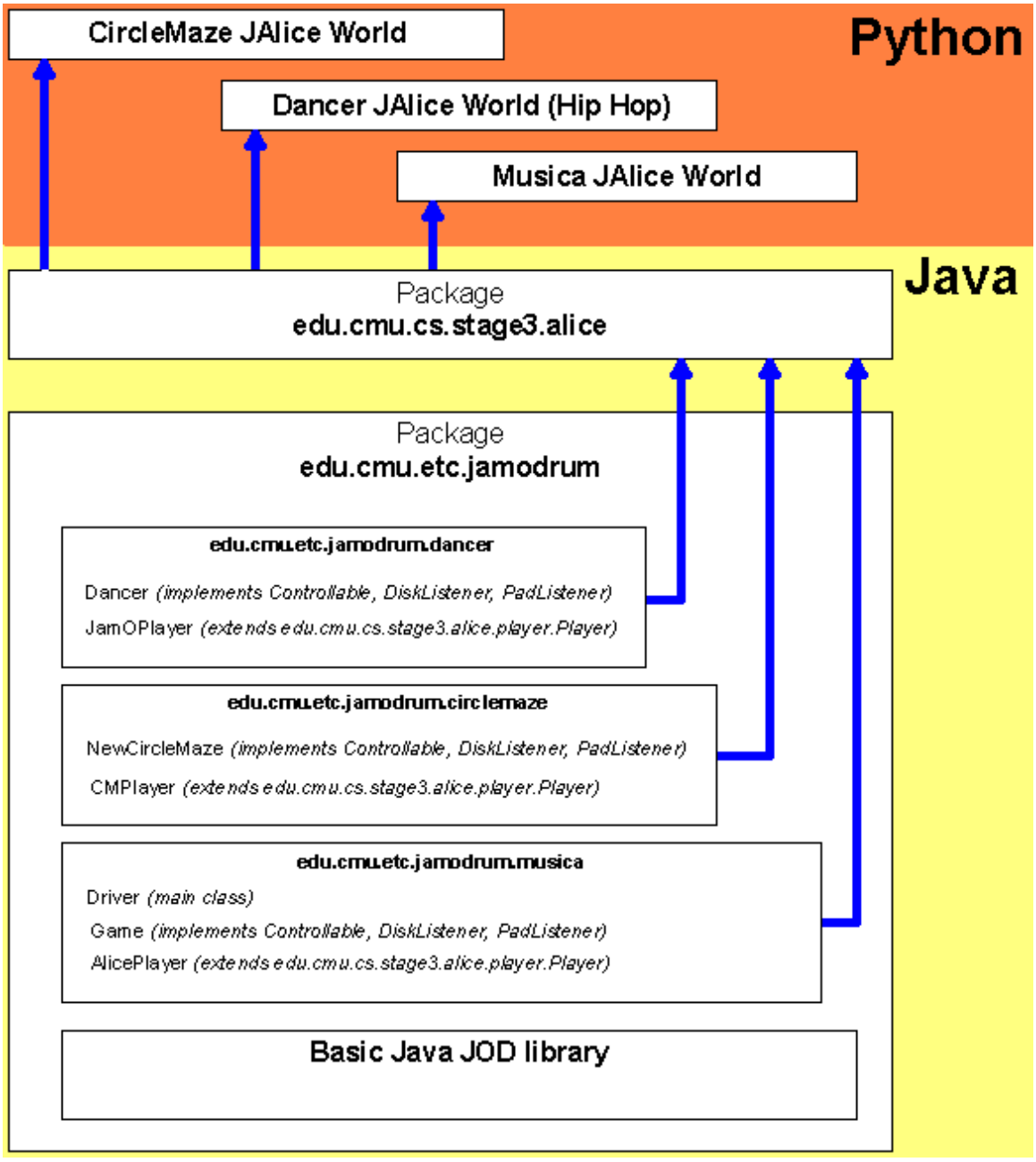


Figure 7. Java-Alice Package Interaction

9.2.4.2. The JAlice Configuration File

From the experience project, an executable file is built that starts the experience. For ***Musica*** this file has been named as JamODrumMusica.exe. It takes the .config that has the same name as the executable to initialize the application. This file specifies some parameters for JAlice and the name of the main class that should be run. Usually this is ZeumControlPanel. An example of such a config file is included as an appendix.

9.2.5. Using JAlice for Animation

9.2.5.1. Introduction: What is JAlice World Script?

JAlice scripts are written in Python, a very-high level scripting language. Under the "Behaviors" section of a JAlice world, there is a section where Python script can be typed directly. All this script is saved in the world folder with the name "script.py". For a tutorial on Python visit http://www.python.org.

The Python script typed in the world script section can directly interact with the JAlice world, and issue JAlice commands. The best documentation on JAlice script commands is found on the Carnegie Mellon Building Virtual Worlds class web site (http://www.alice.org/building.htm). The specific link to the help section is:

http://www-2.cs.cmu.edu/afs/andrew/course/05/331/webSpace/alice_help.html .

One of the most useful functions for understanding JAlice objects is the following:

```
def printActions(object):
    print object.__class__.getMethods()
```

When called with the name of one of the world's objects, a complete list of all the methods for the given object is printed out. This is also a quick way to get an idea of all the possible ways an object can be used in a world.

9.2.5.2. JAlice World Script v. Calling JAlice Script from Java

All coding does not have to be done directly in the JAlice world. Some Jam-O-Drum groups have put as much code in the JAlice world as possible, whilst other groups did most of the logical coding in Java and passed basic object manipulation commands to JAlice. If using Java at all, some balance must be struck that best suits the needs of the project.

What are the tradeoffs? Java has a few obvious advantages. First of all, there is a large existing Java code base for the Jam-O-Drum already. Second, for large amounts of coding, Java lends itself to organization because of its modularity and strong typing. Scripting in JAlice, on the other hand, requires all the code to be in the script.py file with the exception that it is possible to import modules. However, importing modules is inefficient and eats up processor time.

Scripting in JAlice has its advantages as well. Overall, it is better for performance, since JAlice needs all the processing time it can get. If a Java thread is doing constant work, it imposes on JAlice. Also, Python is relatively simple to learn for those that do not know Java either.

9.2.5.3. Methods of Scripting in JAlice

When scripting in JAlice, there are a few techniques that keep popping up as either very useful or necessary for worlds. These are discussed below, using ***Musica*** as an example.

Once of the most useful commands is that which allows an object to be copied. In ***Musica*** there were prototypes for each different type of musical block. Some levels had many copies of the same type of block, although in the JAlice world having all of these present would have been needless clutter. So as ***Musica*** reads in a level, it spawns copies of the blocks it needs using the command:

```
newObject = copy(object, name, classesToShare)
```

One word of warning: expect performance to sink temporarily while copying objects. This is why ***Musica*** restricts its use of the copy() method to only the loading of each new level.

When objects are on the screen that becomes temporarily unnecessary, Musica uses the object.hideRightNow() command. This optimizes JAlice's work process until the objects are needed again. When they are, the object.showRightNow() command summons them back. For instance, the object that is textured with the "You Win!" image is always present and floating above the board. It is actually hidden the whole game, and only shows up via showRightNow() when a set of players wins the game.

Another useful feature of JAlice is its built in ability to change objects textures on the fly (as well as other characteristics). This was most helpful when displaying scores. Each number of the score (up to 7 digits) is an object textured with a "0" at the start of the ***Musica*** game. The JAlice world had textures with the numbers 0 through 9 imported, waiting to be used. When the score needed to be changed, a *PropertyAnimation* was used to switch the seven textures to the textures of the correct numbers. (To activate any animations, they need to be passed as a parameter in the run(animation) command.)

Last, JAlice provides simple support for playing of WAV and MP3 files. By running a *SoundAnimation* animation with the sound as a parameter, the sound is played. Any sound that is imported in the JAlice world can be played this way from the world script. One of these calls looks like this:

```
run(SoundAnimation(blockHit1))
```

Where blockHit1 is a sound in the ***Musica*** world. There is a tradeoff to think about between using WAV files and using MP3 files. MP3 files are relatively small, but need to be decompressed in real time, while WAV files consume inordinate amounts of memory but do not need to be compressed, more on this later.

9.3. Musica Application Source

9.3.1. Java

The core of the ***Musica*** software is the main Java package **edu.cmu.etc.jamodrum.musica**. As a part of the object-oriented software design process, an initial UML static class diagram was produced that shows the internal and external relationships of the classes. A hard copy is available in the diagrams and schematics appendix.

9.3.1.1. Main Classes

Driver

Driver handles the main loop of the game, regulating the state and the thread sleep time. This class is created by the control panel when the user selects the ***Musica*** experience.

Game

This class implements the three interfaces of the JOD library. It is the controllable object passed to the control panel and it receives the input from the Jam-O-Drum.

AlicePlayer

This class extends the class Player of the JAlice package (edu.cmu.cs.stage3.alice.player) and it handles all the interaction with the JAlice engine including initialization, loading and running the JAlice world, as well as handling the JPython methods used for world animations.

Wall, Ball, Paddle, Block

These classes represent game objects and track their position and velocities to varying degrees.

9.3.1.2. Musica Package Interface and Class Summary

Interface Summary

Collidable

Provides an interface for objects that require collision detection to affect each other.

Updatable

Objects that implement this interface have some form of update that gets performed every cycle of some loop, most likely a game loop.

Class Summary

AlicePlayer - Manages all the interaction between the game and JAlice as the user interface

Ball -A ball is an object that is deflected off the paddles to break the musical blocks.

Block -Musical block placed in the board.

Driver - Main class for Musica.

EdgeWall - Walls in the edge of the board that make the ball fall off

FileInputOutput - Class that can be used to read and write from and to text files.

Game - Class that manages the game MUSICA on the Jam-O Drum

Level - Manages the information of a level in the game

Paddle - Represents a paddle in the game.

UpdateManager - Manages all the Updatable and Collidable objects

Vec3D - Class that can be used for representation of 3 Dimensional points or vectors

Wall - These walls are to ricochet the ball and cannot be destroyed

9.3.1.3. Java's Role in Alice

The Musica JAlice world has all the imported models for the animations and JPython methods for all internal game events. These methods are called from the **Game** class, through the **AlicePlayer** class.

9.3.2. Python and Alice

In ***Musica***, all sound management is handled through Python and JAlice's access to the JMF (see the Sound chapter for more information). The basic animations for rotating the blocks and the balls are simple JAlice animation scripts. For more information on JAlice animation scripts, see the ***Musica*** JAlice world script, **script.py**.

10. MODELING AND PAINTING

10.1. Rapid Prototyping

Rapid prototyping is one of the best ways to ensure quality in production. By producing either throwaway or evolutionary trial products every week or so, surveyed estimates may be used to test the feasibility of your experience and its features.

It's important to have rough 3D models as soon as possible, both for conceptual review and for use in an early functional prototype for the game. These models may be basic forms that haven't been painted. These allow you to experiment within Alice to see if your features will be able to live up to your concept art.

10.2. The Level-of-Detail Tradeoff

In creating models, you have the ability to make a really high quality (high polygon) model. The problem is, the more polygons you have in your model the slower Alice will run. So the goal is to make a model with the fewest amount of polygons as possible, without sacrificing visual fidelity. For example, in ***Musica***, the ball was a 900+ polygon model. Four other balls with the same polygon count substantially hindered the frame rate. In the end, the polygon count for a ball was cut to 160. The ball is understandably not as smooth but the detail loss is minimal.

There are a lot of different methods for producing level-of-detail (LOD) compression, both offline and dynamic. 3DS uses a simple method for reducing the polygon count of models, but there are elegant and topology-preserving alternatives such as polygonal decimation, vertex clustering and quadric-error metrics. More information on LOD techniques is available at the ACM's digital library.

10.3. Creating Textures

Most of the textures for ***Musica*** were created in Photoshop. All of the textures applied to the 3D models were stored as Windows bitmaps. Pixel resolution is used when deciding the size of your image. The most important thing to remember about the bitmaps is that the resolution needs to be a power of 2. For example: 2, 4, 16, 32, 64, 128, 256, and 512. The image can be a 128x256 pixel size or some variance of it. Actual image size in inches is irrelevant. The key to efficiency is to use the coarsest

resolution while maintaining a good level of fidelity for your image. This reduces your file size and makes Alice run quicker.

10.4. Mapping Textures

When you have made all of the textures you want in Photoshop, load your model in 3D Studio and open the texture editor. Click on the box to the right of "Diffuse" in the "Maps" section of the editor. Select "Bitmap" then choose your bitmap that you want to add on to your model. Once you select the bitmap click and drag it on to your model. After that press the checkered box button that is under all of the samples, this makes your image appear on the model. After you apply the BMP you need to align it properly on your model. Go to your "Modifier List" and select "UVW Map." This allows you to center your texture on your model. You also need the UV map if you plan on exporting the model to Alice.

DeepPaint was used by the ***Musica*** painting team to help map the textures onto the blocks. The cube bitmap was designed such that it has all of the sides for the cube on one BMP. First the cube was exported from 3D Studio to DeepPaint. This is done by selecting the "hammer tab" and under "utilities" choosing the "More" button. Select the DeepPaint option. You should now have a button below the utilities menu. Now click the "PaintIt" button. In DeepPaint we then proceeded to click on the Map option on the top. All of the sides from the cube were separated and laid out on the bitmap to correlate with the appropriate sides. The file was then sent back to 3D Studio Max.

10.5. Exporting Your Model

It's easiest to export your model when there is only one per 3D Studio file. It is possible to export multiple objects per file, but Alice may also import the other textures in your Material Editor. This means you have to go in to Alice and delete each extra BMP out individually.

The first thing to do is select the object you want to export into Alice. Click on the File menu and select "Export Selected." Select the folder you are exporting to. It's best to keep your ASE file with the bitmaps that are associated with it. The folders may tend to get a little cluttered but Alice has problems importing if it can't locate the bitmap. When selecting ASE exporting options make sure Mesh Definition, Materials, Transform Animation Keys, Mesh Normals, Mapping Coordinates, Vertex Colors,

Geometric, Shapes, and Use Keys are selected. Lighting or cameras may not be imported into Alice. Also, you cannot export entire worlds. It's extremely problematic.

11. ANIMATION

11.1. Keyframe v. Native Alice

An interesting experience is always reinforced by attractive animations of your models. 3D Studio provides some tools for scripting some basic keyframe animations. Unfortunately, these animations rarely import into JAlice successfully as it recognizes only position and rotation information. Thusly, when designing your experience, it's best to prepare yourself for one or the other, depending on which renderer you've chosen to use. If you do choose to use Alice for animation, there are a number of tutorials in Alice involving creating Python scripts in a variety of ways. For more information on scripting in Alice, see section 9.2.5.

11.2. Exporting Keyframe Animations from 3DS

Exporting an object with keyframe animations is done the exact same way as exporting your model. The only difference is when you start JAlice you need to copy the black text at the bottom of the screen and paste it into your behaviors script for the world. Newer versions of JAlice might provide better support for imported animations so check with Stage 3 on the feasibility of what actions you hope to perform. ***Musica*** ultimately did not import any keyframe animations as it was found to be easier to accomplish the simple effects within JAlice. Initially explosions were designed for the blocks but JAlice did not handle this action well due to the structure of the multi-faceted cubes.

11.3. Object Reuse and Resource Conversation

It's important, regardless of what method is used, to conserve graphical resources in your experience. Minimize the number of objects and creations to what is attractive but practical. For instance, if an object is "destroyed" during game play, it is far more prudent to remove the object from view rather than remove it from the game. This way the object may be reused later for another instance. The same goes for animations. If a stock animation can be used for a different purpose with slight modification, create an animation structure that reflects this. Perhaps one block is always used in the same pattern for a certain event. The object may be simplified based on the viewable set of objects in the scene. A crude sort of back-face culling can be applied since the objects

are projected onto a two dimensional surface. Since the view must be identical or very similar from four different viewpoints around the board, 3D animation is rarely of any benefit. This is the sort of mindset that must be employed when working with a rapid-prototyping environment, for the faster and easier it is to build your experience, the lower the degree your implementation can take advantage of the operations you are performing and thusly make for a slower application all together.

12. SOUND

12.1. I Can Hear! (Digital Sound 101)

For a Jam-O-Drum experience, the aural stimulus is just as important as its visual presentation. Sound should be produced often, from both the direct and indirect input of the guest. Themed sound effects should be triggered often, sequences should reward the guest periodically, and consonant samples should be presented for the guest to generate.

Digital sound is an electronic sampling of the acoustic vibration received by the human ear. Without getting into acoustic theory, suffice it to say that CD-quality sound may be reproduced through a compressed .wav file sampled at 44khz in 16-bit stereo. There are three types of sound that you may use in your Jam-O-Drum experience: effects, sequences, and samples.

12.2. Acoustic Magic: Effects

Sound effects are played when certain events occur during the game's play. This includes objects colliding or exploding, lives/balls being lost, or a new hi score being set. There are virtually an endless amount of sound effects that can be used in a game. The key is to provide enough rich and varied effects so that the experience becomes more interesting, but not so much that they interfere with the game play. Too many effects can clog the media pipeline, especially in Alice.

Sound effects are usually stored as .wav files, and less than 300k apiece, depending on duration and quality. They can be created by a variety of means. Sonic Foundry's Sound Forge is an audio editor and can create and modify sounds. A note or series of musical notes can be played by a midi sequencer like Cakewalk Pro Audio and converted to .wav format as well. It's also possible to record effects the old fashioned way with a microphone and a wave recorder. For ***Musica***, sound effects were produced by the Korg X-5 Synthesizer and modified using Cakewalk. All the ***Musica*** effects, sequences and samples are available on the archival CD.

12.3. Writing a Score: Sequences

Sequences are prearranged sets of samples, normally played by the computer's sound card. The synthesizer receives messages from a sequencer, and produces sounds

accordingly. MIDI is a data format for storing such sequences of messages. Though a MIDI sequence is sound card dependent, and not actually digital audio, the output from a played sequence may be recorded to digital audio and later compressed into a MP3 or WMA file.

Sequences are much less frequent than effects, and commonly used at transitional parts of the game, though background music may be played as in the Jam-O-World or CircleMaze application. ***Musica*** plays sequences only at level start or completion, or when the game is idle. There is also a sequence associated when all of a player's balls have been lost.

It's important that the sequences cater to the experience's concept and demographic. Just like a film, the soundtrack must be uniform and engender a particular sort of feeling that the graphics complement. For ***Musica***, all of the sequences are bright and lively, relying mostly on major keys. The duration and complexity of the pieces are also important factors. Since the ***Musica*** experience was intended for a children's museum with a constant flow of traffic, none of the tracks, except for the idle music, are longer than 30 seconds. In fact, the first two level completion sequences are both scarcely more than 15 seconds. They also are limited to eight channels each with simple melody and percussive structures.

12.4. A Poor Man's Music: Samples

Allowing the guest a certain amount of interactivity and music making is one of the key features of the Jam-O-Drum. Since the JOD does have four drum pads, the guest expects a reaction when one is struck, musically if not also graphically. In ***Musica***, this is accomplished by the destruction of the note-blocks on the game board and the playing of samples. A sample is a prerecorded piece of digital audio in a certain timbre (tone color) at a certain frequency (note). Each block in ***Musica*** is a different color and has a musical note (C, D, E, F, G, A, B) associated with it. When the block is destroyed, the note is added to a queue. When a guest strikes one of the drum pads while playing, the sequence of notes is played back in the timbre of the instrument on the guest's pad. The sequence is then cleared and waits to receive a new sequence of notes. Since the Alice media interface does not currently support MIDI messages, samples were made of musical notes. For each of the six possible instruments, .wav

files were generated for each note in one octave. This way, no matter which order the notes are broken in, the resulting sequence is pleasurable.

12.5. Compression and Quality

As with most computer programs, quality and size of the media are inversely proportional. With digital audio, the better the selection sounds, the larger the file will be. Optimization is always necessary, especially with video games, so it's important to think about conservation of your resources when preparing the soundtrack. The sound effects can be of poorer quality (lower frequency, fewer bits, mono) because they are only heard briefly. No sample, however, should be reduced in quality to the point where it has a large amount of noise. Samples can be of middle quality because they're closer to sequences, but still they are short clips played while there are other things going on in the game. The sequences should be of fairly high quality, because they are a reward to the guest and shine through their complex harmonies.

In Alice, this is even more of a necessity than if one were using DirectX to handle the audio. Alice will play .wav and .mp3 files, but each one must be loaded into Alice's before the program is run. For this reason, all of the sequences in ***Musica*** are MP3 files because .wav files are unmanageable if they are longer than a few seconds. Alice decompresses the MP3s during playback and sends the .wav data to the sound card.

Producing versions of varying quality is a good step to take when creating digital audio for your experience. A bit of testing with different combinations of your media can greatly increase both the durability and smoothness of your application.

12.6. Creating Content with Cakewalk Pro Audio

Cakewalk Pro Audio 9 is a multi-track MIDI sequencing application. With it you can perform a wide variety of standard MIDI effects and modifications, as well as record performance and script. Cakewalk was used for all of the ***Musica*** effects, sequences and samples. It's available on all the PCs in the recording studio. As a reference, there is a lengthy user's manual in PDF format on the computers as well.

12.7. The Polyphony Sound Manager

There are many ways to handle the playing of audio in your Jam-O-Drum application. Depending on how advanced your needs are, it's possible to leave most of the grunt work to Alice and the JMF. The Polyphony Sound Manager is a very simple Python

class that makes sound programming quite easy. An instance of the sound manager is declared in the world script in Alice. It has internal procedures for playing individual sounds for certain points in the game. It also has the power to stop any sound that is currently playing. The interface for the game programmer is exceptionally simple. Whenever a sound event is needed in the game, the controlling application (either from Python or Java) calls the polyphony.handleSoundEvent(eventType, sampleType = "none") function with either one or two parameters, depending on the event. The parameter tells the manager what sound to play, and gives it extra information if necessary (like which musical instrument to use). If a sound needs to be stopped, the controlling function simply calls polyphony.stopSound(eventType), where eventType is the sound event to be stopped. The only other work needed is to have the sounds declared in the Alice world before use. Polyphony handles both .mp3 and .wav files. For a more detailed look at the Polyphony sound manager, please see the ***Musica*** JAlice world script, **script.py**.

In ***Musica***, the sound system is restricted to the four speakers in the table and the subwoofer underneath it, all wired in mono. If a more complicated setup is required, it is possible to make use of a simple mixer to route selected channels to different parts of the room. For even more advanced spatiality, Dolby 5.1 is supported by both the SB Live! card and DirectSound. This does, however, involve a substantially larger amount of programming.

13. USABILITY

13.1. The Value of User Testing

User testing such an important phase of your cycle that it deserves its own committee. Since you're most likely developing for a demographic outside of your own, every design decision you make must be validated in user testing. What may seem as natural to you may be completely undiscoverable by the next person. In creating an experience like the Jam-O-Drum, it is clear there will be a wide range of guests interacting with the application on a daily basis. User testing not only answers questions about the ease-of-use of the experience, but also the theme and fun. You will learn very quickly, especially with children, what is and is not entertaining to your audience. The smallest mundane feature in the experience may turn out to be the element that your guests find most intriguing. This is extremely valuable information, and you will need to modify your application to reflect your discoveries. Like all other elements of your cycle, user testing must be carried out with process while keeping in mind a certain level of gravity.

13.2. User Case Scenarios

User case scenarios are a good place to start. The software engineers on your team will probably want to create several diagrams simulating the program flow. You can make good use of these and adapt them slightly to reflect the same data from a user's viewpoint. Walking through the experience, paying close attention to the input and output from the application will be quite fruitful, allowing you to analyze possible weak points in the interface. At any moment the user sends input to the computer (like hitting the drum, turning the pad), the user is going to expect a certain response from the computer. You need to make sure that the response the computer issues is within the expectations of the user. Conversely, when the computer is displaying data looking for an input, you predict what sorts of cues (aural or visual) the user would react to. This is designing for your audience.

By studying user case scenarios, you can discover potential pitfalls in the interface, and know what to expect when conducting a user test. The more difficult side of usability is expecting what the guest will enjoy. Children have a short attention span and will abandon your experience in a heartbeat if it is too hard, too undiscoverable, or too

boring. This can only be helped by rapid prototyping and getting a workable demo out to your user-base early on, perhaps in the form of a throwaway prototype, or a slightly modified previous cycle's experience that embodies your theme and concept.

13.3. Planning User Testing

13.3.1. How Often and When?

Planning user testing is not something that should be put off, mostly because you won't have a lot of experience in it and things will probably change quite a bit before you're finished. User testing should be a consideration from day one of the project and not something shoved in at the end. While the other committees are beginning development, user testing can be planning case scenarios and specific tests with particular goals in mind. Ideally, development can produce a throwaway or evolutionary prototype shortly after mastering the tools. Preliminary user testing can be conducted then to determine the feasibility and value of the major features. It may be discovered that the crux of the experience is less well received than was initially predicted. User testing should also be scheduled a for a week or so before the cycle is over, so final revisions may be made to tailor the experience to the intended audience before release. These estimates are for a seven-week cycle, and should be at least doubled for a fourteen-week cycle; even a third iteration may be necessary. Fortunately test users are easy to find for young demographics.

13.3.2. The User Base

Obtaining users for testing is not a difficult task, but one that must be done early and not at the last minute. Several of the professors in the department have connections to scores of children, whether their own or people who work in childcare and education. An elementary school or a day camp can be a veritable gold mine of information. Children are great testers because they are completely honest; they'll let you know when they don't like something.

13.3.3. Designing the Test

13.3.3.1. Asking the Right Questions

While you can learn a lot from just running the program and letting your guests roam free, it's good to have some structure to how things will be run before you start. Some things to map out are:

- How many users you will have per test and total (1, 2, 3, 4?)
- How long each test group will have with the JOD
- How much (and what) information you want to provide to the group before letting them start.
- How soon you want to step in and help users having trouble.
- What questions do you want the test to answer?

This last bullet is very important. Usability testing is a science, and there are many formal methods of discovering certain types of data from a test. It is wise to put all of the things you want to know in the form of a question, and to make them as quantitative as possible. This is hard, but when surveying the users after the tests, the less room you give them to wander off the topic, the closer to the point they will be and the more valuable the data. For example, say you wanted to find out if a user liked the color scheme on the Jam-O-Drum.

- Are the colors in the game good? (weak)
- Do you like the blue wheel? (good)
- Which do you like more the red wheel or the blue one? (best)

Targeted questions are a good way to extract binary responses from your users. For broader topics, the subject can be wider, but it's best to stick to an optometrist-style line of questioning ("Is that better or worse?").

13.3.3.2. Concept Testing

If you're doing feature surveys or concept tests early on, it may be beneficial to open up the questioning a bit more, and let the user help design the interface. For example, if you were going to build a game that let children put together robots from a set of parts and buttons, you may cut out paper versions of the widgets you would display and let

the users arrange them as they like. By noting path the users take to assembling the interface, and the results they produce, you can gain the perspective you need to design an experience your audience will not only understand, but also enjoy.

13.4. Conducting the Test

When conducting the test, it's important to stick to your schedule and be sure to have the entire team around to assist the experience and watch. Someone should handle surveying the users before and after the test. Another person can keep team and handle the logistics of cycling groups in and out, and giving the necessary background to the guests. While one team member records the data and takes notes, the rest of the team can support the system and help the users when needed.

13.5. Data Mining

After it's all done, a substantial report should be compiled cross-referencing the groups, removing errant data, and searching for trends. If you conduct a thorough and well-planned test, you can gain a lot of insight into the thought and learning patterns of your user base. Key things to look for are times to master the controls, recurring examples of users getting lost or confused, and moments when the user expressed exceptional joy or satisfaction.

Aside from your own personal use, conducting a public user test study and inviting other project teams to visit can be beneficial to all. The data extracted from your user test may be of value to not just future Jam-O-Dream teams, but any team that shares a similar concept or demographic. It would be prudent to do a thorough job documenting your usability test and results, and then publishing them locally for communal profit.

13.6. Revision

When your user test results are processed, you can begin making revisions to the design to improve the interface. Instead of simply distributing the results to your group, hold a group meeting with your advisor and go over the results together. As a team, you can decide if the concept needs modification, and discuss the feasibility of revisions to the visuals, music, or program behavior. Not all user complaints can be brought into accordance, but by resolving similar problems and unifying the changes to maintain that the application stays true to the experience, you can greatly improve the overall user rating for your next iteration.

13.7. More on Usability

This chapter doesn't even scratch the surface of what should go into a commercial product like the Jam-O-Drum. It is strongly recommended that you study more about the subject matter outside of this manual. Usability testing is a large field with a lot to learn about making the most of user feedback. There is much literature on the topic of usability, user testing, and data mining. See the appendix on recommended reading for more information.

PART III:
BEYOND THE CYCLE

14. DEDICATION TO CONTINUITY

14.1. Time Well Spent

It's not a secret that project cycles are insanely short. Corporations spend twelve to eighteen months to release a professional-quality product for distribution. You have seven to fourteen weeks. Admittedly, there are significant differences in scale and market, but a damn good app is a damn good app. The Entertainment Technology Center is petri dish for novel experiences. Every project that's issued to a group is done so in hopes that it turns into something commercial grade, something worth selling. This is, after all, a technical program. If the opportunity came along to sell your app along with a copy of the custom hardware, well…that's the next chapter, but back to the topic at hand.

Project cycles are short, very short, and that's why it's imperative that every group coming into a cycle needs all the help it can get. There is no quicker way to kill the chances of a great idea breaking out of Doherty than a lousy handoff. For this reason (and the divine will of the software engineering gods), **it is your ethical responsibility as a member of this department to produce comprehensive documentation. Period.**

14.2. Document Everything

14.2.1. The Growing Experience Compendium

This manual was devised so that any one person could pick it up and learn virtually anything about the Jam-O-Drum (and its project management) they could possibly desire. Every question that came up and all the materials that were used during the ***Musica*** cycle are in included in this document and its accompanying CD. In a perfect world, you wouldn't need to contact any of the previous experience professionals for assistance in working something out. But this isn't a perfect world or a perfect document, so use the contact information in the appendix if you're lost.

As detailed as this document is, it is not all-inclusive. New problems will be unearthed, new cycles will take different software approaches, and new shortcuts will be discovered. Thusly, it is important that all future teams keep track of their progress in a similar manner. New schematics and drawings will be drafted, class structures and

flowcharts will be produced. All of these materials should be combined along with the requisite history of the committees' development and decisions. For the last week of the cycle, those materials should be assembled and formed as a supplement to this manual. Over time, the supplements will accrue and a veritable epic of the Jam-O-Drum will propagate.

14.2.2. Digital Resources

In addition to the project manual, it's important to include all of the artifacts and tools from a cycle onto a data CD for future reference. The accompanying disc for this manual contains a wide assortment of concept and production materials, including:

- final Alice World
- final SourceSafe Database
- login information for SS database
- all 3DS files
- all DeepPaint files
- all bitmaps (textures)
- all screenshots/images
- all music
- a copy of the finished website
- a copy of the manual
- a copy of the development version of Alice (4.22.01)
- all necessary SDKs

14.3. Evolution of the Jam-O-Drum?

Perhaps in time, with the aid of these materials, there may come a point where the process becomes so well documented and easy to perform that perhaps building experiences can be relegated to classes of younger students. The Jam-O-Drum development challenge could be used as college undergraduate or high school projects as a study in rapid prototyping and developing interdisciplinary experiences. Imagine high school students, apathetic about computer science and technology, now encouraged to work on an assignment about making their own custom video games, their own multimodal experiences. Early on, young programmers realize they cannot perform this task alone and need artists, musicians, and playwrights to help them

produce a compelling experience in half a semester. Half a dozen groups in this new interdisciplinary class compete, each building their own Jam-O-Drum application and registering for time slots to test their creation on the hardware. Getting people to work together and respect each others' talents earlier in education can only aid in accelerating the critical collaboration and communication skills needed for tomorrow's generation.

15. JAM-O-DRUM FOR SALE

15.1. The Price Tag of Reality

The Jam-O-Drum device is an expensive item. The Jam-O-Drum experience is a very expensive item. Hundreds of man-hours go into the creation of a simple, interactive system that lasts little more than ten minutes start to finish. But the value that comes out of the apparatus is priceless. When contemplating the sale of a Jam-O-Drum experience, don't think of it as a custom video game, think of it as an identity.

The Jam-O-Drum is a unique piece of equipment unto itself. That, coupled with the fact that the applications it runs are compelling, immersive, collaborative, and even a little educational make it a very hot item. This chapter attempts to leave the realm of innocent humanitarianism that the Jam-O-Drum was designed around and provide a perspective on the assumed physical value of a system if one so chooses to "sell out" (no pun intended).

15.2. Materials and Labor

The following chart lists the purchase price of all the equipment that is used to create a Jam-O-Drum experience. Consumables are included as well. All charts in this chapter are available on the supplementary CD in .xls format.

(Please see the attached spreadsheets in Appendix G.)

APPENDIX A: MATERIALS PURCHASING PROCEDURE

Like any professional project, you're going to need supplies. Software licenses, hardware, and project materials are all needed for the development of your experience. The technical coordinator is your gateway to procurement, but there are some guidelines to following when requesting materials:

1. All purchases under 100$ may be approved directly by the technical coordinator. Any purchase amount greater than 100$ and less than 1000$ will require the directors approval. Any individual purchase over 1000$ requires a request to the budget office, which will research externally the best vendor and equivalent model (if possible). It's best to try and avoid this last scenario all together, because the process employed for purchases over 1000$ is slow, inefficient, and likely to end up producing something other than what you really need. It could very easily take weeks. For this reason, you need to employ some creative thinking to get the equipment you need.
2. Purchase requests (unless petty) must be submitted in advance to the technical coordinator via email.
3. When submitting a request for materials, the report must detail for each item: the justification of the item, the price of the item, and the vendor and contact information for purchasing the item. A running total of all the items in the request must also be tallied.
4. For petty impulse materials (like electrical tape, solder, etc.) a receipt must accompany the submission to the technical coordinator for reimbursement. Tax will not be reimbursed on any impulse purchase by a team member as Carnegie Mellon is tax exempt.
5. Before making any request for purchase, first check to see if materials are already available. There are a large number of surplus materials in the ETC for use ranging from keyboards, mice and joysticks, to markers, paper, and tools.

APPENDIX B: PREVIOUS JAM-O-DRUM TEAM MEMBERS

Project/Cycle	Name	Committees	Email (cmu.edu)
Musica Fall '01-1	Dan Schoedel	**Modeling/Painting**, **Animation**, Hardware	dschoede@andrew.
Musica Fall '01-1	Dave Ventura	**Music**, Programming, Hardware	dventura@
Musica Fall '01-1	Ingrid Moncada	**Programming**, music, hardware	icm@andrew.
Musica Fall '01-1	Ray Mazza	**Hardware,** programming, animation	rmazza@andrew.
Musica Fall '01-1	Ying-Tzu Lin	Modeling/Painting, Animation, Music, Hardware	yingtzl@andrew.
Musica Fall '01-1	Frank Garvey	**ADVISOR**	fgarvey@andrew.

Project/Cycle	Name	Areas of Expertise	Email (cmu.edu)
Jam-O-World Spring '01-2	Cliff Forlines	Programming, User Testing	forlines@cs.
Jam-O-World Spring '01-2	Kevin AuYoung	Animation, User Testing, Interface Design	auyoung@andrew.
Jam-O-World Spring '01-2	Wil Paredes	Programming, User Testing	paredes+@andrew.
Jam-O-World Spring '01-2	Tina "Bean" Blaine	**ADVISOR**	sabean2@earthlink.net

Project/Cycle	Name	Areas of Expertise	Email (cmu.edu)
Jam-O-World Spring '01-1	Cliff Forlines	Programming, User Testing	forlines@cs.
Jam-O-World Spring '01-1	Philo Chua	Interface Design, User Testing	pchua@andrew.
Jam-O-World Spring '01-1	Rebecca Crivella	Interface Design, User Testing	crivella@andrew.
Jam-O-World Spring '01-1	Wil Paredes	Programming, User Testing	paredes+@andrew.
Jam-O-World Spring '01-1	Tina "Bean" Blaine	**ADVISOR**	sabean2@earthlink.net

APPENDIX C: RECOMMENDED READING

Software Engineering/Project Management

The Mythical Man-Month. Frederick P. Brooks.

An Integrated Approach to Software Engineering. P. Jalote.

Game Programming Gems (series). Various authors, Academic Press.

Human-Computer Interaction/Usability

Bringing Design to Software. Terry Winograd, ed.

Usability Engineering. Jakob Nielsen.

Computer Graphics

Computer Graphics: Principles and Practice. Foley and van Dam.

Advanced Rendering Techniques. Watt and Watt.

OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2. Mason Woo et al.

Graphics Gems (series). Various authors, Academic Press.

Electronic Music/MIDI

MIDI: A Comprehensive Introduction. Joseph Rothstein.

The Complete Guide to Game Audio: For Composers, Musicians, Sound Designers, and Game Developers. Aaron Marks.

Cakewalk Pro Audio 9 User's Guide. Twelve Tone Systems Inc.

ACM

A copious archive of published research papers on computer science is available at the

ACM Digital Library, **http://www.acm.org/dl** (***Highly*** Recommended).

APPENDIX D: SAMPLE EXPERIENCE CONFIGURATION FILE

```
javapath c:/JamODrum/jre/bin/java

# Initial heap size

vmparam -Xms32m


# Maximum Heap size

vmparam -Xmx256m


# Enable incremental garbage collection

vmparam -Xincgc


# python path

vmparam -Dpython.path=c:/JamODrum/jython-2.0/lib

addpath classes


# JRE standard and extension jars

addjars c:/JamODrum/jre/lib

addjars c:/JamODrum/jre/lib/ext


# JAlice jars

addjars c:/JamODrum/lib

addjars c:/JamODrum/externalLib
```

```
#########################################################################
###

# To use the MUSICA WORLD:

# For main class, use either:

#                 edu.cmu.etc.jamodrum.musica.Driver                 or
edu.cmu.etc.jamodrum.ZeumControlPanel

# (To ensure that the joystick inputs work, use ZeumControlPanel
better)

# Either Driver or ZeumControlPanel use the following parameters:

# Path and name of the Jalice world

# Indicator to enable ball falling off the board

#    0 -> The balls will bounce in the edge of the boards as if we had
4 walls # (this is for testing)

#    1 -> The balls will fall off the board like the original game

#

# Number of balls

#

# Example:

# mainclass edu.cmu.etc.jamodrum.ZeumControlPanel "C:\MusicaAlice\Jam-
0-Alice" # 0 2

# will play the game with 2 balls and with the balls bouncing
everywhere

#########################################################################
##

mainclass edu.cmu.etc.jamodrum.ZeumControlPanel "C:\MusicaAlice\Jam-0-
Alice" 1 1 0
```

APPENDIX E: TROUBLESHOOTING

3D Studio MAX

- **Problem:** Why is the applied bitmap not at the right angle, and not on the right face of the object?

  Solution: First, after assigning the texture to the object, make sure that you check the "Show Map in the ViewPort" button in the "Material Editor" to show you the object with texture in your 3D max window. Then, apply the "UVW Mapping" under "Modify" manual to the selected object.

- **Problem:** When I apply a bitmap in 3D max and check the "UVW Mapping", why still can't I make the texture image of the right size and fit the object surface?

  Solution: After you applied the UVW Mapping function to the object, click on the "+" sign of it. That will unfold its sub-function "Gizmo". It will turn yellow when you click on, then you can apply any transformation, like uniform scale, un-uniform scale, or squash, to the bitmap.

JAlice

- **Problem:** Why when I import my ASE file into the JAlice, I can't see anything and my JAlice is stuck there.

  Solution: You can't import your entire 3D Max model in the same time. That will crash your JAlice. All you need to do is save each object in the 3D Max scene as an independent ASE file. Then import these files one by one.

- **Problem:** Why is my texture memory in the JAlice is so big?

  Solution: When you import ASE file of each object into JAlice, you think you import the selected object independently, but actually all the textures or bitmaps in the whole 3D Max file are imported all together. So, to cut down the texture memory in JAlice, try to save each object you want to import as a new .max file first, then export it to .ase before import it to your JAlice world.

- **Problem:** What can I do if I already have bunches bitmap texture in my JAlice world?

  Solution: Unfortunately, all you can do now is to check all objects on the left side column of you JAlice window, click into the TextureMaps, and delete all unneeded TextureMap on the right part by right click and select delete.

- **Problem:** Why I can't find my objects when I import them into JAlice and lose all my animation I make in 3D max?

  Solution: Before you import any of your objects, make sure that you have already reset its pivot point to its center point. Then also check the Manual "Utilities" that you can find on the right top part of your 3D Max window, select the object and click "Reset XForm" button and then check "Reset Transform" appearing below. As to the animation, after you import the animated 3D max object into JAlice instead of creating your own animations in JAlice, copy all the scripts in the bottom window to top "World Script" window in your JAlice. And trigger it by choosing the event in "Behaviors" window, and type the name of your original animation in the "Response" column.

- **Problem:** I'm getting an error in Alice's behavior script, and all I did was comment something out.

  Solution: Usually this means you've commented out the body of a function and left on the top line (the def foo():) which results in an error. Try filling in the body with a placebo command, like foobar = 4. Or you may have commented out something else that Python does not like.

- **Problem:** I press "Play" in Alice and a gray screen pops up, nothing more.

  Solution: This is not your fault. Just save your work and restart Alice. Things will then work again.

JBuilder

- **Problem:** I am trying to open a Jam-O-Drum project in JBuilder, and when I compile, I get lots of errors. Shouldn't it be left in a working state?

  Solution: Yes, it should be left in working order. However, when JBuilder projects are moved to a different computer, the libraries need to be set up again for the project. Activate the menu option "Project→ Project Properties" and a box will pop up. Under the "paths" tab, click on the "Required Libraries" sub-tab. Click on "Add". Click on

"New". Make up a name such as "JAlice External Lib" and then show it the "External Lib" folder in the Jam-O-Drum folder. Finally, click "Add" again. Then repeat this process for the normal libraries, which are located in "Jam-O-Drum/Lib".

The Projector

- **Problem:** The projection doesn't fill the whole table.

  Solution:

  Step 1: Zoom the projector out.

  Step 2: Move the projector to the distance from the table where it's projection just starts to show over the edges of the mirror.

  Step 3: Consult the Hardware section on the Mirror and Projector.

- **Problem:** The projection is warped.

  Solution: Consult the Hardware section on the Mirror and Projector.

Hardware Input

- **Problem:** Direct X cannot "acquire" the turntables encoded as joysticks through the game port.

  Solution:

  Step 1. Make sure the circuitry is plugged in via the +5V DC power supply.

  Step 2. Double check to make sure all wires are connected to the correct locations. Are pins 1, 3, 6, 8, 9, 11, and 13 wired to each other? They should be if they are not.

  Step 3. Use a continuity tester (or the continuity test function on a multimeter) to make sure all solder connections are good. See the section on soldering for instructions on how to do this.

- **Problem:** I can't get input from the drum pads.

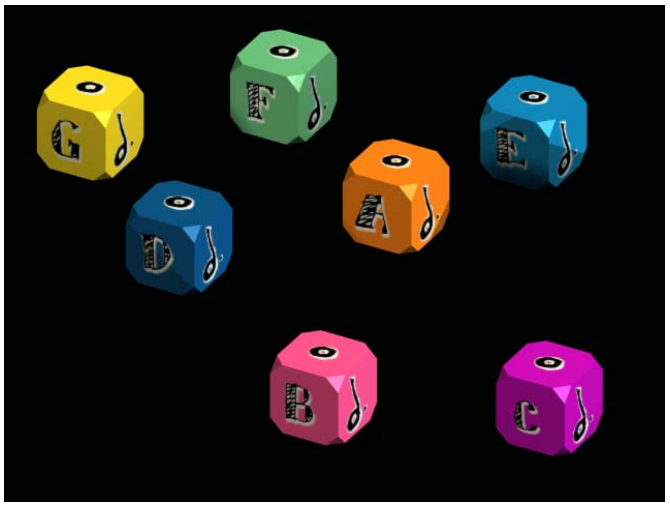
  Solution:

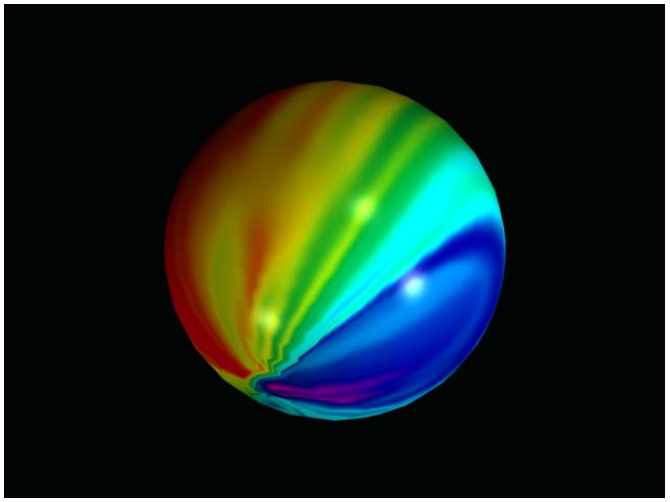
  Step 1: Make sure the pads are plugged into the drum module, and that the drum module is powered on. Also, make sure the drum module is plugged into the MIDI port on the computer.

Step 2: Configure the program to use the proper sound device (in edu.cmu.etc.jamodrum.MIDIInput.java). There is a choice of using either the "SB Live! MIDI UART" sound card or the "MPU-401" on the mother board. Use the SB Live! (unless you are not using a Sound Blaster audio card).

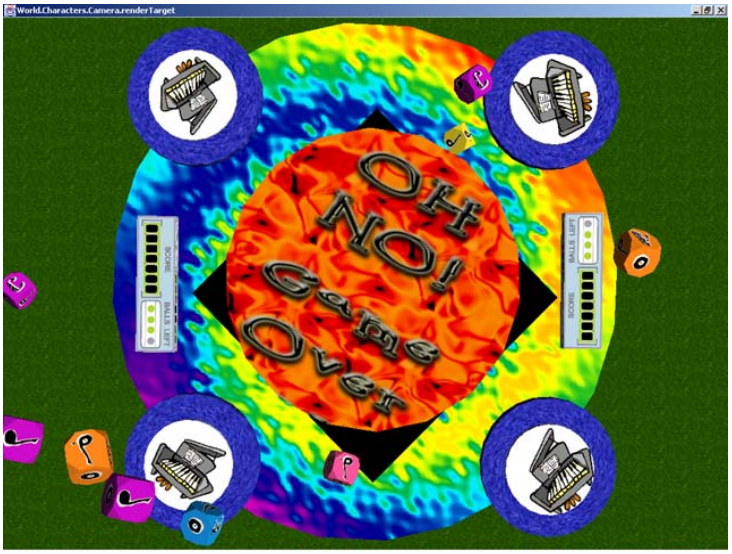
Step 3: Put print statements in edu.cmu.etc.jamodrum.NoteEventObject's constructor to print out the note it makes every time it is called. Hit the four drum pads and record these four notes. In edu.cmu.etc.jamodrum.midiinput.PadInput, change the constants declared for each pad to the notes you just detected. Once this is done, do not change any of the settings on the drum module except for the volume, as doing so may change the notes output by pad hits.

APPENDIX F: MUSICA SCREENSHOTS






Good
JoB!
Next
Level


World.Characters.Camera.renderTarget
OH
NO!
Game
Over


World.Characters.Camera.renderTarget

APPENDIX G.
DIAGRAMS AND SCHEMATICS