Video Games and Software Engineering: A Case Study

Matthew Ginley, John Bowles, and Caroline Eastman

Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208

E-mail: ginley@mailbox.sc.edu

ABSTRACT

Video games, due to their unique nature and relative infancy compared to other software, are just recently being studied with a strong engineering aesthetic like their traditional counterparts. In this case study, we examine the benefits of using a planned, engineered approach to creating a fun and interactive piece of software. This research centers on an original game, developed by the author. Established object oriented software engineering was applied at every step of the development process. End-user testing will measure the game's fun factor, with results being analyzed to determine the relationship between the engineering behind the game and its final playability.

INTRODUCTION

Many possible game ideas were considered. A Japanese language educational game, a top down 2-dimensional fighter, and a traditional role playing game were considered. The author settled with a music game based on drumming and the coordination issues behind the concept of rhythm. Such a game proved to be unique, require a high degree of user interaction, and provided many design elements to examine.

Ultimately, the author settled on this game idea because of the opportunity for intense rhythmic capability not possible with playing a real drum set. This is because in mapping a drum set to the keyboard, all the percussive input comes from your fingers. At first thought, this seems very obvious, but one must contrast this mode of input with that of a real drum set, where input is obtained from four appendages. One may have greater independent control over their arms and legs, but the number of sounds that can be played at once is four, and switching between components will slow down the speed at which various beats can be played.

With a keyboard, the situation changes dramatically. There is less independent control over ten fingers, but one can play up to 10 sounds at once! Also, the distance a finger must travel to strike a different key is a few millimeters compared with a hand moving many feet on a drum set. To put it simply, on a drum set, a player is a human with four slow, but better controlled sources of input, while on a keyboard a player is a less coordinated, lightning fast 10-arm "decapus."

Drum Simulator Specification

The game, which is currently under development is a single player game titled "Drum Simulator." The key concept behind the game is mapping the sounds of a drum set onto the keyboard in a fashion that provides flexibility for players of different abilities. This aspect of the user interface is crucial to the success of the game. Beyond that, the game is broken up into two different modes of play: the first is a practice mode and the second is the real game play mode where the user listens to a drum beat and then must play it back.

The first mode, called *Free Play*, allows the player to play the keyboard layout without timing or scoring constraints. It's essentially an empty game, but it enables you, the player, to become familiar with the control scheme. The second mode, the real game play mode, called *Play That Beat*, is based on rhythmic call and response.

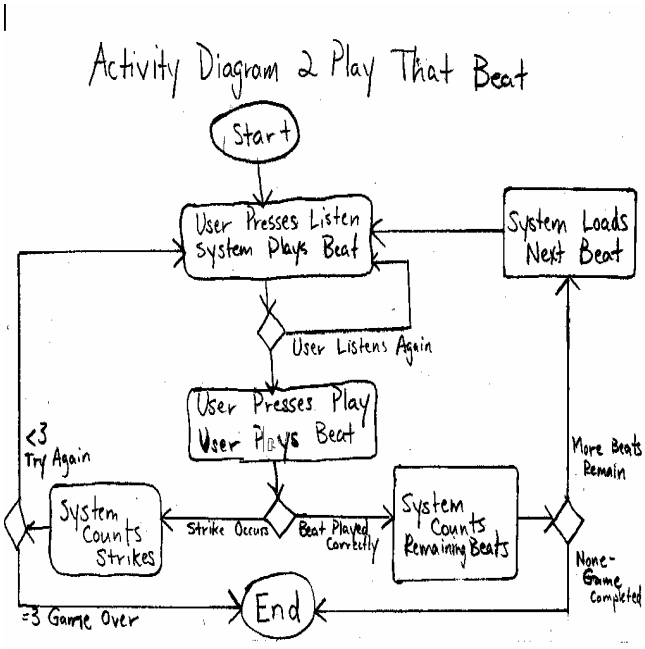

Figure 1 Play That Beat Activity Diagram

Figure 1 is a UML activity diagram of the real game play mode. Note the two different ways the game can come to an end.

When the player is ready, the game outputs a drum beat (audibly and visually) lasting about 5 seconds. Then the player is expected

to play that same beat on the keyboard without mistake. If a mistake is made, the player must try again. If played correctly, the game loads the next beat (the beats become more difficult as the game progresses) and the player listens and plays again. The game ends when the player has either made 3 mistakes, or completed all the beats included with the game. Score is kept according to the progress and how many mistakes have been made.

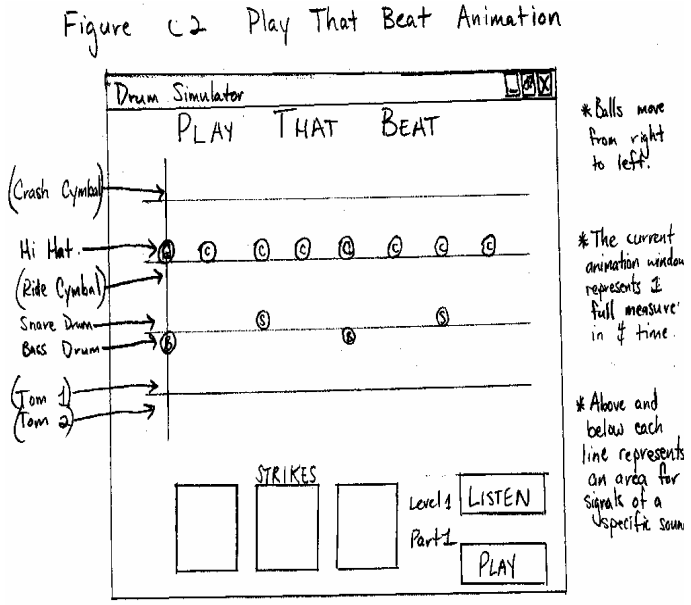

Figure 2 Play That Beat Game Screen

Figure 2 is a screen layout of the second game mode. There are the Listen and Play buttons described earlier, and three boxes to count the strikes, or mistakes, committed by the player. The four horizontal lines represent certain sound domains. Along these lines, markers move from right to left, indicating a sound that needs to be made. The marker striking the left hand side represents the exact moment when the user needs to hit the corresponding key on the keyboard. The top line is for the crash cymbal markers, the second for the closed hi-hat and ride cymbal, the third for the snare drum and bass drum, and the bottom most line is for the tom 1 and tom 2 sounds. The width of the command lines represents one full measure in 4:4 time signature. The screen shown in Figure 2 indicates the standard rock drum beat.

ENGINEERING OVERVIEW

Adhering to a software engineering process has helped game development significantly, and currently the game is in the coding and implementation phase. The planning tasks suggested by (Rucker 2003)—making a schedule, establishing milestones, and predicting risks—have proved invaluable. The constraint triangle was presented first and foremost. The notion of adjusting quality, time, and cost in direct relation to each other helped provide a limit to the number of features in the game specification. With the project time and cost well established from the beginning, the quality (complexity) of any game to be developed had to be limited.

With respect to scheduling, this project would not be where it is if it were not for creating a schedule document with appropriate milestones. The major project milestones were: 1) create game concept; 2) write project abstract; 3) collect requirements; 4) develop specification; 5) illustrate design; 6) code game alpha builds; and 8) write conference paper. Ironically, this project might even be further along if the risks the author identified during the initial planning had been better avoided. The project's primary obstacle at this point is the complexity of the Pop game framework that accompanies (Rucker 2003), and this risk was explicitly identified in the project's initial schedule document.

An undervalued recommendation from (Rucker 2003) was defining the game appearance prior to implementation and coding. Defining the game screens ahead of time provides insight as to what graphics setup will be needed. With the many choices floating around such as Windows MFC, OpenGL, DirectX, or another, knowing the graphical complexity of a game is essential to deciding which graphics package will be used. With Drum Simulator, the visuals are no more than small circles sliding across the screen, so the less powerful and less complicated Windows MFC graphics were chosen.

This recommendation extends to sound in the same fashion. With drumming, many different sounds are played at high speeds and quite often two sounds at the same time. These situations had to be handled in Drum Simulator, or the playability of the game would be ruined. The game would not work if the audio output does not keep up with the player's fingers or if the game could not output a snare and crash cymbal sound at the same time. Thus the innate sound capabilities in the Win32 API were insufficient. The API functions for playing sound are not that fast, and not capable of playing multiple sounds at once. With this audible complexity, the DirectSound API of DirectX was chosen for handling audio output in Drum Simulator (McShaffry 2005).

User Interface Development

A central discovery of this project was the importance of the user interface and control scheme with respect to the game. Morrison (Morrison 2003) emphasizes that a well thought out, intuitive control setup adds immeasurable substance to a game. A game's fun factor is based in large part on the user interface. After all, a game differs from other software in that it is meant to be fun, and a user engages in this sensation through the interaction. Thus, a focus of the engineering (and this project as a whole) was on the development of the user interface.

The first step was to index every component on a drum set. For each component, the following characteristics were examined: which hand(s) play that component, how many sounds that component typically produces, the rhythms typically played on that component, and the usual speed at which that component is played. For example, the snare drum, a very central part of a drum set, is played with both hands but on many beats just with the left hand, it typically produces two sounds (regular strike and

a rim shot), and a variety of rhythms and speeds are used when playing it.

All of these considerations have an impact as to which keys, and how many keys should be assigned for each drum sound. After cataloging the typical drum set components, 14 different sounds needing 22 keys on the keyboard were established. This number is definitely unacceptable—there is no way people can play a game that requires 14 different sounds. Those numbers were reduced to the essential seven sounds on ten keys. Seven sounds is still a large number for the average person, but considering the game relies heavily on only four central sounds (closed hi hat, snare, bass, and crash cymbal), this number will do.

Before placing the sounds on a keyboard layout, some game considerations should be considered. As mentioned, this game provides an opportunity to play some outlandish drumming by taking advantage of the ten fast fingers over four slow limbs concept. To fulfill this ideal, extra keys may be added to the game for some components. The reason is very simple, for if there are only two keys assigned to snare drum sounds, then you can only play snare drum sounds as fast as you can coordinate two fingers. If, however, in the pursuit of a fun, but unrealistic (on a real drum set) opportunity, the game provides four or more keys, the player can then use that many more fingers to play snare sounds. Granted this takes more coordination to use more fingers, but that provides a depth to the user interface and the game that can be taken advantage of by more expert players, while easily ignored by beginning players who can only handle one or two keys. This situation is maximized with the bass drum, where the bass drum is usually played with one foot, or two for advanced drummers, and even still the feet are much less coordinated than a person's arms. By providing four or more bass drum keys, awesome bass drum patterns become possible, possibly making traditional bass drumming in reality seem boring.

After cataloging drum components, totaling possible sounds, and adding key totals, the final step is to map the sounds to the keyboard. The more central parts of a drum set, such as the snare drum and bass drum, were placed first to maintain their priority, with less used sounds placed on more distant keys (farther from the G and H keys). Exactly like on a drum set, the two snare keys are placed dead center, offset to the left by one key to accommodate things on the right. With the snare drum sounds placed on keys F and G, the toms go above that. Tom 1 was placed on keys R and T; Tom 2 following on keys Y and U. The hi hat key was placed on the right of the snare drum for game purposes, as in real life it is on the left which presents a problem with crossing over fingers on the keyboard. The hi-hat was placed at H, the ride cymbal at J, and the bass drum, appropriately at B.

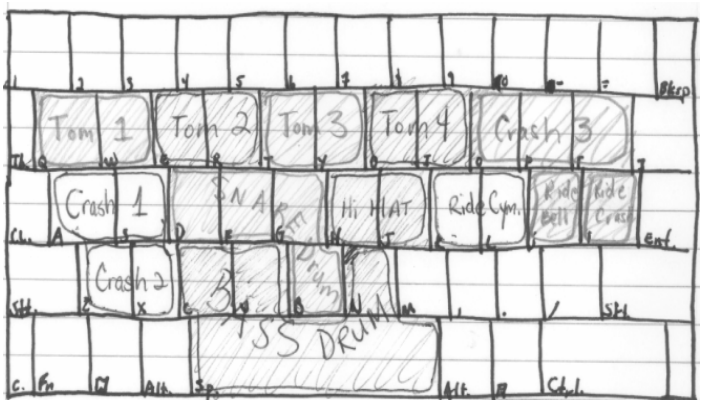

Figure 3 Key Assignments for Game Controls

The mapping shown in Figure 3 is the culmination of the drum set to keyboard analysis. Every component sound was placed accordingly, and with the correct number of keys. The centrality of the snare, bass, and hi-hat keys is important, as these sounds will be used the most in the game. This reflects the nature of learning rock drumming, where most beginning drum beats use only these three sounds. Another important aspect is the alignment of the toms, above the snare drum and arranged in sequential order. Also, the ride cymbal is placed on the far right, as it is rarely played with your left hand.

RESULTS

This research is a work in progress in the sense that the complete game with all of its planned game modes is unfinished as of this writing. In addition, more end-user testing, more interface evaluation, and more internal optimization still need to be done. In order to extend this research further, the case study could examine developing a game that is mouse, trackball, joystick, or mouse and keyboard controlled.

However, the case study has shown the effectiveness of using software engineering techniques to improve the efficiency of video game development. To conclude, the following results are enumerated:

- Specifications can quickly become outlandish. Keep your deadline and the complexity of the game in mind when scheduling.
- Implementation of games is especially difficult due to heavy graphical and timing issues. Maintain rigorous scheduling, cut features if necessary.
- The reality of the game model can interfere with the final game fun factor. Always sacrifice game "realism" for entertainment value.
- User Interface, User Interface, User Interface. Do not proceed a step beyond requirements without heavy consideration for the User Interface and control scheme. A single key function can make or break a game.

ACKNOWLEDGEMENT

This work was done as part of the Research Experience for Undergraduates in Multidisciplinary Computing project at the University of South Carolina and supported in part by the National Science Foundation (award #0353637).

REFERENCES

McShaffry, Mike. 2005. *Game Coding Complete, 2nd Edition*. Paraglyph, Phoenix, AZ.

Morrison, Michael. 2003. *Teach Yourself Game Programming in 24 Hours*. Sams, Indianapolis, IN.

Rucker, Rudy. 2003. *Software Engineering and Computer Games*. Addison-Wesley, Harlow, UK.