An Experiment Comparing Easel with Pygame

Josh Archer, Bryant Nelson, and Nelson Rushton (nelson.rushton@gmail.com)
Computer Science, Texas Tech University, Lubbock, Texas, USA

Abstract – *A framework for developing games using a functional language was designed and implemented at Texas Tech. After it was created, a study took place. It consisted of taking a group of developers with the same task, splitting them in half and giving one group our system, and the other group a well known system. This paper describes the experiment, results, and future work.*

1 Introduction

Easel [Nelson 2014] is a framework for creating real time games by defining pure functions. It was designed principally for the purpose of game programming for math education. An easel game is created by defining the following types and functions in the functional programming language SequenceL [Cooke 2008]:

- *State -- a structure type whose instances are possible states of the game*
- *initialState() -- the starting state of the game*
- *images(S) -- If S is a state, images(S) is a sequence whose members are the images to be displayed in the game window when the game is in state S.*
- *sounds(I,S) -- a sequence of sounds played when input I is accepted in state S.*
- *newState(I,S) -- the new state resulting from accepting input I in state S.*

Given a file containing definitions for the types and functions above (and any helpers necessary), the game algorithm runs as follows until interrupted (typically, by the user closing the game window). Note that the PlayGame algorithm is implemented as a fixed C# program to be linked with SequenceL code written by the student/developer.

Algorithm PlayGame:

State variables:

```
  S: State, C: Click, K: list<char>, I: Input,
lastFrameTime: time-in-seconds
```

Procedure:

```
S := initialState()
while True:
set lastFrameTime to the current time
flip screen display to images(S)
```

If the left mouse button has clicked downward since the last frame, while the mouse was positioned in the game window, store the mouse position in C; otherwise set C equal to (clicked:false).

```
Set K equal to the list of depressed keys
I := (C,K)
play all of the sounds in sounds(I,S)
S := newState(I,S)
Pause until currentTime >= lastFrameTime + 0.0
```

This paper describes an exploratory study designed to test the ability of new game programmers (who are not new programmers) to develop simple games using Easel. The experiment is described in Section 2. The observed results are reported in Section 3, and Section 4 describes new hypotheses and future work.

2 Experimental Design

A group of 35 undergraduate students was divided into two groups alphabetically by last name, and two game design projects were assigned to each group. The first project that was assigned was called *Box Spin*, a simple game where the player can rotate and scale a box drawn on the screen. The second game was *Collision Course*, in which there is a disc in the center of the screen, and darts created by the player move at a constant rate towards the disc until they hit it. The specifications for the games follow, they take place on a 1000x800 pixel screen, with a constant framerate of 30 FPS

The specification for Box Spin is as follows: There is a square box in the center of the screen. The box can rotate left or right, and the box can grow or shrink. The box always remains centered, and the four edges of the box must be visible at any time. The state of the game consists of the length of the box's sides, and the box's orientation angle. The box begins with its size as 10x10 pixels, and its sides parallel to the *x* and *y* axes. The player can press any of the following keys to interact with the game: 'W', 'A', 'S', 'D', 'X. In each frame, the player can perform the following actions:

1. If 'A' is pressed and 'D' is not pressed, box rotates left by 3 degrees.
2. If 'D' is pressed and 'A' is not pressed, box rotates right by 3 degrees.

3. If ‘W’ is pressed and ‘S’ is not pressed and the size of the box is less than or equal to 500x500 pixels, then the sides of the box grow by 4 pixels.
4. If ‘S’ is pressed and ‘W’ is not pressed and the size of the box is greater than or equal to 10x10 pixels, then the sides of the box shrink by 4 pixels.
5. If ‘X’ is pressed then the game returns to the initial state.

The Collision Course game is defined as follows. There is a disc in the center of the screen. Darts (smaller discs) can appear wherever the player clicks on the screen. Darts will always move directly toward the disc in the center at a constant velocity until they reach the center, after which they disappear. The state of the game consists of a collection of darts and their positions, and whether the game is paused or not. The initial state of the game is an empty collection of darts, and unpaused. The player can press ‘X’, or ‘P’ to interact with the game. The player can click at any location on the game window. In each frame,

1. If the player clicks on the game window at point (*x*, *y*) outside of the disc, then a dart is created and centered at (*x*, *y*).
2. If the player presses ‘X’ then the game returns to its initial state.
3. If the player presses ‘P’ and the game is paused then, the game is unpaused.
4. If the player presses ‘P’ and the game is unpaused then, the game is paused.
5. If the game is not paused, then every dart moves directly toward the center of the screen by a distance of 3 pixels.
6. Any dart that reaches the center (will pass through (500,400) in the next frame) disappears.

In Phase I of the study, each student in the class was assigned to write Box Spin, with students in Group I using Easel and SequenceL, and students in Group II using Pygame and Python. The entire class was given the same specification for Box Spin, by which their submissions would be graded for success or failure. In addition to the spec, the class was given a lecture covering the math needed to implement the game. They were encouraged to come ask any questions needed during office hours.

In Phase II the students wrote *Collision Course*, and switched the languages, with Group I now using Pygame and Group II now using Easel. Once again the entire class was given a specification, a lecture on the math needed, and available office hours for help.

During both phases, students received links to the documentation for Python Pygame, SequenceL, and Easel. The documentation for SequenceL and Easel can be found at http://goo.gl/1UcEty. One of the Pygame tutorials students received was http://goo.gl/Ul8wZ4, which discusses the architecture of game loops and how to set the frame rate in a real time game. It is worth noting that most of the students had not used SequenceL before, while most had used Python since it is the CS1 language at Texas Tech.

3 A Priori Hypotheses

Going into the experiment, we had a few patterns that we would look for. Once such pattern would be that, contrary to intuition, the abundance of documentation and examples for Python/Pygame would actually cause difficulties for the students developing in that language. The idea behind this hypothesis is that given a plethora of information written by numerous authors on numerous subject, the developer would be overloaded with information that was not directly relevant to their task: learning to use the language properly for development.

The other hypothesis was that the nature of a functional language would greatly increase the ease of developing a game. We decided to look very closely at the flow of the programs that the subjects would create, seeing if ones built in SequenceL/Easel seemed to allow the developer to implement the specification as closely as possible with minimal translation from spec to product.

4 Observed Results

For project 1, the success rate for students who used Easel was 6 successes out of 18 attempts, and the success rate for students using PyGame was 3 successes out of 17 attempts. A “success” is defined here as writing a game that functions according to its specification. For project 2, the success rate for Easel was 7 out of 17, and the success rate for Pygame was 2 out of 18.

For both projects, incorrect submissions in PyGame were due to framerate most of the time. There were more runnable PyGame submissions than Easel ones. All but two runnable submissions (i.e., submitted programs that did not crash on opening) in Easel were correct.

5 New Hypotheses and Future Work

The most frequent errors in the Pygame programs involved handling the frame rate. We thus hypothesize that this is a stumbling block for new game programmers, and that the fact that it is handled automatically in Easel was a significant reason for the higher success rates for students using Easel. In the Box Spin game, frame rate errors explain all of the difference in success rates. They were, however, not a significant factor in Collision Course.

The large number of tutorials, and large amount of sample code available for Python and Pygame seemed to actually hurt the students’ success rates when using these tools. It seemed that students searched repeatedly for a library function or example that would solve their problems for them, ultimately without success. With Easel, on the other hand, students knew they would have to solve the kernel of the problem themselves, and so they rolled up their sleeves, got to it, and ultimately succeeded at a higher rate.

Ideally, we would like to conduct an experiment in which the students do each project in one observable session. We, as observers, have no real way to gauge exactly what the individual students' issues were with the games since they took the work home. This would also allow us to keep accurate track of the time students spent on each game.

6 References

[Cooke 2008] Daniel E. Cooke, J. Nelson Rushton, Brad Nemanich, Robert G. Watson, Per Andersen: Normalize, transpose, and distribute: An automatic approach for handling nonscalars. ACM Trans. Program. Lang. Syst. 30(2) (2008)

[Nelson 2014] Bryant Nelson, Josh Archer, and Nelson Rushton. Easel: Purely Functional Game Programming. Submitted to SERP 2014.