This is miscellaneous PSP tutorial m2

Introduction

Hi everybody,

This is a short tutorial on creating randomizers. This is only a tutorial to show you what is possible with randomizers. It will result in a program showing a few randomizers on screen which can be done again. There will also be given some information about other functions.

We will use the code from tutorial m1: Timers as the basis for this tutorial. I have deleted all references to the timer object but kept the controls and other functions.

The basics

Before we will implement the seperate randomizers we first have to create the utility object. Well we do not have to but it is better in my opinion because we can add a lot of other functions, other than randomizers. The whole collection of helper functions I always like to put in an util object. Other helper functions will also be given in other tutorials later on.

The util object is a singleton just as the graphics object. See tutorial 0: Setting up the 3D environment for more explanation about singletons. Lets take a look at the header:

UtilObject.h

```
#ifndef UTILOBJECT_H_
#define UTILOBJECT_H_

#include <psputils.h>
```

Here we have our headergaurds again and we include the psputils header.

```
class UtilObject {

private :
   static UtilObject* _instance;
protected :
   UtilObject(void );
   UtilObject(const UtilObject&);
   UtilObject& operator= (const UtilObject&);

   SceKernelUtilsMt19937Context ctx;
```

The class object with all the known singleton functions. The last variable however is one that needs to be explained. As you probebly have seen I will discuss a good and a bad version of the randomizer. This variable holds data of a mersenne twister. We need that for the good randomizer.

```
public :
   static UtilObject* Instance(void );
   ~UtilObject(void );
   void InitRandomizer();
   int getRandomIntBad(int low, int high);
   int getRandomIntGood(int low, int high);

};

#endif
```

In the public area we have a init function and the two functions which will be discussed in the following paragraphs. Both functions take a low and a high value. The resulting random number will be in between those numbers. Let take a quick look at the source for the singleton.

UtilObject.cpp

```
#include "UtilObject.h"
#include <stdlib.h>
#include <time.h>

UtilObject* UtilObject::_instance = 0;// initialize pointer

UtilObject* UtilObject::Instance (void ) {

   if (_instance == 0){
      _instance = new UtilObject; // create sole instance
   }
   return _instance; // address of sole instance
}

UtilObject::UtilObject(void ) {
   // creation of object
}

UtilObject::~UtilObject(void ) {
   // clean up
}
```

We need the stdlib for the random functions and the time lib for our good randomizer. Everything else is just normal singleton code.

Now we have created the base for the two randomizers.

The bad randomizer

You can create randomizers in different ways. This part will demonstrate how to use a bad version. Some of you know the rand() function. This function is a random number generator. I will use this function with the bad version of the randomizer. This function returns random numbers but at some point you can predict the numbers that are returned. This is not always something you want. For example: With boxy II I had a random theme function which cycled through the themes but everytime I started a game I got the same sequence of themes. Sometimes however it has some benefits when knowing what comes. Lets take a look at the code:

UtilObject.cpp

```
int UtilObject::getRandomIntBad(int low, int high) {
```

```
    // return a random number between low and high
    return (rand() % (high-low+1)) + low ;
}
```

Now if you use this on its own the sequence will always be the same (thus predictable). So we have to add something that will make our randomizer a little bit better. We will use the srand() function. You have to pass a number to let the srand() function create a new randomized sequence. If you pass 1 for example it will have sequence A, everytime you call srand(1) again you will have the same sequence A again. This is also preditable but we can create more different random sequences.

We can improve the randomizer even further. We can pass a number to the srand() function which is different everytime we can call the srand function. Time is always different. When we pass time to the srand function everytime a different sequence will be created.

```
void UtilObject::InitRandomizer() {
    srand(time(NULL));
}
```

So now we have a perfect randomizer? not even close. We have to pass an value to the srand() function. The number of sequences are limited and thus it is not a perfect randomizer and it can be predicted. Also there are some problems with the the lowermost bits of the number returned may not be particularly random. So lets create a good randomizer.

The good randomizer

I have already mentioned it but for the good randomizer we will use a mersenne twister. The mersenne twister randomizer returns better real random numbers. I have already given the variable which holds the mersenne twister data and now we are going to use it (put this code in the InitRandomizer() function):

UtilObject.cpp

```
    sceKernelUtilsMt19937Init(&ctx;, time(NULL));
```

We pass the variable ctx as the first parameter and in it the mersenne twister data is saved. The second parameter is the seed value. Again as with the srand we pass the time function for more randomness. Now we have initialized the mersenne twister function. Lets use it:

```
int UtilObject::getRandomIntGood(int low, int high) {
    // return a random number between low and high using a twister.
    u32 rand_val = sceKernelUtilsMt19937UInt(&ctx;);
    rand_val = low + rand_val % (high-low);
    return (int)rand_val;
}
```

The first line uses the sceKernelUtilsMt19937UInt function with the ctx variable to return a random value. With that we create in the second line with that value a value between the given range low and high.

GameApp object.

To get the tutorial to work we have to make some changes to the GameApp object.

GameApp.h

```
#include "UtilObject.h"

protected:
   UtilObject* utils;
   int goodRandom;
   int badRandom;
```

The two variables are to hold the returned random numbers by the two randomizers.

GameApp.cpp

```
   // load the UtilObject
   utils = UtilObject::Instance();

   // initialize the randomizer.
   utils->InitRandomizer();

   // setting the initial state of the variables
   this->goodRandom = 0;
   this->badRandom = 0;
```

This code needs to be in the Load() function. It first creates an utils object. The second line initializes the randomizers as given in the code in the randomizer paragraphs. The last thing we have to do is setting the random values to 0 at the beginning.

```
   if (pad.Buttons & PSP_CTRL_CROSS){
      if(buttonCross == 0) {
         buttonCross = 1;
         this->badRandom = utils->getRandomIntBad(0,9);
      }
   }
   if (pad.Buttons & PSP_CTRL_SQUARE){
      if(buttonSquare == 0) {
         buttonSquare = 1;
         this->goodRandom = utils->getRandomIntGood(0,9);
      }
   }
```

Here is the code where we actually use the functions. When we press the cross button we will use the bad version and when we press the square button we use the good version. This code has to replace the button code already existing.

```
void GameApp::Render() {
   // render information.
   pspDebugScreenSetXY(10, 5);
   pspDebugScreenPrintf("The bad randomizer: %d", this->badRandom);
   pspDebugScreenSetXY(10, 6);
   pspDebugScreenPrintf("The good randomizer: %d", this->goodRandom);
   pspDebugScreenSetXY(4, 15);
   pspDebugScreenPrintf("Press Cross to get bad random number.\n");
   pspDebugScreenSetXY(4, 20);
   pspDebugScreenPrintf("Press Square to get real random number.\n");
   sceDisplayWaitVblankStart();
   sceGuSwapBuffers();
}
```

And this is the render function which renders the numbers on screen as well as the commands.

This is the end of the tutorial. You can now create predictable randomizers as unpredictable. Compile the code and you see that yo ucan

generate random numbers with both methodes. Mess around somewhat with the values, for example do srand(1) and re-run the application a few times and see if the sequence is the same. Try deleting the srand in the initialize function and see what happens.

MakeFile

The makefile for tutorial m1

```
TARGET = out
OBJS = $(wildcard *.cpp) $(wildcard *.c)

INCDIR =
CFLAGS = -O2 -G0 -Wall -g
CXXFLAGS = $(CFLAGS) -fno-exceptions -fno-rtti -g
ASFLAGS = $(CFLAGS)

LIBDIR =
LDFLAGS =
LIBS = -lc -g -lpspgum -lpspgu -lstdc++ -lm -lpsppower

PSPSDK=$(shell psp-config --pspsdk-path)
include $(PSPSDK)/lib/build.mak
```

Compile and see what happens :D

Source files

Download the source files.