Gesture Processing Library

MJaniec BPolnik

28 października 2012

1 Nazwa

Gesture Processing Library

2 Ciekawostka

http://www.imore.com/apple-multitouch-google

3 Nazewnictwo

Nie ma namespace. Proponuje zasymulowac poprzez uzycie prefiksu tl. Proponuje tez pisac pseudo-obiektowo.

- Pliki: **tl_file_name.ext**
- Funkcje: **tlFunctionName**
- Zmienne: **tl_var_name**
- Struktury: **tlStructName**
- "Metody": **tlStructName_MethodName(tlStructName* this...)**
- Stale: **tl_CONSTANT_NAME**

4 Makefile

Uzywamy makefile? mnie sie podoba. Wtedy moznaby podawac flage komplilacji DEBUG lub UNCHECKED. DEBUG wlaczalby biblioteke stdio.h oraz funkcje obslugi bledow.

5 Wersja C

Proponuje C99. Fajne petle, slowko const, bool

6 Typy danych

Trzeba ostroznie z pamiecia. W szczegulnosci proponuje nie floating pointow. W ogole wzorem OpenGL proponuje zdefiniowac wlasne typy. Moje propozycje:

- tlBool - bool
- tlUByte - unsigned char
- tlByte - signed char
- tlWord - sigend short
- tlUWord - unsigned short
- tlInt - tlWord
- tlUInt - tlUWord
- tlFloat - tlWord -¿ fixed point
- tlChar - char
- tlVoid - void
- tlString char*

Do debugowania wlasny printf jakis tlPrintf trzaby obslugiwac

- %s - tlString
- %c - tlChar
- %f - tlFloat
- %d %i - tlInt, tlWord
- %ud %ui - tlUInt, tlUWord
- %b - tlBool

7 Obsluga bledow

- tl_errno -numer bledu. 0=brak bledu
- tl_ERRSTR - stringi z opisami bledow
- tl_ERRSTR[tl_errno] - Opis ostatniego bledu
- tl_ERRSTR[0]= 'OK'

Wszystkie funkcje obsluguja bledy w TEN SAM sposob.
Informacja a blendzie tylko w tl_errno
Kazda funkcja na poczatku zeruje tl_errno
jesli po wykonaniu funkcji tl_errno!=0 to blad i wartosc zwrocona z funkcji moze byc invalid

Proponuje makra

- $fun
- $if
- $c
- $e

Definicja funkcji

```
tlType tlFunctionName(tlType1 var1,...){
  $fun;
  code...
  $if (condition) error_code; //if condition rise error error_code;
  code...
  return retval;
}
```

Wywolanie funkcji

```
tlFunctionName(var1,...)$c; //continues
tlFunctionName2(var2,...)$e; //exit
```

Dzialanie makr

```
$fun <=> tl_errno=0
$if condition error_code <=> if(condition){tl_errno=error_code; return;} //zwraca smiecia
//wypisuje informacje o bledzie i kontynuje wykonanie
$c <=> ; {if (tl_errno) print(Filename, linenumber tl_ERRSTR[tl_errno], line of code);}
//wypisuje informacje o bledzie i zamyka program
$e <=> ; {if (jw) print(jw); exit(tl_errno); }
```

Takie podejscie ma 2 zalety.
1. Zaimplementowalem juz cos z grubsza podobnego na potrzeby sysopow, wiec jest praktycznie gotowe.
2. Makra mozna uzaleznic od innego makra. Dzieki temu mozna zamienic wszystkie instrukcje obslugi bledow na blanki przy pomocy parametru kompilacji. W spomniane wyzej DEBUG, UNCHECKED

8 edytor

Nie wiem jescze czego uzywac bedziemy do edytowania, ale wiele edytorow posiada mozliwosc modyfikowania syntax highlightingu. Mozna by ódkeywordowacśtandardowe typy. Dodac typy uzywane w bibliotece. Dodac wyzej wymienione makra. I ponaddto zmienna tl_errno. oraz warto by bylo zrobic jakas stala tl_null=(tlVoid*)(0)

9 input

Na wejscie tablica dwu wymiarowa + jej wymiary jaks funkcja
tlVoid tlInit(tlInt width, tlHeight, tlByte** (getData*)());
Kod klienta by sobie refreshowal
tlVoid tlRefresh();

10 output

Na wyjsciu informacja o wykonywanym gescie , lub jego braku + dodatkowe parametry. Opisane w pliku external.

Kolejka zdazen. Klient moze zarejstrowac pewne zdazenia takie jak scrolldown. wtedy do kolejki beda wrzucane takie gesty. Bedzie to jedynie informacja ze uzytkownik cos zrobil. Rozni sie to tym ze to pojawia sie tylko po wykonaniu gestu i pociaga za soba konkretna akcje.

11 propozycja

Proponuje znacznie obciąć funkcjonalność gry, bo ma ona służyć tylko do prezentacji - wydaje mi się, że wystarczyłoby przesuwanie kocków, czas, punkty i inne nie są potrzebne. Pokazanie zoom'u na grze dodaje wg mnie niesamowicie duzo pracy. Warto by jednak przemyśleć, czy nie lepiej stworzyć ćoś", po czym można wykonywać gesty i informacja jaki to gest pojawiała sie na dole ekranu - wydaje mi sie, że wtedy jest łatwiej coś pokazać, łatwiej do tego dodać gesty i można sie mniej napracować, a prowadzącemu to chyba obojętne z tego, co mówił.

12 lista gestow

Wydaje mi sie, ze jedynym sposobem dodawania gestow w momencie pracy biblioteki jest utrzymywanie jakos zaalokowanej pamieci, gdzie mozna je przechowac. Dlatego albo dajemy np. wymog alokowania pamieci uzytkownikowi - dodajGest(pamiec, tablica_z_wygladem_gestu), ale wtedy nigdy jej nie zwolnimy, albo jest jeszcze jeden sposob - wystawiamy w interfejsie wskaznik na funkcje ktorej uzywamy do alokowania, a user nam odpowiednia funkcje przypisze pod ten wskaznik - znowu kwestia zwalniania pamieci, ale jezeli juz wystawilismy jeden wskaznik, to czemu nie wystawic drugiego do zwalniania??. Wydaje mi

sie jednak, ze oba rozwiazanie dodaja niesamowicie duzo pracy i chyba lepiej byloby albo porzucic dodawanie nowych gestow, albo dodac je na etapie kompilacji, jednak sam nie wiem jak mialoby to wygladac (moze klient wystawia globalny wskaznik, a my w headerze mamy extern??). Jednak summa summarum, jak dla mnie to bez dodawania gestów biblioteka jest wystarczajaca trudna do napisania. Zawsze mozna po prostu zdefiniowac na starcie wiecej gestow.

Btw. Jak wczytamy gesty? Z pliku nie bardzo, bo nie mamy biblioteki.... Napiszemy jakiś program, konfigurator biblioteki generujący gesty i tworzacy kod, ktory potem bedziemy linkowac??

13 additional info

Patrzyłem jak to działa dla javy i ogólnie wygląda to tak, że tworzysz sobie GestureOverlayView, do niego dodajesz jako listener cos, co implementuje interfejs OnGesturePreformed. Implementator interfejsu ma wywoływaną metodą, która dostaje gest i GestureOverlayView jako parametry. Przykład jest: tutaj Wnioski z tego są takie - przetwarzanie gestów może być jednowątkowe, wczytanie gestów załatwia api z androida.

Poczytałem też troszke jak to działa, ze kod w c, czy tam bardziej c++ jest wykorzystywany w androidzie. Działa to przez jni. Czyli my sobie generujemy nagłówki z kodu javy do C++, w nim korzystamy z środowiska do mapowania typów, i teraz korzystamy z naszej biblioteki w C wywołując jej odpowiednie funkcje. Z czym się to wiąże?? Appke robimy w javie. Robimy całe to GestureOverlayView, dodajemy javowego listenera, który jako biblioteke ma coś napisane w javie z użyciem metod natywnych. Jeżeli chodzi o mapowanie gestu do czegoś bardziej w c, to gest zawiera liste przycisniec (GestureStroke), a ono ma tablice punktów i długość.

To o co bardzo trzeba sie pilnować, to to, aby to była biblioteka do przetwarzania gestów - czyli rozpoznawania, a nie do obsługi gestów. Odczytanie samemu gestu w c jest chyba niemożliwe.

Pytałeś, czy przetwarzamy gesty na żywo, czy jednak nie. Myśle, że to co napisałem powyżej odpowiada na to pytanie binarnie. in gest -¿ out list of Prediction(nazwa + score);