

ALL ABOUT GAME MAKING



Zestaw artykułów o Game Makerze OFF-LINE

Zestaw ten został stworzony, dla osób, które chcą przeglądać, lub wydrukować artykuły zamieszczone na naszej stronie. Miłej lektury.

Game Makerowe Pop-upy

W Game Makerze Pop-Upami nazywamy każde wyskakujące okienko, np. tablicę wyników. Można to nazwać zaletą GMA, ponieważ konkurencyjny The Games Factory nie posiada tego.

Teraz przejdźmy na stronę kodu GML. Napewno Was interesuje jak wysłać takie okienka. Więc zacznijmy:

show_text(fname,full,backcol,delay) - pokazuje tekst w pop-upie. Fname to nazwa pliku (tylko *.txt i *.rtf) położonego w katalogu z grą, full to oznaczenie czy pop-up ma być na cały ekran, backcol to kolor tła tekstu, delay to jak długo ma być wyświetlany tekst (w milisekundach).

show_image(fname,full,delay) - pokazuje obrazek w pop-upie. Fname to nazwa pliku (tylko *.bmp, *.jpg i *.wmf) położonego w katalogu z grą, full to oznaczenie czy pop-up ma być na cały ekran, delay to jak długo ma być wyświetlany obrazek (w milisekundach).

show_video(fname,full,loop) - pokazuje film w pop-upie. Fname to nazwa pliku (tylko *.avi i *.mpg) położonego w katalogu z grą, full to oznaczenie czy pop-up ma być na cały ekran, loop to oznaczenie czy plik ma być zapętłony.

show_info() - poprostu pokazuje okienko informacyjne gry.

show_message(str) - pokazuje okienko dialogowe z dowolnym komunikatem. Pod str wpisujesz swój komunikat.

show_error(str,abort) - pokazuje błąd. Pod str wpisujesz treść błędu, a abort oznacza czy gra ma się wyłączyć po okazaniu tego błędu.

highscore_show(numb) - pokazuje tabelę wyników. Wartość numb to nowy wynik. Możemy ją podmienić zmienną z punktami zdobytymi przez gracza.

highscore_add_current() - pokazuje tabelę wyników z dodaniem wyniku będącego w zmiennej score.

Dla was narazie tyle wystarczy, dlatego przejdziemy do innych spraw, czyli jak to będzie w eventach wyglądać.

EVENT: Keyboard W

Kod:

```
{  
  highscore_add_current();  
}
```

EVENT: Keyboard F1

Kod:

```
{  
  show_info();  
}
```

EVENT: Keyboard F2

Kod:

```
{  
  show_video('intro.avi', full);  
}
```

itd.

Myślę, że to już wszystko dla Was. Mam nadzieję że pomoże wam to jakoś w tworzeniu gry, np. jak chcemy wyświetlić intro.

Execute Shell

Dzięki `execute_shell` możemy uruchamiać programy przez GMA. Na początku przyjrzyjmy się bliżej tej funkcji. Opis z helpa jest następujący:

`execute_shell(prog,arg)` Executes the program (or file) in the shell.

Co po polsku znaczy:

`execute_shell(prog,arg)` Włącza program (albo plik) w shellu systemu (co znaczy że zrobi to nie wyłączając programu).

Podstawową czynnością jest odpalanie programów. Możemy to zrobić w następujący sposób:

`execute_shell('gra.exe',0);` // zero oznacza że bez argumentu.

Dozwolone jest również odpalania plików. Robimy to tak:

`execute_shell('readme.txt',0);`

Teraz was zadziwię. Można też nim odpalać strony internetowe (nie potrzebujemy mieć zarejestrowanego GM):

`execute_shell('http://aagm.dog.pl',0);`

Myślę że pomoże wam to chociaż trochę w tworzeniu gier. Dziękuję za uwagę.

Korzystamy z argumentów, czyli optymalizacji cz. 1

Jak napewno dobrze wiecie, w Game Makerze mamy skrypty. Są one w specyficznym języku GML. Tylko co zrobić, żeby nie powtarzać skryptów, tylko ze zmianą kilku zmiennych, zaśmiecając przy tym nieźle plik gry? To będzie temat tego artykułu. Nauczymy się optymalizacji poprzez argumenty.

Artykuł dzielę na kilka części omawiającej każdy element.

UWAGA! Artykuł jest przystosowany do GM6, jednak można korzystać z tych funkcji, w GM5, po daniu kilku poprawek.

1. Co to są argumenty i jak z nich korzystać?

Argumenty nazywają expressions (zmienne, stringi itp.). Dzięki nim możemy nie musimy powielać kodu kilkakrotnie zmieniając niektóre zmienne. Są one ustawiane podczas otwierania skryptu i wpisywanego do niego przy korzystaniu z niego. Dzięki temu możemy dać skrypt chodzenia, w którym podczas otwierania będziemy mogli ustawić czy postać będzie biegać czy chodzić.

Przykład na chodzenie i bieganie w lewo:

skrypt lewo

Kod:

```
if place_free(x-2,y+0)
x=x-argument0
y=y+0
```

event na chodzenie

Kod:

```
lewo(4);
```

event na bieganie

Kod:

```
lewo(8);
```

Pamiętaj!

Jednorazowo możesz skorzystać aż z 16 argumentów (argument0, argument1...argument15), ale przy korzystaniu z klocków możesz wprowadzić najwyżej 5 argumentów!

2. Określanie argumentów.

Argumenty najłatwiej jest określić przez klocki. Tutaj nie ma żadnej filozofii. Lecz możemy też z wykorzystaniem wcześniej zainicjowanych zmiennych. Najpierw jednak cała najważniejsza prawda, czyli jak wygląda wzór na to:

Kod:

```
nazwa_skryptu(argument0, argument1, argument2, argument3, argument4, argument5,
argument6, argument7, argument8, argument9, argument10, argument11, argument12,
argument13, argument14, argument15);
```

Jest to maksymalna ilość argumentów dla jednego skryptu.

Pamiętajcie, że zawsze usuwamy te zbędne, np. jak mamy tylko 3 argumenty to kod będzie

wyglądać tak:

Kod:

```
nazwa_skryptu(argument0, argument1, argument2);
```

Ważne jest też to, żeby nie zostawiać nie wykorzystanych argumentów, np. błędny kod:

Kod:

```
nazwa_skryptu(15, zmienna12, argument2, argument3, argument4, argument5,  
argument6, argumen7, argument8, argument9, argument10, argument11, argument12,  
argument13, argument14, argument15);
```

prawidłowy kod:

Kod:

```
nazwa_skryptu(15, zmienna12);
```

Jak zauważyliście, podałem w powyższym przykładzie zmienną. Je też możemy dodawać do argumentów.

Narazie to tyle co byście potrzebowali wiedzieć. Wspomnę też, że można dać też globalne i stringi, ale mam nadzieję, że sobie już poradzicie, bo ten artykuł to miałbyć tylko taki wstęp.

Snake – chodzenie

Więc tak. Głowa i ogon to muszą być oddzielne obiekty. W kolizji obiektu głowa wpisze:

Kod:

```
dlugosc += 1;
```

Zmienna ta oznacza długość tego snake

W create obiektu głowa wpisze:

Kod:

```
czasruchu = 7;  
alarm[0] = czasruchu;  
dlugosc = 10;
```

Kod ten oznacza, że zmienna czasruchu będzie oznaczać z jakim opóźnieniem ma się snake przesunąć. Ustawiamy alarm na tą zmienną. Późniejsze to tylko ustalenie długości początkowej.

Dalej w alarm0 dajemy:

Kod:

```
instance_create(x, y, ogon);  
nastX = x + 8 * cos(degtorad(direction));  
nastY = y - 8 * sin(degtorad(direction));  
x = nastX;  
y = nastY;  
alarm[0] = czasruchu;  
image_single = direction div 90;
```

Oznacza to że ogon ma się wydłużać i przesuwać tak jak w różnych grach z pocket gamesów, czyli kratka po kratce. Mam nadzieję, że mnie zrozumieliście.

UWAGA! W powyższym skrypcie ustawiasz wielkość jednej części snake. Domyślnie dałem 8 dla x i y. Ustawiacie tam własne wartości!

Platformówka – skakanie

W momencie przycisku na skakanie bierzecie execute the piece of code a w nim:

Kod:

```
if not place_free(x+0,y+1)
{
vspeed=-10;
}
```

Tworzyście później skrypt który wlepiacie do stepa:

Kod:

```
if place_free(x+0,y+1)
{
gravity_direction=270;
gravity=0.5;
}
else
{
gravity_direction=270;
gravity=0;
}
if (vspeed>12)
{
vspeed=12;
}
```

Krótkie to było i proste. Jak coś nie będzie działać poprostu napiszcie na forum, nie krępujcie się.

Obsługa joysticka

Joystick był i nadal jest najpopularniejszym kontrolerem gier, zastępując klawiaturę łatwością grania oraz tym, że nas palce nie boją od naciskania klawiszy. Z tego też powodu, obsługa joysticka raczej powinna być w grze. Game Maker, o którym jest w tym artykule mowa, obsługuje maksymalnie dwa joysticki (zazwyczaj jest podpięty do komputera tylko jeden, więc zupełnie to wystarcza). Wszystko co jest związane z joystickiem w GMie, odbywa się za pomocą poniższych funkcji.

UWAGA! ID to numer joysticka, czyli jeżeli chcemy żeby gracz grał joystickiem numer 1, to w miejsce ID wpisujemy 1.

joystick_exists(id) - sprawdza czy joystick o podanym ID istnieje (1 lub 2);

joystick_name(id) - sprawdza nazwę joysticka o podanym ID;

joystick_axes(id) - sprawdza ilość dostępnych kierunków w joysticku o podanym ID (zazwyczaj jest to 8);

joystick_buttons(id) - sprawdza ilość przycisków w joysticku o podanym ID;

joystick_has_pov(id) - sprawdza czy joystick o podanym numerze ID ma tzw. point-of-view;

joystick_direction(id) - sprawdza keycode (od vk_numpad1 do vk_numpad9) działających jak kierunki w joysticku;

joystick_check_button(id,numb) - sprawdza czy przycisk na joysticku został naciśnięty (numb to wartość od 1-32);

joystick_xpos(id) - sprawdza pozycję X drążka joysticka;

joystick_ypos(id) - sprawdza pozycję Y drążka joysticka;

joystick_zpos(id) - sprawdza pozycję Z drążka joysticka (jeżeli taka istnieje);

joystick_rpos(id) - sprawdza pozycję Rudder drążka joysticka (lub 4 drążka);

joystick_upos(id) - sprawdza pozycję U drążka joysticka (lub 5 drążka);

joystick_vpos(id) - sprawdza pozycję V drążka joysticka (lub 6 drążka);

joystick_pov(id) - sprawdza pozycję point-of-view i zwraca ją w stopniach obrotu (0 na przód, 90 w prawo, 180 w tył, 270 w lewo, a gdy nie ma POV, to zwraca wartość -1).

Mini-słowniczek:

Point-of-view - dodawane do joysticka (zazwyczaj w postaci trójkąta którym można obracać jak jakimś drążkiem) urządzenie do zmiany kierunku, co najlepiej działa w symulatorach lotu, ponieważ zwykły drążek nie potrafi dać tak dobrej dokładności w stopniach.

Zbiór pochodzi z dnia: 25.01.05

Wszystkie artykuły są mojego autorstwa (czyli Marmota).

Dziękuję za uwagę!

Marmot