# GAME MAKER TUTORIAL

Tutorial po polsku uczący od podstaw robić gry w game makerze.

**Dziś czytasz książkę, jutro tworzysz historię gier.**

Autor: Sebastian (Seka) Kąkolewski 2011

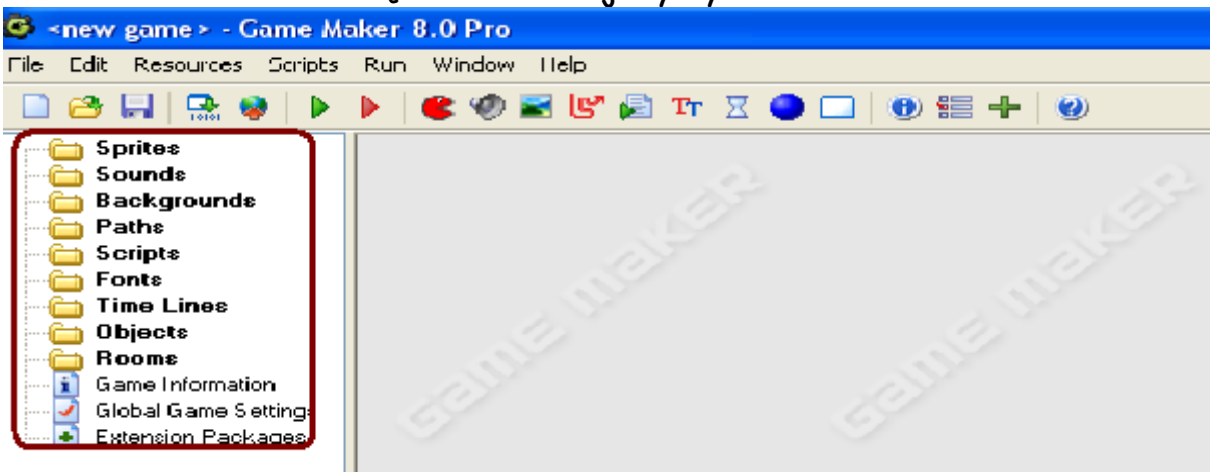
# Game Maker 8 [Tutorial]

## Wstęp

Aby zrobić grę, najpierw należy opracować temat, najlepiej zrobić jakieś „szkice" w głowie. Najpierw zajmijmy się grami treningowymi. Są to najczęściej minigry, pomagające zdobyć doświadczenie. Jeżeli od razu zajmiemy się mega produkcją, to po kilku dniach zrazimy się do Game Maker-a. Ważne jest też to że to nie jest „klikany program". Co prawda da się wszytko zrobić metodą podnieś i upuść, ale to na dłuższą metę nie zda egzaminu. Ja pracuje na Game Maker 8.0 Pro. Pragnę dodać, że istnieje darmowy odpowiednik tego programu – Game Maker 8 Lite. Ma on kilka funkcji mniej, ale na początek może być.

## Rozdział 1 – Zapoznanie z programem

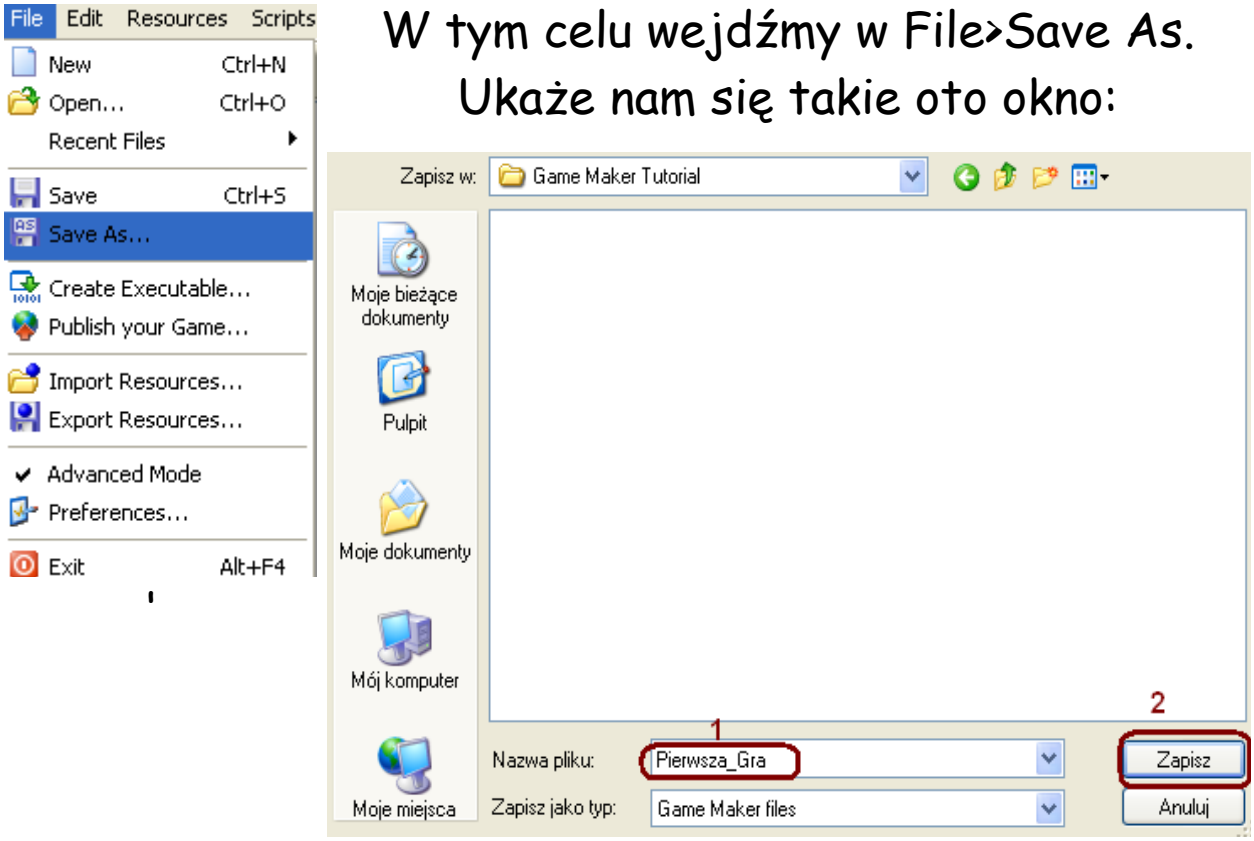
Po włączeniu GM ujrzymy takie okno:



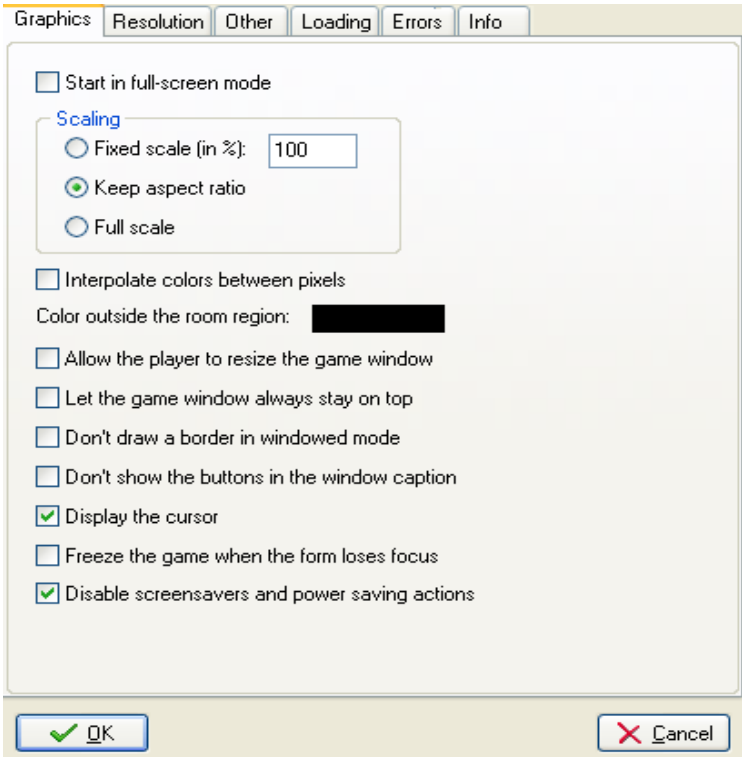

Po lewo mamy takie katalogi:

*Sprites – Grafika.*
*Sounds – Dźwięki.*
*Backgrounds – Tła.*
*Paths – Ścieżki.*
*Scripts – Funkcje, skrypty.*
*Time Lines – Osie czasu.*
*Objects – Obiekty.*
*Rooms – Pokoje (poziomy gry).*
*Game Information – Info o grze.*
*Global Game Settings – Ustawienia gry.*
*Extension Packages – Pliki rozszerzenia aplikacji(Tylko wersja PRO).*

Najpierw należy zająć się stworzeniem nowego projektu. W tym celu wejdźmy w File>Save As. Ukaże nam się takie oto okno:




W nazwę projektu wpisujemy „Pierwsza_Gra" i klikamy „Zapisz". Następnie ustawmy najważniejsze dane. Wejdźmy po lewo w „Global Game Settings". Teraz ukaże nam się takie okno opcji:

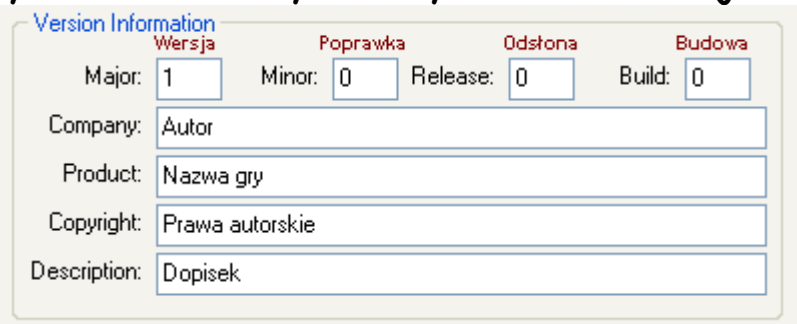



**Start in Full-screen mode**
***Czy ma otwierać się w trybie Pełnego Ekranu. Niżej mamy ustawienia co do trybu okna, powiększenia itp.***

**Display the cursor**
***Opcja ta daje nam wybór czy mamy widzieć kursor podczas gry.***

**Tak naprawdę wystarczy nam (na razie) tyle wiedzy na temat ustawień graficznych. Teraz przejdźmy do „Other". Na samej górze są ustawienia „Deflaut keys" Czyli ustawienia przycisków.**

Na samym dole mamy rubrykę informacji o wersji.




Wypełniamy według uznania. W „Loading" mamy ustawienia ładowania, oraz ikony, w errors ustawienia debugowania, a w info – informacje.

Teraz nie zapomnijmy o zaznaczeniu „Advanced Mode" w File.


## Rozdział 2.1 – Stworzenie Pierwszej gry (Grafika)

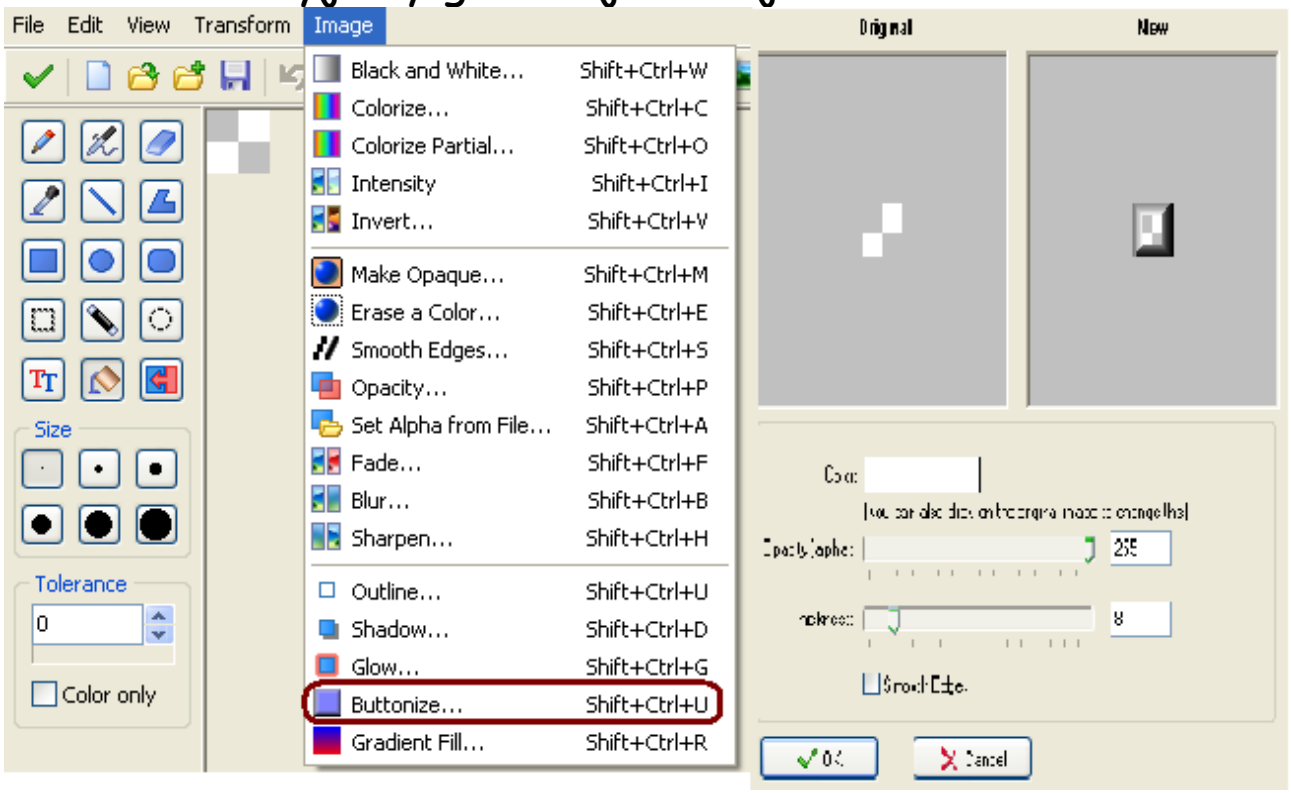
Teraz zajmijmy się tworzeniem gry. Przygotuj nowy projekt według tego, co pokazałem w ostatnich rozdziałach. Teraz stwórzmy nową grafikę. Będziemy korzystać z tego paska:

Kliknijmy na postać „Pacmana". W wyskakującym oknie przygotujmy grafikę. W name wpisz „spr_gracz". Jest to nazwa grafiki. Następnie wciśnij „Edit Sprite" i zobaczymy okno edycji klatek. W tym oknie wybieramy File>New. Ukaże nam się okno ustawień wielkości.

Wpiszmy w obie rubryki 32. Po potwierdzeniu ukaże nam się nowa klatka. Kliknijmy na nią dwukrotnie LPM. Ukaże nam się edytor grafiki. Użyjmy lupy, i włączmy siatkę.

Stwórzmy bohatera. Mój wygląda tak:

Zamykamy okno edycji grafiki,potwierdzając zapis. Tworzymy nowego sprite znów klikając na „Pac-mana", tym razem nazwijmy go „spr_przeszkoda". Postępujmy podobnie, wielkość znów 32x32. Do zrobienia tej grafiki użyjemy gotowej funkcji buttonize.



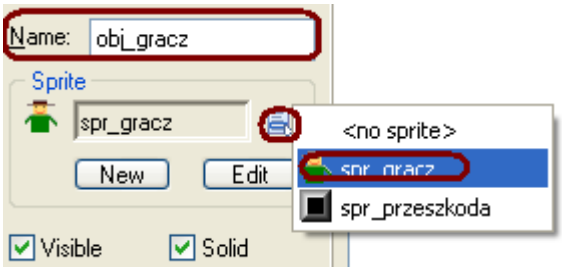

Koleinie wylewamy czarny kolor w środek. Powinno to wyglądać tak:

Zamykamy grafikę, potwierdzając zapis.
*Jeżeli macie wersje lite, nie możecie użyć buttonize. Zastąpcie więc go czym chcecie.

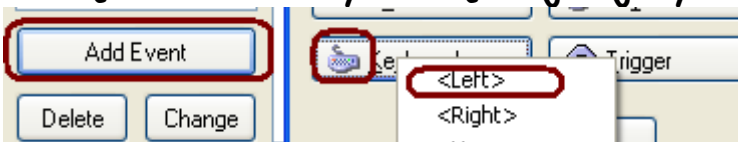
## Rozdział 2.2 – Stworzenie Pierwszej gry (Obiekty)

Teraz stwórzmy obiekty. Najpierw stwórzmy nowy obiekt.

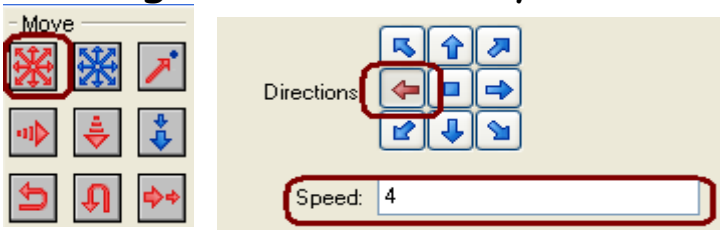
Kliknijmy na niebieską kulkę. Nowy obiekt nazwijmy „obj_gracz". Wybierzmy grafikę. Kliknijmy na ikonę pod napisem sprite, i wybierz grafikę „spr_gracz". Teraz zaznacz opcje Solid, opcja ta oznacza obiekt jako stały. Powinno to wyglądać tak:




Teraz przejdźmy do edycji eventów gracza. Eventy to coś takiego co pozwala na np.: poruszanie się gracza. Więc właśnie tym się zajmijmy.



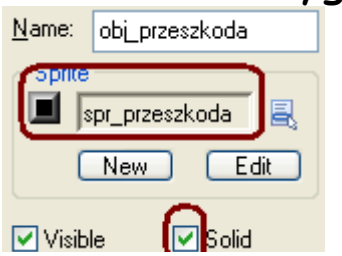

Wybieramy Add Event>Keyboard>Left. Ta akcja wykona się podczas wciśnięcia guzika „lewo", czyli strzałki w lewo.




Wybierzmy czerwone strzałki, czyli poruszanie. Wyskoczy okno ustawień. Wybieramy kierunek – w tym przydatku lewo, oraz prędkość, w tym przypadku 4.

Potwierdzamy. Podobnie robimy z pozostałymi kierunkami. Różnica jest taka,że wybieramy keyboard>kierunek, zamiast left. Na przykład dla prawo jest to keyboard>right. A w właściwościach poruszania wybieramy inny kierunek. Powinno wyglądać to tak:

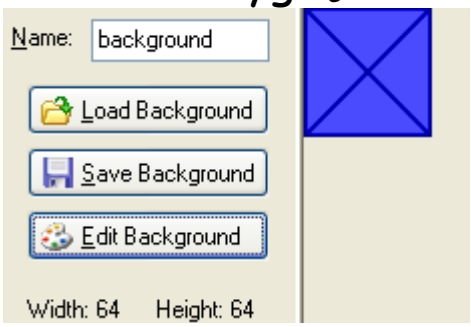

<Left>- lewo, <Up> - góra, <Right> - prawo, <Down> - dół. Teraz tworzymy obiekt o nazwie „obj_przeszkoda", a jako grafikę wybieramy „spr_przeszkoda". Zaznaczamy mu „Solid". Powinno to wyglądać tak:




## Rozdział 2.3 – Stworzenie Pierwszej gry (Tło)

Wybieramy ikonę obrazu z menu górnego.

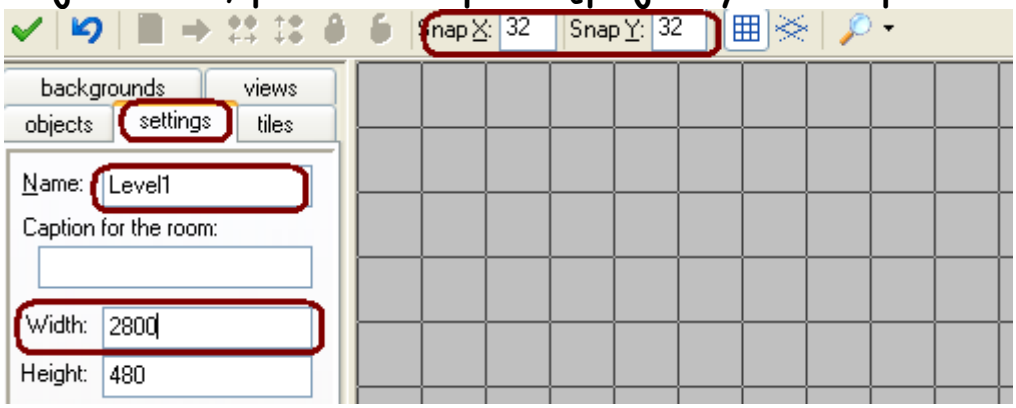
Tło nazywamy „background". Przechodzimy do edycji. Postępujemy podobnie do sprite. Wielkość ma być 64x64. U mnie to wygląda tak:



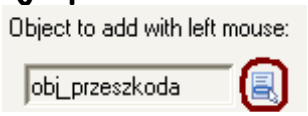
## Rozdział 2.4 – Stworzenie Pierwszej gry (Rooms i testy)

Stwórzmy nowy room.

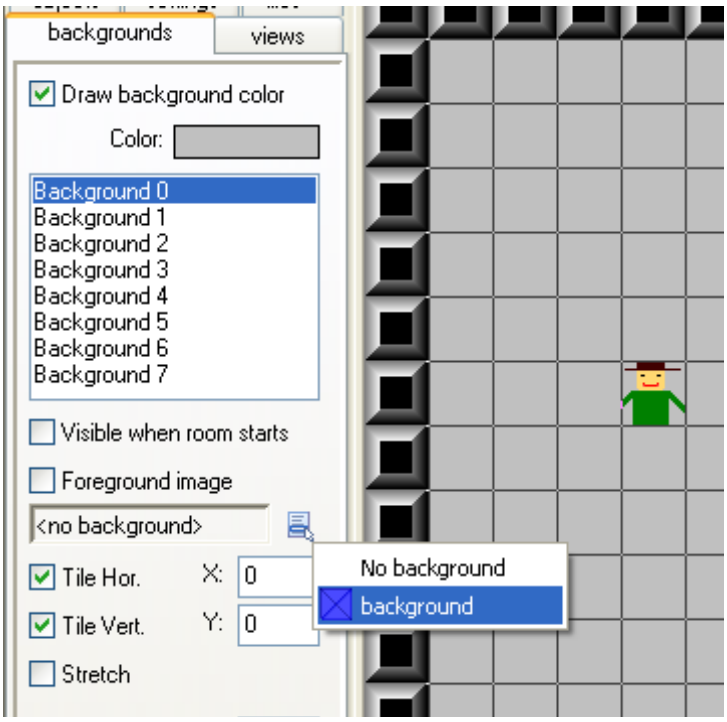
Najpierw wchodzimy w zakładkę settings. Wpisujemy nazwę „Level1", następnie wybieramy szerokość 2800px. Teraz ustawmy siatkę. Dlatego, że nasze obiekty mają rozmiar 32x32 to wybieramy Snap X jako 32, podobnie postępujemy z Snap Y.

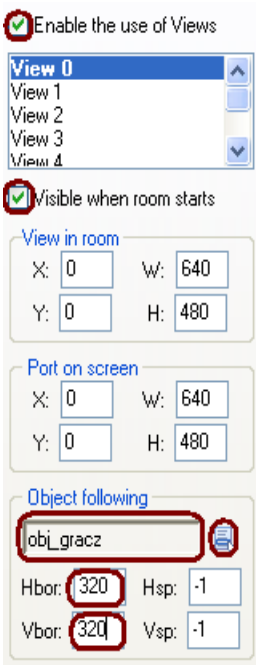



Teraz wchodzimy w zakładkę objects i wybieramy obiekt obj_przeszkoda.



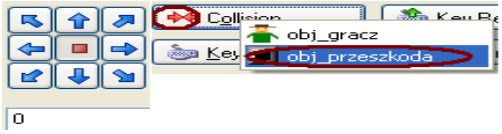

Teraz wciskamy shift, i przeciągamy po roomie myszką, tworząc przeszkody. W moim wypadku będą one na skraju mapy. Teraz postawmy obiekt obj_gracz. Po tych czynnościach wchodzimy w backgrounds i wybieramy tło.



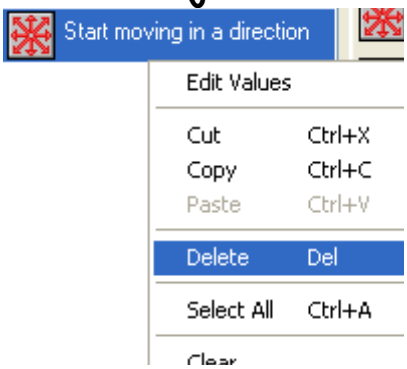

Koleinie przechodzimy do views. Zaznaczmy „Enable the use of Views" - czyli używaj viewa. Wybieramy „Visible when room starts" - czyli używaj jako domyślnego. Wybierzmy teraz obiekt za którym ma podążać view – czyli podążanie ekranu. Następnie wpisujemy Hbor i Vbor, czyli krawędzie po przekroczeniu których ekran się porusza. Powinno wyglądać to tak jak po lewo. Teraz testujemy grę, klikając na zieloną strzałkę u góry.


Teraz zajmijmy się naprawą błędów. Ja znalazłem bug, przechodzenie przez ściany. Więc wchodzimy w obiekt gracza i w evencie colision with przeszkodą wybieramy czerwone strzałki i ustawiamy speed na zero, a kierunek jako stop.

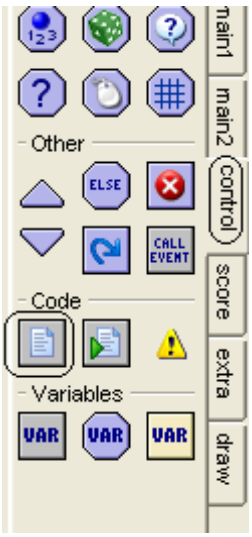


# Rozdział 3 – Pierwsze zderzenie z kodem

Teraz czas na nauczenie się kodzenia. Polecam program Decoder Blocks autorstwa Mateusza Bacławskiego, oraz polskie forum Game Makera gmclan.org, jeżeli macie jakieś pytania walcie tam. Ale zajmijmy się programowaniem. Ulepszmy system chodzenia. Zamieńmy najpierw left na kod. Wejdźmy więc i usuńmy klocek ze strzałkami. Kliknij PPM a następnie:




I wstawiamy klocek codu.




Wpisujemy:

x=x-4;

Oznacza, że pozycja x zostaje zmniejszona o 4px co tworzy efekt przemieszczania się obiektu. Oś x to inaczej oś pozioma. Oś y to inaczej oś pionowa. Ale wracając do kodu. Można go skrócić zmieniając na:

x-=4;

Taki zapis oznacza to samo. A teraz podobne skróty/operatory:

**logiczne(używane w działaniach):**

**[zmienna1] = [zmienna2] //zmienna 1 staje się zmienna2**
**[zmienna1] -= [zmienna2]; //zmienna 1 staje się o zmienna2 mniejsza**
**[zmienna1] += [zmienna2]; /zmienna 1 staje się o zmienna2 większa**
**[zmienna1] *= [zmienna2]; //zmienna 1 staje się o zmienna2 razy większa**
**[zmienna1] /= [zmienna2]; //zmienna 1 staje się o zmienna2 razy mniejsza**

**warunkowe(używane do porównań):**

**[zmienna1] > [zmienna2]; //zmienna 1 jest mniejsza od zmiennej2**

```
[zmienna1]  < [zmienna2]; //zmienna 1 jest większa od zmiennej2
[zmienna1] => [zmienna2]; //zmienna 1 jest większa bądź równa zmiennej2
[zmienna1] <= [zmienna2] //zmienna 1 jest mniejsza bądź równa od zmiennej2
[zmienna1] != [zmienna2] //zmienna 1 jest różna od zmiennej2
[zmienna1] == [zmienna2] //zmienna 1 jest równa zmiennej2
```

Według tego w prawo musimy zwiększyć wartość x, ponieważ osie są liczone od lewego górnego rogu. Czyli x jest liczony od lewej strony. Zwiększenie x to poruszenie x w prawo. Więc poruszanie w prawo będzie wyglądać tak:

```
x+=4;
```

Teraz pobawmy się y. Aby poruszać się w górę musimy pomniejszać y. Więc:

```
y-=4;
```

W dół powiększamy y:

```
y+=4;
```

Sprawdźmy, wyłącz edytor kodu potwierdzając zapis. I ponownie użyjmy testera, czyli zielonej strzałki.

## Rozdział 4 – Zmienne

Teraz zajmijmy się czym właściwie jest zmienna. Zmienna to tak jakby magazyn liczb. Dajmy na to zmienna „health" odpowiada za życie i jest to zmienna. Dlaczego więc nie użyjemy liczby? To proste. Gdy zmienimy wartość liczby, to i jej „nazwa" się zmieni. Potrzebujemy czegoś co zmienia wartość, ale nie zmieniając „nazwy". To właśnie jest zmienna. Porównanie zmiennych i liczb:

```
5+=2; //piątka ustąpi miejsce 7, ponieważ 5+2 to 7.
x+=2; //x choć będzie większy, nadal będzie tak samo się nazywał
```

Więc jak widzisz, życie bez zmiennych byłoby trudne. Warto zrozumieć zmienne, gdyż bez nich „ani rusz".
*(Zmienne mogą zawierać zarówno liczby, jak i znaki)

## Rozdział 5 – Warunek „if”

Jeżeli chcemy określić, kiedy ma się coś wydarzyć często używamy warunków. Jak to działa? Używając operatorów z wcześniejszego rozdziału możemy określić się. Składnia if:

```
if([zmienna1] [operator] [zmienna2])
{
[treść]
}else
{
[treść]
};
```

Czyli if to „jeżeli coś to coś”. Można także użyć if bez else. Else oznacza inaczej, jeżeli nie. Przykład if bez else(1) i z else(2):

```
if(health<0)
{
lives-=1;
health=100;
};
```

```
if(kasa=>100)
{
batonik+=1;
kasa-=100;
}else
{
show_message('Nie masz kasy');
};
```

Pierwsza pętla oznacz: jeżeli health jest mniejsze od 0, to lives staje się mniejsze o 1, a health staje się 100.

Druga oznacza: jeżeli kasa jest większa bądź równa 100, to batonik powiększa się o 1, a punkty stają się większe o 100. Jeżeli jednak kasa nie jest większa ani równa 100, to wyskakuje komunikat o treści „Nie masz kasy".
Do naszej gry wprowadzimy najpierw pierwszą pętle.
W obj_gracz w evencie game start – czyli start gry, ustal, aby życia były równe 5.

```
lives=5;
```

Teraz w evencie step zajmijmy się określeniem co się stanie po utracie życia. W naszej grze, gra będzie się „restartować". Więc wpisz:

```
if(lives<0)
{
game_restart();
};
```

Event step to taki event, który wykonuje się cały czas, dopóki obiekt istnieje. Więc teraz cały czas będzie się sprawdzało czy mamy jeszcze życia. Jeżeli nie będziemy mieli żyć to gra się zrestartuje.

Jest jeszcze możliwe kilka warunków. Jeżeli użyjemy „&&" to będą musiały wykonać się oba warunki, jeżeli „||" choć jeden, jeżeli „^^" tylko jeden. Można korzystać z tych argumentach w większości pętli. Przykład:

```
if(a>2 && b<5)
{
a=5;
};
```

# Rozdział 6 – Pętla While

Pętla while służy do określania twardego warunku. Dopóki warunek ten nie będzie wykonany dalszy kod nie wykona się, więc należy uważać jak i gdzie jej używamy, ponieważ może spaść znacząco wydajność aplikacji, jeżeli niewłaściwie jej użyjemy. Teraz coś o samej pętli. Pętla to tak jakby dopóki. Czyli dopóki „coś jest coś w stosunku do coś to będzie się wykonywać coś". Budowa tej pętli wygląda tak:

```
while ([zmienna1] [operator] [zmienna2])
{
[treść]
}
```

Różni tak więc się tym od if, że wszystko wykonywane jest nieustannie dopóki warunek jest zgodny, a if 1 raz na każdy kod w evencie. W step czy draw trudno zauważyć różnice, bo kody te wykonują się i tak średnio 1/klatkę. Dam wam przykład. Jeżeli umieścimy coś w colision, wykona się po każdym zderzeniu. Tak więc, aby kod wykonał się kilka razy, musimy odsuwać się i ponownie uderzać w obiekt.  I tu przychodzi nam z pomocą „while". Teraz przykład pętli while:

```
while(xp>maximum_xp)
{
lvl+=1;
maximum_xp*=2;
}
```

W tej pętli też można używać kilku warunków, korzystając z metody, którą poznaliśmy w rozdziale 5.

## Rozdział 7 – Pętla For

Osobiście nie lubie tej pętli., ale mimo wszystko powinniście ją znać. Ta pętla jest tak jakby while, tylko z tą różnicą, że zmienna będzie odpowiadać za ilość wykonanych akcji, będzie odrębna. Chodzi o to, że pętla dajmy na to, będzie wykonywać się do tąd, aż n nie będzie równe 0, a każdorazowa, zmienna n będzie się zmieniać. Struktura:

```
for([zmienna (ile razy?)]=[wartość];[zmienna(dopóki)] [operator] [zmienna2(dopóki)];[jak ma się zmieniać co pętle zmienna kierująca;)
{
[akcja]
}
```

Przykład:

```
for(i=health;i>0;i-=1;)
{
health-=1+a;
}
```

## Rozdział 8 – Switch

Switch to wybór. Jeżeli dajmy na to, zmienna może mieć tylko kilka wartości, możemy użyć switch zamiast if. Budowa:

```
switch ([zmienna(odpowiadająca za warunek)])
{
case [wartość zmiennej warunkowej]:
[akcja]
break;

case [wartość zmiennej warunkowej]:
[akcja]
break;

default: [akcja jeżeli niedojdzie do żadnej z możliwości(niekonieczne jest wpisywanie default)];
} ;
```

Przykład:

```
switch(wybor)
{
case 1:
draw_text(0,0,'wybrałeś 1');
break;

case 2:
draw_text(0,0,'wybrałeś 2');
break;

case 3:
draw_text(0,0,'wybrałeś 3');
break;

case 4:
draw_text(0,0,'wybrałeś 4');
break;

default:
draw_text(0,0,'wybrałeś złą liczbę');
break;
};
```

## 9 – Pętla repeat

Pętla repeat to najbardziej „prymitywna" pętla. Oznacza ona, że coś będzie wykonywane ileś razy. Budowa:

```
repeat([zmienna])
{
[akcja]
}
```

Przykład:

```
repeat(20)
{
health+=1;
}
```

## Rozdział 10 – Pętla Do Until

Tym razem pętla podobna do while. Jest jednak różnica. Pętla ta wykonuje się, a na końcu sprawdza czy nadal warunek jest spełniony – jeżeli nie to po prostu pętla się zakańcza. Dużo nie ma co pisać o niej, bo reszta jest podobna jak w while, zasady itp. Więc ta pętla wykona się chociaż raz. Struktura:

```
do
[treść pętli]
until
[warunek]
```

Przykład:

```
do
a+=1
until
a<0;
```

## Rozdział 11 – Tablice

Tablice to takie zmienne w zmiennej. Jedna tablica może mieć kilka zmiennych w których są kolejne zmienne itp. Przydatne jest to w wielu przypadkach, np.: przy tworzeniu strzelanek często używa się jako oznaczenie amunicji i broni. Struktura tablicy jedno wymiarowej:

```
nazwa[nr indexu]
```

Struktura tablicy 2 wymiarowej

```
nazwa[nr indexu,nr indexu]
```

Tablice mogą być tylko 2 lub 1 wymiarowe.

Przykład:

```
tablica[2,5]+=tablica[9,12];
```

# Rozdział 12 – Draw

Draw czyli rysować. W tym rozdziale zajmiemy się właśnie rysowaniem. Ale uwaga, jeżeli jakikolwiek obiekt używa komendy draw_[...] to automatycznie przestaje rysować swoją grafikę(sprite), jeżeli chcesz, żeby go rysował wpisz w (evencie)draw:

draw_sprite(sprite_index,image_index,x,y);

Komenda ta rysuje sprite obiektu. Zajmijmy najpierw się komendą draw_text(x,y,<co ma rysować>) X i y to namiary, oraz co ma rysować. Chodzi o text i stringi. Stringi czyli zmienne. Przykład draw z użyciem textu i zmiennych:

draw_text(10,12,'Życia: '+string(lives));

Tej komendy użyjemy w naszej grze. Więc stwórz nowy obiekt o nazwie „obj_controller" i nie dawaj mu grafiki. W evencie draw wpisz tę komendę. Ustaw obiekt na mapie. Następnie zmień kolor tekstu. Tu użyj komendy

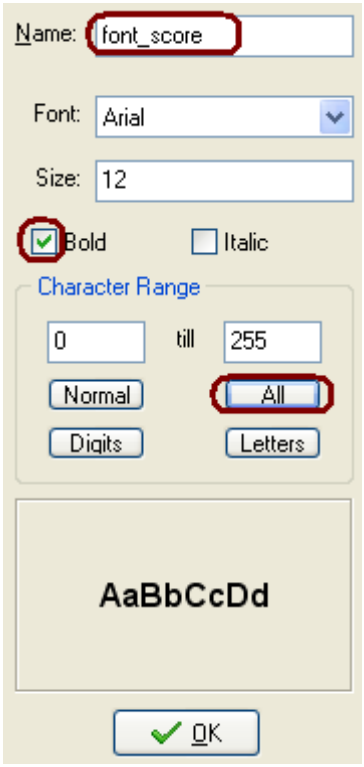
draw_set_color(c_red);

Czyli ustaw kolor na czerwony. Można zamiast red wpisać dowolny inny kolor. Teraz tekst jest widoczny, ale nie „chodzi" za ekranem. Więc na początku dopisz:

x=view_xview;y=view_yview;

A „draw_text(10,12,'Życia: '+string(lives));" zamień na „draw_text(x+10,y+12,'Życia: '+string(lives));". Teraz ustawmy polską czcionkę, ponieważ nie widać „Ż". W tym celu utwórz nową czcionkę. W tym celu użyj rysunku litery „T".

Wypełnijmy etykietę tak jak po lewo.




Teraz w draw znowu na początku naszego kodu dopisz „draw_set_font(font_score);" Czyli zmieni nam się czcionka na tą zrobioną przez nas, z polskimi znakami. Nasz kod w draw powinien wyglądać tak:

```
draw_set_font(font_score);
x=view_xview;y=view_yview;
draw_set_color(c_red);
draw_text(x+10,y+12,'Życia: '+string(lives));
```

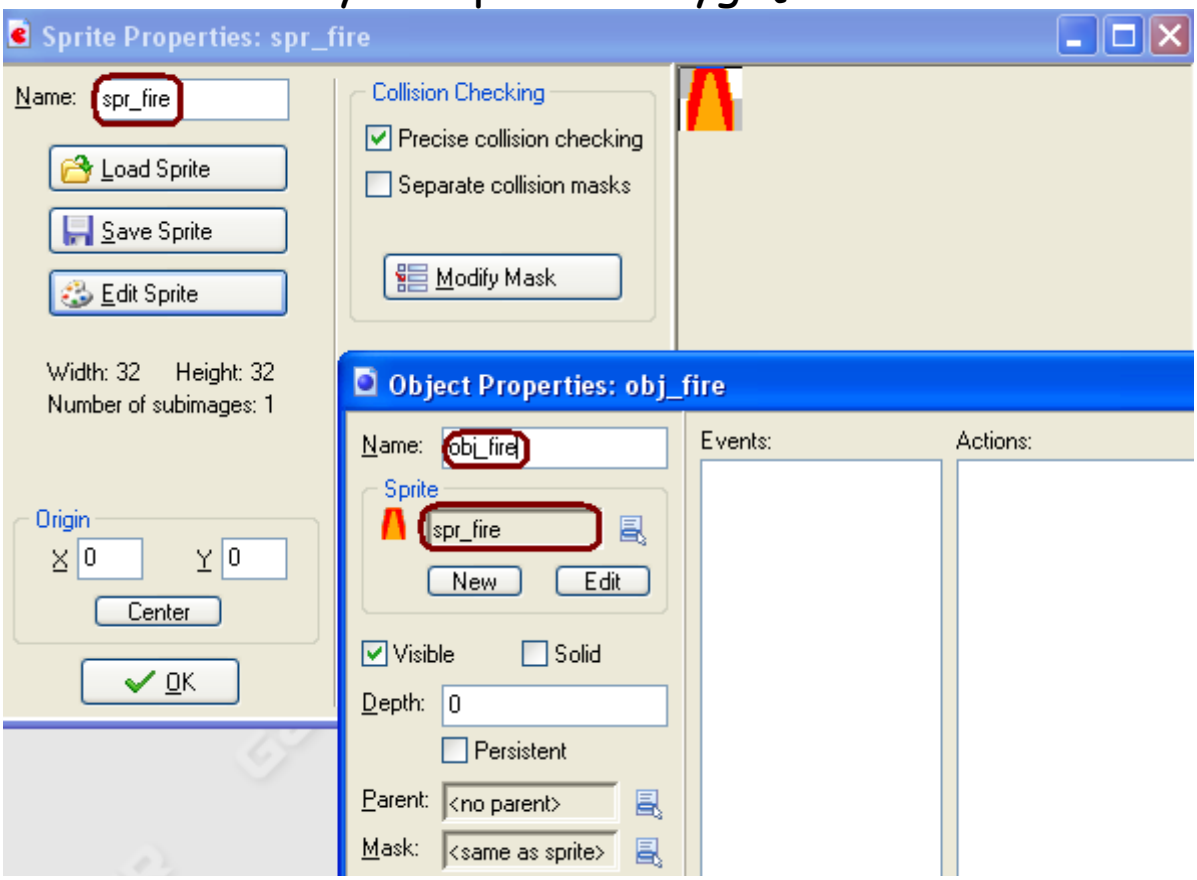
Przetestuj. Warto też zapoznać się z rysowaniem innych rzeczy. Teraz powiem wam jak zrobić pasek np.: życia. Nie przyda się to w naszej grze, ale warto wiedzieć. Pasek życia składa się z tła, które musi mieć tyle px ile można mieć max. życia, oraz drugiego paska, czyli tego właściwego o wielkości aktualnego stanu życia. Przykład:

```
draw_set_color(c_red);
draw_rectangle(100,32,200,38,0)
draw_set_color(c_green)
draw_rectangle(100,32,health+100,38,0)
```

Kod tworzy czerwony pasek tła, następnie zielony pasek z życiem. Dlatego znajduje się „+100" za health, ponieważ pasek ma być zaczęty od 100px. Rysowanie prostokąta, w tym wypadku „paska" wygląda tak. Najpierw komenda draw_rectangle, a następnie w nawiasach x startu, y startu, x końca, y końca i grubość konturów.

## Rozdział 13 – Ostatnie poprawki w grze.

W naszej grze, posiadamy życia, ale nie posiadamy czegoś co może je odebrać, oraz nasz level jest pusty. Najpierw stwórzmy grafikę ognia, i obiekt „obj_fire" z tą grafiką. Wszystko powinno wyglądać tak:

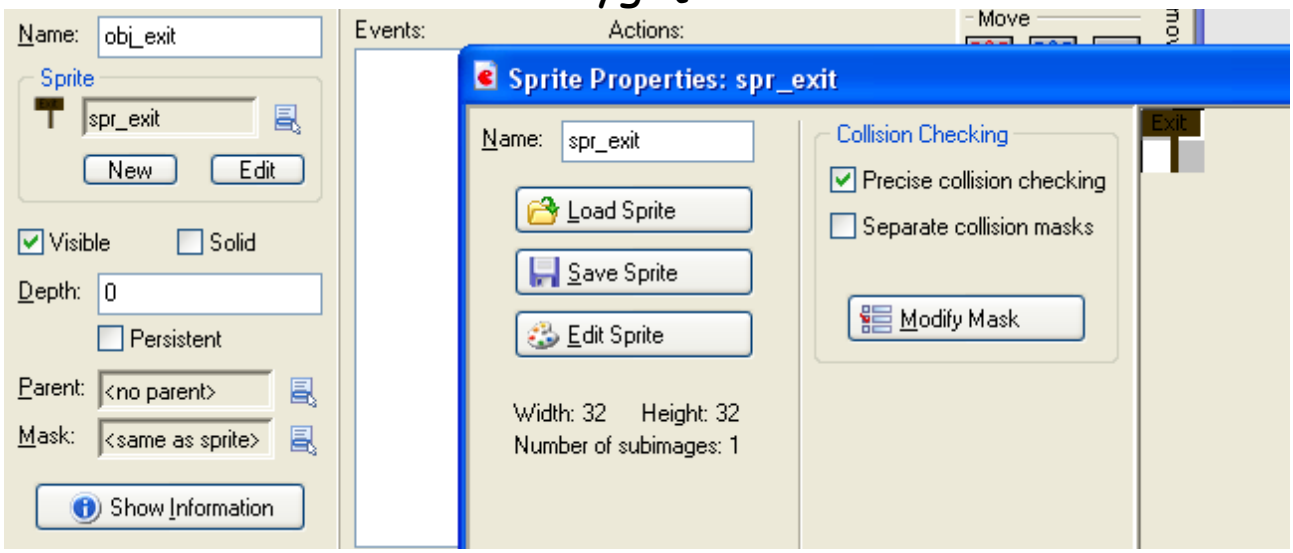



Jeżeli tak jest przejdźmy do obiektu gracza. Stwórz event kolizji z ogniem. W nim należy uwzględnić utratę życia, oraz rozpoczęcie pokoju od nowa. Kod powinien wyglądać więc tak:

```
lives-=1;
room_restart();
```

Teraz stwórz planszę. Twórz nowe pokoje i edytuj je tak jak w rozdziale 2.4. Wypełnij pokoje pułapkami itp. Teraz brakuje jeszcze czegoś co nas przeniesie do następnego

poziomu. Zrób więc tabliczkę „exit". Grafikę oraz obiekt. U mnie wygląda to tak:




W obiekcie exit(w evencie colision z graczem) stwórz kod. Kod musi działać tak, aby przenosić cię do następnego pokoju chyba że ten jest ostatni, wtedy powinien pojawić się komunikat, że przeszyłeś grę , oraz wyłączyć gra. Użyjemy więc if. Jeżeli istnieje następny pokój to nas przeniesie, jeżeli nie to to co wspomniałem wcześniej. Oto kod:

```
if (room_next(room)>-1)
{
room_goto_next();
}else
{
show_message('koniec gry');
game_end();
};
```

Teraz testy. Jeżeli skończyłeś edycje leveli. Pamiętaj, że zawsze możesz ulepszać grę. Przykładem jest stworzenie monet i punktów, oraz controlera, rysującego punkty. Pragnę jeszcze dodać, że przed zmiennymi dopiski: global.zmienna – oznacza że zmienna nie zmienia się wraz z zmianą pokojów, i jest dostępna dla wszystkich. globalvar zmienna – róznica jest taka, że w takim

wypadku nie musimy w następnej części kodu dopisywać przed zmienną „global.”.
var zmienna – zmienna dostępna tylko w danym skrypcie.
Nazwa_obiektu.nazwa_zmiennej daje dostęp do prywatnych zmiennych takich jak x, czy y. Jeżeli będziesz miał jakieś problemy, polecam zajrzeć na polskie forum – 'gmclan.org'. Aby grę przetworzyć na plik wykonawczy(*.exe) należy kliknąć ikonę konwertera.