Kurs tworzenia gier z silnikiem XenoN Core

Część 1: Samochód - sterowanie, fizyka oraz rysowanie

Przygotowanie:
Aby zacząć tworzenie potrzebny nam jest silnik XenoN Core w wersji 0.7.5.100, jego dokumentacja (w której znajdują się niezbędne informacje o silniku jak i opisy funkcji) oraz program Game Maker w wersji zarejestrowanej. Silnik wraz z dokumentacją znajdują się w folderze z kursem oraz na stronie http://www.xenon-core.yoyo.pl/sdk, zaś program Game Maker można ściągnąć z http://www.yoyogames.com, gdzie również się go rejestruje. Kurs pisany podczas używania programu w wersji 6.1.

Zaczynamy:
Najpierw tworzymy room **room_game**, następnie obiekty **ground**, **engine** oraz **car**. W roomie **room_game** ustawiamy te obiekty na dowolnej pozycji, lecz w odpowiedniej kolejności: najpierw koniecznie obiekt **engine**, następnie obiekt **ground** a na końcu **car**. Ważne: nie zmieniaj wielkości rooma na większy niż ma być ekran gry – nie jest to konieczne ani potrzebne teraz. Najlepiej ustawić wielkość 800x600.
Na początek zajmiemy się obiektem **ground**. Będzie on zawierał mało kodu, gdyż XenoN ułatwia Nam sprawę z terenem – jego ustawieniem i rysowaniem. Ustawiamy jego **depth**=1, a w create dajemy kod:

```
// ustawienie obiektowi właściwości terenu
xenon_set_terrain_simple(x,y,0, // pozycja terenu
                         15,15, // ilość segmentów poziomo i pionowo
                         1024,1024, // szerokość i wysokość segmentu
                         background_get_texture(bg_ground), // textura
                         c_white,1) // kolor i alpha

// pętla ustawiająca losową wysokość wierzchołków terenu
for(i=1;i<15;i+=1)
for(j=1;j<15;j+=1)
xenon_terrain_simple_set_depth(i,j,random(256+128))
```

Zaś event draw uzupełniamy kodem:

```
xenon_draw_terrain_simple() // rysujemy teren
```

A teraz zajmiemy się obiektem **engine**. Będzie on odpowiedzialny za kontrolę nad silnikiem i jego ustawieniami. Ustawmy mu **depth**=1000.

```
show_info() // pokazujemy informacje o kursie ;P

xenon_core_header() // skrypt nagłówkowy, potrzebny w każdym obiekcie
korzystającym ze skryptów XenoN Core

// inicjacja silnika
xenon_core_emulate_initiation('XenoN Core.dll', // nazwa pliku DLL
                              XE_NULL) // flaga - określa m.in. czy silnik ma
włączyć własny render do okna aplikacji. Domyślnie: XE_NULL

// ustawienia bryły odcinania, czyli zmienne grupy RENDER
xenon_core_set_double(XE_RENDER_WIDTH,room_width)
xenon_core_set_double(XE_RENDER_HEIGHT,room_height)
xenon_core_set_double(XE_RENDER_ANGLE,45)
xenon_core_set_double(XE_RENDER_NEAR,1)
xenon_core_set_double(XE_RENDER_FAR,10000)

// ustawienia trybów fizycznych
xenon_core_set_double(XE_PHYSICAL_COVALENCE_FORCE_MODE,XE_TRUE) // włączenie
trybu sił kowalencyjnych (międzycząsteczkowych)
xenon_core_set_double(XE_PHYSICAL_COVALENCE_FORCE_FIRST_STREAM,XE_TRUE) //
aktywacja pierwszego strumienia obliczeń sił kowalencyjnych

// zmienne kamery
z=0
alpha=0
beta=0

d3d_set_culling(false) // wyłączenie cullingu - ukrywania tylnich powierzchni
(backface culling)
d3d_set_fog(true, // mgła aktywna
            background_color, // kolor mgły
            0, // odległość początkowa
            1024*4) // odległość końcowa
```

Tryby fizyczne są nam potrzebne do tego, by świat gry „ożył” – stał się bardziej realistyczny. Tryb sił kowalencyjnych odpowiada za obliczenia utrzymujące substancje (podstawowe składniki fizyki XenoN Core) w odpowiednich odległościach od siebie. Tryb ten też jest podzielony na dwa podtryby: pierwszego i ostatniego strumienia. Standardowo aktywujemy tylko pierwszy strumień, a drugi nas nie obchodzi. Możemy włączyć drugi strumień, gdy np. włączymy też tryb kolizji promieniowej z trybem bezwładności.
Po omówieniu eventu create i jego kodu przechodzimy do eventu step:

```
xenon_core_physic_update() // aktualizacja środowiska fizycznego
```

Jak widać, event ten bogaty w kod nie jest, a obsługuje najważniejszą część fizyki – aktualizuje cały świat gry zbudowany z elementów.
Aby po skończeniu pracy silnik nie zaśmiecał pamięci trzeba go z niej zwolnić (co prawda po wyłączeniu aplikacji sam sprząta po sobie, ale i tak lepiej jak my go uprzedzimy – będziemy mieli pewność że na pewno zakończył pracę bezproblemowo). Zrobimy to w evencie Room End (zamiast Game End, po to, by z restartem gry nie pakował się na nowo do pamięci):

```
d3d_end() // wyłączenie trybu 3d
xenon_core_closure() // odłączenie silnika od aplikacji
```

No to w obiekcie **engine** pozostał event draw. W nim będziemy ustawiać kamerę wedle naszego uznania :).

```
// ustawienie kamery
xenon_core_emulate_camera_set(alpha,beta,0, // kąty obrotu kamery: alpha(oś Z), beta(oś Y) oraz gamma(oś Z; domyślnie 0, gdyż kąt gamma jest obsługiwany tylko w pełnej wersji silnika)
                              x,y,z, // pozycja kamery
                              128, // odległość oka od kamery
                              true) // zamiana miejscami pozycji oka i kamery (true - patrzenie na punkt (widok w trzeciej osobie); false -  patrzenie z punktu (widok w pierwszej osobie))
```

I w ten sposób zakończyliśmy kodowanie obiektu kontrolującego silnik i kamerę. Ostatnim obiektem, który Nam pozostał to obiekt **car** i jego kod eventu create:

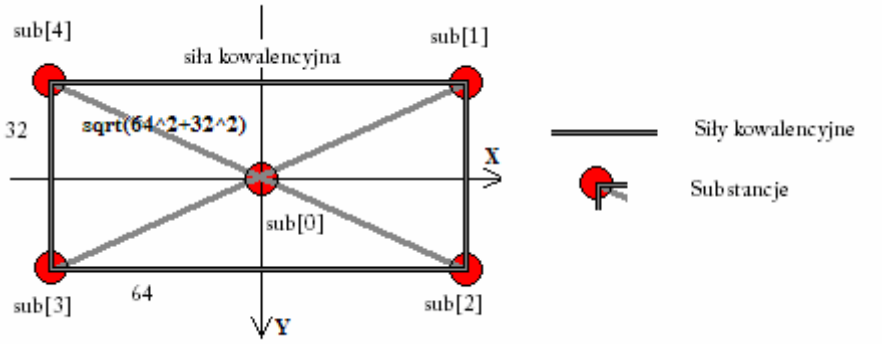
```
xenon_core_header() // skrypt nagłówkowy

// model auta
model[0]=d3d_model_create()
d3d_model_block(model[0], // index modelu
                -64,-32,-8, // pozycja pierwszego wierzchołka
                64,32,16, // pozycja ostatniego wierzchołka
                1,1) // ilość powtarzalnej tekstury poziomo i pionowo
model[1]=d3d_model_create()
d3d_model_block(model[1],-64+16,-32,16,64-48,32,16+24,1,1)
model[2]=d3d_model_create()
d3d_model_ellipsoid(model[2], // index modelu
                    -64+24-16,-32-16,-16, // pozycja pierwszego wierzchołka
                    -64+24+16,-32+16,16, // pozycja ostatniego wierzchołka
                    1,1, // ilość powtarzalnej tekstury poziomo i pionowo
                    12) // jakość elipsoidy - ilość powierzchni
d3d_model_ellipsoid(model[2],64-24-16,-32-16,-16,64-24+16,-32+16,16,1,1,12)
d3d_model_ellipsoid(model[2],64-24-16,32-16,-16,64-24+16,32+16,16,1,1,12)
d3d_model_ellipsoid(model[2],-64+24-16,32-16,-16,-64+24+16,32+16,16,1,1,12)

// grawitacja auta
grav_rate=1
for(i=0;i<6;i+=1)
grav[i]=0

// zmienne odpowiadające za prędkość, pozycję i kierunek auta
```

```
spd=0
spd_max=64
height=32
x=7.5*1024
y=7.5*1024
z=xenon_collision_terrain_simple_get_z(ground,x,y)+height // pozycja Z równa
wysokości terenu na pozycji X Y + wysokość zawieszenia auta
alpha=0
beta=0
gamma=0

// tablica pozycji substancji
for(i=0;i<5;i+=1)
{
subx[i]=x
suby[i]=y
subz[i]=z
}

// a teraz zabawa z substancjami i siłami kowalencyjnymi
// substancje
sub[0]=xenon_core_substance_create() // tworzymy substancję, która będzie
środkiem "podwozia" auta
xenon_core_physic_set(XE_SUBSTANCE,sub[0]) // kopiujemy naszą substancję do
odnośnika elementu aktywnego (by mieć do niej szybki dostęp)
xenon_core_substance_set_position(XE_NULL, // wartość XE_NULL stawiamy wtedy,
gdy odwołujemy się do odnośnika elementu aktywnego. Gdybyśmy wpisali po prostu
sub[0] uzyskalibyśmy ten sam efekt, lecz dostęp do tej substancji trwał by
dłużej, zależnie od ilości substancji
                                  x,y,z) // pozycja substancji
xenon_core_substance_set_mass(XE_NULL,100) // ustawiamy masę substnacji
xenon_core_substance_set_range(XE_NULL,height/2) // a tutaj jej promień

// odtąd kolejne substancje to koła auta
sub[1]=xenon_core_substance_create()
xenon_core_physic_set(XE_SUBSTANCE,sub[1])
xenon_core_substance_set_position(XE_NULL,x+64,y-32,z)
xenon_core_substance_set_mass(XE_NULL,100)
xenon_core_substance_set_range(XE_NULL,height)
sub[2]=xenon_core_substance_create()
xenon_core_physic_set(XE_SUBSTANCE,sub[2])
xenon_core_substance_set_position(XE_NULL,x+64,y+32,z)
xenon_core_substance_set_mass(XE_NULL,100)
xenon_core_substance_set_range(XE_NULL,height)
sub[3]=xenon_core_substance_create()
xenon_core_physic_set(XE_SUBSTANCE,sub[3])
xenon_core_substance_set_position(XE_NULL,x-64,y+32,z)
xenon_core_substance_set_mass(XE_NULL,100)
xenon_core_substance_set_range(XE_NULL,height)
sub[4]=xenon_core_substance_create()
xenon_core_physic_set(XE_SUBSTANCE,sub[4])
xenon_core_substance_set_position(XE_NULL,x-64,y-32,z)
xenon_core_substance_set_mass(XE_NULL,100)
xenon_core_substance_set_range(XE_NULL,height)

// siły kowalencyjne
// łączymy koła z podwoziem
val=sqrt(64*64+32*32) // odległość od koła do środka auta

_cf=xenon_core_covalence_force_create()
xenon_core_physic_set(XE_COVALENCE_FORCE,_cf)
xenon_core_covalence_force_set_substance(XE_NULL, // odnośnik elementu aktywnego
                                         XE_COVALENCE_FORCE_SUBSTANCE_1, //
substancja pierwsza
```

```
                                        sub[0]) // substancja naszego
"podwozia"
xenon_core_covalence_force_set_substance(XE_NULL, // odnośnik elementu aktywnego
                                        XE_COVALENCE_FORCE_SUBSTANCE_2, //
substancja druga
                                        sub[1]) // jedno z kół auta
xenon_core_covalence_force_set_force(XE_NULL,val) // siła, czyli odległość na
jaką utrzymywane będą obie substancje
xenon_core_covalence_force_set_resilience(XE_NULL,1) // sprężystość siły -
ustala jak bardzo mogą się oddalić od siebie dwie substancje
_cf=xenon_core_covalence_force_create()
xenon_core_physic_set(XE_COVALENCE_FORCE,_cf)
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_1,
sub[0])
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_2,
sub[2])
xenon_core_covalence_force_set_force(XE_NULL,val)
xenon_core_covalence_force_set_resilience(XE_NULL,1)
_cf=xenon_core_covalence_force_create()
xenon_core_physic_set(XE_COVALENCE_FORCE,_cf)
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_1,
sub[0])
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_2,
sub[3])
xenon_core_covalence_force_set_force(XE_NULL,val)
xenon_core_covalence_force_set_resilience(XE_NULL,1)
_cf=xenon_core_covalence_force_create()
xenon_core_physic_set(XE_COVALENCE_FORCE,_cf)
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_1,
sub[0])
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_2,
sub[4])
xenon_core_covalence_force_set_force(XE_NULL,val)
xenon_core_covalence_force_set_resilience(XE_NULL,1)

// teraz łączymy koła ze sobą
val=32*2
_cf=xenon_core_covalence_force_create()
xenon_core_physic_set(XE_COVALENCE_FORCE,_cf)
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_1,
sub[1])
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_2,
sub[2])
xenon_core_covalence_force_set_force(XE_NULL,val)
xenon_core_covalence_force_set_resilience(XE_NULL,1)
val=64*2
_cf=xenon_core_covalence_force_create()
xenon_core_physic_set(XE_COVALENCE_FORCE,_cf)
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_1,
sub[2])
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_2,
sub[3])
xenon_core_covalence_force_set_force(XE_NULL,val)
xenon_core_covalence_force_set_resilience(XE_NULL,1)
val=32*2
_cf=xenon_core_covalence_force_create()
xenon_core_physic_set(XE_COVALENCE_FORCE,_cf)
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_1,
sub[3])
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_2,
sub[4])
xenon_core_covalence_force_set_force(XE_NULL,val)
xenon_core_covalence_force_set_resilience(XE_NULL,1)
val=64*2
```

```
_cf=xenon_core_covalence_force_create()
xenon_core_physic_set(XE_COVALENCE_FORCE,_cf)
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_1,
sub[4])
xenon_core_covalence_force_set_substance(XE_NULL,XE_COVALENCE_FORCE_SUBSTANCE_2,
sub[1])
xenon_core_covalence_force_set_force(XE_NULL,val)
xenon_core_covalence_force_set_resilience(XE_NULL,1)
```

Uff, dużo tego. No to zaczynamy :).
Te elementy… A może pokażę Wam lepiej na rysunku, to łatwiej zrozumiecie, o co chodzi.



Jak widać ten układ substancji i sił międzycząsteczkowych sprawi, że nasze auto będzie stabilne i substancje nie będą uciekać gdzie zechcą. Reszta jest wyjaśniona w kodzie, więc nie będę się rozpisywał.
Następnym eventem jest step. W nim będziemy dokonywać akcji sterujących nim.

```
xenon_core_physic_set(XE_SUBSTANCE,sub[0]) // kopiujemy pierwszą substancję do
odnośnika elementu aktywnego

// akcje ruchu auta
if(z<=xenon_collision_terrain_simple_get_z(ground,x,y)+height+6)
{
if(keyboard_check(vk_up))spd+=1
else if(keyboard_check(vk_down))spd-=1
else spd-=0.5
if(keyboard_check(vk_space))spd-=4
if(spd<0)spd=0
if(spd>spd_max)spd=spd_max
if(keyboard_check(vk_left))rot=3
else if(keyboard_check(vk_right))rot=-3
else rot=0
}
else rot=0

for(i=0;i<5;i+=1)
{
// pobieranie pozycji poszczególnych substancji auta
subx[i]=xenon_core_substance_get(XE_NULL,XE_SUBSTANCE_X) // pozycja X
suby[i]=xenon_core_substance_get(XE_NULL,XE_SUBSTANCE_Y) // pozycja Y
subz[i]=xenon_core_substance_get(XE_NULL,XE_SUBSTANCE_Z) // pozycja Z
xenon_core_physic_next(XE_SUBSTANCE) // przeskok odnośnika aktywnego elementu o
jedną pozycję dalej
}
repeat(5)
```

```
xenon_core_physic_back(XE_SUBSTANCE) // przeskok odnośnika aktywnego elementu o
jedną pozycję wstecz

// przesunięcie auta o wektor prędkości
for(i=0;i<5;i+=1)
{
subx[i]+=xenon_core_lengthdir_x(alpha,beta,spd)
suby[i]+=xenon_core_lengthdir_y(alpha,beta,spd)
subz[i]+=xenon_core_lengthdir_z(alpha,beta,spd)
}
repeat(4)xenon_core_physic_back(XE_SUBSTANCE)

// obrót auta
for(i=1;i<5;i+=1)
{
ta=xenon_core_direction_alpha(subx[0],suby[0],subz[0], // pozycja pierwszego
punktu
                              subx[i],suby[i],subz[i]) // pozycja drugiego
punktu
tb=xenon_core_direction_beta(subx[0],suby[0],subz[0],subx[i],suby[i],subz[i])
tl=xenon_point_distance(subx[0],suby[0],subz[0],subx[i],suby[i],subz[i]) //
dystans pomiędzy dwoma punktami
ta-=rot*sin(degtorad(spd/spd_max*90)) // obrót względny do prędkości
subx[i]=subx[0]+xenon_core_lengthdir_x(ta,tb, // kąty alpha i beta
                                       tl) // długość
suby[i]=suby[0]+xenon_core_lengthdir_y(ta,tb,tl)
subz[i]=subz[0]+xenon_core_lengthdir_z(ta,tb,tl)
xenon_core_physic_next(XE_SUBSTANCE)
}

// obliczenia pozycji substancji
for(i=0;i<5;i+=1)
{
tz=xenon_collision_terrain_simple_get_z(ground,subx[i],suby[i]) // wysokość
terenu na pozycji danej substancji
// sprawdza czy substancja koliduje z terenem. Jeśli tak, to ustawia jej
wysokość równą wysokości terenu + height i zeruje grawitacje danej substancji
subz[i]-=grav[i]
grav[i]+=grav_rate
if(subz[i]<tz+height)
{
grav[i]=0
subz[i]=tz+height
}

// aktualizacja pozycji obiektu auta
if(i=0)
{
x=subx[i]
y=suby[i]
z=subz[i]
}

xenon_core_substance_set_position(XE_NULL,subx[i],suby[i],subz[i]) //
aktualizuje dokonane przez nas zmiany w substancji
xenon_core_physic_next(XE_SUBSTANCE)
}

// aktualizacja kierunku auta
// punkt pomiędzy dwoma sąsiednimi kołami
tx=mean(subx[1],subx[2])
ty=mean(suby[1],suby[2])
tz=mean(subz[1],subz[2])
// kąty alpha i beta między środkiem auta a punktem wyżej wyliczonym
```

```
alpha=xenon_core_direction_alpha(x,y,z,tx,ty,tz)
beta=xenon_core_direction_beta(x,y,z,tx,ty,tz)
gamma=-
xenon_core_direction_beta(subx[1],suby[1],subz[1],subx[2],suby[2],subz[2])

// ustawiamy pozycje i kąt obrotu kamery, takie jak u naszego auta
engine.x=x
engine.y=y
engine.z=z
engine.alpha=-alpha+180
engine.beta=-15
```

Na początku kopiujemy pierwszą substancję auta do odnośnika elementu aktywnego, byśmy mieli łatwy dostęp do niego i reszty kolejnych. Wystarczy wtedy tylko przesuwać iterator tegoż odnośnika w przód lub wstecz i nie tracimy przy tym fps-ów przy dużej ilości kodu edycji/pobierania danych z XenoNa. Pierwszy warunek, sprawdza czy auto dotyka ziemi. Jeśli dotyka to sprawdza stan klawiatury i ustala prędkość lub kierunek względny auta. Jeśli zaś nie dotyka ziemi to zeruje kierunek względny. Następnym krokiem jest pobranie pozycji wszystkich substancji auta i zapisanie ich do odpowiednich zmiennych. Następnie dzięki funkcjom lengthdir 3d XenoNa przesuwamy auto o wektor prędkości, potem używając substancji **sub[0]** („podwozia") będziemy obracać względem niej resztę substancji o kierunek względny (zmienne **rot**). Końcowe obliczenia dotyczą grawitacji punktów, by nasze auto nie fruwało. Potem tylko aktualizujemy zmienne kierunku auta i przekazujemy obiektowi **engine** pozycję i kierunek auta dla kamery.

Na koniec słów kilka:
Jak widać gry 3d należą do tych, które dobre efekty uzyskują dzięki dłuższej pracy nad nimi. Nie twierdzę, że ten kurs jest idealnym przykładem gry 3d. Wręcz przeciwnie – wiele rzeczy zostało w nim uproszczonych, choćby kąt gamma auta, który wymaga dokładniejszych wyliczeń. Ale to w końcu miał być prosty kurs prostej gry, tyle że „upiększonej" silnikiem XenoN Core. Dla początkujących pisanie gier 3d powiem, że to nie jest ani trudne, ani proste – to trzeba po prostu zrozumieć. Ponadto potrzebna jest choć minimalna wiedza z zakresu matematyki. Ale bez obawy - w każdym z kolejnych kursów postaram się Wam wyjaśnić najlepiej jak się da zagadnienia trójwymiarowego świata i panujących tam zasad.

Autor: *PsichiX* (ΨΧΞ)

Zapraszam na:

- ***GMClan.org***
- ***XenoN Core Website***
- ***YoYoGames***