Samouczek do programu Game Maker

Samolotowa strzelanka

Napisane przez : Mark Overmars (*Copyright © 2007-2009 YoYo Games Ltd), 23.12.2009*
Tłumaczenie : Grzegorz Madejski
Korzysta z: Game Maker 8.0, Edycja Lite lub Pro, Tryb: Simple Mode, Advanced Mode
Poziom : początkujący

Strzelanki to gry akcji stosunkowo łatwe do stworzenia w programie GameMaker. W procesie tworzenia takiej gry nauczysz się nowych rzeczy takich jak korzystanie ze zmiennych.

Ogólnie można opisać je jako gry, w których gracz kontroluje obiekt taki, jak samolot, statek kosmiczny lub samochód, a ich cechą charakterystyczną jest też przewijające się tło dlatego nazywane są „przewijającymi się strzelankami" (scrolling shooter). Na planszy pojawiają się przeszkody, wrogie jednostki, które trzeba zestrzelić, a czasem bonusy np. apteczki leczenia, amunicja, nowa broń itp. Z reguły wraz z postępem gry, wzrasta trudność. W tym samouczku stworzymy grę pod tytułem *1945*, w której gracz leci samolotem nad morzem i musi walczyć z samolotami wroga.

Zajmiemy się wieloma aspektami gry, takimi jak tworzenie iluzji przesuwającego się tła, kontrola samolotu, tworzenie wrogów i pocisków, wyświetlanie wyniku, obsługa stanu zniszczenia samolotu czy liczba żyć. Jednak zanim to zrobimy musimy zrozumieć czym są zmienne i jak ich użyć by wzbogacić grę.

Zmienne i właściwości

Czy jest więc zmienna (ang. *variable*)? Można ją rozumieć jako właściwość instancji obiektu. Jak już wiesz, niektóre właściwości (ang. *properties*) obiektu można zdefiniować podczas jego tworzenia np. czy jest widzialny (Visible) lub stały (Solid). Są też akcje zmieniające te właściwości np. zmiana położenia lub szybkości. Każda instancja ma pewną ilość takich właściwości. Są też właściwości globalne, takie jak wynik, które nie są związane z żadną instancją. Wszystkie właściwości przechowywane są w tak zwanych **zmiennych**, z których każda jest inaczej nazwana.

O zmiennych można myśleć jak o etykietkach dla właściwości, przykład z życia wzięty: trójkąt ma wysokość, chcąc ją policzyć na lekcji matematyki musisz dać tej wysokości jakąś nazwę np. *h*. Nazwaliśmy więc zmienną *h* i przechowujemy w niej jakąś właściwość (liczbową) np. *h*=10 cm. Poniżej przedstawiamy właściwości/zmienne, które posiada każda instancja obiektu:

`x` współrzędna x obiektu
`y` współrzędna y obiektu
`hspeed` szybkość w poziomie (horyzontalna), jednostka to ilość pikseli na krok
`vspeed` szybkość w pionie (wertykalna), jednostka to ilość pikseli na krok
`direction` aktualny kierunek ruchu wyrażony w stopniach (0-360; 0 to poziomo w prawo)
`speed` aktualna szybkość w danym kierunku
`visible` zmienna oznajmiająca czy obiekt jest widoczny (1) lub niewidoczny (0)
`solid` zmienna oznajmiająca czy obiekt jest stały (1) lub niestały (0)

A teraz wymieńmy zmienne globalne:
score aktualna wartośc liczbowa wyniku
lives aktualna liczba żyć
mouse_x aktualna współrzędna x kursora myszki
mouse_y aktualna współrzędna y kursora myszki
room_speed aktualna szybkość pokoju (w krokach na sekundę)
room_caption napis pojawiający się na pasku tytułowym okna
room_width szerokość pokoju w pikselach
room_height wysokość pokoju w pikselach

Jest wiele innych zmiennych, zarówno dla instancji (czyli obiektów lokalnych), jak i globalnych. Wszystkie są opisane w dokumentacji program *GameMaker*. Zmienne można zmieniać uruchamiając różne akcje, albo zmienić je bezpośrednio. Co więcej, możesz zdefiniować własne zmienne i ich używać. Na przykład, samolot, który stworzymy w naszej grze będzie mógł strzelać tylko raz na pięć kroków, więc będziemy musieli mu nadać zmienną „można strzelać?" którą nazwiemy **can_shoot**. Uwaga: nazwa zmiennej może składać się tylko z liter i podkreślnika, litery duże i małe są rozróżniane więc **Can_shoot** to inna nazwa. Podczas tworzenia samolotu ustawiamy tą zmienną na 1 (1 będzie zwykle oznaczać prawdę, czyli odpowiedź „tak" na pytanie „można strzelać"). Kiedy gracz chce strzelać, sprawdzamy czy ta zmienna jest ustawiona na 1. Gdy strzelimy, zmienną **can_shoot** ustawiamy na 0 (co oznacza, że strzelanie jest czasowo zabronione). Następnie używamy zdarzenia alarmującego (budzik) by odliczył 5 kroków a następnie ustawił tą zmienną z powrotem na 1. Jest to bardzo ważne, bo tak obsługuje się wiele atrybutów statku - jak pole ochronne, ulepszenia broni, itp. Opiszemy to ponownie szczegółowo w dalszej części samouczka.

Są dwie ważne akcje, które zajmują się zmiennymi bezpośrednio. Można je znaleźć w zakładce **Control** (Kontrola):

Set the value of a variable (Ustaw wartość zmiennej). Jak sama nazwa mówi, akcja ta służy do zmianny wartości zmiennej, zarówno wbudowanej w program jak i stworzonej przez ciebie. Wystarczy podać nazwę zmiennej i jej nową wartość. Przy naciśnięci okienka **Relative**, podana wartość jest dodawana do obecnej – jednak tylko pod warunkiem gdy zmienna ma już jakąś wartość. Zamiast podawać wartość bezpośrednio, możesz wpisać też wyrażenie. Na przykład by podwoić wynik w okienku wartości zmiennej **score** wpisz **2*score** (gwiazdka to zwyczajowo znak mnożenia).

If a variable has a value (Jeśli zmienna ma wartość). Tą akcją możesz sprawdzić jaką wartość aktualnie przechowuje zmienna. Jeśli wartość zmiennej jest równa podanej przez ciebie liczbie, to zwracana jest wartość „prawda" (true) i następne akcje są wykonywane. Jeśli nie jest, akcje znajdujące się za nią, nie są wykonywane. Można też pytać o to czy wartość jest mniejsza lub większa od podanego numeru. Możesz oczywiście używać nie tylko liczb, ale też różnorakich wyrażeń.

Poniżej zobaczymy kilka przykładów użycia takich akcji.

Jest jeszcze jedna rzecz, którą musisz wiedzieć. Jak zostało wyżej powiedziane, zmienne mogą być lokalne (przypisane do instancji) lub globalne. Gdy korzystasz z własnych zmiennych, są one zawsze lokalne. Jeśli jednak chciałbyś zdefiniować własną globalną zmienną musisz poprzedzić jej nazwę napisem **global** a następnie kropką. Możesz więc użyć zmiennej **global.bonus** do określenia punktów bonusowych zbieranych przez gracza. Uważaj by przypadkiem nie użyć zmiennej, która nie istnieje. Uważaj także by nie nazwać zmiennej tak samo jak jakiegoś obrazka lub dźwięku. Jak już zauważyłeś różnym obiektom nadawaliśmy nazwy z prefiksem **spr_, snd_, obj_**. Dla zmiennych lokalnych takim prefiksem może być **var_** .

1945

Spójrzmy teraz na grę, którą chcemy zrobić. Wpierw stwórzmy opis naszego projektu. Dla anszej gry *1945* zrobimy krótki opis by nie rozpychać tego samouczka, jednak takie opisy zwykle są długi na wiele stron. Gra będzie wyglądała mniej więcej tak:

Gracz kontroluje duży żółty samolot który leci do góry. Z naprzeciwka lecą trzy samoloty przeciwnika. Na dolnej listwie mamy liczbę żyć, pasek stanu samolotu i wynik (Score).

1945 – dokument projektu gry

Opis

W grze kontrolujesz samolot lecący nad morzem. Napotykasz wzrastającą liczbę wrogich samolotów, które próbują cię zniszczyć. Musisz omijać lub zestrzelić. Celem gry jest przeżycie jak najdłuższy czas i zniszczenie jak największej liczby samolotów.

Obiekty w grze

Tło jest złożone z przewijającego się morza z kilkoma wyspami. Samolot gracza leci nad tym morzem. Możesz strzelać pociskami, które niszczą wrogie samoloty. Wróg dysponuje 4 typami samolotów:

- samoloty, które musisz niszczyć,
- samoloty, które strzelają na dół po linii prostej,
- samoloty, które strzelają w stronę gracza,
- samoloty, które wylatują z dołu zamiast góry.

Dźwięki
Użyjemy dźwięków eksplozji i muzyki w tle.

Kontrola gry
Gracz kontroluje grę za pomocą klawiatury. Klawisze ze strzałkami sterują samolotem, spacją się strzela, ale tylko raz na 5 kroków w grze.

Przebieg gry
Gra rozpoczyna się od razu. Gracz dostaje 3 życia. Gdy zużyje wszystkie życia, pojawi się tabela z najlepszymi wynikami. Po naciśnięciu klawisza <F1> (pomoc) pojawi się krótki opis wyjaśniający, a po naciśnięciu <Esc> gra się kończy.

Poziomy
Jest tylko jeden poziom gry, ale samolotów wroga będzie pojawiać się coraz więcej, i coraz trudniejszych.

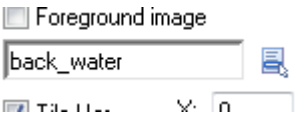
Iluzja ruchu

Strzelanka "z przewiajniem", jak sama nazwa mówi, działa w ten sposób, że świat gry (w naszym przypadku jest to morze z wyspami) przewija się przez ekran, zwykle z dołu do góry, lub z prawa do lewa. To sprawia wrażenie, że gracz się porusza. W naszej grze 1945 świat przewija się pionowo. Nawet jeśli samolot gracza stoi nieruchomo na ekranie, odnosisz wrażenie, że leci nad przewijającym się tłem. Kontrolujesz pozycję samolotu latając nim po ekranie. To daje wrażenie przyśpieszania gdy lecisz do przodu i wrażenie spowolnienia gdy lecisz do tyłu. Konieczne jest by samolot nie poruszał się szybciej do tyłu niż przewijające się tło. To sprawiłoby efekt lecenia do tyłu, co oczywiście jest niemożliwe.

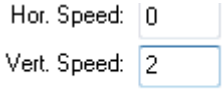
Jak więc stworzyć przewijające się tło w GameMakerze? Są dwie możliwości. Pierwsza, łatwiejsza to wstawienie tła sklejonego z kafelków wody, które poruszają się w dół. Drugie, bardziej skomplikowane podejście zakłada zbudowanie olbrzymiego pokoju, ale pokazywanie na ekranie tylko kawałka tego pokoju (używają tak zwanego widoku ang. *view*). Widok powoli przesuwa się w górę pokoju. My użyjemy pierwszej opcji, ale później przyjrzymy się i drugiej.

Nasza gra dzieje się nad morzem, więc potrzebujemy obrazka tła przypominającego morze widziane z lotu ptaka. Stwórz więc nową grę i nazwij ją 1945, następnie dodaj obrazek tła **water.png** z załączonego folderu do zasobów tła (Background resources) i nazwij go **`back_water`**.

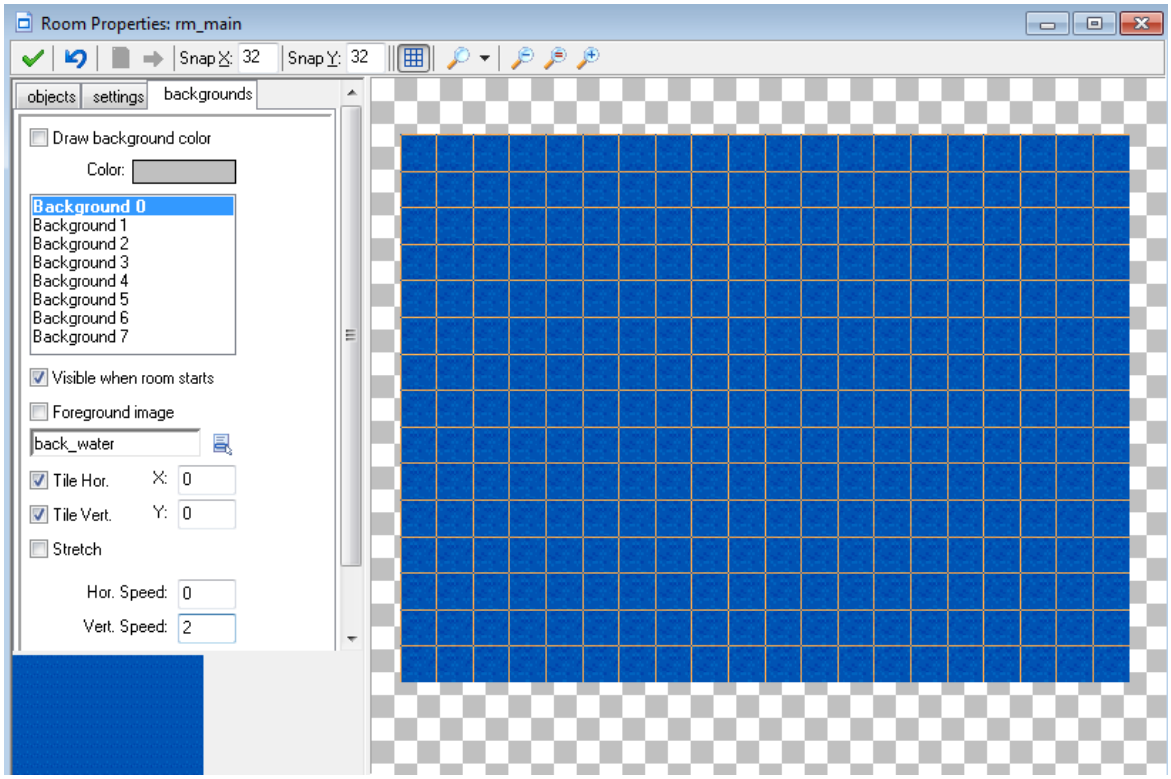
Wypełnienie tła tym kafelkiem da ładnie wyglądające morze. Stwórz pokój sposobem, który już znasz. Nazwij go **`rm_main`** (zakładka **settings**). W zakładce **backgrounds** musimy pozmieniać trochę ustawień. Ponieważ zamierzamy pokój wypełnić obrazkiem, nie potrzebujemy wypełnienia kolorem – odznacz okienko **Draw background color**. Po drugie, wypełnij tło obrazkiem wody, który dodałeś przed chwilą.



Domyślne ustawienie sprawie, że cały pokój wykafelkujemy wodą. I to jest to, co chcemy. Na zakończenie musimy jeszcze sprawić, że tło będzie się poruszało. Ustaw wobec tego pionową szybkość pokoju na 2.



Formatka ustawień pokoju powinna wyglądać tak:



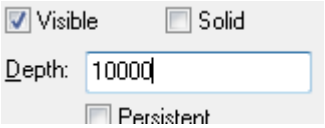
Możesz zapisać i odpalić grę, by sprawdzić efekt lotu nad morzem. Od tej chwili będziemy korzystać z paru opcji dostępnych tylko w trybie zaawansowanym (Advanced Mode). Zmień zatem tryb na zaawansowany poprzez menu **File**, następnie **Advanced Mode**.

By zwiększyć odczucie ruchu, dodamy parę wysp. Prostym sposobem byłoby stworzenie dużego obrazku wyspy i dodanie go do tła. Niestety minusem tego rozwiązania jest to, że obraz ten będzie pojawiał się regularnie i gracz dostrzeże efekt „przelatywania ciągle nad tą samą wyspą". Wybierzemy więc trochę bardziej skomplikowany sposób i dodamy wyspy jako obiekty a nie tło. Dodajmy 3 sprajty bazujące na poniższych obrazkach (skorzystaj z folderu resources):

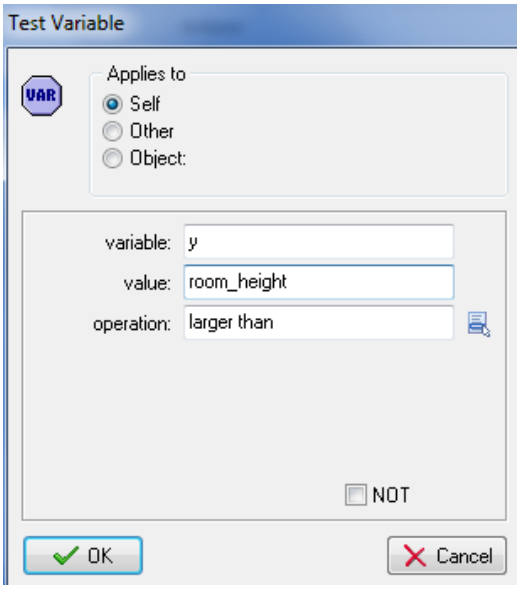
Zauważ, że obrazki te mają przezroczyste tło, jest to cecha m.in. obrazków z rozszerzeniem png. Dodając sprajty odznacz opcję **Precise Collision Checking** (Precyzyjne Badanie Kolizji), jako że nie będziemy badać kolizji obiektów z wyspami. Nazwij je kolejno `spr_island1`, `spr_island2`, `spr_island3`.

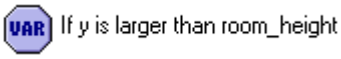
Dla każdej wyspy stwórz obiekt bazujący na stworzonych sprajtach, nazwij je `obj_island1`, `obj_island2`, `obj_island3`. Do każdego z tych obiektów dodajmy zdarzenie tworzenia „Create", a jako akcję do tego zdarzenia dajmy pionowy ruch w dół  również z prędkością 2. Dzięki temu zarówno wyspy jak i morze będą poruszać się z jednakową prędkością, tak jakby były częścią jednego tła. By wyspy nie pojawiały się nad samolotem, tylko samolot nad wyspami, trzeba ja

umieścić głębiej w naszym pokoju. Zwiększmy im zatem głębokość (**Depth**) do wysokiego poziomu: 10000. Im wyższa ta wartość, tym szybciej rysowane na tle są obiekty. Skoro wyspy mają tak wysoką głębokość, to będą rysowane pierwsze, a różne inne obiekty na wierzchu, czyli na nich.



Musimy zrobić jeszcze jedną rzecz. Gdy wyspa zniknie na dole ekranu, musi ponownie pojawić się u góry i znów lecieć na dół (w ten sposób będziemy mieli wrażenie, że lecimy nad kolejnymi wyspami). By to zrobić, dodajmy dla każdej wyspy jako obiektu zdarzenie **Step** (Krok), które dzieje się co… krok (czyli 1/30 sekundy - to bardzo częste zdarzenie). Co krok (czyli 1/30 sekundy) będziemy sprawdzać czy wyspa zniknęła na dole ekranu, i jeśli tak, to przesuniemy ją na górę ekranu by ponownie mogła się przesuwać w dół. Przydatna tutaj będzie zmienna **`y`** wyspy, która pozycję pionową wyspy (**`y=0`** oznacza, że wyspa jest na samej górze). Zmienna **`room_height`** oznacza wysokość pokoju, więc gdy zmienna **`y`** wyspy jest większa niż **`room_height`**, to znaczy, że wyspa wyjechała na dole poza ekran. Dla zdarzenia **Step**, użyjemy więc akcji do testowania zmiennej **Test Variable** uzupełniając ją następująco:

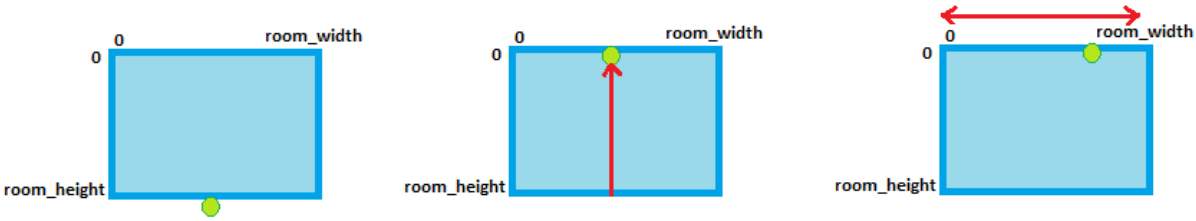


Nasz test wygląda następująco: 

Czy **`y`** jest **`większe`** niż **`room_height`**? Jak widzisz, w okienka można wpisywać nie tylko liczby, ale nazwy innych zmiennych, a nawet wyrażenia. Jeśli ten test zwróci wartość „prawda”(true) tzn odpowiedź na to pytanie będzie pozytywna, to wykonane będą kolejne akcje. Musimy więc dodać te akcje. Dodamy akcję skok do pozycji, ale nie określonej, tylko losowej gdzieś u góry ekranu, by wzorzec powtarzających się wysp był mniej regularny (a gracz nie miał wrażenia monotonii). Ale jak dać losową wartość? Do tego służy funkcja **`random()`**. A co to takiego funkcja? Funkcja to coś co oblicza dla ciebie jakąś wartość, po tym jak podasz jej inne wartości (zwane argumentami), w nawiasach za nazwą funkcji.

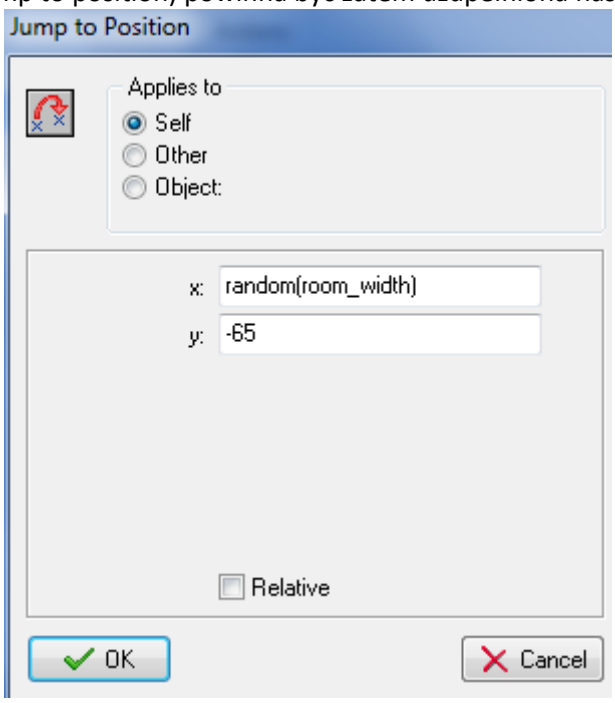
Przykład z życia wzięty: wiemy, że najprostszy wzór fizyczny na obliczanie przebytej drogi **s** znając prędkość **v** i czas **t** to s=vt. Możnaby zrobić funkcję **`oblicz_drogę(v,t)`**, która wymaga podania dwóch wartości i na ich podstawie wyliczy trzecią wartość np. **`oblicz_drogę(3,10)`** wyniesie **`30`**.

Wracamy do naszej funkcji **`random()`**. Można ją wpisywać w różne okienka tak samo jak wartości i nazwy zmiennych. Można to też robić z wieloma innymi funkcjami, które istnieją w GameMakerze. W nawiasy naszej funkcji można wpisać jakąś wartość np. **`random(room_width)`** - to spowoduje, że funkcja random obliczy (a raczej wylosuje) nam wartość spomiędzy **0** a liczby **`room_width`**, taka liczba jest idealna dla współrzędnej x naszej wyspy, która może być po lewej stronie planszy (współrzędna pozioma x: **0**) lub po prawej (współrzędna pozioma x: **`room_width`**), lub gdzieś pomiędzy. Można sobie to zobrazować następująco:

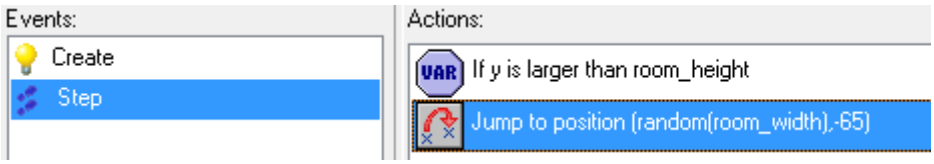


Najpierw wyspa zjeżdża na dół ekranu. Przesuwamy ją z powrotem do góry (współrzędna pionowa y=0 na nawet jeszcze wyżej np. -65 by wyspa była nad ekranem). Oraz losujemy jej współrzędną poziomą za pomocą funkcji random na coś pomiędzy 0 a **`room_width`**. Wszystko to dzieje się w ułamku sekundy więc nie zauważymy żadnego skoku, wszystko będzie płynne.

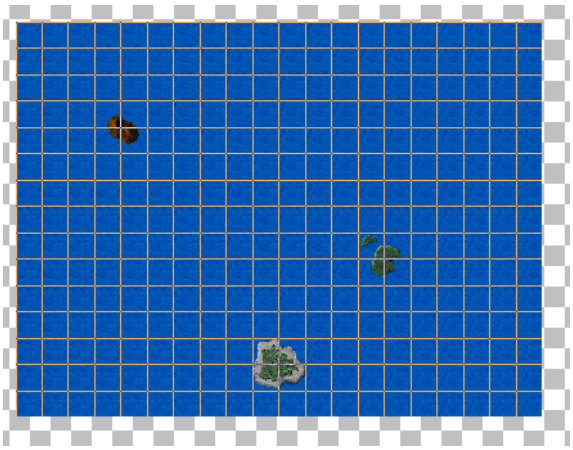
Akcja Skok do pozycji (Jump to position) powinna być zatem uzupełniona nastepująco:



Użyliśmy –65 dla współrzędnej y by mieć pewność, że wyspa po przeniesieniu na górę będzie totalnie schowana nad pokojem. Z powodu jej ruchu w dół, ponownie pojawi się na ekranie. Musimy to zrobić oczywiście dla wszystkich trzech wysp (można zrobić dla jednej i zduplikować, po czym zmodyfikować obiekty).



Oczywiście, musimy też wszystkie wyspy umieścić w pokoju (po 1 z każdego rodzaju) w różnych miejscach na morzu (najlepiej na różnych wysokościach, by wyspy nie pojawiały się seriami tylko pojedynczo). W taki sposób tło będzie gotowe. Możesz wprowadzić dodatkowy element losowości poprzez zastąpienie wartości -65 dla zmiennej y, jakąś wartością losową np. -65-random(100).
Zapisz i uruchom grę, by zobaczyć wyspy w akcji.

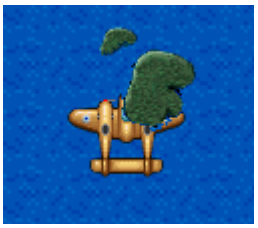


Główny samolot

Teraz, gdy tło jest gotowe możemy zająć się tworzeniem samolotu, który będzie kontrolował gracz. Jest to zadanie nietrudne. Zaczniemy od obrazka samolotu. Nasz samolot będzie miał śmigła. By sprawić wrażenie, że śmigła się kręcą nasz sprajt będzie składał się z trzech obrazków. W każdym z obrazków śmigło jest inaczej ustawione. Oto obrazek **`myplane_strip3.png`** z folderu Resources.

Zwróć uwagę na nazwę pliku, takie animowane sprajty mają nazwę pliku kończącą się na **_stripXX**, gdzie **XX** oznacza liczbę podobrazków (w tym przypadku 3) – to gwarantuje, że *GameMaker* rozpozna 3 klatki naszej animacji samolotu. Podczas tworzenia sprajta musimy zrobić jedną ważną rzecz. Ustawiamy **X** i **Y** dla **Origin** (Źródło) na liczbę 32. To oznacza, że źródło (czy raczej punkt zaczepienia, punkt za który chwytamy samolot i przyklejamy go na planszę, punkt jego współrzędnych) jest w środku obrazka samolotu, a nie w jego lewym górnym rogu. Jest to ważne, by pociski wylatywały właśnie ze środka samolotu, a nie z lewej strony. Zrób więc sprajta **`spr_myplane`** według wskazówek podanych wyżej.

Następnie zrób obiekt naszego samolotu **`obj_myplane`**. Jako sprajta dodaj ten, który przed chwilą zrobiliśmy i nadaj mu głębokość -100, by zagwarantować, że samolot leży nad pociskami, które stworzymy później. Musisz nauczyć się mądrze nadawać odpowiednią głębokość obiektom. Praktycznie w każdej grze, trzeba wiedzieć co leży na czym. Przy źle ustawionej głębokości samolot wleciałby pod wyspę (zobacz obok)!



Na razie nasz obiekt zaopatrzymy w obsługę ruchu. Gdy gracz nic nie robi, samolot się nie porusza (choć przesuwające się tło nadal sprawia, że lecimy do przodu). Jeśli gracz naciśnie którąś ze strzałek na klawiaturze, samolot poleci w tym kierunku. Musimy też zadbać, by samolot nie wyleciał poza pokój. Zamiast nadawać samolotowi prędkość, zrobi ręczną obsługę zmiany położenia.

Spójrzmy na ruch towarzyszący naciśnięciu strzałki w lewo. Dla samolotu dodaj zdarzenie naciśnięcia klawiatury (Keyboard), a konkretniej strzałki w lewo (<Left>). Dla tego zdarzenia dodajmy teraz odpowiednie akcje. Najpierw sprawdzamy czy możemy lecieć w lewo tzn. czy nie jesteśmy za blisko

lewego brzegu planszy. W akcjach dajemy więc test na zmienną **x**, która musi być większa niż 40 (podobnie jak badaliśmy współrzędne wyspy). Jeśli test zwróci prawdę przesuwamy samolot w lewo od aktualnej pozycji. Do tego użyjemy akcji **Jump to position** (Skok do pozycji) z wartościami -4 dla **x** i 0 dla **y** (oczywiście trzeba zaznaczyć **Relative**). Dla strzałki w prawo (<Right>) robimy to samo, tyle że test na zmienną będzie inny: czy **x** jest mniejsze niż **room_width**-40? W przypadku odpowiedzi pozytywnej skok będzie dodawał do **x** wartość 4, a dla **y** nadal 0. Ze strzałkami w górę i dół też sobie poradzisz. Musisz jedynie wiedzieć, że zmieniamy tym razem zmienną **y** o -2 lub 2 (pamiętaj, że nigdy nie możemy ruszać się szybciej do tyłu niż tło). Przy testowaniu czy samolot nie wylatuje na dół ekranu użyj większego marginesu 120. Potrzebne nam będzie tam miejsce na umieszczenie informacji o punktach, życiach i uszkodzeniu.

Nasz samolot może już latać. Umieść jego instancje w pokoju i przetestuj latanie.

Strzelanie

Ale co to za strzelanka, jeśli nie można strzelać? Rozszerzymy teraz naszą grę o nowe elementy – zaczniemy od dodania możliwości strzelania naszemu samolotowi. Potrzebny nam będzie obrazek naboju **bullet.png**, który zapiszemy jako sprajt **spr_bullet**. By uczynić grę bardziej spektakularną, nabój ten będzie dość duży w stosunku do samolotu (takie przesadzanie jest bardzo powszechne w tworzeniu gier ☺).

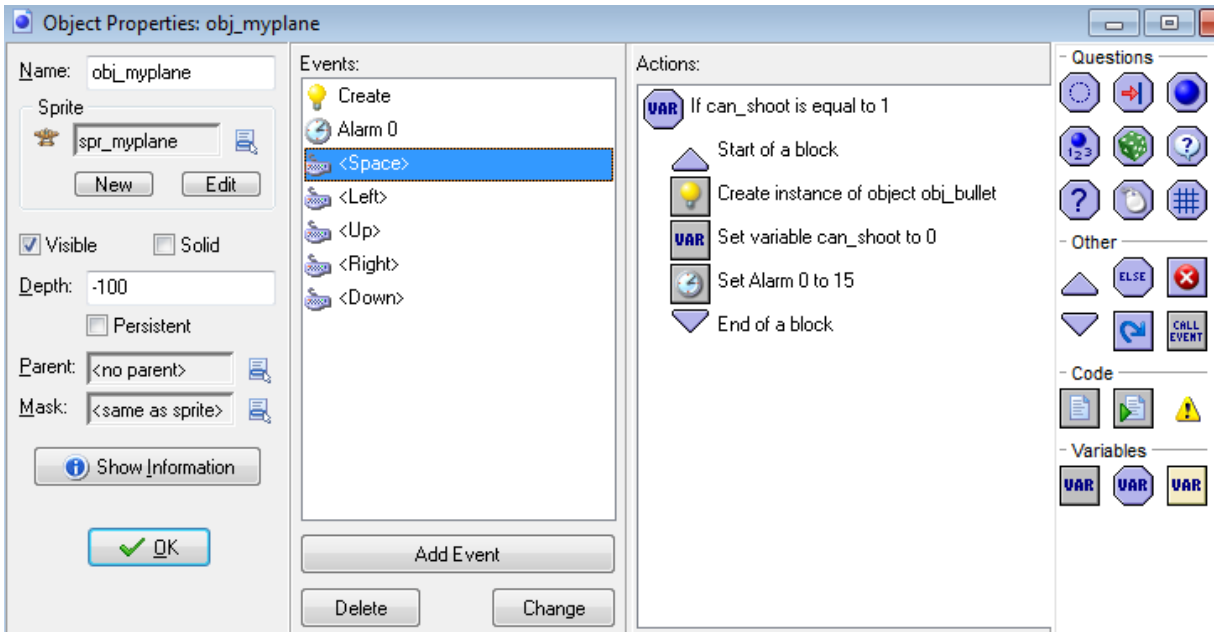
Stwórzmy obiekt na podstawie tego sprajta (dobrą nazwę dla obiektu jesteś już na pewno w stanie wymyślić) i ustaw jego głębokość na 0 tak, aby przy wystrzeleniu pojawiał się nad wyspami (głębokość 10000), ale pod samolotem (głębokość -100). Zachowanie pocisku jest proste, w zdarzeniu tworzenia nadajemy mu prędkość pionową o wartości -8, by poruszał się do góry. Pocisk po wyleceniu poza pokój będzie nadal leciał w nieskończoność, musimy go zatem zniszczyć gdy zniknie nam z oku. Można to łatwo zrobić. Co krok (zdarzenie **Step**) sprawdzaj czy jego współrzędna y jest mniejsza niż -16 odpowiednim testem. Jeśli tak, to użyj akcji **Destroy the instance** (Zniszcz instancję) – która usuwa pocisk i odciąża nasz komputer od myślenia o nim. (Zamiast sprawdzania co krok zmiennej y, możesz też użyć zdarzenia „Poza pokojem” **Outside room**).

Pocisk powinien wylecieć po tym jak gracz naciśnie spacje. Tak jak w większości strzelanek naboje powinny lecieć dopóki klawisz jest wciśnięty. Jednak nie chcemy by samolot wystrzeliwał z siebie naraz zbyt dużo amunicji, bo gra byłaby zbyt prosta. Chcemy by gracz mógł strzelić tylko raz na pół sekundy, czyli 15 kroków. By wprowadzić takie ograniczenie użyjemy nowej zmiennej **`can_shoot`**, o której wspomnieliśmy wcześniej. W zdarzeniu tworzenia samolotu dodaj akcję ustawiającą tą zmienną na 1, co oznacza, że samolot może strzelić. Do samolotu dodajemy zdarzenie naciśnięcia spacji (<Space>), a do niego akcję sprawdzającą czy zmienna **`can_shoot`** jest równa 1. Jeśli tak, to wykonamy trzy akcje:

- tworzymy pocisk (akcja **Create instance**) przed samolotem w pozycji względnej (Relative) o współrzędnych (0,-16),
- ustawiamy zmienną **`can_shoot`** na 0, co oznacza, że czasowo blokujemy strzelanie,
- ustawiamy alarm 0 na 15 (pamiętaj, że alarm tyka co krok – czyli ustawiliśmy budzik na dzwonienie za pół sekundy).

Ponieważ test na zmienną **`can_shoot`** dotyczy wszystkich tych trzech akcji, to musimy je spiąć w blok używają odpowiednich cegiełek akcji. Przedstawiono to poniżej na rysunku.

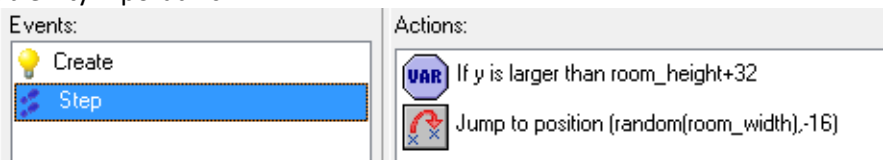
A co się stanie gdy alarm zadzwoni? Dla takiego zdarzenia ustawiamy akcję zmiany zmiennej **`can_shoot`** z powrotem na 1 (tzn. że po pół sekundy od wystrzelenia znowu możemy strzelać).



Zmiana częstotliwości strzelania jest możliwa poprzez zmianę czasu odpalenia budzika. (W niektórych grach można przyśpieszyć strzelanie poprzez ciągłe naciskanie spacji, a nie przytrzymanie jej. Takie coś można również stworzyć w *GameMakerze* za pomocą podobnych zabiegów jak zrobiliśmy u góry, ale bez konieczności badania zmiennej)

Wrogie samoloty

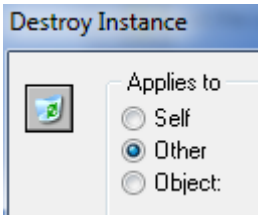
Pora na dodanie wrogów. Naszym pierwszym wrogiem będzie mały samolot, który po prostu leci na dół i nie strzela. Jednak gdy się zderzy z naszym samolotem gra się kończy. Stwórz na podstawie obrazka `enemy1_strip3.png` sprajta `spr_enemy1` z punktem zaczepienia (Origin) ustawionym na (16,16). Na jego podstawie stwórz obiekt `obj_enemy1` i w zdarzeniu tworzenia dodaj prędkość pionową (Vertical speed) z wartością 4, by samolot leciał na dół. Gdy samolot zleci na dół, poza pokój powinien pojawić się u góry pokoju by znowu lecieć na dół. Robiłeś to samo z wyspami, więc powinieneś sobie z tym poradzić.



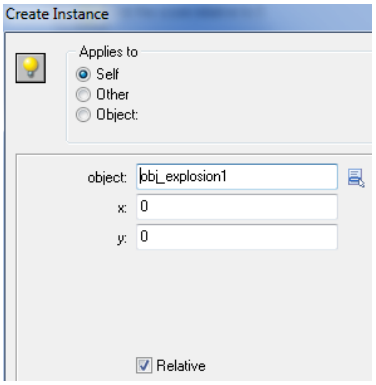
Następnie musimy dodać dwa ważne zdarzenia związane z kolizjami: kolizja z pociskiem naszego samolotu i kolizja z naszym samolotem (która niszczy nasz samolot i zakańcza grę). Zaczniemy od kolizji z pociskiem. Zdarzenie to pociąga za sobą kilka akcji:

- Odegranie dźwięku eksplozji. Oczywiście najpierw musimy dodać dźwięk eksplozji `snd_explosion1.wav` w znany już Tobie sposób. Mając dźwięk w naszych zasobach, można już łatwo dodać akcję odegrania tego dźwięku w zdarzeniu kolizji.

- Usunięcie pocisku. Używamy do tego akcji usunięcia instancji, ale w okienku zamiast **Self** ustawiamy **Other**. Oznacza to, że nie zniszczy się samolot wroga, tylko to, z czym się zderza (pocisk).



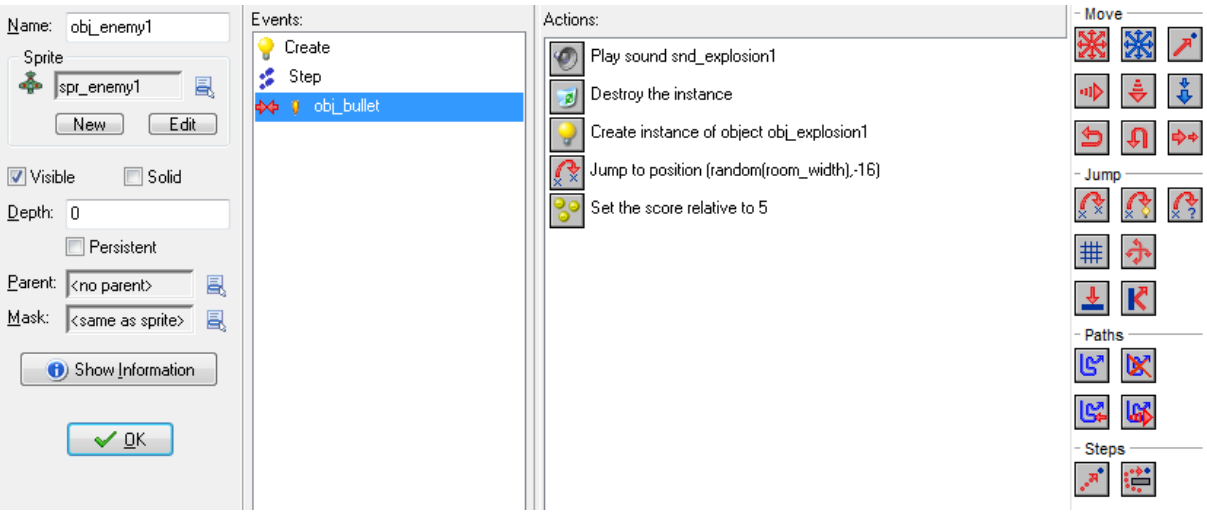
- Odegranie animacji eksplozji. Do tego potrzebujemy sprajta na podstawie obrazka `explosion1_strip6.png`. Ustawmy jego punkt zaczepienia czy źródło (Origin) na (16,16). Na podstawie sprajta stwórzmy obiekt. Będzie on miał tylko jedne zdarzenie – koniec animacji (**Animation end**), dla którego akcją będzie zniszczenie instancji (gdy wszystkie klatki animacji eksplozji się skończą to jest ona trwale usuwana). Mając obiekt eksplozji jesteśmy gotowi dodania trzeciej akcji do kolizji wrogiego samolotu z pociskiem. Jest to akcja stworzenia instancji obiektu eksplozji w pozycji względnej (Relative) o współrzędnych (0,0). To zagwarantuje, że eksplozja stworzy się dokładnie w środku wrogiego samolotu (pamiętasz gdzie ustawialiśmy środek **Origin** wrogiego samolotu, żeby faktycznie był to środek obrazka?).



- Teraz moglibyśmy zniszczyć wrogi samolot, ale zamiast tego… wykorzystamy go jeszcze raz jako nowego wroga. Wystarczy przesunąć go nad pokój w losowej pozycji. Samolot znowu wyleci (bo ma ustawioną prędkośc pionową w dół) więc gracz będzie myślał, że to nowy samolot. Trzeba dodać akcję **Jump to position** z odpowiednimi parametrami. Powinieneś wiedzieć jakimi, bo podobnie zarządzaliśmy wyspami.

- Ostatnią akcją będzie dodanie 5 punktów do wyniku. Pamiętaj, aby zaznaczyć opcję **Relative** – to gwarantuje dodanie 5 punktów do obecnej wartości, a nie ustawienie punktów na wartość 5.

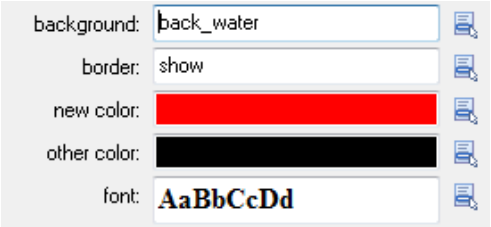
Zdarzenie i jego akcje powinny wyglądać ostatecznie mniej więcej tak:



Kolejne zdarzenie związane z wrogim samolotem to zderzenie z samolotem gracza. Znów będziemy potrzebować dźwięku eksplozji (trochę głośniejszego) i sprajta eksplozji (trochę większego). Dodaj więc dźwięk i sprajt na podstawie plików `snd_explosion2.wav` i `explosion2_strip7.png`. Przy tworzeniu sprajta eksplozji punkt **Origin** musi wskazywać środek obrazka. Skoro obrazek eksplozji ma 65 pikseli szerokości I wysokości, to jaki punkt leży w środku? Tworzymy dla sprajta eksplozji obiekt. Tym razem ten obiekt będzie miał więc akcji ponieważ musi też zakończyć grę. W akcji zakończenia animacji **Animation end** dodajemy kilka akcji:

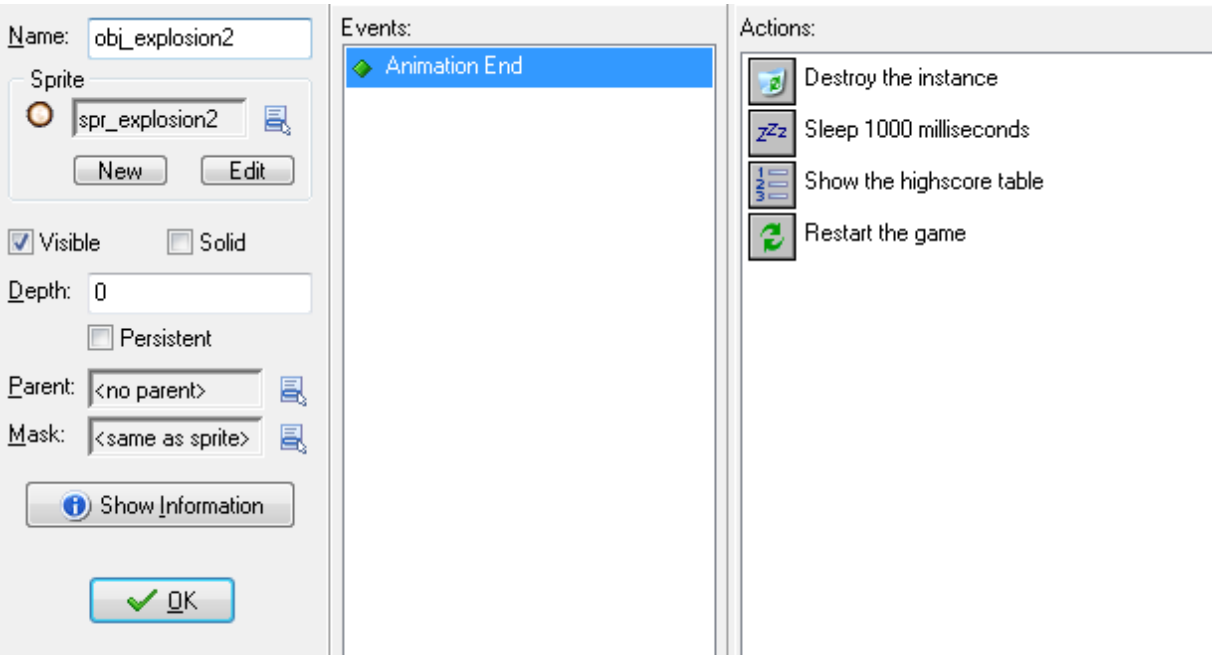
- Niszczymy instancję eksplozji, by zniknęła i nie odtwarzała się w kółko.

- Potem czekamy 1 sekundę, aż skończy się odtwarzać dźwięk eksplozji (dopiero gdy dźwięk wybuchu się skończy to kończymy grę). Wykorzystamy do tego akcję **Sleep** (Śpij) z wartością 1000 milisekund.
- Następną akcją będzie pokazanie tabeli najlepszych wyników (**Highscore table**). Nie namęczymy się z tym (wystarczy jedna akcja), bo program GameMaker sam tworzy taką tabelę i w dodatku gracz ma tam miejsce na wpisywanie swojego imienia. Można jednak w okienku tej akcji ustawić jak ma wyglądać lista najlepszych wyników (kolor okna, tło, czcionkę – przykład po prawej).



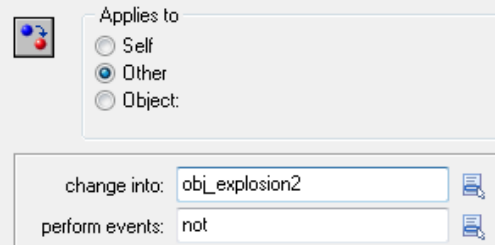
- Ostatnią akcją musi być restart gry **Restart the game**.

Zdarzenie z jego akcjami wyglądać powinno następująco:



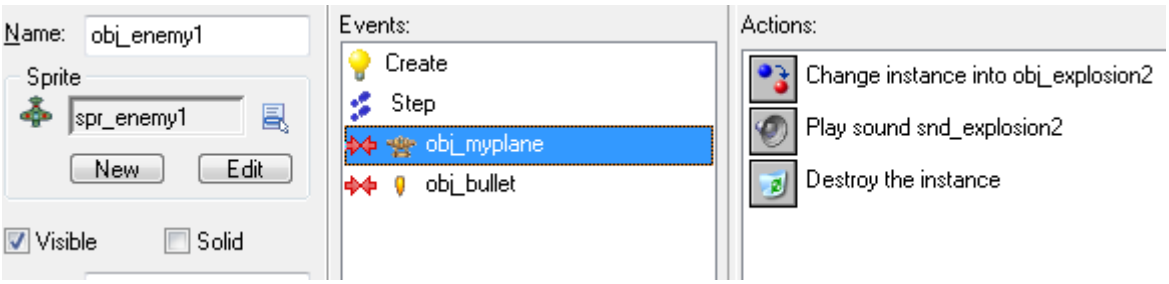
Wracamy do naszego wrogiego samolotu i dodajemy mu nowe zdarzenie: kolizję z samolotem gracza. Dla tego zdarzenia musimy dodać parę akcji:

- Wrogi samolot zderza się z samolotem gracza. Samolot gracza znika, pojawia się eksplozja. Musimy sprytnie zamienić samolot gracza na eksplozję. Można to zrobić za pomocą zamiany instancji samolotu na instancję eksplozji **Change instance**. Pamiętaj, żeby zaznaczyć **Other**, bo **Self** zniszczyłoby instancję samolotu wrogiego.

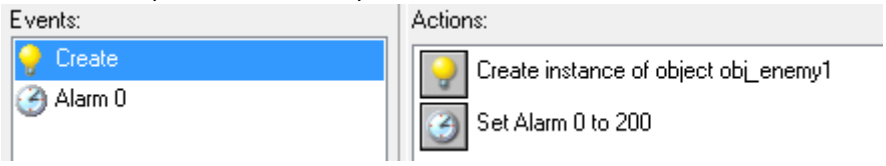


- Drugą akcją będzie odpalenie dźwięku eksplozji, który dodaliśmy.
- I wreszcie usuwamy wrogi samolot za pomocą akcji **Destroy instance**.

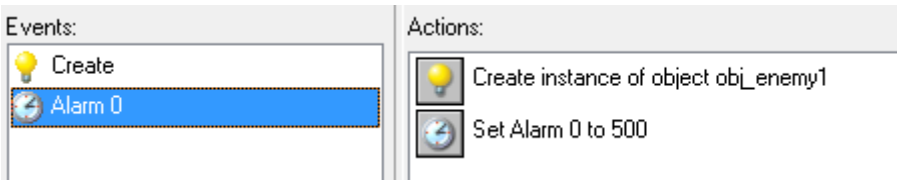
Powinno to wyglądać tak:



Ostatnią rzeczą, którą należy zrobić jest umieszczenie wrogich samolotów w pokoju. Nie będziemy tego robić ręcznie, bo to spowoduje, że cały czas będzie taka sama ilość samolotów w grze (tyle ile wstawimy na początku). Chcielibyśmy, aby możliwe było dodanie większej ilości samolotów z czasem. Stworzymy zatem specjalny obiekt o nazwie **`controller_enemy`**. Będzie on kontrolował tworzenie wrogich samolotów. Tworząc go, nie dodajmy żadnego sprajta i ustawiamy go jako niewidzialnego (odznaczamy pole **Visible**). W zdarzeniu tworzenia (Create) dla tego obiektu dodajmy akcję tworzenia nowego samolotu wroga w losowej pozycji tuż nad pokojem oraz dodajemy akcję ustawienia alarmu na 200 (niecałe 7 sekund).



Dodajemy teraz zdarzenie związane z odpaleniem alarmu. W reakcji na to zdarzenie dodajemy kolejny samolot wroga (znów ponad pokojem) i ustawiamy budzik na 500 (około 16 sekund).



I to wszystko. Efekt będzie taki, że na początku gry pojawi się tylko jeden samolot. Po 200 krokach (7 sekund) pojawi się następny. Kolejne samolot będą pojawiały się w odstępach 15 sekundowych.

Dlaczego takie duże odstępy czasowe? Przecież gra będzie zbyt wolna!
Otóż wcale nie! Pamiętaj, że wrogi samolot po zestrzeleniu wcale nie znika, tylko pojawia się ponownie u góry ekranu (ustawialiśmy to w zderzeniu z pociskiem gracza). Czyli liczba samolotów będzie stale rosła.

Wejdź do pokoju i wrzuć obiekt **`controller_enemy`** gdziekolwiek na planszę. Zapisz i odpal grę by zobaczyć efekty.

Wynik, życia, zdrowie

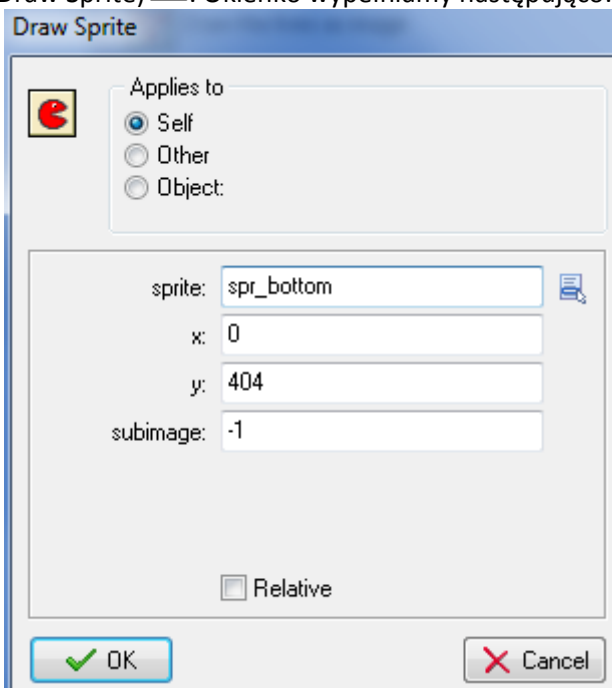
Gra dotychczas miała mało satysfakcjonujące zakończenie. Zawsze kończy się gdy zostaniesz uderzony. By była bardziej interesująca, sprawimy, że samolot będzie miał pewną ilość punktu zdrowia. Na skutek ataku wrogów punkty zdrowia będą zmniejszane, a gdy dojdą do 0, samolot zostanie zniszczony. Dodamy też kilka żyć i stworzymy pasek z wszystkimi tymi informacjami, wraz w wynikiem. Na szczęście będzie to proste, bo *GameMaker* ma wbudowane mechanizmy do kontroli wyników, żyć i zdrowia.

Zacznijmy od stworzenia nowego obiektu kontrolującego stan naszego samolotu **`controller_life`**. Nie potrzebuje on sprajta - co więcej, rysowanie (np. pasku życia) będziemy tym razem robić wykorzystując różne zdarzenia i akcje. Sprajt ma to ograniczenie, że jest rysowany zawsze (w każdym kroku gry, czyli klatce animacji). Wykorzystując akcje w zdarzeniu rysowania, możemy jednak już bardziej kontrolować proces rysowania tzn. co i kiedy będzie narysowane. Mnóstwo akcji rysujących jest w zakładce **Draw** (Rysuj), które powinno się wstawiać do zdarzenia rysowania 

. Do tego zdarzenia można oczywiście wstawiać i akcje z innych zakładek. Na odwrót jednak zrobić nie można, tzn. nie powinno się wstawiać akcji rysujących do innych zdarzeń, bo nie będą działały.

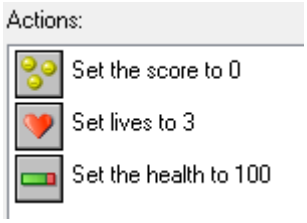
Mamy obiekt **`controller_life`**. Teraz stwórzmy sprajta dla paska informacji (możesz nazwać go **`spr_bottom`** lub inaczej), który wygląda następująco:

Pasek ten będzie pokazywał wynik (Score), zdrowie (czarny pasek po lewej) i liczbę żyć, które pozostały w formie rysunku samolotów. Do obiektu **controller_life** dodaj zdarzenie rysowania, w nim akcję rysowania sprajta (Draw Sprite)  . Okienko wypełniamy następująco:

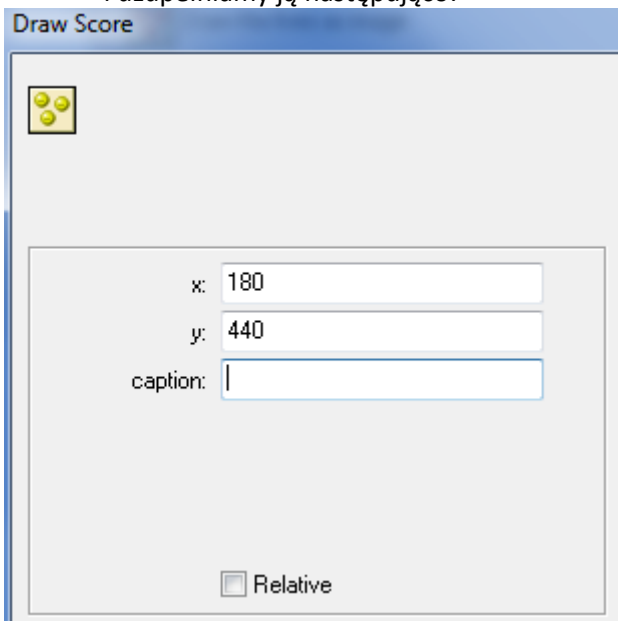


W ten sposób narysujemy sprajta przedstawiającego pasek (spr_bottom), na współrzędnych x=0 (zaczynając od lewej strony) i y=404 (czyli na dole ekranu). Pole subimage oznacza, który obrazek ma być rysowany w przypadku, gdy sprajt składa się z wielu podobrazków. U nas jest tylko jeden obrazek więc zostawiamy -1, co oznacza „rysuj aktualny obrazek". By mieć pewność, że pasek będzie leżał zawsze na wierzchu nadamy obiektowi **`controller_life`** głębokość (Depth) -10000.

Dodajmy zdarzenie kreacji dla obiektu **`controller_life`**. W nim ustawmy wynik na 0, liczbę żyć na 3 i zdrowie na 100. Wszystkie akcje potrzebne by to zrobić znajdują się w zakładce **Score**.

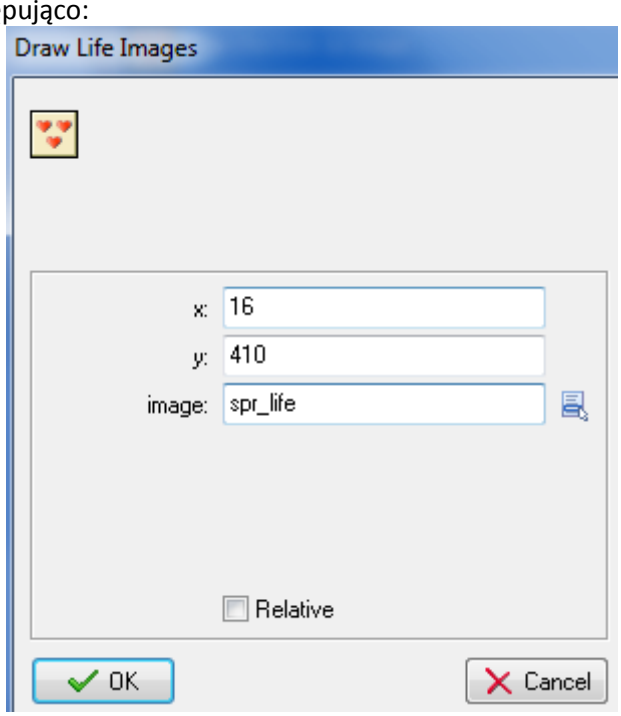


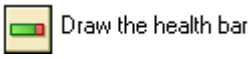
Wracamy teraz do zdarzenia rysowania. W nim, oprócz rysowania paska, wstawimy jeszcze parę akcji. By narysować wynik, najpierw ustawimy kolor czcionki na żółty. Wykorzystamy do tego akcję Set the color a w niej ustawiamy kolor na żółty. Następnie dodajemy trzecią akcję rysowania wyniku Draw the value of score i uzupełniamy ją następująco:



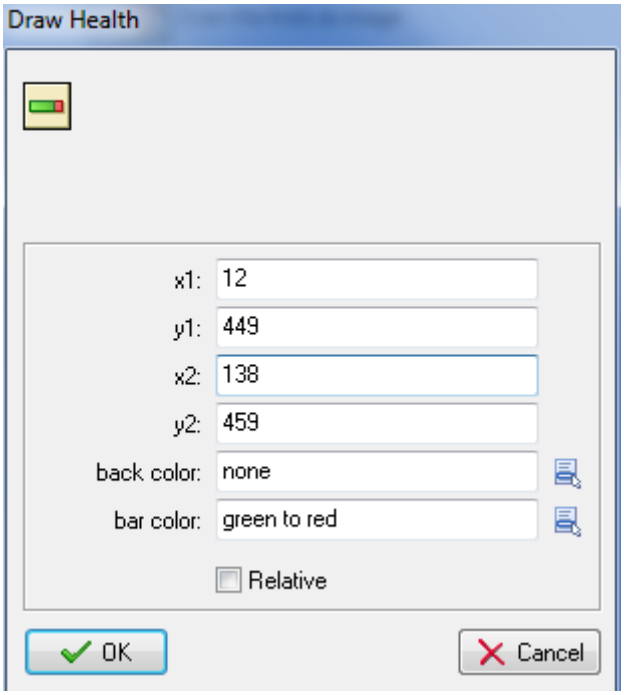
Ważne jest by dobrze ustawić współrzędne wyniku, tak aby pojawiał się w odpowiednim miejscu na pasku informacji (czasem trzeba poeksperymentować by dobrze trafić). Pole **Caption** (czyli napis, etykietka tekstowa) zostawiamy puste, bo na pasku już jest napis **Score**, czyli Wynik.

Do rysowania żyć użyjemy innego mechanizmu. Każde życie będzie reprezentowane przez mały rysunek samolotu. Użyjemy do tego obrazka naszego samolotu w pomniejszeniu **`life.png`**, na podstawie którego należy stworzyć sprajta np. **`spr_life`**. Teraz jesteśmy gotowi by dodać kolejną akcję rysowania do zdarzenia rysowania (Rysuj Obrazki Żyć) czyli Draw the lives as image. Akcję uzupełniamy następująco:



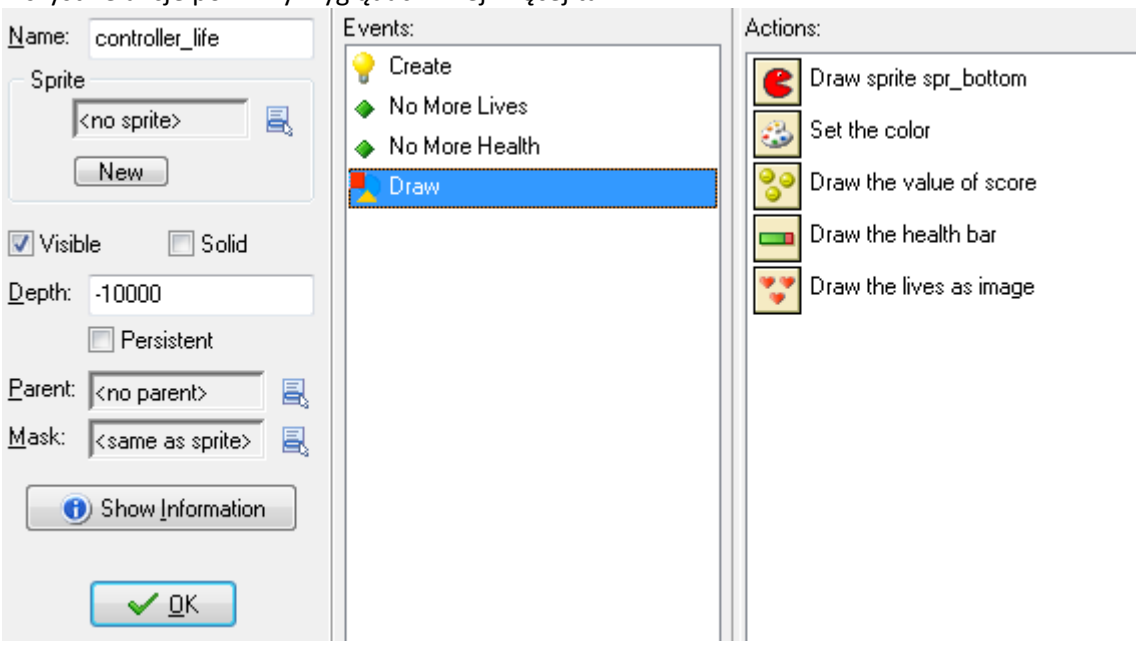
Również dla paska zdrowia mamy specjalną akcję: 

. Można w niej ustawić pozycję, wielkość I kolor. Ustawimy te parametry następująco:



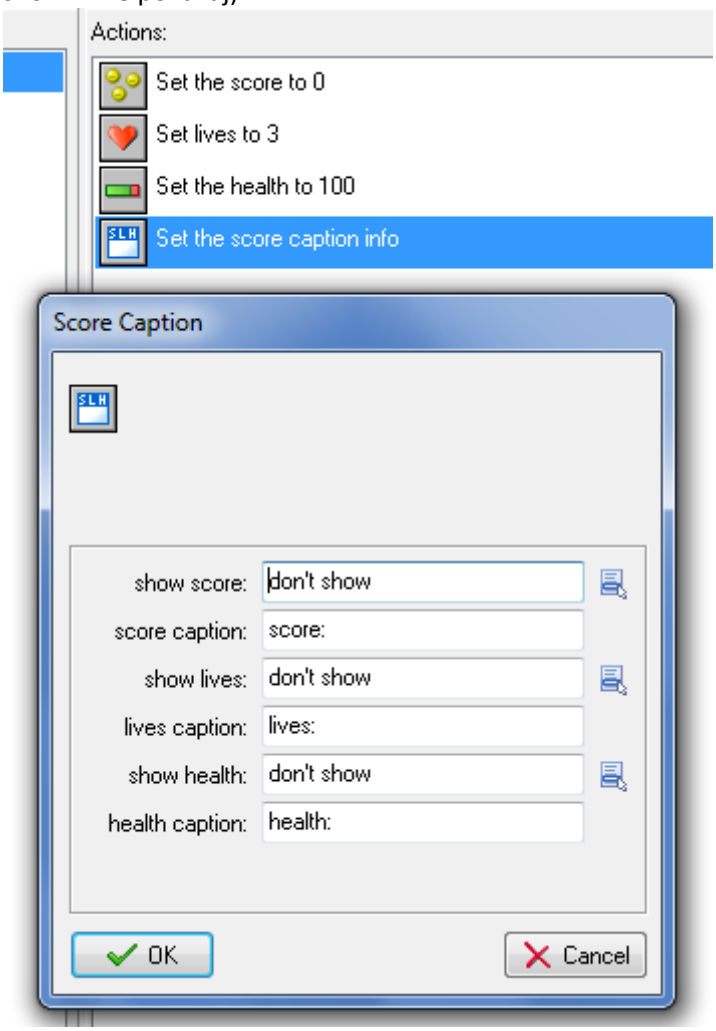
Liczby x1,y1 to współrzędne lewego górnego rogu paska, x2,y2 to prawy dolny róg. Musimy te liczby wpisać takie, aby pasek był dokładnie w czarnym polu na naszym panelu informacji. Kolejne dwa pola oznaczają kolor tła paska, oraz kolor paska (Green to red = zielony do czerwonego, w zależności od stanu zdrowia samolotu).

Wszystkie akcje powinny wyglądać mniej więcej tak:

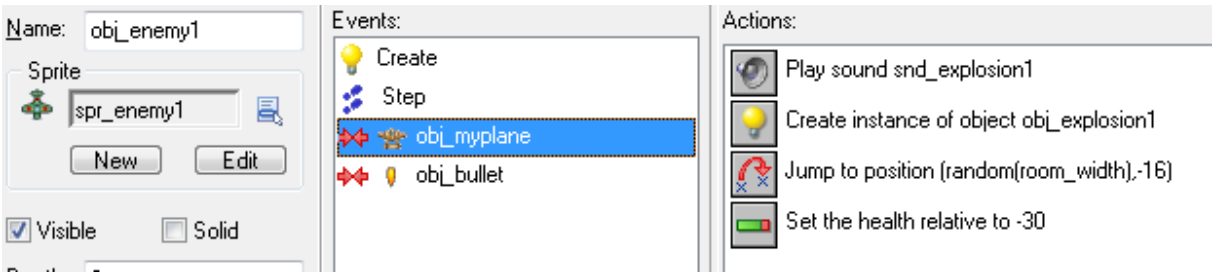


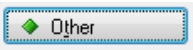
Ponieważ wynik i życia będą pokazywały się na naszym pasku informacji, to już nie muszą pokazywać się na górnym pasku okna. Usuniemy je za pomocą specjalnej akcji. Wejdź do zdarzenia kreacji w obiekcie **`controller_life`** i dodaj 4 akcję służącą do ustawiania informacji na górnym pasku

(znajdziesz ją w zakładce **Score**). Uzupełnij ją tak, aby wynik, zdrowie i życia nie były wyświetlane na górnym pasku (*don't show* = nie pokazuj):



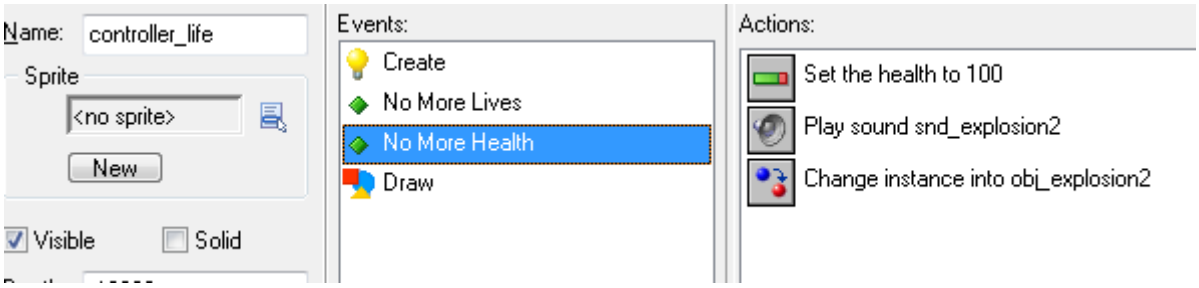
Nadal jednak nie dodaliśmy sprawdzania stanu zdrowia i liczby żyć. Musimy wiedzieć czy zdrowie lub życia spadły do 0. Na początek zmienimy trochę to co dzieje się podczas kolizji naszego samolotu z samolotem wroga. Nie powinno to niszczyć instancji naszego samolotu ani zamieniać go na instancję eksplozji (musimy dodać zdarzenie tworzące instancję eksplozji o współrzędnych 0,0 + Relative). Dodatkowo powinniśmy zmniejszyć zdrowie samolotu o 30 (tak by mniej więcej wytrzymał 3 uderzenia). Powinno to wyglądać tak:



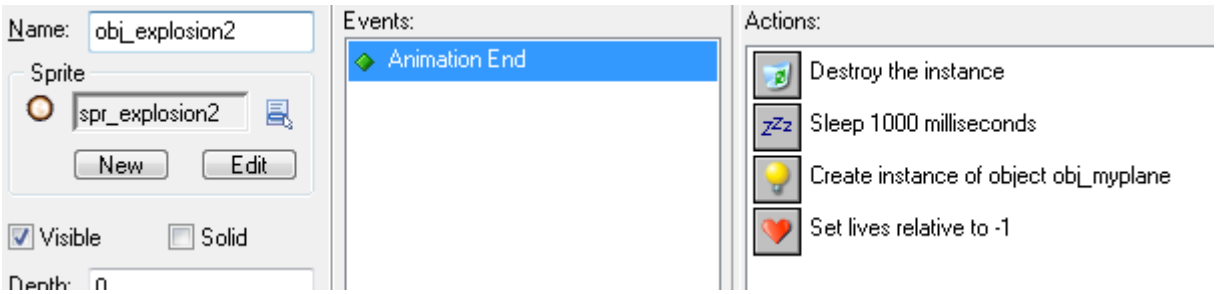
Teraz obiektowi **`controller_life`** każemy sprawdzać czy zdrowie spade poniżej 0. Służy do tego specjalne zdarzenie **No more health** (Brak zdrowia), kryjące się pod guzikiem 

. Akcje dla tego zdarzenia są trzy: resetujemy pasek z powrotem na 100, odpalamy dźwięk głośnej

eksplozji, pozbywamy się samolotu czyli zmieniamy instancję **`obj_myplane`** na **`obj_explosion2`** za pomocą akcji **Change instance**. Powinno to wyglądać tak:

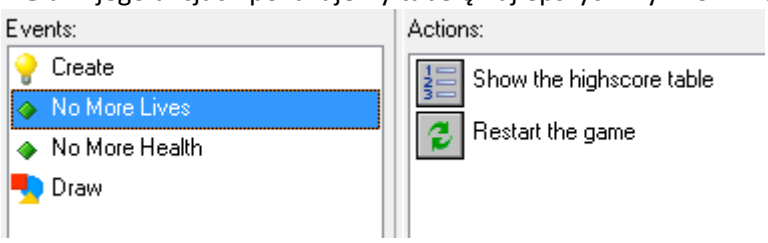


Trzeba teraz zagwarantować by eksplozja naszego samolotu nie kończyła gry tylko zmniejszała liczbę żyć. W zdarzeniu **Animation End** musimy wprowadzić parę zmian:



Pierwsza akcja usuwa eksplozję, druga czeka na zakończenie dźwięku eksplozji, trzecia tworzy nowy samolot w pozycji **Relative** i współrzędnych 0,0 i liczba żyć jest zmniejszana o 1 (również pamiętaj o opcji **Relative**).

Na koniec, musimy sprawdzić czy wyczerpały nam się życia. Na to również mamy specjalne zdarzenie **No more lives** (Brak żyć) kryjące się pod guzikiem **Other**. Dodajemy je do obiektu **`controller_life`** a w jego akcjach pokazujemy tabelę najlepszych wyników i restartujemy grę.



Teraz wystarczy wrzucić **`controller_life`** gdziekolwiek do pokoju i odpalić grę. Nasz pasek powinien ładnie przedstawiać informacje o grze:

Prześledźmy jeszcze raz nasze 3 parametry.

- **Zdrowie** na początku jest ustawiane na 100 (tworzenie obiektu kontrolującego życie). Przy każdym zderzeniu z wrogim samolotem zmniejszane jest o 30. Gdy zdrowie spada poniżej 0 (zdarzenie *No more health*) to nasz samolot wybucha i zmniejszana jest liczba żyć. Co krok (czyli co 1/30 sekundy) pasek zdrowia jest rysowany o takiej długości i kolorze jaki jest aktualny stan punktów zdrowia.

- **Życia** na początku są ustawiane na 3 (tworzenie **`controller_life`**). Wraz z wybuchem samolotu liczba żyć jest zmniejszana o 1. Gdy liczba żyć spada do 0 (zdarzenie *No more lives*), gra jest kończona. Zdarzenie **Draw**, sprawdza co 1/30 sekundy jaki jest aktualny stan żyć i rysuje na pasku odpowiednią ilość samolotów.
- **Wynik** jest ustawiony na początku na 0 automatycznie. W zdarzeniu wrogiego samolotu z naszym pociskiem jest zwiększany o 5. Zdarzenie **Draw**, co 1/30 sekundy sprawdza jaki jest aktualny stan punktów i umieszcza odpowiednią informację na pasku informacji.

Więcej wrogów

W tym rozdziale opiszemy jak dodać więcej wrogich samolotów różnych typów. Na razie mamy samoloty lecące pionowo dół. Dodamy trzy nowe typy samolotów:

- Samoloty lecące w dół, strzelające w dół,
- Samoloty lecące w dół, strzelające w kierunku naszego samolotu,
- Samoloty lecące w górę.

Ostatni typ samolotu jest trudny do zestrzelenia, więc pojawi się dopiero w późniejszych etapach gry.

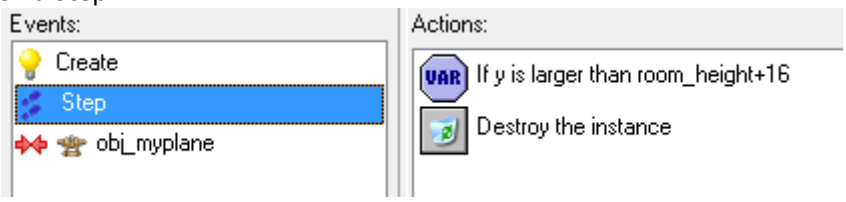
By stworzyć pierwszy typ samolotów (strzelające w dół) musimy najpierw dodać sprajta, na podstawie obrazka:

Jak widać, jest podobny do obecnych już samolotów, ale ma inny kolor.

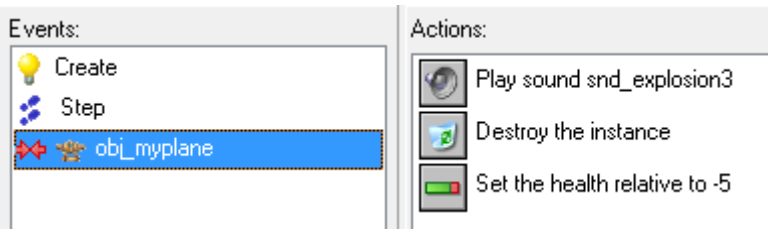
Po drugie, potrzebujemy obiektu. By nie robić całego obiektu od początku i ułatwić sobie pracę, zduplikujemy nasz wcześniejszy samolot (naciśnij prawym przyciskiem myszy w spisie obiektów na **`obj_enemy1`** i wybierz **Duplicate**). Taki sklonowany obiekt ma dokładnie takie same właściwości, zdarzenia i akcje jak pierwowzór. Jedynie nazwa się zmienia. Musisz teraz pozmieniać parę rzeczy w tym sklonowanym obiekcie:

- Nadaj mu nową nazwę np. **`obj_enemy2`**.
- Zmień mu sprajt na ten, który niedawno dodałeś.
- Ponieważ, to trudniejszy do zestrzelenia samolot. To zmień liczbę punktów na 10 za jego zestrzelenie (trzeba edytować jedną akcję w zdarzeniu kolizji z pociskiem).

Teraz musimy zająć się strzelaniem tego samolotu. Po pierwsze, dodaj sprajta na podstawie obrazka (enemybullet1.png). Stwórz na podstawie tego sprajta obiekt pocisku o nazwie np. **`obj_enemybullet1`**. Dodaj zdarzenie kreacji do tego obiektu, a w nim akcję pionowego ruchu w dół (Vertical speed) z prędkością 8. Czyli w skrócie, jak wrogi pocisk pojawi się na planszy, to będzie leciał w dół. Tak jak z pociskiem gracza, musimy postąpić i z pociskiem wrogiego samolotu tzn. jeśli wyleci poza pokój na dole, to zostanie usunięty. Do sprawdzania czy obiekt wyleciał poza pokój używamy zdarzenia Step:

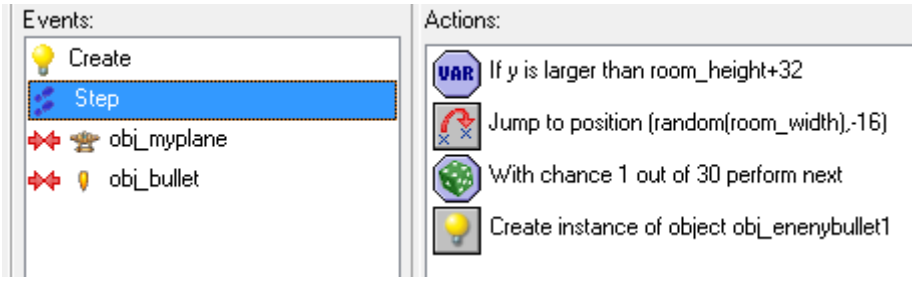


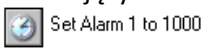
Gdy wrogi pocisk uderzy o samolot gracza, jego zdrowie zmniejsza się o 5, pocisk jest niczszczony, i odgrywamy dźwięk eksplozji.



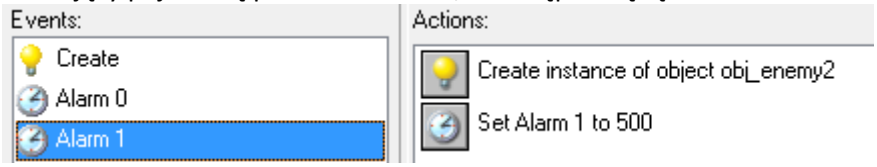
W ten sposób opisaliśmy zachowanie pocisku.

Teraz pora zaprogramować nasz wrogi samolot tak, by strzelał. Chcemy, żeby strzelał nieregularnie, losowo. Użyjemy do tego rzutu kostką (ang. *dice*). Komputer będzie losował, kiedy samolot strzeli, a kiedy nie. Zrobimy to w zdarzeniu kroku (step) w samolocie dając akcję rzutu kostką (dice) z parametrem 30 (czyli rzucamy 30-ścienną kostką). W drugiej akcji tworzymy instancję wrogiego pocisku. Oznacza to, że z szansą 1/30 wrogi samolot strzeli pociskiem co krok. Ponieważ kroków w sekundzie jest 30, więc samolot będzie strzelał średnio raz na sekundę:



Pozostało nam jeszcze dodawanie samolotu w planszy przez obiekt **`controller_enemy`**. Budzik nr 0 (Alarm 0) tworzy nam już samoloty niestrzelające. Stworzymy więc drugi budzik (Alarm 1), który będzie zajmował się dodawaniem samolotów strzelających. W zdarzeniu tworzenia (Create) dodajemy akcję ustawiającą alarm 1 na 1000 

. W zdarzeniu odpalenia budzika 1 (alarm 1) tworzymy instancje wrogiego samolotu i znowu ustawiamy alarm 1, tym razem na 500. Czyli samolot strzelający pojawi się po 30 sekundach, a następne będą dochodzić co około 15 s.

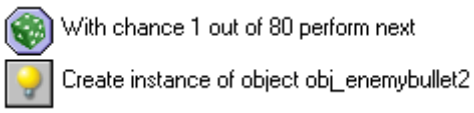


W tym momencie możesz zapisać i przetestować grę.

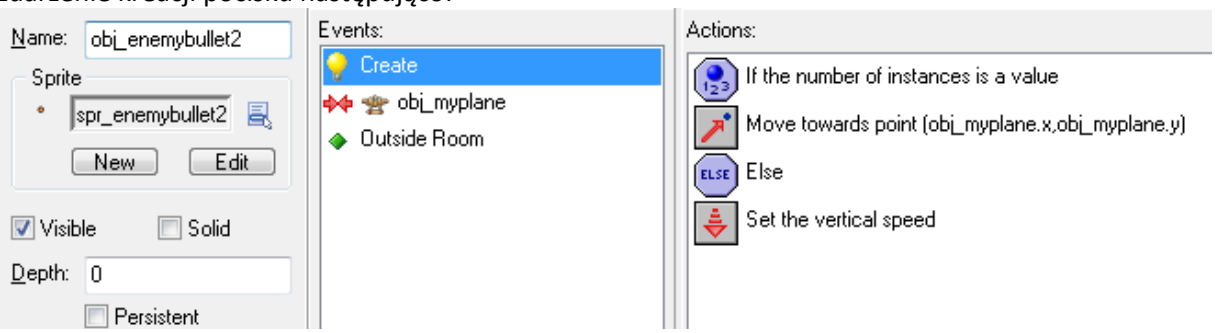
Pora na następny typ wrogiego samolotu, który zawsze strzela w stronę naszego samolotu. Będziemy potrzebować dla niego nowego sprajta i nowego sprajta dla pocisków, którymi będzie strzelać. Znajdziesz je w folderze *Ressources*.

Znowu duplikujemy samolot (wroga strzelającego) i jego kopię zamieniamy na naszego wroga nr 3 (znowu zmieniamy nazwę, sprajta oraz liczbę punktów na 20). Zduplikujemy również obiekt odpowiadający wrogiemu pociskowi dla samolotu strzelającego w dół, a następnie zmieniamy tej kopii nazwę, sprajta (na świeżo dodanego).

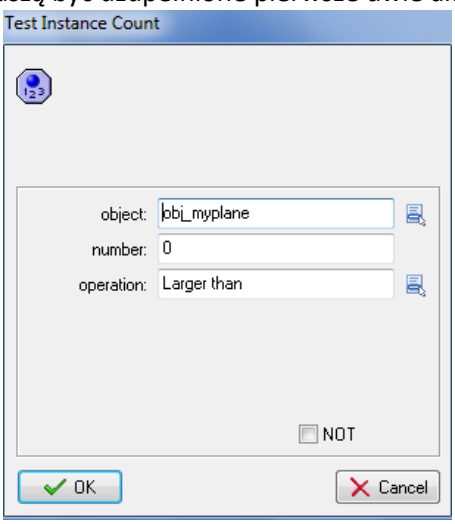
Mamy więc **`obj_enemy3`** i **`obj_enemybullet2`** (lub inaczej nazwane obiekty). Musimy teraz zająć się samym procesem strzelania. Po pierwsze, zmieniamy pociskowi rzut kostką z 30 na 80, bo chcemy, by pociski rzadziej zostały wystrzeliwane (są trudne do unikania). W akcji za kostką trzeba zmienić tworzenie instancji pocisku na pociski typu 3 (**`obj_enemybullet2`**).

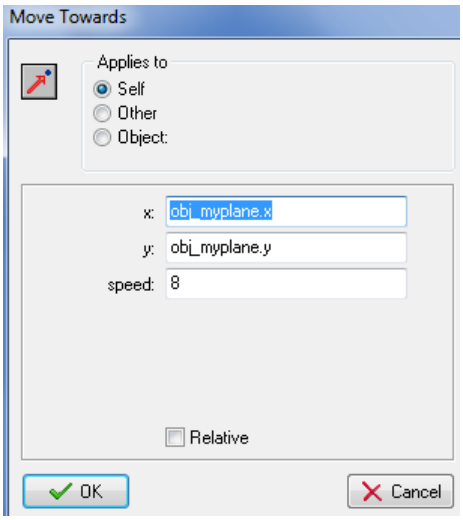


Nowy pocisk będzie działał następująco. W jego zdarzeniu kreacji, dodajemy akcję poruszania się w kierunku pozycji. Ale jaką pozycję podać? Musimy podać pozycję naszego samolotu, jego współrzędne x i y w pokoju. Łatwo to zrobić w GameMakerze, wystarczy w okienku wpisać x, poprzedzone nazwą obiektu i kropką czyli: **obj_myplane.x** oraz **obj_myplane.y**. A co by było gdyby w pokoju były dwa samoloty obj_myplane? Wtedy pocisk leciał by w kierunku pierwszego, który dodaliśmy. Gdy zaś nie będzie samolotów w pokoju, to wyskoczy nam błąd. To może być problem w przypadku gdy, samolot wrogi strzela a nasz samolot akurat wybuchł bo zderzył się z czymś i nie ma go tymczasowo na planszy. Jest akcja, która sprawdza ile jest instancji konkretnego obiektu w pokoju. Użyjemy jej by sprawdzić czy jest samolot w pokoju, a jeśli tak, to wrogie samoloty będą strzelały w jego kierunku. W przeciwnym wypadku, pocisk będzie leciał pionowo na dół. Musimy uzupełnić zdarzenie kreacji pocisku następująco:



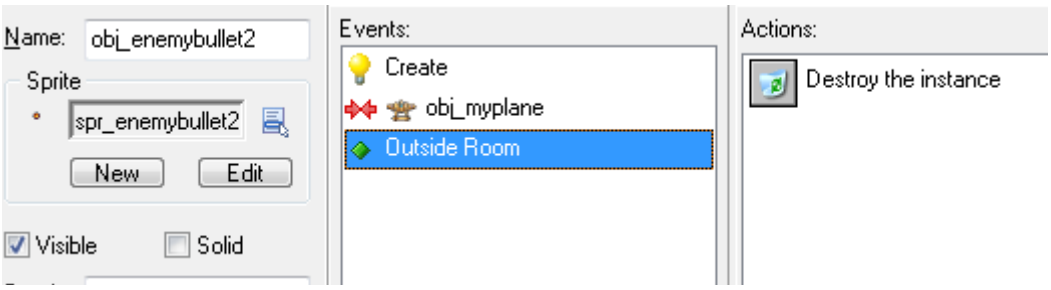
Oto jak muszą być uzupełnione pierwsze dwie akcje:





Akcja „Else" oznacza „W przeciwnym wypadku". Wszystkie 4 akcje tworzą coś w rodzaju zdania: „Jeśli liczba samolotów gracza jest większa niż zero (akcja 1) to strzel w kierunku jego samolotu (akcja 2), w przeciwnym wypadku (akcja 3) strzel pionowo na dół (akcja 4). Stawianie takich warunków jest bardzo częstym zabiegiem programistycznym.

Musimy jeszcze poprawić badanie czy pocisk jest wewnątrz pokoju. Ponieważ nasze nowe pociski latają we wszystkie strony, musimy sprawdzić czy nie przeleciał on przez lewą, prawą, dolną lub górną ścianę pokoju. Jest jednak zdarzenie, które sprawdza czy obiekt wyleciał poza pokój (przez dowolną ścianę). Jest to zdarzenie **Outside Room** (Na zewnątrz pokoju). W nim wystarczy wpisać akcję niszczącą instancję obiektu. Pocisk zatem będzie miał trzy akcje:



Ostatnia rzecz, to obsługa pojawiania się nowych samolotów przez obiekt **`controller_enemy`**. Dodajemy nowy budzik (alarm 2) ustawiając go w zdarzeniu tworzenia na 2000. Dodajemy zdarzenie odpalenia alarmu 2, w którym tworzymy nowy samolot i ponownie ustawiamy alarm 2, ale tym razem na 1000. Wszystko jest podobne jak dla samolotu wcześniejszego, zmieniony jest tylko numer alarmu i liczby kroków w alarmach.
Zapisz i przetestuj jak zachowuje się nowy typ samolotu.

Ostatni typ samolotu powinien pojawiać się na dole i lecieć do góry. Powinieneś poradzić sobie z nim, bo w zachowaniu jest bardzo podobny do pierwszego samolotu (który pojawiał się u góry i leciał na dół). Możesz go zduplikować i pozmieniać niektóre rzeczy. Nie zapomnij o dodaniu nowego sprajta dla tego samolotu:  . W obiekcie powinieneś oczywiście dodać kolejny alarm, ale i z tym powinieneś sobie poradzić.

Zapisz i przetestuj grę. Gra jest już grywalna i naprawdę może wciągnąć. Osiągnięcie wysokich wyników jest dość trudne. Jedyne co możemy zrobić, to trochę dopieścić grę by stała się jeszcze bardziej interesująca. Możesz dodać muzykę w tle (tak jak w poprzedniej grze) korzystając z pliku **`background.mid`**.