

Czytając ten artykuł, dowiesz się jak stworzyć przeciwnika w grze platformowej. Nie będzie to zbyt skomplikowane AI, lecz będzie posiadało wszystkie istotne i najpotrzebniejsze rzeczy - aby przybliżyć wam na jakiej zasadzie mniej więcej to działa. Zawarty tu też jest prosty systemik walki - który radzę wam rozbudować a nawet napisać własny. Należy zaznaczyć, że podczas pisania tego artykułu będę pracować na obrazkach o rozmiarach **48x48** i do niego są dostosowane obliczenia. Gdy zamierzasz użyć innych rozmiarów - musisz je odpowiednio pozmienić w całym kodzie. Na końcu artykułu znajdują się 2 przykłady oraz przyjemniejsza w czytaniu wersja PDF. :) *ps. wybaczcie za różne kolory składni (w czasie pisania artykułu nie pomyślałem i je zmieniłem w GM).*

Naszym pierwszym krokiem, do zaprogramowania „myślącego” wroga będzie stworzenie nowego obiektu - naszego przeciwnika. Nazwijmy go **obj_enemy**. W tym miejscu należy zaznaczyć, że istnieje pewien dobry zwyczaj. Gdy tworzymy nowe zasoby, przed ich właściwymi nazwami możemy wpisywać pewne niezmiennie przedrostki (inne dla każdego rodzaju zasobów). Na przykład **spr_** dla obrazków (sprites), **bac_** dla tła (backgrounds). Służy to temu, aby nie pogubić się w kodzie oraz dla klarowności. Wróćmy teraz do naszego obiektu.

Nasz wróg musi posiadać jakąś prędkość poruszania. Oprócz tego, musimy określić co robi w danym momencie - stoi w miejscu lub się porusza. Posłuży nam do tego zmienna **stan**. Zmienna **stan** przechowywać będzie zatem w naszym założeniu 2 wartości.

Najlepiej w tym celu stworzyć **stałe** :

- ❑ **s_move** - potworek się porusza (stała o wartości 1)
- ❑ **s_stand** - potworek stoi w miejscu (stała o wartości 0).

Nasz przeciwnik musi potrafić walczyć z naszym bohaterem. Co to by było za AI bez tego ważnego szczegółu. Dlatego właśnie zadeklarujemy specjalną zmienną **attack**, która będzie określać co wróg robi jeśli chodzi o walkę.

W naszym obiekcie dodajmy nowe wydarzenie (event) **create** i wpiszmy kod :

```
spd = 2; // szybkość poruszania
stan = s_move; // aktualny stan
attack = false; // 0 - brak, 1 - zauważenie, 2 - pogon za graczem, 3 - walka
dir = 0; // kierunek poruszania
sloth = 2; // współczynnik "lenistwa", przedział 0 - 10
```

Możemy sprawić, aby potwór chodził raz w jedną, raz w drugą stronę, rozglądał się, zatrzymywał, przeskakiwał przeszkody. Na razie ograniczymy się do dodania 2 zmiennych : **dir** oraz **sloth**. Zmienna **dir** odpowiada za to, w jakim kierunku porusza się nasz przeciwnik. Zważając, że robimy AI do platformówki, przyjmuje ona tylko 2 wartości - **0** (prawo) oraz **180** (lewo). Zmienna **sloth** odpowiada za to, jak często przeciwnik będzie się zatrzymywał. Im wyższa wartość zmiennej tym przeciwnik jest bardziej „leniwy”.

Zmienna **attack** przechowywać będzie natomiast 4 wartości. Gdy jest równa **false**, oznacza to, że potworek nie toczy w tej chwili walki. Gdy przyjmie wartość **1** - oznacza, że zauważył gracza. Gdy jest w trakcie pogoni - przyjmuje wartość **2**, i wreszcie jak jest już przy graczu i może go naparzać - wartość **3**. Więcej o tej zmiennej w ostatnim punkcie - gdzie szczegółowo zajmiemy się walką.

Wszystkie obiekty typu **solid** (czyli platformy, schody itp.) musimy przypisać do jednego parenta, którego nazywamy **BLOCKS**.

Zajmijmy się teraz zachowaniem potworka, gdy nie ma w pobliżu jego wroga (czyli Ciebie!).

Gdy nie ma w pobliżu gracza

Na początek należy zaznaczyć, że dość często (w różnych przypadkach) nasz potworek będzie zmuszany do zmiany kierunku ruchu (zakręt na krańcu platformy, naturalny obrót, przeszkody). Aby nie przepisywać ciągle tego samego kodu, stworzymy skrypt - dla czystej wygody i porządku w kodzie. Nazwijmy go **change_dir**.

```
if dir == 180 and position_meeting( x + 25, y + 49, BLOCKS )
{
    dir = 0;
}
else if dir == 0 and position_meeting( x - 25, y + 49, BLOCKS )
{
    dir = 180;
}
```

Skrypt uniemożliwi też naszemu potworkowi zmianę kierunku, gdy za jego plecami jest przepaść (zdarzyć to się może sporadycznie w niektórych przypadkach [dla dociekliwych: gdy potworek skończy za nami gonitwę na skraju platformy]). Zaplanujmy zachowanie naszego wroga. Chodzi w różnych kierunkach po platformie. Gdy dojdzie do jej końca - zawraca. W międzyczasie może zawrócić podczas ruchu gdzieś na środku platformy, stanąć na chwilę w miejscu, lub też się rozglądać. Co do rozglądania zasada jest prosta - potworek wylosował (o tym później) zmianę kierunku oraz zatrzymanie. Po kolejnym wylosowaniu tego samego, będzie to wyglądało jakby się rozglądał.

Na jakiej zasadzie nasz wróg będzie się poruszał? Będzie on sprawdzał, czy miejsce kilka pikseli przed nim jest **wolne od przeszkód**. Jeśli będzie, to znaczy, że mamy wolną przestrzeń przed nami. Jeśli pierwszy warunek okaże się prawdziwy, musimy dodatkowo stwierdzić, czy platforma się nie skończyła. Dopiero wtedy możemy zacząć się poruszać. Wklejmy taki kod do wydarzenia **step** (nie zapomnijmy wcześniej wkleić grawitacji - choćby z gmclanowego FAQ)

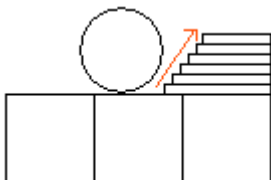
```
if stan == s_move // jesli sie porusza
{
    if dir == 0 // w prawo
    {
```

```

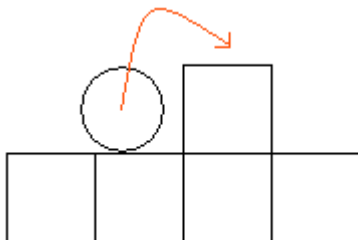
        if ! position_meeting( x + 24 + spd, y + 20, BLOCKS ) //czy miejsce
wolne od przeszkód
        {
            if ! place_free( x + spd + 48, y + 49 ) //czy koniec platformy
            {
                x += spd; //poruszanie
            }
        }
    }
else if dir == 180 // w lewo
{
    if ! position_meeting( x - 24 - spd, y + 20, BLOCKS )
    {
        if ! place_free( x - spd - 48, y + 49 )
        {
            x -= spd;
        }
    }
}
}
}

```

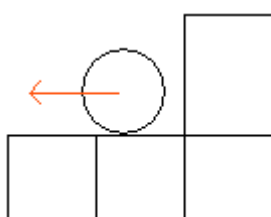
A co by się stało gdyby np. miejsce nie było wolne od przeszkód, lub platforma by się skończyła? Z naszym aktualnym kodem, nasz wróg by się zatrzymał co nie byłoby zbyt efektowne. Musimy zmodyfikować nasz kod. Gdy platforma się skończy, nasz wróg w prostym następstwie powinien zawrócić. Wykorzystamy do tego wcześniej przygotowany skrypt **change_dir**. A przeszkody? Założmy, że dla naszego wroga mogą to być schody. Aby je pokonać posłużymy się automatycznym przesunięciem do góry (wchodzenie po schodach).



Oprócz tego nasz potwór powinien umieć wskoczyć na wystarczająco niskie przeszkody



Jeśli jednak przeszkoda będzie zbyt wysoka (tzn. nie będą to schody) przeciwnik powinien zmienić kierunek.



Zajmiemy się każdym z tych zachowań po kolei.

Zawracanie na końcu platformy

Ogranicza się do dodania jednego wyrażenia **else**, w którym zamieścimy skrypt **change_dir**.

```
if stan == s_move // jeśli sie porusza
{
    if dir == 0 // w prawo
    {
        if ! position_meeting( x + 24 + spd, y + 20, BLOCKS )//czy miejsce
wolne od przeszkód
        {
            if ! place_free( x + spd + 48, y + 49 ) //czy koniec platformy
            {
                x += spd; //poruszanie
            }
            else if vspeed == 0 // jeśli koniec platformy
            {
                change_dir();// zmien kierunek
            }
        }
    }
    else if dir == 180 // w lewo
    {
        if ! position_meeting( x - 24 - spd, y + 20, BLOCKS )
        {
            if ! place_free( x - spd - 48, y + 49 )
            {
                x -= spd;
            }
            else if vspeed == 0
            {
                change_dir();
            }
        }
    }
}
```

Wchodzenie po schodach

Skrypt sprawdzający możliwość poruszania, sprawdza czy przestrzeń dokładnie **przed naszymi nogami** jest pusta. Teraz już możemy się domyśleć, dlaczego nie sprawdzamy przestrzeni umieszczonej np. tuż przed głową. Schody są po prostu niskie (u mnie **24 x 12** jeden stopień) i dlatego musimy szukać ich niżej. Gdy nasz potwór wykryje takowe schody przestanie funkcjonować. Musimy „powiedzieć” mu co zrobić gdy przestrzeń przed jego nogami **nie będzie** pusta, jak do tej pory.

```
if stan == s_move // jeśli sie porusza
{
    if dir == 0 // w prawo
    {
        if ! position_meeting( x + 24 + spd, y + 20, BLOCKS )//czy miejsce
wolne od przeszkód
```

```

    {
        if ! place_free( x + spd + 48, y + 49 ) //czy koniec platformy
        {
            x += spd; //poruszanie
        }
        else if vspeed == 0 // jesli koniec platformy
        {
            change_dir();// zmien kierunek
        }
    }
    else if ! position_meeting( x + 24 + spd, (y + 20)-13, BLOCKS ) //
schody
    {
        x += 6;
        y -= 13;
    }
}
else if dir == 180 // w lewo
{
    if ! position_meeting( x - 24 - spd, y + 20, BLOCKS )
    {
        if ! place_free( x - spd - 48, y + 49 )
        {
            x -= spd;
        }
        else if vspeed == 0
        {
            change_dir();
        }
    }
    else if ! position_meeting( x - 24 - spd, (y + 20)-13, BLOCKS )
    {
        x -= 6;
        y -= 13;
    }
}
}
}

```

Jeśli przestrzeń tuż przed naszymi nogami nie jest pusta (stopień schodów), skrypt sprawdza czy kilka pikseli wyżej (gdzie stopień się kończy) - już jest. Na podstawie tego wróg wykryje obecność schodów i zgrabnie po nich wejdzie.

Gdy ściana jest za wysoka

Reakcja wroga powinna być taka sama jak na końcu platformy - zawrócenie. Obecność ściany zostanie wykryta automatycznie, gdy nie zostanie ona rozpoznana jako schody (czyli gdy tuż nad nią nie ma wolnej przestrzeni). Wystarczy więc dopisać kolejne **else**.

```

if stan == s_move // jesli sie porusza
{
    if dir == 0 // w prawo
    {
        if ! position_meeting( x + 24 + spd, y + 20, BLOCKS )//czy miejsce
wolne od przeszkód
        {
            if ! place_free( x + spd + 48, y + 49 ) //czy koniec platformy
            {
                x += spd; //poruszanie
            }
        }
    }
}

```

```

        }
        else if vspeed == 0 // jesli koniec platformy
        {
            change_dir();// zmien kierunek
        }
    }
    else if ! position_meeting( x + 24 + spd, (y + 20)-13, BLOCKS ) //
schody
    {
        x += 6;
        y -= 13;
    }
    else // gdy natrafimy na sciane
    {
        change_dir();
    }
}
else if dir == 180 // w lewo
{
    if ! position_meeting( x - 24 - spd, y + 20, BLOCKS )
    {
        if ! place_free( x - spd - 48, y + 49 )
        {
            x -= spd;
        }
        else if vspeed == 0
        {
            change_dir();
        }
    }
    else if ! position_meeting( x - 24 - spd, (y + 20)-13, BLOCKS )
    {
        x -= 6;
        y -= 13;
    }
    else
    {
        change_dir();
    }
}
}
}

```

Wskakiwanie na niskie przeszkody

Jak już wielu z was może się domyśleć, „niskie przeszkody” to po prostu „wyższe schody” po których potworek normalnie nie byłby w stanie wejść. Zbudować więc możemy je nawet z obiektu „stopień schodów”. Nie są one tak wysokie jak ściana opisana wcześniej - więc potwór może je przeskoczyć. Warunek musimy umieścić w trochę innym miejscu. Dlaczego? Gdyż potworek powinien zacząć skok trochę wcześniej, a nie zaraz pod ścianą - aby wyglądało to naturalnie. Poza tym, potworek mógłby się zaklinować podczas skoku. Musimy go wpisać zaraz pod warunkiem, który sprawdza czy miejsce przed nogami jest puste. Teraz musimy sprawdzić przestrzeń nieco **dalej** (około 60 pikseli), gdyż skok musi być wykonany wcześniej (właśnie te 60 czy 50 pikseli przed przeszkodą).

```

if stan == s_move // jesli sie porusza
{

```

```

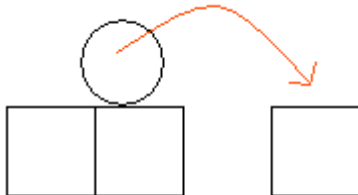
if dir == 0 // w prawo
{
    if ! position_meeting( x + 24 + spd, y + 20, BLOCKS )//czy miejsce
wolne od przeszkód
    {
        if position_meeting( x + 60, y - 12, BLOCKS )
        {
            if ! position_meeting( x + 60, y - 24, BLOCKS )
            {
                vspeed = -8; // skok
            }
        }
        if ! place_free( x + spd + 48, y + 49 ) //czy koniec platformy
        {
            x += spd; //poruszanie
        }
        else if vspeed == 0 // jesli koniec platformy
        {
            change_dir();// zmien kierunek
        }
    }
    else if ! position_meeting( x + 24 + spd, (y + 20)-13, BLOCKS ) //
schody
    {
        x += 6;
        y -= 13;
    }
    else // gdy natrafimy na sciane
    {
        change_dir();
    }
}
else if dir == 180 // w lewo
{
    if ! position_meeting( x - 24 - spd, y + 20, BLOCKS )
    {
        if position_meeting( x - 60, y - 12, BLOCKS )
        {
            if ! position_meeting( x - 60, y - 24, BLOCKS )
            {
                vspeed = -8; // skok
            }
        }
        if ! place_free( x - spd - 48, y + 49 )
        {
            x -= spd;
        }
        else if vspeed == 0
        {
            change_dir();
        }
    }
    else if ! position_meeting( x - 24 - spd, (y + 20)-13, BLOCKS )
    {
        x -= 6;
        y -= 13;
    }
    else
    {
        change_dir();
    }
}

```

```
}  
}
```

Skok przez przepaść

Dodajmy jeszcze jedną umiejętność naszemu złoczyńcy - przeskakiwanie przez „przepaść” gdy wystarczająco blisko jest inny klocek. Z naszym aktualnym kodem taka sytuacja byłaby niemożliwa :



Gdyż nasz potworek będzie zawracał zawsze, gdy skończy się platforma.

Aby kod działał poprawnie musimy stworzyć zmienną pomocniczą **jump**. Dopiszmy więc do create :

```
jump = false; // skok przez przepasc
```

i zmodyfikujmy kod w step :

```
if stan == s_move // jesli sie porusza  
{  
    if dir == 0 // w prawo  
    {  
        if ! position_meeting( x + 24 + spd, y + 20, BLOCKS ) //czy miejsce  
wolne od przeszkód  
        {  
            if position_meeting( x + 60, y - 12 , BLOCKS )  
            {  
                if ! position_meeting( x + 60, y - 24, BLOCKS )  
                {  
                    vspeed = -8; // skok  
                }  
            }  
            if ! place_free( x + spd + 48, y + 49 ) or jump //czy koniec  
platformy  
            {  
                x += spd; //poruszanie  
            }  
            else if vspeed == 0 // jesli koniec platformy  
            {  
                if position_meeting( x + 75 + spd, y + 27, BLOCKS )  
                {  
                    vspeed = -8; jump = true // skok przez przepasc  
                }  
                else  
                {  
                    change_dir(); // zmien kierunek  
                }  
            }  
        }  
    }  
    else if ! position_meeting( x + 24 + spd, (y + 20) - 13, BLOCKS ) //  
schody  
    {
```



```

        x += 6;
        y -= 13;
    }
    else // gdy natrafimy na sciane
    {
        change_dir();
    }
}
else if dir == 180 // w lewo
{
    if ! position_meeting( x - 24 - spd, y + 20, BLOCKS )
    {
        if position_meeting( x - 60, y - 12, BLOCKS )
        {
            if ! position_meeting( x - 60, y - 24, BLOCKS )
            {
                vspeed = -8; // skok
            }
        }
        if ! place_free( x - spd - 48, y + 49 ) or jump
        {
            x -= spd;
        }
        else if vspeed == 0
        {
            if position_meeting( x - 75 - spd, y + 27, BLOCKS )
            {
                vspeed = -8; jump = true;
            }
            else
            {
                change_dir();
            }
        }
    }
    else if ! position_meeting( x - 24 - spd, (y + 20)-13, BLOCKS )
    {
        x -= 6;
        y -= 13;
    }
    else
    {
        change_dir();
    }
}
}
}

```

oprócz tego musimy dodać kod, przy kolizji z **BLOCKS**.

```

if jump {
jump = false; }

```

Do tej poru nasz wróg poruszał się tylko gdy miał „grunt pod stopami” (by uniknąć spadku z platformy). Dlatego właśnie stworzyliśmy zmienną **jump** - aby był w stanie poruszać się także w powietrzu. Ustawia się ona na **true** - gdy zostanie wykryta platforma za przepaścią, a przy kolizji z podłożem powraca do false.

Należy dodać, że przeciwnik będzie wykonywał wszystkie opisane powyżej czynności niezależnie czy jest w trakcie walki (gonitwy za graczem), czy też nie.

Niby wszystko gotowe, ale nasz potworek nadal zachowuje się nieco sztucznie. Musimy wprowadzić do jego zachowania trochę losowości. Posłużą nam do tego m.in. **alarmy** (<http://www.gmclan.org/index.php?czytajart=51>) i funkcja **random**. Zajmiemy się teraz wspomnianym wcześniej rozglądaniem, zakręcaniem oraz stawaniem w miejscu, w różnych momentach - jest to chyba najprostsza część tego kursu. Co jakiś losowy czas wywoływany jest alarm, w którym losujemy czy wróg ma się zatrzymać czy nie (pamiętacie zmienną **sloth** ? - to od niej będzie zależała częstość zatrzymań), wylosujemy dodatkowo czy potwór ma zmienić kierunek. Napišmy więc prosty kod. Jedyną rzecz, którą musimy zrobić w „create” jest ustawienie alarmu.

```
alarm[ 0 ] = 1;
```

Cała reszta obliczeń zostanie wykonana w alarmie[0], gdzie wklejmy taki kod :

```
alarm[ 0 ] = choose( 30, 60 ); // losowy czas wywołania alarmu
if attack <> 0 or vspeed <> 0 exit;

var _s;
_s = round( random(10) );

if _s > sloth
{
    stan = s_move;
}
else
{
    stan = s_stand;
}

if _s < 2 and vspeed == 0
{
    change_dir(); // zmiana kierunku
}
```

Jak już wcześniej wspomniałem, zachowania te nie będą miały miejsca, gdy wróg jest w trakcie ataku - stąd druga linijka kodu. Potem losujemy czas ponownego wywołania alarmu. Następnie tworzymy zmienną tymczasową, która przybiera wartość od 0 do 10. Jeśli **_s** jest większa od naszego „współczynnika lenistwa” potwór się nie zatrzyma. Dlatego właśnie - im większa wartość **sloth**, tym potwór częściej będzie stawał w miejscu.

Teraz możemy już naszego potworka ustawić na platformie i cieszyć się jego samodzielnością.

Nie takiego moba jednak oczekujemy. Nasz stworek będzie ignorował naszego bohatera, a przecież ma być jego potencjalnym mordercą. Przed zmodyfikowaniem samego AI,

musimy napisać jakiś prosty system walki, który możecie potem rozbudować do swojej gry.

Prosty system walki

Jest to element dodatkowy artykułu. Dowiesz się z niego, jak napisać system walki (możesz napisać własny), dzięki któremu będziesz mógł powalczyć z dobrze ci już znanym „potworkiem”, aby wreszcie dojść do ostatniego punktu dotyczącego AI podczas walki. Nadszedł czas na stworzenie drugiego obiektu w naszym projekcie - nazwijmy go **obj_hero**. Nasz bohater musi posiadać punkty życia, mane (opcjonalnie), szybkość, kierunek poruszania, obrażenia oraz szybkość ataku. Wklejamy więc do create kod:

```
hp = 100;
hp_max = hp;
mp = 100;
damage = 20; // obrażenia
attack_speed = 40; // szybkość ataku
```

Każda jednostka uczestnicząca w walce musi posiadać te zmienne (musisz je więc wpisać zarówno w bohatera jak i w potworku). Myślę, że zmienne od hp do damage są dla ciebie oczywiste. Jak jednak uzyskać szybkość ataku? Jak zapewne dobrze ci wiadomo, polega to na tym, że pomiędzy pojedynczymi ciosami upłynie jakiś ustalony czas, zapisany w zmiennej **attack_speed**. Przy wszystkich zdarzeniach związanych z upływem czasu posługujemy się alarmami, które mam nadzieję już poznałeś chociażby czytając ten artykuł.

Bohater może uderzyć tylko po upływie danego czasu. Musimy zatem stworzyć zmienną, która nie pozwoli nam atakować bez przerwy. Nazwijmy ją przykładowo **can** (czyli **możliwość**, u nas: **możliwość ataku**). Będzie ona przybierała dwie wartości: **true** lub **false** (omówimy ją później). Dopiszmy więc do create potwora i bohatera:

```
can = true; // możliwość ataku
```

Stwórzmy dwie zmienne **fight_left** oraz **fight_right**, które będą przechowywały id wroga po lewej bądź prawej stronie (aby mieć dane z kim walczymy, komu odebrać hp).

```
fight_left = noone; // przeciwnik po lewej
fight_right = noone; // przeciwnik po prawej
```

Pamiętajmy, że nasz potworek również ich potrzebuje.

Kolejnym krokiem będzie stworzenie skryptu o nazwie **hit**. Będzie on odpowiedzialny za zadawanie obrażeń zarówno od strony bohatera, jak i potwora.

```
if dir == 0
{
    if fight_right <> noone
    {
        fight_right.hp -= damage;
```

```

        with( fight_right )
        {
            if hp <= 0
            {
                fight_left.fight_right = noone;
                instance_destroy();
            }
        }
    }
}
else
{
    if fight_left <> noone
    {
        fight_left.hp -= damage;
        with( fight_left )
        {
            if hp <= 0
            {
                fight_right.fight_left = noone;
                instance_destroy();
            }
        }
    }
}
}

```

Jak już mówiłem zmienne **fight_right** oraz **fight_left** przechowują id instancji (o tym, jak dochodzi do zapisu - później), dlatego skrypt może dotyczyć zarówno potwora jak i bohatera (gdy zmienne nazwane są u nich tak samo). Skrypt sprawdza czy po lewej (prawej) stronie jest potwór (bohater). Jeśli warunek jest spełniony - odejmuje mu hp i ewentualnie niszczy, gdy skończy mu się życie. Po zabiciu przeciwnika, musimy także zwolnić id zabitego ze zmiennej.

Teraz możemy wykorzystać zadeklarowaną wcześniej zmienną **can**. Dodajmy nowe wydarzenie w **obj_hero** - **key press <control>**, lub dowolny inny klawisz, którym mamy wprowadzać uderzenie i wklejamy kod :

```

if can
{
    hit(); //uderzenie
    can = false; // nie moze atakowac
    alarm[ 1 ] = attack_speed;
}

```

Oprócz tego, wklejmy w **alarm[1]** :

```

can = true;

```

Atakować możemy tylko wtedy, gdy **can = true**. Zaraz po ataku zmienna jest ustawiana na **false**, a dopiero po czasie ustalonym w **attack_speed** powraca do **true**, który umożliwia nam atak. Co do ataku wroga, jest on częścią ostatniego punktu, gdyż jest powiązany z AI.

Właśnie napisaliśmy prosty system walki, który wykorzystamy w ostatnim punkcie. Dowiemy się tam, jak naszego moba zmienić w diabła z baranka.

AI wroga - walka

Niech wróg nie goni bohatera od razu po zauważeniu - wprowadzimy krótki czas reakcji (możemy dodatkowo puścić wtedy jakiś dźwięk dla lepszego efektu). Dopiszmy do create:

```
reaction_time = 10;
```

Musimy ustalić również odległość po której wróg nas zauważy.

```
dist = 96;
```

Teraz musimy przystąpić do wypisania kilku warunków i akcji, które pokierują naszym wrogiem podczas walki. Poniższy kod umieścimy w **step**, ale **przed** kodem na poruszanie, a zaraz po grawitacji :

```
if not attack
{
    if distance_to_object( obj_hero ) <= dist and not jump
    and not collision_line( x, y, obj_hero.x, obj_hero.y, BLOCKS, 1, 0 )
    {
        attack = 1; // zauwazenie
        alarm[ 1 ] = reaction_time;
        stan = s_stand; // stoi w miejscu

        if obj_hero.x < x and dir != 180
        {
            dir = 180;
        }
        else if obj_hero.x > x and dir != 0
        {
            dir = 0;
        }
    }
}
```

Jeśli wróg nie atakuje - sprawdza czy bohater jest wystarczająco blisko. Potem musimy sprawdzić, czy bohatera i potwora nie dzieli żadna ściana. Następnie zmienna atak zmienia wartość na 1 (jest to stan „zauważenia” o którym wcześniej już wspomniałem) i minie krótka chwila, nim potworek zacznie nas gonić. Włączamy alarm, oraz zatrzymujemy wroga (zmieniając wartość zmiennej **stan**). Poza tym, obracamy wroga w kierunku gracza. Teraz wklejmy kod do alarm [1] :

```
if attack == 1 // jesli zauwazyl gracza
{
    attack = 2; // pogon za graczem
    stan = s_move; // odblokowanie poruszania
}
```

W powyższym alarmie, rozkazujemy potworkowi gonić gracza (zmienna **attack** ustawiona na 2) i wznawiamy ruch (zmienna **stan**).

Sam kod pogoni jest prosty - musimy dbać jedynie o kierunek wroga oraz odległość od gracza (jeśli jest poza zasięgiem - musimy anulować gonitwę). Zmodyfikujmy więc nasz ostatni kod z step :

```
if not attack
{
    if distance_to_object( obj_hero ) <= dist
    and not collision_line( x, y, obj_hero.x, obj_hero.y, BLOCKS, 1, 0 )
    {
        attack = 1; // zauwazenie
        alarm[ 1 ] = reaction_time;
        stan = s_stand; // stoi w miejscu

        if obj_hero.x < x and dir != 180
        {
            dir = 180;
        }
        else if obj_hero.x < x and dir != 0
        {
            dir = 0;
        }
    }
}
else if attack == 2 // gonitwa
{
    if distance_to_object( obj_hero ) <= dist // jesli jest w zasiegu
    and not collision_line( x, y, obj_hero.x, obj_hero.y, BLOCKS, 1, 0 )
    {
        if obj_hero.x < x
        {
            dir = 180;
        }
        else
        {
            dir = 0;
        }
    }
    else if not jump // jesli poza zasiegiem - anulujemy gonitwe
    {
        attack = false;
        stan = s_stand;
        change_dir();
    }
}
```

Całą resztę mamy załatwioną w kodzie poruszania, którym zajmowaliśmy się na początku artykułu (omijanie przeszkód).

Mamy już potworka goniącego bohatera. Teraz trzeba by zapisać, co ma robić, gdy już dojdzie do kolizji między nim a bohaterem. Stwórzmy więc w potworku event **collision obj_hero** i wpiszmy tam następujący kod :

```
if attack == 2 // jesli goni ( a raczej „dogonił” gracza )
{
```

```
    attack = 3; // rozpocznij walke
    stan = s_stand; // stoj w miejscu
}
```

Ustawiamy zmienną **attack** na 3 (walka w zwarciu), i zatrzymujemy wroga w miejscu.

Teraz przyszedł czas na wykorzystanie naszego wcześniej napisanego systemu walki. Skąd tak właściwie potworek, albo bohater ma wiedzieć, w kogo uderzyć ? Jak zapewne pamiętacie, służyć do tego będą dwie zmienne : **fight_left** oraz **fight_right**. Musimy więc zmienić nasz kod, aby id potworka jak i bohatera wpisały się do odpowiedniej zmiennej :

```
if attack == 2 // jesli goni gracza
{
    attack = 3; // rozpocznij walke
    stan = s_stand; // stoj w miejscu
    alarm[ 2 ] = attack_speed; // atak

    if x < other.x
    {
        fight_right = obj_hero; // przeciwnik potworka
        if other.fight_left = noone // jeśli nie jest zajęte
            other.fight_left = id; // przeciwnik gracza
    }
    else
    {
        fight_left = obj_hero;
        if other.fight_left = noone // jeśli nie jest zajęte
            other.fight_right = id;
    }
}
```

Powyższe akcje, zostaną wykonane, tylko **zaraz po pogoni** – to znaczy jednorazowo, gdyż potem zmienna **attack** ustawia się na **3** i skrypt już się nie powtórzy. Musimy jednak wziąć pod uwagę, że bohater może się bić po jednej stronie z więcej niż jednym wrogiem naraz. Wtedy, gdy zabijemy jednego wroga, zmienna **fight_left** bądź **fight_right** powinna automatycznie przeskoczyć na następnego moba. Zamieńmy poprzedni kod na taki :

```
if attack == 2 // jesli goni gracza
{
    attack = 3; // rozpocznij walke
    stan = s_stand; // stoj w miejscu
    alarm[ 2 ] = attack_speed; // atak

    if x < other.x
    {
        fight_right = obj_hero; // przeciwnik potworka
        if other.fight_left = noone // jeśli nie jest zajęte
            other.fight_left = id; // przeciwnik gracza
    }
    else
    {
        fight_left = obj_hero;
        if other.fight_left = noone // jeśli nie jest zajęte
            other.fight_right = id;
    }
}
```

```

}
else if attack == 3 // w trakcie walki
{
    if x < other.x and other.fight_left == noone
    {
        other.fight_left = id; // przeciwnik gracza
    }
    else if x > other.x and other.fight_right == noone
    {
        other.fight_right = id; // przeciwnik gracza
    }
}
}

```

Teraz przyszedł czas, na zadawanie obrażeń. Posłużymy nam do tego alarm, wywołany w powyższym kodzie. Wpiszmy w ten alarm następujący kod :

```

if attack == 3
{
    hit(); // atak
    alarm[ 2 ] = attack_speed;
}

```

Po zadaniu obrażeń, ponawiamy alarm.

Gdy potwór jest zbyt wymagający, powinniśmy mieć możliwość ucieczki. Wtedy mob powinien oczywiście znowu nas gonić (zmienna **attack** ustawiona ponownie na 2) ponadto, należy zwolnić id bohatera i wroga ze zmiennych **fight_left** oraz **fight_right**. Aby umożliwić tę niemal oczywistą sposobność, musimy powrócić do naszego skryptu walki w **step** i zamienić go na :

```

if not attack
{
    if distance_to_object( obj_hero ) <= dist
    and not collision_line( x, y, obj_hero.x, obj_hero.y, BLOCKS, 1, 0 )
    {
        attack = 1; // zauwazenie
        alarm[ 1 ] = reaction_time;
        stan = s_stand; // stoi w miejscu

        if obj_hero.x < x
        {
            dir = 180;
        }
        else
        {
            dir = 0;
        }
    }
}
else if attack == 2
{
    if distance_to_object( obj_hero ) <= dist // jesli jest w zasiegu
    and not collision_line( x, y, obj_hero.x, obj_hero.y, BLOCKS, 1, 0 )
    {
        if obj_hero.x < x
        {
            dir = 180;
        }
    }
}

```



```

    }
    else
    {
        dir = 0;
    }
}
else // jesli poza zasiegiem - anulujemy gonitwe
{
    attack = false;
    stan = s_stand;
    change_dir();
}
}
else if attack == 3 // jesli jest w trakcie walki
{
    if distance_to_object( obj_hero ) > 8 // jesli gracz ucieka
    {
        fight_left = noone; fight_right = noone;
        obj_hero.fight_left = noone; obj_hero.fight_right = noone;

        attack = 2; // pogon za graczem
        stan = s_move;
    }
}
}

```

Dajmy też taką możliwość potworkowi, gdy straci np. 75 % swojego życia. Aby tego dokonać, musimy stworzyć jeszcze jedną zmienną. Możemy ją nazwać np. **escape**. Dlaczego ją tworzymy? Ponieważ z naszym aktualnym kodem potwór nie byłby w stanie „uciec” od gracza - ponieważ jest cały czas nastawiony na gonitwę. Jeśli zmienimy mu kierunek, on i tak w mgnieniu oka zwróci się w stronę gracza. Zmienna **escape** zablokuje mu taką możliwość, a raczej „wkradnie” się w obliczenia ustawiając właściwy kierunek. Po oddaleniu się wystarczająco daleko potworek powróci do stanu wyjściowego (attack = 0, stan = s_stand). **Escape** będzie przyjmować dwie wartości :

- ❑ **0** - Potwór nie ucieka
- ❑ **1** - A tutaj owszem

Zacznijmy od zadeklarowania tej zmiennej w potworku :

```
escape = 0;
```

Założmy, że nie każdy potworek może uciekać (tzw. Kamikadze :D). Zadeklarujmy więc zmienną, która będzie o tym informować.

```
can_escape = true; // czy może uciekać ( można ustawić od razu „false” )
```

Jak zapewne się domyślasz zmienna **escape** nie może stale być ustawiona na **1** - ponieważ potworek będzie od tej pory uciekał za każdym razem gdy oberwie. Po ustawieniu zmiennej na **1**, należy również ustawić zmienną **can_escape** na **false** (jeśli początkowo była zadeklarowana na true) - aby ucieczka była jednorazowa.

Teraz musimy zmodyfikować już wcześniej napisany skrypt **hit()**. Na początku sprawdzimy, czy **otrzymującym cios** jest potworek (ponieważ skrypt dotyczy również bohatera) oraz czy potwór ten ma możliwość ucieczki (`can_escape = true`). Jeśli tak - obliczamy, czy jego poziom życia spadł poniżej np. 25 procent. Teraz musimy zadbać, aby zmienna **escape** ustawiła się na **1** a **can_escape** na **0**. Zmienną **attack** ustawiamy na 2, gdyż będzie to „odwrócona gonitwa”, ruszamy potwora z miejsca (gdyż do tej pory stał przy bohaterze), oraz zwalniamy lewą lub prawą stronę bohatera oraz własną.

```
if dir == 0
{
    if fight_right <> noone
    {
        fight_right.hp -= damage;
        with( fight_right )
        {
            if hp <= 0
            {
                fight_left.fight_right = noone;
                instance_destroy();
            }
            else if object_index == obj_enemy
            {
                if can_escape and (hp*100)/hp_max <= 25
                {
                    escape = 1; // moze uciekac
                    can_escape = false;
                    attack = 2; // "gonitwa" - tak naprawde ucieczka
                    stan = s_move;
                    fight_left.fight_right = noone;
                    fight_left = noone;
                }
            }
        }
    }
}
else
{
    if fight_left <> noone
    {
        fight_left.hp -= damage;
        with( fight_left )
        {
            if hp <= 0
            {
                fight_right.fight_left = noone;
                instance_destroy();
            }
            else if object_index == obj_enemy
            {
                if can_escape and (hp*100)/hp_max <= 25
                {
                    escape = 1; // moze uciekac
                    can_escape = false;
                    attack = 2; // "gonitwa" - tak naprawde ucieczka
                    stan = s_move;
                    fight_right.fight_left = noone;
                    fight_right = noone;
                }
            }
        }
    }
}
```

```

    }
  }
}

```

Jak pewnie pamiętacie, potworek podczas gonitwy musiał poruszać się zawsze w kierunku bohatera. Teraz musi być zupełnie na odwrót. Po zakończeniu ucieczki, zmienna **escape** znów przyjmie wartość 0, jednak **can_escape** nie umożliwi powrócenia jej do wartości 1 (warunek jest sprawdzany w skrypcie **hit**). Musimy ponownie zmienić nasz kod w stepsie (dotyczący AI walki) - tym razem podam jedynie fragment dotyczący gonitwy :

```

else if attack == 2
{
    if distance_to_object( obj_hero ) <= dist // jeśli jest w zasięgu
    and not collision_line( x, y, obj_hero.x, obj_hero.y, BLOCKS, 1, 0 )
    {
        if obj_hero.x < x and dir != 180 - ( 180*escape )
        {
            dir = 180 - ( 180*escape );
        }
        else if obj_hero.x > x and dir != 0 + ( 180*escape )
        {
            dir = 0 + ( 180*escape );
        }
    }
    else if not jump // jeśli poza zasięgiem - anulujemy gonitwe
    {
        attack = false;
        stan = s_stand;
        change_dir();
        escape = 0;
    }
}

```

Przyjrzyjmy się, jakie zaszły różnice. Jeśli **escape** będzie równe 0, to potworek zawsze będzie podążał w kierunku bohatera. Jednak gdy **escape** otrzyma wartość 1 - wróg poruszać się będzie dokładnie na odwrót. Możecie podstawić sobie te wartości, aby się przekonać. Dalej po wystarczającym oddaleniu zerujemy **escape**. Wróg nie będzie miał już ponownej możliwości ucieczki i zapewne czeka go śmierć z naszej ręki. xD Niech wróg ucieka dopóki nie stanie mu na drodze jakaś przeszkoda (wymuszenie zmiany kierunku). Zatrzymać więc proces ucieczki możemy w skrypcie **change_dir()**. Wystarczy na samym końcu dopisać

```

escape = false;

```

Do tej pory, gdy doszło do kolizji między bohaterem a wrogiem - wróg musiał się zatrzymać. Teraz jednak musi otrzymać możliwość „oderwania” się od bohatera. Dopiszmy więc na samym początku skryptu :

```

if escape exit;

```

Zakończenie

W taki oto sposób stworzyliśmy moba do platformówki. Mam nadzieję że przeczytałeś choć część, tego przydługawego artykułu, a nie zjechałeś od razu na dół. :D Jak widzisz stworzenie AI wcale nie jest takie trudne (długość artu nie ma znaczenia), więc myślę nawet ci mniej doświadczeni powinni go zrozumieć. Polecam przetestowanie przykładu dołączonego do artykułu. Znajduje się tam jeszcze kilka skryptów nie powiązanych bezpośrednio z artykułem (takie pierdołki jak ruch gracza), aby można było go bez problemów przetestować.