Kolizje w grach 2D (sheadovas/poradniki/goto/kolizje/kolizje-w-grach-2d/)

Cze 07, 2015 / Kolizje (sheadovas/category/poradniki/goto/kolizje/)

Zestawienie najpopularniejszych rodzajów kolizji w grach (2D).

Mimo, że kilka kilka (chyba 2) poradników o kolizjach w grach 2D to są one nieco rozrzucone (SAT (sheadovas/piszemy-gre-w-sfmlu-lekcja-4/), bounding-boxy (sheadovas/pozycja-rotacja-skalowanie-przeksztalcanie-obiektow/)), dlatego postanowiłem zebrać razem w 1 poradniku najbardziej popularne sposoby wykrywania kolizji.

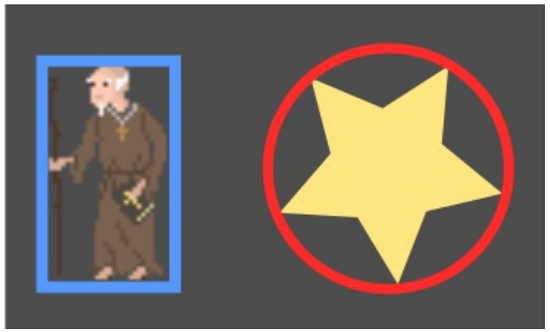
Będę tutaj używał opisów typowo matematycznych przy użyciu języka c++, więc nie jest wymagana znajomość żadnych bibliotek do pisania gier. Do zrozumienia tego co tu się dzieje wystarczy znajomość programowania (w dowolnym języku) na poziomie podstawowym oraz wiedza matematyczna na poziomie gimnazjum/liceum.

Czym są kolizje?

Kolizja w grach to nic innego jak wykrycie, czy 2 obiekty przedstawione przy użyciu figur geometrycznych mają części wspólne, inaczej mówiąc: czy się przecinają. Wykrywanie kolizji służy do m.in. zapobiegania przechodzenia przez siebie dwóch obiektów fizycznych (np. kolizje pomiędzy graczem, a ścianami), wykrycia kliknięcia myszą na obiekcie etc.

Do wykrywania kolizji posługujemy się uproszczonymi wersjami obiektów w zależności od potrzeb, im większe uproszczenie obiektu tym szybsze wykrycie kolizji, jest to szczególnie ważne w momencie gdy na scenie mamy kilka set lub więcej obiektów fizycznych. Jednak im większe uproszczenie tym mniej dokładnie wykrycie kolizji (ilustruje to obrazek poniżej).

W grach dwuwymiarowych najczęstsze uproszczenia obiektów to kolizje pomiędzy: punktami, czworokątami i okręgami i nimi się właśnie zajmiemy, pozostałymi zajmiemy się w ramach innych artykułów (np. wielokątami).



(https://i0.wp.com/www.shead.ayz.pl/wp-content/uploads/2015/06/typy-kolizji.png)
Reprezentacje kolizji w grze

Notacja

W dalszej części poradnika będę korzystał z klas takich jak *Vector2*, *Box*, *Circle* i są one zdefiniowane następująco:

```
Definicje prymitywów                                                    C++
```

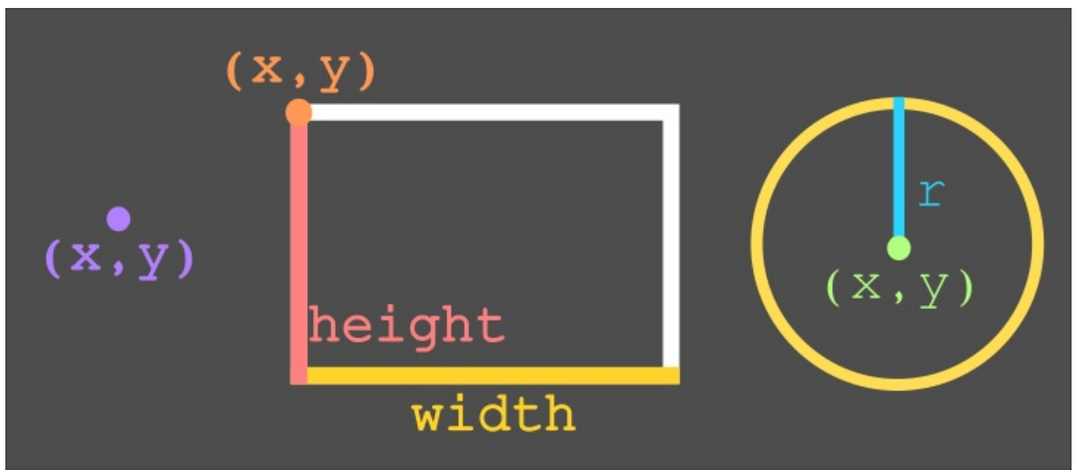
```cpp
class Vector2
{
public:
    int x;
    int y;
};


class Box
{
public:
    Vector2 position;                   // pozycja (x,y) krawędzi prostokąta/kwadratu

    unsigned int width;                 // szerokość
    unsigned int height;                    // wysokość
};


class Circle
{
public:
    Vector2 center;                     // środek okręgu
    unsigned int r;                     // promień okręgu
};
```

Jak widzimy są to najprostsze z możliwych implementacji tych prymitywów i w większości przypadków w zupełności nam wystarczą.



(https://i1.wp.com/www.shead.ayz.pl/wp-content/uploads/2015/06/obiekty.png)

Implementacja

We wstępie tego paragrafu chciałbym zaznaczyć, że nie znajdziecie tutaj implementacji SAT, a to dlatego że mówimy o prostych algorytmach do wykrywania kolizji, a on już takim prostym nie jest.

Punkt-Punkt

Najprostszy z możliwych wypadków, jak wiemy kolizja pomiędzy dwoma obiekatmi zachodzi wtedy gdy istnieją punkty wspólne (punkty przecięcia) pomiędzy tymi obiektami. W przypadku punktów, kolizja zachodzi wtedy gdy punkty są sobie równe.

punkt-punkt C++

```cpp
bool isCollsion(Vector2 p1, Vector2 p2)
{
    if (p1.x == p2.x && p1.y == p2.y)
        return true;
    else
        return false;
}
```

Punkt-Okrąg

Jak wiemy z matematyki, to punkt jest wewnątrz okręgu wtedy i tylko wtedy gdy odległość pomiędzy nim, a środkiem okręgu jest mniejsza bądź równa jego promieniowi.

Punkt-Okrąg C++

```cpp
bool isCollision(Vector2 p, Circle circle)
{
    // liczymy odległość pomiędzy dwoma punktami
    float d = sqrt(pow(p.x - circle.center.x, 2) + sqrt(pow(p.y - circle.center.y, 2)));

    if (d <= circle.r)
        return true;
    else
        return false;
}
```

Punkt-Prostokąt (Box)

Tutaj sprawa robi się nieco ciekawsza, ponieważ musimy sobie założyć, że boki naszego czworokąta są równoległe odpowiednio do osi OX i OY, ponieważ opisany poniżej sposób nie działa w przypadku obiektów na których zadziałaliśmy rotacją. Zatem, kiedy punkt jest wewnątrz prostokąta? Chyba zgodzisz się ze mną, że wtedy gdy jego pozycja x jest większa lub równa od lewej pionowej krawędzi oraz mniejsza lub równa pozycji jego prawej krawędzi, analogicznie powtarzamy nasze rozumowanie dla pozycji y.

Punkt-Box C++

```cpp
// Punkt - Box
bool isCollision(Vector2 p, Box box)
{
    if (p.x >= box.position.x && p.x <= box.position.x + box.width &&
        p.y >= box.position.y && p.y <= box.position.y + box.height)
        return true;

    else
        return false;
}
```

Okrąg-okrąg

Wkraczamy na tereny ciekawszych sytuacji, ponieważ o ile sytuacje z punktem były zazwyczaj dość intuicyjne i trywialne, tak tutaj warto jest sięgnąć po wiedzę matematyczną, którą zdobywa się (o ile dobrze pamiętam) w liceum. Skorzystamy tutaj z twierdzenia o styczności okręgów, czyli przypomnijmy sobie kryterium na styczność okręgów:

Niech: (O,r), (P,R) 2 okręgu o środkach w dowolnych punktach O i P oraz dowolnych promieniach: r,R. Wtedy dla dowolnych okręgów O i P prawdziwe są zależności:

- *okręgi są styczne zewnętrznie (mają 1 punkt wspólny) <=> |OP| = r + R*
- *okręgi przecinają się (mają 2 punkty wspólne) <=> |r-R| < |OP| < r + R*
- *okręgi są styczne wewnętrznie (1 punkt wspólny) <=> |OP| = |r-R|*

- *okręgi są rozłączne wewnętrznie (mniejszy okrąg jest wewnątrz większego) <=> |OP| < |r-R|*

Mamy tu potencjalnie sporo liczenia, wszystkie sytuacje powyżej przedstawiają wypadki gdy okręgu posiadają chociaż 1 punkt wspólny, czyli moment gdy występuje kolizja.

Jednak istnieje trik, który pozwala ułatwić wiele rzeczy i zminimalizować ilość obliczeń: jeżeli rozpatrzymy sytuację gdy okręgi NIE mają części wspólnych, tzn. sytuację gdy nie występuje kolizja i zanegujemy wartość tego wyrażenia to otrzymamy pożądany wynik.

Wiemy, że okręgi NIE mają części wspólnych <=> |OP| > r + R.

Okrąg-Okrąg C++

```cpp
bool isCollision(Circle o, Circle p)
{
    // odległość pomiędzy środkami okręgów
    float d = sqrt(pow(o.center.x - p.center.x, 2) + pow(o.center.y - p.center.y, 2));

    if ( !(d > o.r + p.r) )
        return true;
    else
        return false;
}
```

Z negacji korzysta się dość często, ponieważ jak widzimy czasami jest łatwiej sprawdzić kiedy coś nie zachodzi i później zanegować wynik, niż szukać go w tradycyjny sposób.

Prostokąt-Prostokąt (Box-Box)

Kolejna dość ciekawa sytuacja pod kątem matematycznym. Tak jak poprzednio musimy założyć, że ściany czworokąta są równoległe do osi OX i OY.

Tutaj musielibyśmy sprawdzać, czy kolejne wierzchołki prostokątów się zawierają w drugim prostokącie, co oczywiście potrafimy już sprawdzić jednak byłoby to pracochłonne i jak poprzednio skorzystamy z negacji i sprawdzimy kiedy 2 prostokąty są rozłączne zewnętrznie (nie mają części wspólnych).

Box-Box C++

```cpp
bool isCollision(Box b1, Box b2)
{
    Vector2 p1 = b1.position;
    Vector2 p2 = b2.position;

    if (p1.x > p2.x + b2.width || p1.x + b1.width < p2.x ||
        p1.y > p2.y + b2.height || p1.y + b1.height < p2.y)
        return false;

    else
        return true;
}
```

Zakończenie

To wszystko jeżeli chodzi o najczęściej używane i najczęściej spotykane sytuacje, jeżeli jesteś ciekawy jak wykryć kolizję pomiędzy okręgiem, a czworokątem lub pomiędzy dowolnymi wielokątami to zapraszam do kolejnego artykułu o kolizjach 2D.

Dzięki za przeczytanie i jak zawsze:

Code ON!