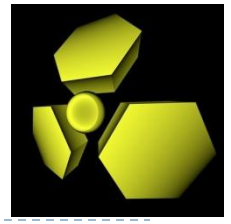
C++ w programowaniu gier

Adam Sawicki - www.asawicki.info - 30 października 2010

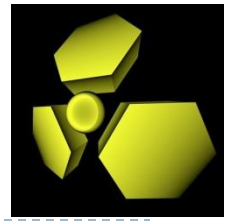
Dlaczego C++?



- W jakim języku pisze się gry?

- Poważne tytuły na PC i konsole - tylko C++!
- Inne platformy – różnie
  - iPhone, iPad – Objective C (oraz C++)
  - Android – Java (oraz C++)
  - Web – ActionScript (Flash), JavaScript, …
- Amatorskie projekty – dowolnie
  - XNA, Java, Python, cokolwiek…

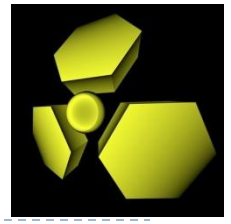
Dlaczego C++?



C++ to nie najważniejszy język na świecie.
TIOBE Programming Community Index for October 2010
[TIOBE]

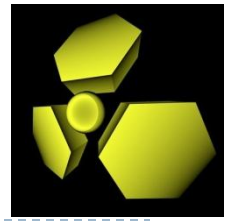
Position Oct 2010	Position Oct 2009	Delta in Position	Programming Language	Ratings Oct 2010	Delta Oct 2009	Status
1	1	=	Java	18.166%	-0.48%	A
2	2	=	C	17.177%	+0.33%	A
3	4	↑	C++	9.802%	-0.08%	A
4	3	↓	PHP	8.323%	-2.03%	A
5	5	=	(Visual) Basic	5.650%	-3.04%	A
6	6	=	C#	4.963%	+0.55%	A
7	7	=	Python	4.860%	+0.96%	A
8	12	↑↑↑↑	Objective-C	3.706%	+2.54%	A
9	8	↓	Perl	2.310%	-1.45%	A
10	10	=	Ruby	1.941%	-0.51%	A

Dlaczego C++?



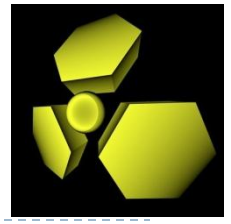
- C++ nie jest idealny [C++FQA]
  - Jest trudny i obszerny
    - Np. wskaźniki, szablony, przeciążenia
  - Wiele rzeczy nieustandaryzowane
    - Np. stringi, kontenery, obsługa błędów, brak ABI
  - Uboga biblioteka standardowa
  - Niski poziom
    - Potrzeba dużo kodu, żeby cokolwiek napisać
    - Łatwo o różnorodne błędy, np. w zarządzaniu pamięcią

Dlaczego C++?



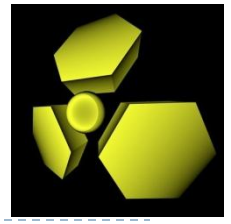
- Jednak C++ to najlepszy/jedyny wybór do gier
  - Dostatecznie wysoki poziom, by pisać złożone programy
    - Np. programowanie obiektowe
  - Dostatecznie niski poziom, by pisać wydajny kod
    - Np. wskaźniki, brak wirtualnej maszyny, ręczne zarządzanie pamięcią
  - Jest szeroko wspieranym standardem
    - API, biblioteki i silniki do gier mają zwykle interfejs do C/C++
  - Istnieją kompilatory na interesujące nas platformy
    - PC/Windows, Xbox 360, PlayStation 3

Nie kombinuj



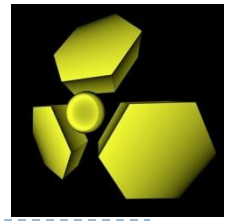
- W programowaniu gier chodzi o to, żeby napisać grę
- Warto pisać wszystko jak najprościej
- Dlatego nie kombinuj!
  - Nie musisz być mistrzem inżynierii oprogramowania
    - Nie powinieneś przekombinować z programowaniem obiektowym i wzorcami projektowymi
  - Nie musisz znać na pamięć standardu C++
    - Nie powinieneś przekombinować z preprocesorem, szablonami i przeciążaniem operatorów

C++ a programowanie gier

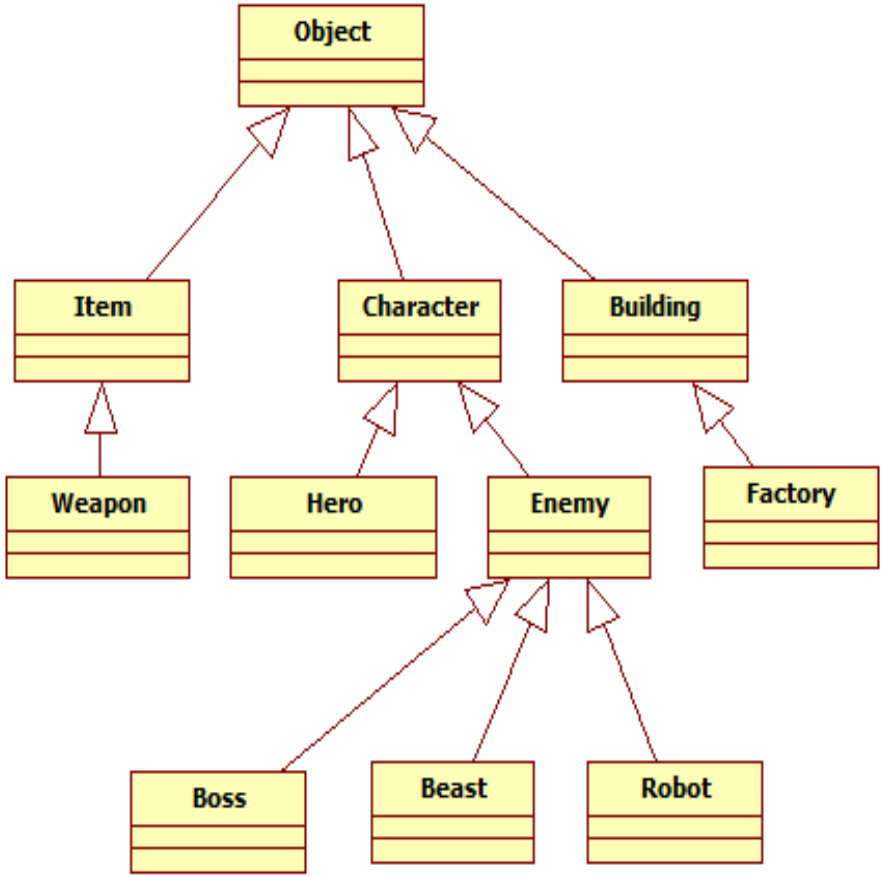


- Programowanie gier dopiero zaczyna się tam, gdzie kończy się nauka C++
  - Problem z konstruktorem klasy? Zrobić metodę Init!
  - Problem z const correctness? Immutability, koncepcja deskryptora.
  - Problem z singletonem? Jawnie inicjalizować i finalizować podsystemy!
  - Za dużo getterów i setterów? Pisać struktury, pola publiczne!
  - Wiele zagadnień typowych w programowaniu gier [GPPATTERNS]
    - Np. podwójne buforowanie, architektura komponentowa, pula obieków

Data-Driven



☹

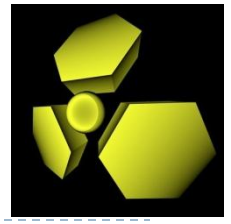


☺

```
"Boss": {
   "Life": 1000,
   "Armor": 200,
   "Weapon": "Laser"
},

"Laser": {
   "ParticleEffect":
     "LaserEffect02",
   "Damage": 1000,
   "Duration": 2.5,
   "Cooldown": 0.7
}
```

Optymalizacja

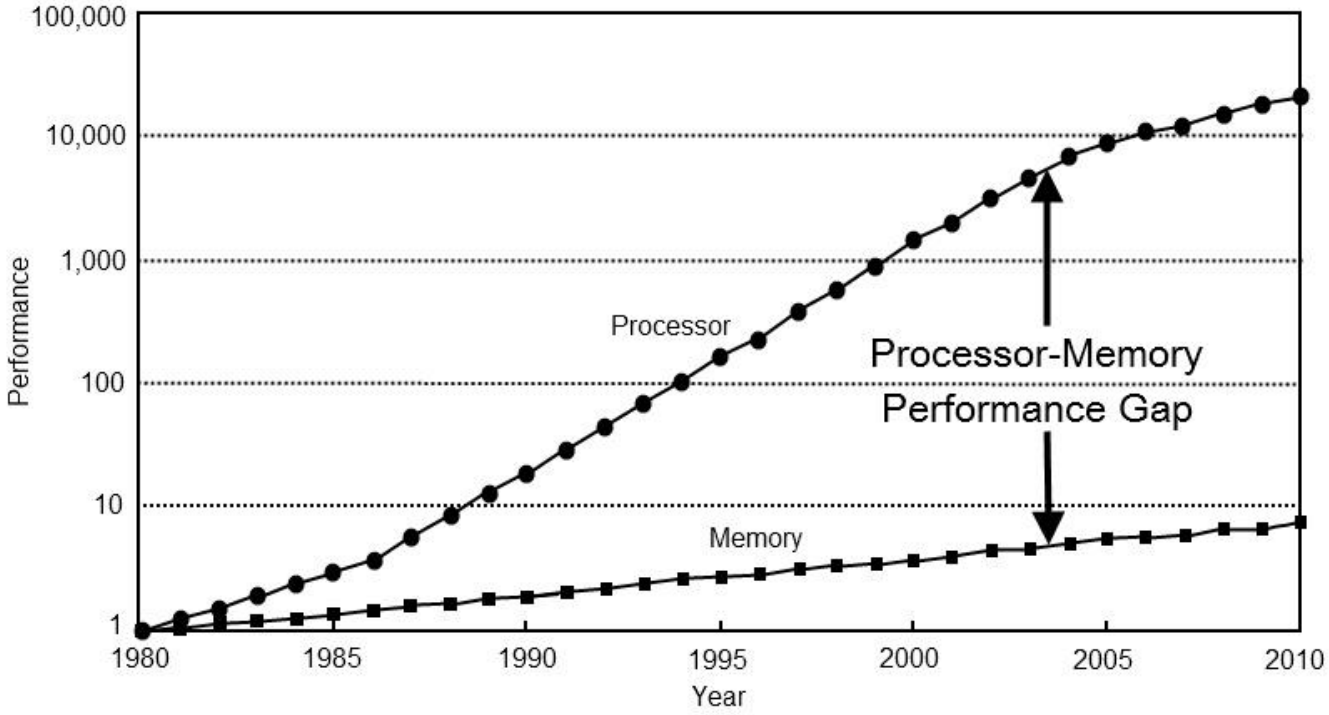


- Optymalizacja – co w praktyce oznacza?
  - Jednym kojarzy się z doborem algorytmu o dobrej złożoności asymptotycznej
  - Innym kojarzy się z przepisaniem algorytmu na asembler
- Prawda leży pośrodku!
  - Duże obiekty przekazywać i zwracać przez wskaźnik lub referencję, nie przez wartość
  - Upraszczać obliczenia matematyczne, mnożyć zamiast dzielić
  - Nie definiować złożonych zmiennych wewnątrz pętli
  - Nie ufać optymalizacji kompilatora ☺

Wydajność


Dodawanie, odejmowanie, mnożenie
Dzielenie, funkcje transcendentalne
Branching, metody wirtualne, skok pod wskaźnik
Dynamiczna alokacja pamięci
Zasoby systemowe – tekstury, wątki, gniazda
Wejście-wyjście – pliki, sieć

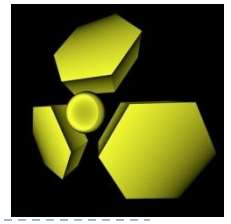
Optymalizacja



- Nietrafienie w pamięć podręczną – cache miss
  - 1980: Odwołanie do pamięci RAM ≈ 1 cykl procesora
  - 2009: Odwołanie do pamięci RAM ≥ 400 cykli procesora

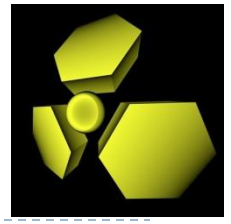
[POOP]

Optymalizacja



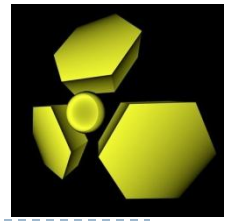
- Lokalność odwołań ważniejsza, niż ilość obliczeń
  - Programowanie obiektowe ☹
  - Programowanie zorientowane na dane ☺
  - Obiekty alokowane osobno, rozrzucone po pamięci ☹
  - Obiekty w tablicy, w ciągłym obszarze pamięci ☺
  - Pakowanie danych
    - short, char
    - Flagi bitowe
    - Pola bitowe
    - Nawet zmiana kolejności pól może wpłynąć na wydajność!

Najlepsza optymalizacja



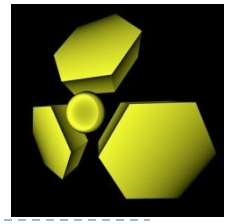
- „Najszybszy kod to taki, który nigdy się nie wykonuje”
- Precalc – przygotować dane wcześniej, nie liczyć za każdym razem
  - Podczas wczytywania gry NIE parsować plików tekstowych, XML, modeli OBJ, tekstur PNG, nie kompilować shaderów
  - Edytor lub inne narzędzia powinny przygotować assety w docelowym formacie: modele do wczytania prosto do VB/VBO, tekstury w DDS (ewentualnie JPEG), własne formaty plików, VFS
  - Podczas działania gry NIE liczyć w każdej klatce tego, co można przygotować raz lub policzyć raz na jakiś czas (np. w AI)
- Stosować culling i podział przestrzeni, aby nie przetwarzać tego, czego nie widać lub co nie jest istotne
- LOD – Level od Detail – kiedy mimo wszystko trzeba przetwarzać dużo danych, można mniej szczegółowo

Wyjątki



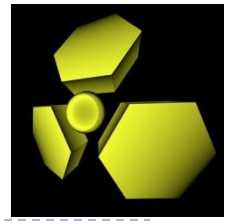
- W kodzie poważnej gry wyjątki powinny być nieużywane, a ich obsługa wyłączona
  - Brak standardowego sposobu obsługi błędów w C++
  - Niedoskonałość wyjątków C++ - brak finally, RAII nie jest powszechne
  - Pojęcie błędu w grze nie ma takiego sensu jak w programie
  - Włączone wsparcie dla wyjątków w kompilatorze C++ to duża strata wydajności, nawet kiedy wyjątki nie są faktycznie używane
  - Kompilatory C++ na konsolach nie wspierają wyjątków [KosztWyjątków]

STL



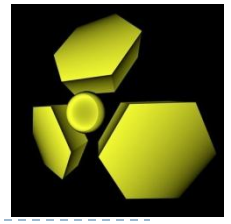
- STL-a też często nie używa się w programowaniu gier
  - Wprawdzie warto mieć szablony kontenerów i mogą działać wydajnie,
  - jednak STL robi zbyt dużo dynamicznych alokacji pamięci.
  - Stringów używać tylko do pokazywania tekstu użytkownikowi, nie wewnętrznie w kodzie gry
  - Unikać std::list, std::set, std::map, najlepszy jest std::vector
  - EASTL – implementacja STL dostosowana do programowania gier [EASTL], udostępniona ostatnio za darmo [EASTL/GIT]

Programowanie równoległe



- To dziedzina obszerna, trudna, wciąż dojrzewająca,
- ale bardzo ważna, opłacalna i już obowiązkowa!
- Wątki i muteksy to dopiero początek…

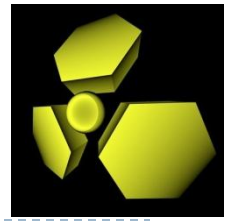
Narzędzia



- Jedyna słuszna platforma: Windows :)
- Jedyne słuszne IDE: Visual C++ :)

- Narzędzia pomocnicze:
  - Śledzenie wycieków pamięci: własny alokator pamięci, Visual Leak Detector
  - Profilowanie: własna instrumentacja, Very Sleepy, AMD CodeAnalyst, Intel Vtune
  - Debugowanie bibliotek: Debug version of Direct3D, PIX, PhysX Visual Debugger
  - Statyczna analiza kodu: CppCheck

Bibliografia



- [TIOBE] TIOBE Programming Community Index for October 2010
  - http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
- [C++FQA] C++ FQA, Yossi Kreinin
  - http://yosefk.com/c++fqa/
- [GPPATTERNS] Game Programming Patterns, Robert Nystrom
  - http://gameprogrammingpatterns.com/
- [POOP] Pitfalls of Object Oriented Programming, Tony Albrecht (Sony Computer Entertainment Europe)
  - http://bit.ly/90fCdE
- [KosztWyjątków] Koszt wyjątków, forum.gamedev.pl
  - http://forum.gamedev.pl/index.php/topic,19151.msg229335.html#msg229335
- [EASTL] EASTL -- Electronic Arts Standard Template Library
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2271.html
- [EASTL/GIT] paulhodge / EASTL
  - http://github.com/paulhodge/EASTL