Programowanie gier

dr inż. Paweł Wojciechowski
Instytut Informatyki
Politechniki Poznańskiej
2011

Z czego będziemy korzystać

- SDL?
- Microsoft XNA
- Farseer Physics
- NewtonSDK/Ogre3D
- Ogre3D
- Blender
- evaLUAtion
- FX Composer

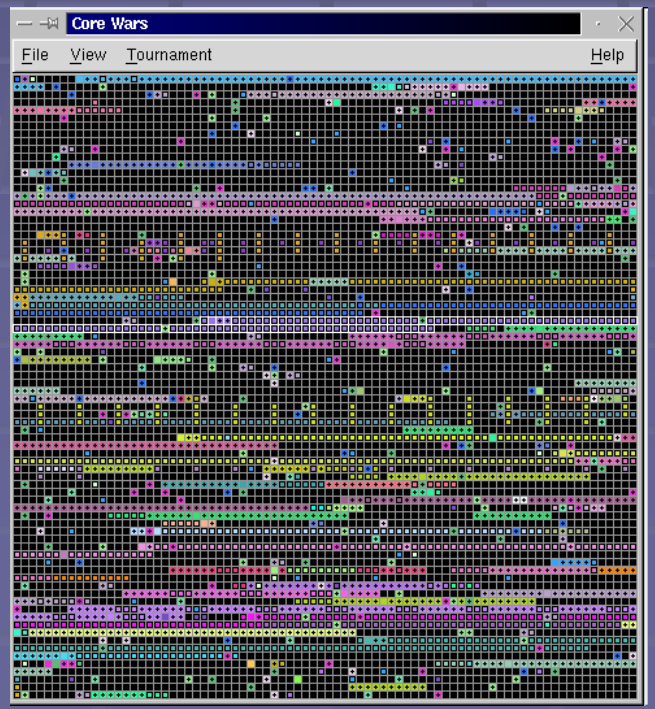
evaLUAtion

Geneza powstania

- Wojny rdzeniowe

  „Gra pomiędzy programami napisanymi w asemblerze na danej maszynie lub symulatorze, gdzie celem jest zabicie programu przeciwnika przez jego nadpisanie.”

  The Jargon Lexicon



evaLUAtion

- autorem evaLUAtion jest Piotr Sienkowski.
- jest to framework do nauki pisania skryptów sztucznej inteligencji dla gier komputerowych
- Środowisko do testowania skryptów kontrolowanych przez boty”/agentów

> BOT - user who is actually a program
>
> The Jargon Lexicon

- Agent może być sterowany poprzez:
  - skrypty Lua,
  - zbiory rozmyte,
  - mysz/klawiatura.

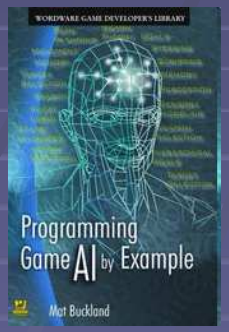
http://svn2.assembla.com/svn/evaLUAtion-lynx/

Lua to…

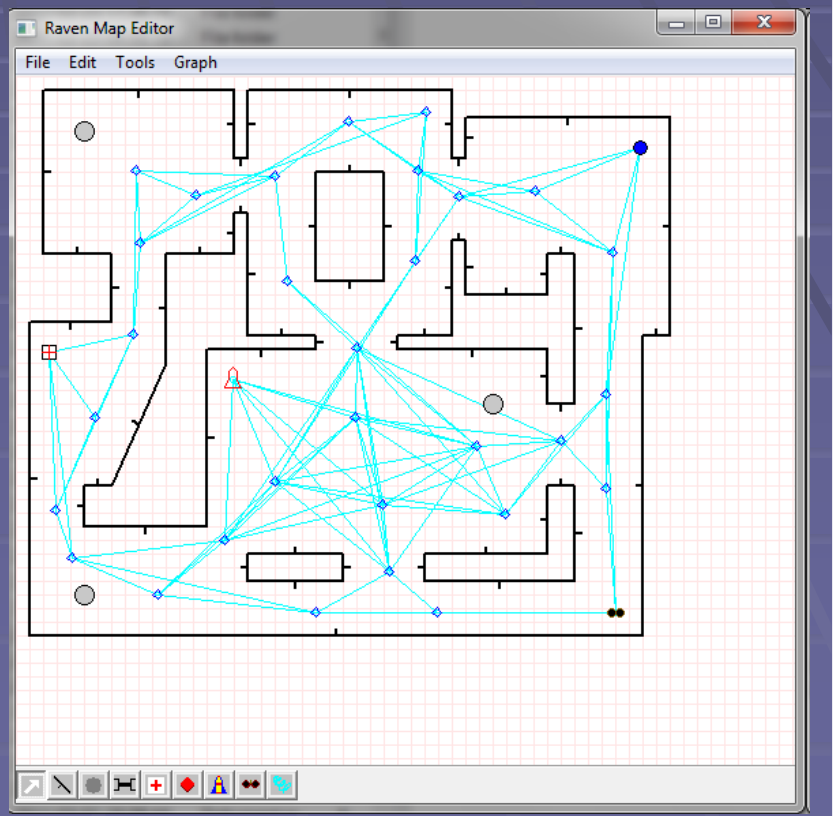
- **sprawdzone rozwiązanie**
  - zastosowania biznesowe (Adobe's Photoshop Lightroom),
  - gry (World of Warcraft)
- **szybkośc** – jeden z najszybszych interpretowanych języków skryptowych
- **przenoszalność** (Unix, Windows, Symbian, Pocket PC, embedded microprocessors – Lego MindStorm)
- **„osadzalność"** – może zostać użyty do rozszerzenia programów napisanych w C, C++, Java, C#, Fortran, Ada, Perl and Ruby)
- **mały**
- **prosty**
- **darmowy**

evaLUAtion - mapy

- do budowy map wykrzystano edytor dostarczony z książką



Mat Buckland
Programming Game AI by Example
2005



mapa zapisana jest w pliku tekstowym

http://svn2.assembla.com/svn/evaLUAtion-lynx/

Plik konfiguracyjny

```
#plik evaluation.cfg
height=768
width=1024
#respawns=yes

[map]
filename=maps/sparseDM1.emf
actors.file=actors.eaf

[actor]
speed=50
maxHealth=300

[weapon]
chaingun.ammo=100
railgun.ammo=1
rocket.ammo=2
shotgun.ammo=5
```

Plik opisujący aktorów

```
#actors.eaf
#nazwa        rodzaj         team   pozx  pozy    wyglad
Lynx           ms             1      45    35     skeleton.bow.
Fuzzy          fz             1      40    70     skeleton.bow.
RobsonFz       fz             2      70    380    man.staff.
PawelFz        fz             2      320   70     skeleton.bow.
BonesScript    sc:bones.lua   3      380   380    man.staff.
FleshScript    sc:flesh.lua   3      250   250    man.staff.
```

Pierwszy skrypt - wymagania

```
#actors.eaf
Lynx          ms             1      45     35     skeleton.bow.
dumbScript    sc:dumb1.lua   2      250    250    man.staff.
```

```
--dumb1.lua
counter = 0
function dumb1whatTo( agent, actorKnowledge, time)
   io.write( counter, ”\n”)
   counter = counter + 1
end;

function dumb1onStart( agent, actorKnowledge, time)
   io.write( counter)
end
```

Klasa Agent

Klasa którą sterujemy zachowaniem agenta

```
class LuaAgent
{
   void selectWeapon(Enumerations::WeaponType weapon);
   void moveDirection(Vector4d direction);
   void moveTo(Vector4d target);
   void shootAtPoint(Vector4d vect);
   void rotate(Vector4d direction);

   double randomDouble();
   void reload();
   void wait();
   void continueAction();
}
```

Klasa ActorKnowledge

Klasa, dzięki której wiemy, co dzieje się dookoła.

```
class ActorKnowledge
{
public:
    int getAmmo(Enumerations::WeaponType type);
    Vector4d getPosition();
    Vector4d getDirection();
    Navigation * getNavigation();
    unsigned short getTeam();
    int getHealth();
    Enumerations::WeaponType getWeaponType();
    int getArmour();
    bool isMoving();
    bool isLoaded(Enumerations::WeaponType type);
    Vector4d getLongDestination();
    Vector4d getShortDestination();
    const char* getName();

    ActorInfo getSelf();
    std::vector<ActorInfo> getSeenFriends();
    std::vector<ActorInfo> getSeenFoes();
    std::vector<ActorInfo> getSeenActors();

    int getEstimatedTimeToReach(Vector4d self, Vector4d target);
};
```

Klasa ActorInfo

```
class ActorInfo
{
public:
   Vector4d getPosition();
   Vector4d getDirection();
   void setPosition(Vector4d position);
   void setDirection(Vector4d direction);
   void setActionType(Enumerations::ActionType type);
   unsigned short getTeam();
   void setTeam(unsigned short team);
   int getHealth();
   void setHealth(int health);
   int getArmour();
   void setArmour(int armour);
   const char* getName();
   Enumerations::WeaponType getWeaponType();
   void setWeaponType(Enumerations::WeaponType type);
};
```

```
--dumb2.lua

function dumb2whatTo( agent, actorKnowledge, time)

end;

function dumb2onStart( agent, actorKnowledge, time)

   io.write( "My name is: ", actorKnowledge:getName(), "\n")
   io.write( "Number of Navigation Points is: ",
    actorKnowledge:getNavigation():getNumberOfPoints() ,"\n")
end
```

```
--dumb3.lua
counter = 0
function dumb3whatTo( agent, actorKnowledge, time)
   if ( actorKnowledge:isMoving()) then
   dist = actorKnowledge:getLongDestination()
   io.write( "ide do ")
   showVector( dist)
   io.write( "\t")
   showVector( actorKnowledge:getShortDestination())
   io.write( "\n")
   else
   if ( time > 100) then
        agent:moveTo( Vector4d( 40, 40, 0, 0))
   end
   end
end;

function dumb3onStart( agent, actorKnowledge, time)
   io.write( counter)
   io.write( "Mam na imie: ", actorKnowledge:getName(), "\n")
end

function showVector( vector)
    io.write( "(", vector:value(0), ",", vector:value(1), ",")
    io.write( vector:value(2), ",", vector:value(3),");")
end;
```

UWAGA: jeżeli pozycja do której chcemy się ruszyć jest nieosiągalna - nic się nie stanie.

```
function dumb4whatTo( agent, actorKnowledge, time)
   if ( actorKnowledge:isMoving()) then
       dist = actorKnowledge:getLongDestination()
       io.write( "ide do ")
   showVector( dist)
   io.write( "\t")
   showVector( actorKnowledge:getShortDestination())
   io.write( "\n")
   else
   if ( time > 100) then
       agent:moveTo( Vector4d( 40, 40, 0, 0))
   end
   end

   enemies = actorKnowledge:getSeenFoes()
   if ( enemies:size() > 0) then
   agent:shootAtPoint( enemies:at(0):getPosition())
   end
end;
```

Zmiana broni

```
enemies = actorKnowledge:getSeenFoes()
if ( enemies:size() > 0) then
agent:shootAtPoint( enemies:at(0):getPosition())
end

if ( actorKnowledge:getAmmo( actorKnowledge:getWeaponType()) == 0)
then
weapon = actorKnowledge:getWeaponType()
weapon = weapon + 1
    agent:selectWeapon( weapon)
end
```

Operacje na wektorach

```
function dumb6whatTo( agent, actorKnowledge, time)

  enemies = actorKnowledge:getSeenFoes()
  if ( enemies:size() > 0) then
     dir = enemies:at(0):getPosition() - actorKnowledge:getPosition()
     agent:rotate( dir)

     io.write( "on trzyma ", enemies:at(0):getWeaponType())
     io.write( " dist: ", dir:length(), "\n")

  end

end;
```

Wartości losowe

```
function dumb7whatTo( agent, actorKnowledge, time)

  enemies = actorKnowledge:getSeenFoes()
  if ( enemies:size() > 0) then
     dir = enemies:at(0):getPosition() - actorKnowledge:getPosition()
     --agent:rotate( dir)
     agent:moveDirection( dir*(-1))
     if ( dir:length() < 20) then
    dest = Vector4d( agent:randomDouble()*800, agent:randomDouble()*800, 0,0)
         agent:moveTo( dest )
     end

     io.write( "on trzyma ", enemies:at(0):getWeaponType())
     io.write( " dist ", dir:length(), "\n")
  end

end;
```

Amunicja, apteczki i pancerz

```
class Navigation {
public:
    /**checks if ray colides any solid thing on the way
    */
    bool anyRayCrateColision(Vector4d displacementVector,
                                        Vector4d positionVector);
    Vector4d getNodePosition(int index);
    /**finds shortest way between two points
    */
    std::vector<int> searchWay(Vector4d from, Vector4d to);
    int getNumberOfTriggers();
    Trigger * getTrigger(int index);
    int getNumberOfSpawnPoints();
    Vector4d getSpawnPoint(int index);
}
class Trigger : public Entity
{
public:
    enum TriggerType {
          Weapon,
          Armour,
          Health
    };
    TriggerType getType();
    bool isActive();
    Vector4d getPosition();
    double getBoundingRadius();
}
```

Amunicja, apteczki i pancerz

```
function dumb8whatTo( agent, actorKnowledge, time)

  nav = actorKnowledge:getNavigation()

  for i=0, nav:getNumberOfTriggers() -1, 1 do
     trig = nav:getTrigger( i)

     if ( trig:getType() == Trigger.Health and
          trig:isActive()) then
   agent:moveTo( trig:getPosition())
     end

  end

end;
```

Uwagi dotyczące drużyn

- można uszkodzić kolegę z drużyny ;)
- zmienne globalne
- wymiana informacji – pliki?

Regulamin konkursu 1/4

1. Oficjalny framework turniejowy dostępny jest tutaj (Ostatnia aktualizacja frameworku: 03.03.2011 g. 11:50 Lista zmian).
2. Turniej odbędzie się na dwóch kolejnych wykładach tj. 16.05.2011 i 30.05.2011.
3. Rozwiązania należy nadsyłać do 13.05.2011 do godziny 23:59.
4. Eliminacje odbędą się 9.05.2011 i będą niejawne.
5. Eliminacje mają na celu wyłącznie podział na koszyki: silniejszy i słabszy. Chcemy w ten sposób uniknąć sytuacji, gdy do jednej grupy trafiają bardzo silne drużyny. Eliminacje nie są obowiązkowe!! Drużyny które nie wezmą udziału w kwalifikacjach od razu trafiają do słabszego koszyka. W przypadku, gdy zostanie zgłoszonych za mało drużyn do kwalifikacji, podział na koszyki (i grupy) będzie losowy.
6. Osoba, która nie wystawi swoich skryptów w turnieju, w celu zaliczenia przedmiotu musi napisać skrypty, które będą w stanie pokonać zwycięską drużynę z poprzedniego roku!!! Oczywiście skrypty z którymi należy wygrać nie zostaną udostępnione.
7. Zgłoszenie powinno zawierać skrypt(y), opis aktorów (co należy wpisać w pliku konfiguracyjnym oraz opis działania skryptów.
8. Każdy z graczy ma do dyspozycji 3 boty, z których każdy może (ale nie musi) być sterowany innym skryptem.
9. Czas działania skryptu jest ograniczony do rozsądnych granic. Symulacja powinna przebiegać płynnie. W przypadku kłopotów z czasem reakcji skryptów drużyna może zostać ukarana.

Regulamin konkursu 2/4

10. Respawn jest wyłączony.
11. Pojedynek trwa do wyeliminowania wszystkich botów należących do jednej drużyny bądź określony czas (2-5minut). Dokładny czas symulacji zostanie ustalony w dniu rozgrywania turnieju.
12. W pojedynku biorą udział dwie drużyny. Przy czym bitwa między dwoma drużynami składa się z 2 pojedynków.
13. Każdy z pojedynków punktowany jest oddzielnie.
14. W drugim pojedynku następuje zamiana miejsc startowych botów w stosunku do pierwszego pojedynku.
15. Punktacja jest następująca:
    - "**eliminacja**": eliminacja wszystkich botów przeciwnika: **5** pkt
    - "**zwycięstwo**": po upływie czasu pojedynku, pozostało więcej botów naszej drużyny niż przeciwnej: **3** pkt
    - "**remis**": po upływie czasu pojedynku, pozostało tyle samo botów obu drużyn: **2** pkt
    - "**porażka**": po upływie czasu pojedynku, pozostało więcej botów drużyny przeciwnej (włączając sytuację gdy ostatni strzał zabija wszystkie boty obu drużyn): **1** pkt
    - "**nokaut**": wszystkie boty drużyny zostały wyeliminowane przed czasem: **0** pkt
16. Liczba punktów życia zostanie zwiększona do 500(1000) tak, żeby zmusić skrypty do poszukiwania amunicji, apteczek itd.

Regulamin konkursu 3/4

17. Przewiduje się 2 fazy turnieju:
    - grupowa - wszystkie drużyny zostają podzielone na grupy, w ramach której odbywają się bitwy każdej drużyny z każda.
    - pucharowa - zwycięża drużyna, która uzyskała więcej punktów w bitwie
18. W celu wyważenia składu grup w fazie grupowej, dopuszcza się możliwość przeprowadzenia rundy kwalifikacyjnej (patrz eliminacje).
19. Liczba grup zostanie dobrana na podstawie liczby zgłoszonych drużyn.
20. Do fazy pucharowej przechodzą co najmniej po 2 najlepsze drużyny z grupy. Liczba ta może ulec zmianie i zostanie uściślona w dniu rozpoczęcia turnieju.
21. Dopuszcza się jednorazowe, NIECELOWE zawieszenie pojedynku. W takim przypadku pojedynek zostanie zrestartowany. Kolejne zawieszenie pojedynku przez skrypt powinno zostać wykryte i ukarane walkowerem.
22. Dostępne są następujące kary, o których nałożeniu na drużynę decyduje komisja, składająca się z prowadzącego i uczestników turnieju.
    - **dyskwalifikacja** - za skrypty uniemożliwiające przeprowadzenie turnieju (np. restarty, deadlocki itd).
    - **wykluczenie** - skrypty które nie stosują się do punktu 6 regulaminu można wykluczyć z pojedynku/bitwy. W takim przypadku drużyna składa się z mniejszej liczby botów. Drużyna poszkodowana decyduje, czy wynik pojedynku w ramach którego doszło do naruszenia p. 6 regulaminu ma pozostać aktualny, czy ma się odbyć powtórka pojedynku po nałożeniu kary.

Regulamin konkursu 4/4

23. Nagrody:
    - student, którego drużyna wygra turniej otrzymuje ocenę 5.0 z przedmiotu
    - student, którego drużyna zajmie 2 miejsce dostaje podwyższenie oceny o 1.0, lub 4.0 z przedmiotu
    - student, którego drużyna zajmie 2 miejsce dostaje podwyższenie oceny o 0.5, lub 3.0 z przedmiotu
24. W przypadku unikania walki przez drużyny w pojedynku możliwe jest przeprowadzenie kolejnego/restart pojedynku na innej mapie.
25. Zakłada się uruchomienie skryptów z inną (zmniejszoną) liczbą początkową amunicji niż dotychczasowa.
26. Przykładowy skrypt z zeszłego roku (oczywiście zgłoszenie go lub jego drobnych modyfikacji jako swojego traktowane jest jak brak zgłoszenia do turnieju) jest tutaj.