

Использование Python в качестве скрипт-языка в игровых движках

Лыфарь Дмитрий

Overview

- Средства интеграции с C++
- Использование в играх
 - Производительность
 - Memory management
 - Сборка проекта
 - Загрузка\сохранение
- Краткое сравнение python vs. lua.

Почему Python можно и нужно использовать в качестве Game Scripting Engine?

- Легкая интеграция
- Затраты на написание maintainable кода меньше в 5-10 раз
- Наличие высокоуровневых примитивов(list, tuple, etc) и множество модулей, облегчающих написание кода
- Наличие сборщика мусора и, как следствие, повышенная «помехозащищенность»
- Отсутствие link-time, возможность изменять и выполнять скрипты «на лету»
- Стабильность и большое community
- Поддержка ООП

Python может помочь в автоматизации сборки

- В нашем проекте python очень облегчил сборку конечной версии для тестирования (кроме того, python интегрирован с svn)
- Набор скриптов для экспорта из графических пакетов
- Скрипт для сборки игры «одним запуском» на машине тестера

Как использовать C++ код?

- Написать wrapping для каждой функции и обеспечить проверку значений
- Таблица экспортируемых функций
- Функция инициализации модуля

Что умеет Boost.Pyste?

- Автоматизация написания glue-code при помощи интерфейсных .pyste файлов
- Пример .pyste файла:

```
NPCPlayer_wrapper = Class('NPCPlayer', 'NPCPlayer.h')  
getAI = Function('getAIPlayer', 'NPCPlayer_wrapper.h')  
set_policy(getAI, return_value_policy(reference_existing_object));
```

- Поддержка:
 - Классов и шаблонов
 - Перегрузки функций
 - Виртуальных функций
 - Boost::shared_ptr и std::auto_ptr
 - etc

Что умеет Boost.Pyste?(продолж.)

- Пример сгенерированного кода для модуля NPCPlayer

```
void Export_c_Work_cherpaev_include_pyste_npcplayer_wrapper()
{
    class_ < NPCPlayer_wrapper >("NPCPlayer_wrapper", init< const NPCPlayer_wrapper& >())
        .def(init< unsigned int, const std::string& >())
        .def_readwrite("mInfo", &NPCPlayer_wrapper::mInfo)
        .def_readwrite("mDraught", &NPCPlayer_wrapper::mDraught)
        .def_readwrite("mScriptPassed", &NPCPlayer_wrapper::mScriptPassed)
        .def("findPath", &NPCPlayer_wrapper::findPath)
        .def("findTakeOffPath", &NPCPlayer_wrapper::findTakeOffPath)
        .def("checkFreeBonusVisibility", &NPCPlayer_wrapper::checkFreeBonusVisibility)
        .def("getRealDraughtIndex", &NPCPlayer_wrapper::getRealDraughtIndex)
        .def("getRealEnemyDraughtIndex", &NPCPlayer_wrapper::getRealEnemyDraughtIndex)
        .def("isPointInSight", &NPCPlayer_wrapper::isPointInSight)
        ;

    def("getAIPlayer", &getAIPlayer, return_value_policy< reference_existing_object >());
}
```

Наследование от C++ классов

- Используя Python можно легко определять собственные потомки классов C++ и их поведение
- ```
class Base {
 public: virtual int f(std::string x) { return 1; }
};
```
- Диспатчер  

```
struct BaseWrap : Base
{
 BaseWrap(PyObject* self_) : self(self_) {}
 PyObject* self;
 f_default(std::string x) { return this->Base::f(x); }
 f(std::string x) { return call_method<int>(self, "f", x); }
};
```
- Экспорт этого класса:  

```
class_<Base, BaseWrap>("Base") .def("f", &Base::f, &BaseWrap::f_default)
```

# Object Interface в Boost.Python

- Вы можете писать на C++, так как будто делаете это на Python:

```
object s("hello, world");
object ten_Os = 10 * s[4]; // -> "oooooooooooo"
```

- Используя шаблон `extract<T>` из Boost.Python можно преобразовать тип `object` в тип C++:

```
double x = extract<double>(obj);
```

# Memory Allocation and its problems

- Любая сущность в Python – объект
- Все объекты reference-counted
- Возможность падения frame rate во время работы gc
- При небрежном использовании встроенного gc возможно занятие значительного пространства виртуальной памяти

# Memory Allocation problems resolution

- Модуль gc позволяет управлять работой сборщика мусора и уровнями отладки(DEBUG\_SAVEALL, DEBUG\_STATS)
- Использование пулов памяти (Pymalloc, используется сейчас в Python по умолчанию)

# Производительность

- Сложные вычислительные задачи не для Python
- Переложите все критичные участки кода на плечи C++
- Использование python в качестве game-loop, AI logic и для написания прототипов вполне оправданно
- И все же в некоторых случаях Python быстрее:
  - Работа со строками(reference-count позволяет избежать копирование в некоторых случаях, в которых оно используется в C++)
  - Большинство операций с `std::map` –  $O(\log(n))$ , Python hash-tables –  $O(1)$

# Производительность(продолж.)

- Помните о правиле 90/10
- Используйте встроенные модули для профилирования(`profiler`, `trace`)
- Набор “универсальных советов”:
  - Не читайте файлы построчно, пользуйтесь буфером
  - Короткие имена для переменных в Python это не шутка
  - Избегайте `heavy-use` переменных в области видимости модуля, используйте локальную

# Загрузка\сохранение состояния игрового мира

- На C++ это большое неповоротливого количество кода
- Модуль cPickle позволяет загружать и восстанавливать объекты любой сложности

- Загрузка

```
gameObject = cPickle.load(open("C:\game.sv"))
```

- Сохранение:

```
import cPickle
gameObject = {'Attack': 3, 'Defence': 2, 'Health': 89, 'name': 'Ogre_1'}
cPickle.dump(gameObject, open("C:\game.sv", "wb"))
```

# Проблемы и недостатки

- Трудно отлаживать код
- Утомительная разработка и поддержка glue-code
- Сложность из-за взаимосвязей между двумя разными языками
- Отсутствует compile-time проверки, как в C++, из-за чего много ошибок проявляются только в run-time

# Python vs Lua

## ■ Python

### □ Против

- Неповоротливые и неудобные отладчики
- Довольно сильные падения производительности в некоторых местах

### □ ЗА

- Множество хорошо документированных библиотек, выполняющих большую часть рутинной работы
- Простой и удобный синтаксис
- Большое community
- Лучшая поддержка unicode и locales

# Python vs Lua

## ■ Lua

### □ Против

- Не очень хорошая документация
- Меньшая built-in функциональность
- Плохая поддержка locales

### □ За

- Быстрее и использует меньше памяти(<http://shootout.alioth.debian.org>)
- Хорошая интеграция с C++

# Где использовался?

- Open source 3D редактор Blender3D
- Open source 3D engine Nebula
- Commercial
  - Eve Online
  - Battlefield
  - Freedom force
  - Severance: Blade of Darkness
  - Tooltown
  - etc