转 Handling Complexity in the Halo 2 AI

2015年09月27日 06:10:45 wolf96 阅读数 845

原文：http://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php

Developers of game AI are always interested in cramming more complexity into the virtual brains they build. However complexity often ha rather has many prices: poor run-time, poor scalability, a lack of directability and worst of all, a murky experience for the player in which t to act "randomly" rather than "intentionally". We will discuss the sources of complexity, the various ways in which complexity can manifes also some of the architectural approaches we took in *Halo 2* to mitigate these negative effects. This discussion will center on the problem decision-making, but will also touch on questions of memory, knowledge model, and control representations for the scripting and influen AI by level designers.

The Brute Force Approach to Common Sense

When it comes to game AI, more is often better, in the sense that the more well-rounded an AI's behavioral repertoire is, the more unique AI can recognize, and the more unique ways an AI can respond, the more likely we are to see the AI as a common-sensical kind of creatur "common sense" AI is a long-standing goal for much of the research AI community. For many (for example, [Lenat95] and [Stork99]), the c common sense is intimately connected to the problems of knowledge acquisition and representation. After all, common sense can simply considered the massive database of mundane, everyday knowledge that is so obvious to the walking, seeing, thinking human being that i needs to be taught or even expressed. This makes common sense a very elusive thing indeed.

For games, or at least for *Halo 2*, we are far less interested in encoding factual knowledge (birds have wings, water gets things wet) than v encoding behavior, which is perhaps a different sort of knowledge. This is the knowledge that says that when you are sitting in a vehicle s have to get out of the seat before you can walk through the door, or that in order to stop someone from shooting you, you need to move place a large sturdy barrier between you and your attacker. In both styles of common sense however, the solution is the same: quantity. T AI knows, the better.

Quantity, of course, *is* complexity, especially when considered along with some of the other constraints that The Game forces upon us. It i enough that the AI be able to do it a lot of things, it is equally important that they do all those things *right*, at the right *times*, and in a wa not break the illusion of life, or threaten the player's understanding of the AI's intentions or motivations. In *Halo 2*, the AI works best whe believes he is fighting a living breathing (evil) creature, and can respond to and predict that creature's actions accordingly. As authors of t of our primary goals is to facilitate the on-going narrative that is taking place in our player's head: "oh, the grunt just ran away screaming pulled out my energy sword and it was scared, but when I had it cornered, it turned around and started fighting again".

Attaining both these goals - quantity on the one hand and what we might term behavioral integrity on the other - is a huge *architectural* Because whatever the content of the knowledge we encode, we need an appropriate container to put it all in, hopefully a container that a perennial large-system-design concerns of scalability, modularity, transparency and so on.

We pay for complexity in a number of ways:

- Coherence: If behavior is action over time, we need to make sure that our AIs start, stop and change actions at appropriate times. Anc avoid at all costs the problem of dithering (the rapid flipping back and forth between two or more actions).
- Transparency: given the AI's outward stance, it must be possible for the untrained observer to make reasonable guesses as to the AI's state as well as explain and predict the AI's actions.
- Run-time: The most obvious of all constraints. The AI has to run at 30Hz or more.
- Mental bandwidth: When we lose the ability to reason about what's going on in the system, we lose control over it.

Quantity in service of common sense is not the only sources of complexity. Consider these others:

- Usability: The AI must be directable enough to support the larger fictional setting of the game. The "user" in this case is not the playe level designer, who must craft a drama over the course of the level through character placement, scripting and high-level direction.
- Variety: Different AIs behave in different ways according to their character. How do we design a system that provides a base of robust sense behavior but that also allows for character-to-character variety?
- Variability: AIs should behave different in different situations, especially when those situations are directed by the designers in service (for example, one scene might demand that the player-ally AI be holed up, defending themselves from an onslaught that the player w rescue them from, while the next might send the same AI out on an assault with the player).

- Run-time: the one concern that can both suffer from and contribute to complexity. Much of the complexity of an architecture like *Ha* from our desire to avoid work we don't need to do.

This paper will discuss some of the techniques that Bungie used in the implementation of the *Halo 2* AI to handle the burgeoning comple problem. The first half of the paper will deal mostly with questions of internal architecture, particularly as it relates to memory and decisio The second half of the paper will present some of the useful level-design tools we used to control AI and script levels.

Core Combat Cycle

In the beginning, it's all very simple. It probably starts out looking something like Figure 1. This is the kind of diagram a game designer m up with to describe the ways in which a player may interact with AI. Clearly each of the states shown describes very different modes of be our characters, preferably with their own animation styles (sneaking for searching, flailing panic for flight, etc.) How might we go about im this scheme?

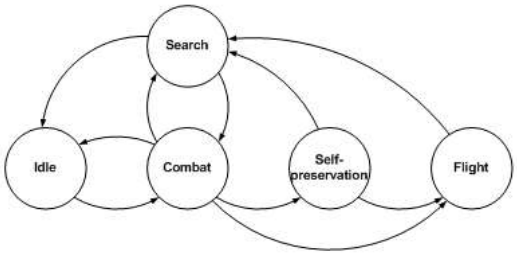

Figure 1: The combat cycle

The first thing to recognize is that the figure contains all kinds of hidden complexities. For example, for each of the arrows we have a ques "when is it appropriate to follow this transition?" Some of the transitions are voluntary (for example, the decision to give up searching anc idle). Others are forced by perception: clearly from combat we are forced along a transition, either to *idle* or to *search*, when our target st an obstacle. In other words, the diagram is a useful conceptual tool (particularly for designers), but falls far short of being implementable.

Behavior

What does the actual control structure look like? Like many systems, the *Halo 2* AI implements a hierarchical finite state machine (HFSM) tree, or even more specifically, a behavior DAG (directed acyclic graph), since a single behavior (or behavior subtree) can occupy several lc the graph. An example is shown in Figure 2. This is a highly abbreviated version of the actual core behavior DAG of *Halo 2*, which contains order of 50 different behaviors.

HFSMs are a well-known and time-honored technique for decision-making. We will therefore confine our current discussion to some of th features" we found useful in *Halo 2*.

Decision routines

In a typical HFSM scheme, the role of non-leaf behaviors is to make decisions (specifically, decisions about which of its children to run), w of the leaf behavior is to get something done. When it comes to the decision-making process that takes place in the former, there are twc approaches: (a) the parent behavior can make the decision using custom code, or (b) the children can compete, with the parent making th choice based on child behavior desire-to-run, or relevancy. Both options will in fact be useful to us at different times, so we leave the abili customized decision routines on the table.

Design Principle #1: Everything is customizable

Where we can, we will use the more general mechanism (option b), particularly for some of the core components of the combat cycle, eac will be parent to many children (on the order of ten to twenty). Using this approach for these parents is a good idea, since writing hard-cc choose between one of twenty options can be tedious, as well as unscalable.

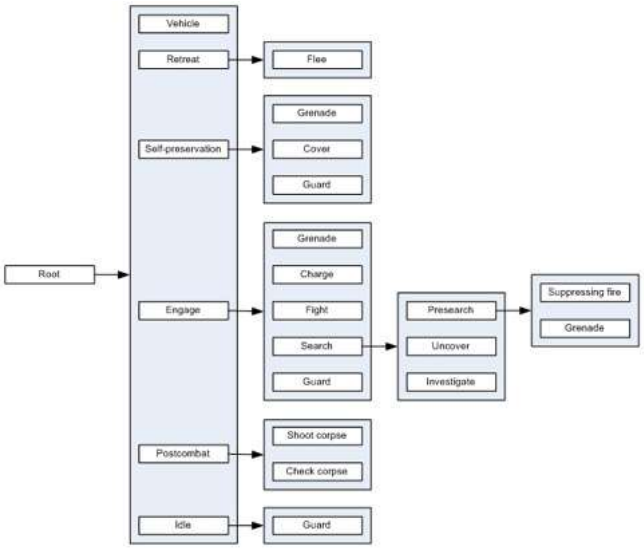

Figure 2: example behavior dag

Assuming we do use a child-competitive decision model, how do we actually go about picking a winner? Numerous systems feature an ar activation desire: each child provides a floating point number indicating its relevancy, and the child with the highest relevancy wins (with t tick's winner given an added bonus to avoid dithering). This does, however, again face a scalability problem once the number of competir gets above a certain number, especially when a very specific set of priorities is desired (for example, "fight the target, unless the player dri vehicle, in which case get in his vehicle"). Tweaking floats in order to get a specific set of rules or priorities is feasible when there are two c choices, but when there are twenty or more it is almost impossible.

We will simplify this scheme considerably by making relevancy a binary test. Using this approach, we were able to define a small number c decision schemes:

- *prioritized-list*: march down a prioritized list of the children. The first one that can run, does, but higher-priority siblings can always int winner on subsequent ticks.
- *sequential*: run each of the children in order, skipping those that are not currently relevant (and never revisiting it). When we reach th list, the parent behavior is finished.
- *sequential-looping*: same as above, but when we reach the end of the list, we start again.
- *probabilistic*: a random choice is made from among the relevant children.
- *one-off*: pick in a random or prioritized way but never repeat the same choice.

Of these, by far the most commonly used is the *prioritized-list* scheme. It has a number of great advantages, not the least of which is that in line with the way that we generally think of solving problems: we think first of the best thing to do, but failing that we will consider the the third best and so on. Whichever we choose, when a better option opens up, we immediately switch to it.

Behavior Impulses

However, this presents a new problem: what about when the priority is not fixed? In other words, under certain circumstances behavior A over behavior B ("fight rather than getting into a nearby vehicle") but under other circumstances, B has priority over A? ("Unless the playe vehicle, in that case do get in.") To solve this problem, we use a behavior **impulse**. An impulse is a free-floating trigger which, like a full be provides a binary relevancy, but is itself merely a reference to a full behavior. When the impulse wins the child competition either the curr running behaviors is redirected to the position of the referenced behavior, or the referenced behavior is simply run in the position of the i the example given above, we are interested in the latter. Our priority list becomes

```
player_in_vehicle_impulse
fight_behavior
enter_vehicle_behavior
```

The important point here is that we have explicitly separated out the condition that would have made the *enter_vehicle_behavior* more de a separate impulse that nonetheless references the same behavior.

Design Principle #2: Value explicitness above all other things

As mentioned, impulses can also serve to redirect the current behavior stack to another portion of the tree. For example, there might be s preservation impulses (self-preserve due to damage, self-preserve when facing a scary enemy, etc.) that are children of the *engage* behav impulses are only considered when the AI is engaging. When one of these impulses is chosen, rather than running *self-preservation* unde simply pop up a level in the tree and run self-preservation in its native position. The semantics for how this redirection is performed (in pa level of the tree to search for the referenced behavior, and what limitations to place on the reference itself) are somewhat involved. Suffic that impulses can at times act as "pointers" to other branches of the tree and cause execution to jump to that branch.

Impulses serve us well in another way. Consider an impulse that never returns a positive relevancy: this impulse will never provide us with behavior to run. On the other hand, this is an arbitrary, lightweight piece of code that can itself be run at very specific point in the behavic What might we use this code for? Anything. Perhaps we could make a data-logging call, to record the fact that we reach that point in the Or perhaps we wish to spew some debugging information to the console. Or perhaps we wish the character to make a certain sound whe certain condition is met. The fact is, that the code does not need to be explicitly part of a behavior to do something useful. Might this be hack? In some cases, yes, since we are specifically bypassing the behavior performance step (which says that a behavior can only do real w is officially chosen), but in fact this is one of the design purposes of the impulse construct: to give us a convenient way to place arbitrary p code at specific points in the behavior DAG.

Design Principle #3: Hackability is key

Hacks are going to happen. When they do, we must make sure we have a way of containing them. This system also imposes a healthy disc our hacks, since one is required to label and, in the case of the *Halo 2* codebase, list them in a global list of impulses, thus making it very u we will lose track of them and forget that they are there.

Behavior Tagging

As trees grow to large sizes, we can easily imagine that determining behavior relevancy would become one of the principle contributors to After all, we are often checking the relevancy of numerous behaviors and impulses that are not actually running. Often, however, we find t the basic relevancy conditions are the same across many candidates. For example, in *Halo 2*, vehicle status (is the actor a driver, a passeng foot) and alertness status (is the AI in visual contact with a target, simply aware of a target, or unaware of any targets) are practically alway when determining relevancy.

The idea of behavior tagging is to move these common conditions out of the relevancy function (thereby avoiding having to write the sar and over again) and encode them in a tag for the behavior, which is checked directly at decision-time. In *Halo 2*, these conditions are enc bitvector, which is then simply compared with another bitvector representing the AI's current actual state. Behaviors and impulses whose are satisfied undergo the full check for relevancy. The others are ignored entirely.

While this can be considered simply a way to speed up the relevancy check, there is another interesting interpretation. We can see these c locking and unlocking large portions of the behavior tree, thus modifying its fundamental structure. For a passenger of a vehicle, for exam unlocked portions of the tree are very limited: major branches controlling fleeing, self-preservation and searching, for example, are unava vehicle driver has much more available to it, but still not as much as an infantry AI. If we were to look closely at the engage behavior we w something else: that the fighting behaviors of a driver and an infantry unit are different, the infantry unit using the *fight_behavior* and the the more specialized *vehicle_fight_behavior* (the latter keeps the AI moving around constantly, whereas the former tends to pick points ar there). Similarly the process of searching is very different for a driver versus an infantry unit, mostly for the presence in the case of the latt number of coordination behaviors that make searching a group activity.

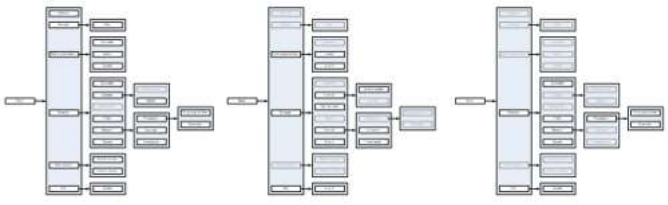

Figure 3: Allowable behaviors for infantry, drivers, and passengers

This is the first of several techniques we will present that affects the decision-making process through direct modification of the structure itself.

Stimulus behaviors

Here is another redundancy concern: imagine a "flee when leader dies" impulse. This impulse essentially waits until an "actor died" event o it springs into action, testing whether the actor that died was a leader, whether there are other leader actors in the vicinity, etc. If all its co satisfied, it triggers a flee behavior. The problem is that given the architecture we've described, this impulse would need to be tested ever We would like to avoid the need to evaluate this impulse continually when we KNOW that no "actor died" event has occurred. We would sense, to make this impulse "event-driven".

One way we might consider doing this is through a **stimulus behavior**. This is a behavior or impulse that does not appear in the static tre but is instead dynamically added by an event-handler to a specific point in the tree. In the given example, the actor would receive an "act event asynchronously with its main update loop (in *Halo 2* these sorts of event notifications happen through a callback). Assuming it is th determined that the actor that died was of a leader class, this causes a *flee_because_leader_died* stimulus impulse to be added to the beh the receiving actor. This means that for a given a period of time (one or two seconds in *Halo 2*), that impulse will be considered for execut with all the other static behaviors and impulses.

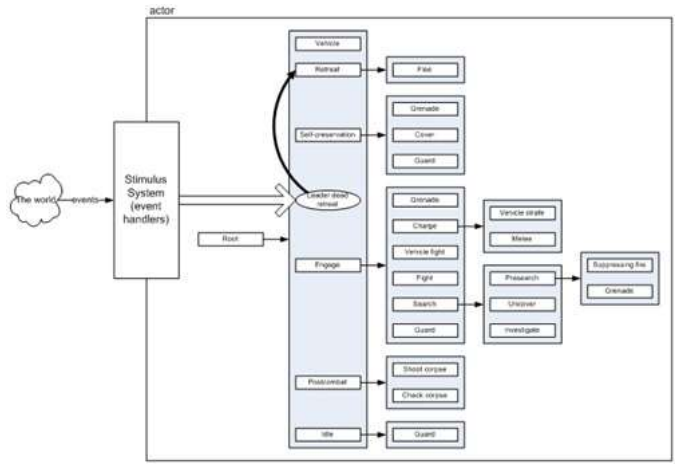

Figure 4: a dead-ally event is turned into a "leader dead retreat" impulse which is dynamically added to the root child list. This impulse will cause behavior, interrupting *engage*, *postcombat* and *idle* but not self-preservation or retreat itself (if the AI is already running it).

Why is it important that the impulse be placed into the actual behavior tree? After all, we could simply force the actor to start running a b based on some decision local to the event-handling code. We don't do this, because it would not, in a sense, be a well-thought-out decis was made in the context of the full tree. In the above example, we would not want to consider fleeing if we were already running *enter_pi* but could if we were simply running *engage*. It would be ludicrous, not to mention highly unscalable, to test these conditions in the event Only by placing the stimulus behavior into the tree itself can we be assured that all the higher-level and higher-priority behaviors have ha before the stimulus behavior can consider taking action.

This is an important point, because it underlines the fact that tree-placement constitutes as large a part of the decision process for a beha impulse as does its relevancy function. Note also that nothing prevents the stimulus behavior from being a non-leaf behavior, thereby allo addition of entire behavior subtrees in an event-driven way. Thus we are again finding a way to modify the structure of the behavior tree get the precise set of behaviors considered.

Halo-博客设置存储的实现 阅读数 461
在开发Halo的时候，有很多设置需要保存在数据库里，比如站点标题，关键字等等。... 博文 来自： RYAN0U...

hexo 添加标签 阅读数 14
---title:title#文章標題date:2016-06-0123:47:44#文章生成時間categories:"... 博文 来自： weixin_3...

Halo博客搭建 阅读数 1996
Halo项目地址：https://github.com/ruibaby/halo最近重新搭建了一下博客系统，使... 博文 来自： juemuren...

光晕2XP系统运行补丁 06-06
光晕2XP系统中文版完美破解补丁，使用补丁可用在XP下完美运行光晕2 下载

什么是Cyclomatic Complexity（圈复杂度） 阅读数 62
圈复杂度(CyclomaticComplexity)是一种代码复杂度的衡量标准。它可以用来衡量一... 博文 来自： sui102的...

Handling Complexity in the Halo 2 AI - wolf96的专栏 - CSDN博客
Handling Complexity in the Halo 2 AI2015年09月27日 06:10:45 wolf96 阅读数 836 原文:http://www.gamas...

HFSM,多层次状态机 - ceciyang的专栏 - CSDN博客
Handling Complexity in the Halo 2 AI 09-27 阅读数 818 原文:http://...博文 来自: wolf96的专栏 Halo,Halo 2...

分布式访问框架halo-dal设计思想及原理 阅读数 307
halo-dal原帖地址http://www.iteye.com/topic/1123284近期刚刚完成halo-dal的... 博文 来自： ak478288

Handling Complexity in the Halo 2 AI - wolf96的专栏 - CSDN博客
转Handling Complexity in the Halo 2 AI 2015年09月27日 06:10:45 wolf96阅读数:747更多个人分类: AI 原文:...

Halo,Halo 2游戏的人工智能设计讲座 - 踏入神的领域 - CSDN博客
Handling Complexity in the Halo 2 AI 09-27 阅读数 837 原文:http://...博文 来自: wolf96的专栏 HFSM,多层次...

平板/笔记本亮度调节工具halo(WINDOWS) 阅读数 1545
笔记本一般都有亮度调节功能,使用特定的Fn功能键或者其他键.而桌面PC只能使用显... 博文 来自： 精彩的生...

个人服务器docker使用管理 阅读数 15
2019独角兽企业重金招聘Python工程师标准>>>... 博文 来自： weixin_3...

Halo is coming。 - CSDN博客
Handling Complexity in the Halo 2 AI 原文:http://www.gamasutra.com/view...wolf96 2015-09-27 06:10:4...

AI资源 - wolf96的专栏 - CSDN博客
2015年09月14日 11:04:51 wolf96 阅读数:1473 from http://www.red3d....AIin action games: complete inter...

Spring Boot 阅读数 227
静态资源外部配置@ConfigurationpublicclassMyWebAppConfigurerextendsWeb... 博文 来自： zishinan...

Convex Optimization - Algorithms and Complexity 02-26
Convex Optimization - Algorithms and Complexity Sébastien Bubeck Theory Group, Microsoft Research 下载

面试笔试杂项积累-leetcode 41-45 - wolf96的专栏 - CSDN博客

OCLint的部分规则（Size 部分） 阅读数 1172
OCLint的部分规则（Size部分）对OCLint的部分规则进行简单翻译解释，有部分进行... 博文 来自： fool宋的...

如何去除set、get方法，@Data注解的使用 阅读数 912
LombokLombok能以简单的注解形式来简化java代码，提高开发人员的开发效率。例... 博文 来自： 水越帆的...

简单了解空间复杂度（Space complexity） 阅读数 364
简单了解空间复杂度（Spacecomplexity）空间复杂度（Spacecomplexity）=指令... 博文 来自： smallerxu...

《复杂性》+《Think Complexity》 11-16

两本关于大数据的好书：《复杂性：一种哲学概观》+《Think Complexity》 下载

Systems Thinking：Managing Chaos and Complexity(ed3) 09-05

Systems Thinking：Managing Chaos and Complexity(ed3) 下载

Halo v1.0 正式版发布，一款惊艳的动态博客系统 阅读数 9

前言Halo从去年5月开源以来，广受小伙伴们的喜爱，在此非常感谢使用Halo发表博... 博文 来自： weixin_3...

combinatorial optimization-algorithms and complexity.pdf 03-06

Clearly written graduate-level text considers the Soviet ellipsoid algorithm for linear programming; efficient algorith... 下载

软件设计度量工具structure101学习(四)：complexity的使用与... 阅读数 1786

度量一个函数(方法)是否复杂，最常用的方法就是计算函数的圈复杂度cyclomaticcom... 博文 来自： aty

Unity 自带光晕组件学习 阅读数 3642

今天把vikingvillage拿出来看了下，毕竟是3年前的工程，效果确实赶不上现在的《死... 博文 来自： Deveupe...

利用重构降低圈复杂度（Cyclomatic Complexity） 阅读数 85

什么是CyclomaticComplexity，以及其计算方法，这里不做讨论。圈复杂度可以用P... 博文 来自： banner

知识点6：计算的复杂度(computational complexity) 阅读数 56

Ifwehaveareallyefficientalgorithm,wecanminimizetheamountofresourceswehav... 博文 来自： weixin_3...

Arithmetic complexity of computations 11-18

Winograd记录了用于为m和r的任何选择生成最小滤波算法F（m，r）的技术。该构造使用中国剩余定理生成线性卷积的最小化算法，该算法等... 下载

手机游戏破解工具HALO 2.0中文版 12-17

简介：【说明】本工具运行需要java运行环境支持,请首先到 http://www.java.com/zh_CN/ 去下载JDK安装 进入H A L O 打开一个需要破解... 下载

Domain-Driven Design: Tackling Complexity in the Heart of Software 10-28

《Domain-Driven Design: Tackling Complexity in the Heart of Software》，中文名《领域驱动设计：软件核心复杂性应对之道》，该资源... 下载

自动汉化破解工具Halo V2.0 02-21

可以破解jar软件的收费功能，电脑里要先装有java软件才可以运行，解压后运行Halo.exe 下载

如何安装MySQL，MySQL两种安装方式 阅读数 2

MySQL是一个关系型数据库管理系统，由瑞典MySQLAB公司开发，目前属于Oracle... 博文 来自： weixin_3...

RxJava 学习笔记（九） --- Error Handling 错误处理操作 阅读数 1576

onErrorReturn指示Observable在遇到错误时发射一个特定的数据onErrorResumeN... 博文 来自： ZekingLee

Python异常 阅读数 1万+

异常的处理异常是指程序中的例外，违例情况；在Python3中，BaseException是所以... 博文 来自： .|Zihuata...

github学习 阅读数 63

注册github都好久了，最近才利用起来，上传了自己测试的一些小demo，下载了几... 博文 来自： sinaihalo...

Computational Complexity：A Modern Approach 09-16

《Computational Complexity: A Modern Approach》是一部将所有有关复杂度知识理论集于一体的教程。将最新进展和经典结果结合起来，... 下载

complexity of lattice problems--A Cryptographic Perspective 09-03

作者: Daniele Micciancio 和 Shafi Goldwasser. 是一本标准的参考文献 下载

HALO修改器，破解jar手机游戏 10-07

破解工具HALO，该包分三个部分，随便解压一个即可打开。（HALO需要电脑安装java运行平台才能打开） 下载

数据库问题记录 阅读数 73

目录1.OperationnotallowedafterResultSetclosed1.OperationnotallowedafterRes... 博文 来自： HaloTrrig...

mybatis结果映射遇到的问题 阅读数 570

错误如下：org.apache.ibatis.exceptions.PersistenceException:###Errorquerying... 博文 来自： weixin_4...

HALO2中文语音 07-02

HALO2中文语音BT种子，XP系统需要下载破解补丁 下载

iQ4bis Business Intelligence 05-09

iQ4bis BI为美国领先BI供应商（其产品名称现已改名为Halo Source /Halo Prism ） 下载

quartz定时任务创建成功但无法生成实例 阅读数 458

quartz在oracle中无法生成实例描述：使用quartz完成定时任务，连接oracle库时可... 博文 来自： kane040...

DDD实战进阶第一波(三)：开发一般业务的大健康行业直销系统（... 阅读数 155

DDD实战进阶第一波(三)：开发一般业务的大健康行业直销系统（搭建支持DDD的轻... 博文 来自： malaoko...

内存MCE错误导致暴力扩充messages日志 以及chattr记录 阅读数 1514

由于放假，好久没登过服务器，今天登上服务器查看日志意外发现：/var/log/messa... 博文 来自： Webber's...

精确运算避免使用float和double 阅读数 169

1.概述float和double类型的主要设计目的是为了科学计算和工程计算。它们执行二进... 博文 来自： iteye_113...

【技术综述】深度学习自动构图研究报告 阅读数 105

文章首发于微信公众号《有三AI》【技术综述】深度学习自动构图研究报告今天带来... 博文 来自： hacker_lo...

内存越界问题 阅读数 1125

一、背景最近手上的项目出现一个如下的BUG，在网上查了查原来是内存越界的问题... 博文 来自： light_in_d...

FANUCLR Handing Tool操作说明书 03-12

LR Handing Tool操作说明书 下载

mybatis pagehelper分页查询时候报错 阅读数 3375

今天再做开发的时候遇到了mybatisPagehelper 分页查询的错误。Theerroroccurred... 博文 来自： wittdong...

北大AI讲座公开课-精华 阅读数 2318

最近在观看北大AI讲座公开课，觉得受益良多，稍做笔记。观看地址如下：https://c.... 博文 来自： CrazyBull...

机器学习教程 Objective-C培训 交互设计视频教程 颜色模型 设计制作学习

mysql关联查询两次本表 native底部 react extjs glyph 图标 ai技术培训 ai培训

没有更多推荐了，返回首页

wolf96

博客专家

关注

原创	粉丝	喜欢	评论
229	858	480	238

等级： 博客 访问： 106万+

积分： 1万+ 排名： 1845

勋章：

TA的个人主页 >

友情链接

我的Github

欢迎Follow、Fork、Star
我的个人博客

aras的博客
Gamasutra
计算机视觉论坛
凯奥斯的知乎文章

最新文章

ARM Unity开发者图形优化指南总结
用Projector来优化阴影
Volumetric Cloudscapes(一):理论
开启stripping遇到的问题
Planer Reflection

博主专栏

unity3d shader实战练习
文章数：74 篇　访问量：33万+

游戏中人工智能的学习与应用
文章数：6 篇　访问量：2万+


游戏开发设计模式
文章数：5 篇　访问量：2万+

游戏开发中的相机技术
文章数：2 篇　访问量：4283

虚幻4引擎源码学习笔记
文章数：2 篇　访问量：776

个人分类

Unity3D	163篇
Unity Client Programming	7篇
OpenGL	4篇
Graphic Tips	57篇
Shader	130篇

展开

归档

2019年6月	1篇
2019年4月	2篇
2019年2月	3篇
2019年1月	1篇
2018年12月	5篇

展开

热门文章

世界传说 换装迷宫2 所有人物及所有技能及奖励技能 && 传说系列各秘奥技和台词
阅读数 31419

Unity3d HDR和Bloom效果（高动态范围图像和泛光）
阅读数 31347