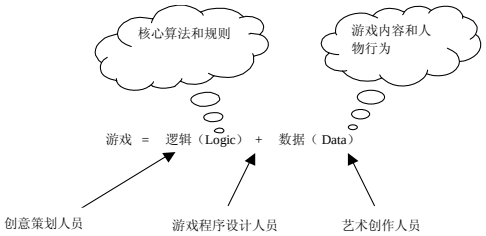
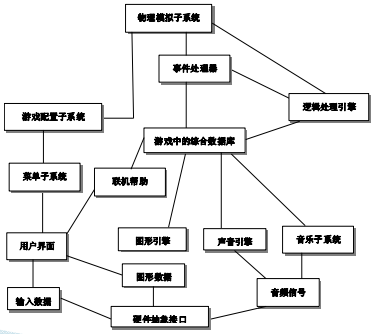
《计算机游戏程序设计》
游戏程序设计概述

关于“游戏”

- 游戏只是某些“逻辑”与“数据”的结合体

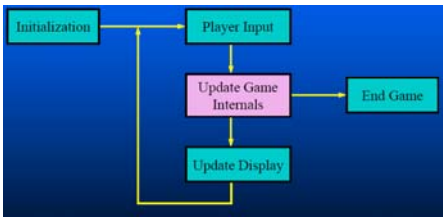


游戏的基本组成



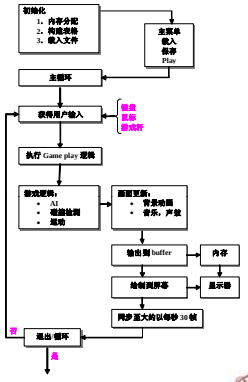
游戏循环

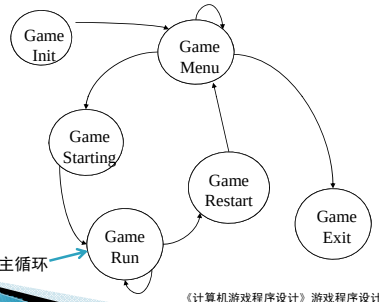
- 所有游戏都采用这个类似的过程



更详细的过程

- **游戏其实就是一个不断按某种逻辑更新各种数据（画面、声音等）的过程。**
  - 游戏的基本流程只是一个连续的循环，它不断地按某种逻辑来绘制新的图像，并刷新画面
  - 游戏可被类比为一个带有前置终端的实时数据库，该终端实时地接受用户（玩家）输入的各种交互指令，取出相应的数据，并“优雅”地将这些数据以各种形式（视觉、听觉等）展现给用户。





一个例子

- 看一个特定的游戏，包括很多状态


```
// defines for game loop states
#define GAME_INIT          1    // the game is initializing
#define GAME_MENU          2    // the game is in the menu
#define GAME_STARTING      3    // the game is about to run
#define GAME_RUN           4    // the game is now running
#define GAME_RESTART       5    // the game is going to restart
#define GAME_EXIT          6    // the game is exiting
// game globals
int game_state  = GAME_INIT;    // start off in this state
Int error       = 0;            // used to send errors back to OS

// main begins here
Void main( )
{       // implementation of main game loop
```



```
While (game_state != GAME_EXIT)
{       // implementation of main game loop
        switch(game_state)
        {
                case GAME_INIT:         // the game is initializing
                {       // allocate all memory and resources
                        Init( );
                        game_state = GAME_MENU;
                } break;
```

```
case GAME_MENU:     // the game is in the menu
{       // call the main menu function and let it switch states
        game_state = Menu( );
        // note: we could force a RUN state here
} break;
case GAME_STARTING:     // the game is about to run
{       // this state is optional, but usually used to set things up right
        // before the game is run you might do a little more housekeeping
        Setup_For_Run( );
        // switch to run state
        game_state = GAME_RUN;
} break;
```


```
case GAME_RUN:      // the game is now running
{       // this section contains the entire game logic loop
        Clear( );       // clear the display
        Get_Input( );   // get the input
        Do_Logic( );    // perform logic and AI
        Render_Frame( ); // display the next frame of animation
        Wait( );        // synchronize the display
        // the only way that state can be changed is thru user interaction
        // in the input section or by maybe losing the game.
} break;
```


```
case GAME_RUN:      // the game is now running
{       // this section contains the entire game logic loop
        Clear( );       // clear the display
        Get_Input( );   // get the input
        Do_Logic( );    // perform logic and AI
        Render_Frame( ); // display the next frame of animation
        Wait( );        // synchronize the display
        // the only way that state can be changed is thru user interaction
        // in the input section or by maybe losing the game.
} break;
```


```
case GAME_RESTART: // the game is restarting
{       // this section is a cleanup state used to fix up any loose ends
        // before running again
        Fixup( );
        // switch states back to the menu
        game_state = GAME_MENU;
} break;
```


游戏开发的基本理念

- 数据驱动的理念
  - “逻辑”方面是一款游戏的灵魂，主要由游戏设计人员来负责完成；
  - “数据”方面主要起到描述性和修饰性的作用，主要由程序设计人员来处理。
    - 这样的主次关系决定了只有高效地、灵活地处理和对付这些“数据”部分，才能让游戏设计人员把更多的时间和精力花费在“逻辑”部分，
  - 游戏编程人员必须把这些“数据处理”工作变得简便和快捷，才能保证游戏开发的成功。

数据驱动的策略

- 游戏程序设计人员在编写代码时，要采用可以很容易修改游戏数据的方式，保证游戏数据的动态可调整性
- 把游戏程序中的一些常数放置在文本文件中，这样，不需要重新编译就可以很容易地对这些参数进行修改
- 在编码实现中，时刻牢记把“逻辑”和“数据”分开的核心准则，所有的数据都不放在代码内
- 使用脚本来控制游戏的流程
- 在大型的复杂游戏开发中，需要另外的编辑工具代替手工编辑方式，方便于产生游戏数据的文本文件，如关卡编辑器、脚本编辑器等
- 避免重复的数据

游戏开发的原型法

- 在投入大量的人力，物力之前，在限定的时间内，用最经济的方法开发出一个可实际运行的系统模型，测试人员在运行使用整个原型的基础上，通过对其评价，提出该进意见对原型进行修改，统一使用，评价过程反复进行，使原型逐步完善，直到完全满足需求为止
  - 可玩性测试的原型系统：专门用于检验游戏设计的合理性和用户的可接受程度；
  - 用户界面的原型系统：检查玩家如何与游戏进行交互；
  - 各个子系统的原型系统：测试该子系统的功能，并可检查各个子系统的交互关系和数据接口；
  - 算法测试的原型系统：可用来检查各种算法。尤其是特定领域的一些复杂算法需要通过原型系统进行不断地改进和提高。

游戏的调试与测试

- 游戏的调试是编程人员的任务，在这个阶段中要找出游戏程序的内在缺陷，并加以修正
- 游戏BUG的出现可能是由于程序员的技术问题，也可能是由于策划的设计问题，或者是因为美工的一时疏忽
- 参与游戏测试的人员则包括游戏设计、开发以及玩游戏的人员
- 游戏在整体上完成后，在进入全面测试阶段的时候，就可以进行游戏参数的调整

游戏开发的基本准则

- 所有的游戏开发都必须为今后的重用作好准备。
  - 尽量使用可重用的模块将会最大程度上缩短游戏的开发周期。
- 开发文档不是可有可无的。
  - 好的开发文档不仅能为软件重用提供有力的技术保证，而且在游戏开发过程中，它能让其他的开发人员清楚地知道你在干什么。
- 先设计，后编程。
  - 游戏的设计和编程实现是明显分开的，因此，游戏的开发一定要先进行设计，后进行编程实现。
  - 游戏开发是一个不断进行自我完善的过程，在开始编程的时候，大概只完成了80%的游戏设计工作量，其余的设计工作随着开发进程的推进而逐步地进行精致和完善。
- 灵活有效地安排开发进程。
  - 在游戏开发过程中，一定要让每一个游戏编程人员知道他们的开发目标。虽然执著地实施既定的开发计划很重要，但是，更重要的是如何在开发进程落后的时候，重新校准新的开发计划。
- 及时地发现错误。
  - 在开发过程中，及时地发现和纠正错误十分重要。时间越长，这些潜在的错误就越容易“发酵腐烂”，最终导致开发的失败。

游戏开发的辅助工具

- 声音数字转换器
- 音乐编辑器、编曲机
- 图形工具：Photoshop等
- 摄像头
- 视频采集卡
- 图形库：包括建模和绘制
- 游戏引擎
- 。。。。。。

游戏开发人员的基本素质要求

- 软件设计的基本技能
  - 丰富的知识面
    - 软件工程、数据结构、数据库、算法设计等
    - 2D/3D图形学、人工智能、音频/视频处理 、人机交互、计算机网络
    - 坚实的数学和物理基础：线性代数、欧氏几何、牛顿物理学
  - 经验和能力
    - 综合运用各片断技术的经验
    - 不断学习新技术的能力
    - 游戏的开发常常受限于运行时间或者存储空间等资源因素，但却处处追求高效率和高性能。
- 其他
  - 游戏引擎，如DirectX，OpenGL, 及其他3D游戏引擎
  - 一些数据处理和建模工具，如Photoshop,Maya,3DS Max等
  - 。。。。。。

游戏引擎

- 游戏引擎技术的出现是游戏程序设计技术发展的里程碑之一，并已成为当前计算机游戏开发的关键技术和核心平台
- 游戏编程人员就不需要从头做起，能够高质量地在很短的周期内开发出新游戏
- 游戏引擎相当于游戏的底层框架平台，如果把游戏引擎比拟为一个“游戏操作系统”，那么最终的游戏产品则可比拟为一个个具体地运行在“游戏操作系统”上的应用程序
- 游戏引擎已经发展为一套由多个子系统共同构成的复杂系统

常用引擎简介

- 商用游戏引擎
  - Doom
  - Quake
  - Unreal
  - Unity
  - Virtools

- 开源游戏引擎
  - SDL（2D）
  - Allegro（2D）
  - Ogre3D
  - Horde3D
  - Crystal Space
  - Irrlitch
  - Panda3D（Python）
  - Delta3D

游戏的娱乐效果

- 游戏可玩性 – 挑战与动作
- 美学 – 风格协调性
- 讲故事 – 交互式故事叙述
- 风险与回报 – 失败与胜利
- 新奇 – 多样
- 学习 – 严肃游戏
- 自我表现型玩法 – 自主选择
- 沉浸 – 图形、战术、策略、叙述
- 社会化 – 联网操作

用户体验
可玩性设计

以玩家为中心的界面设计

- 一致性
- 提供好的反馈
- 记住玩家是发号施令的人
- 限制采取一个动作所需要的步骤的数目
- 允许简单的动作撤销
- 最小化身体压力
- 不要损害玩家的短期记忆
- 对屏幕上的反馈机制进行分组
- 为有经验的玩家提供快捷键

交互模型

- 基于化身的模型
  - 玩家的动作大部分由控制游戏中的一个角色组成
- 多处存在模型
  - 给玩家一个允许他看到各种区域的视角，并可以改变该视角
- 基于团队的模型
  - 一小组角色通常保持在一起作为一个组
- 竞赛模型
  - 玩家回答问题并作出决定，不需要导航
- 桌面模型
  - 模拟一个计算机（或实际的）桌面

视角

- 第一人称视角
- 第三人称视角
- 空中视角
  - 上下视角 – 顶视图
  - 等距视角 – 斜45度
  - 自由徘徊视角 – 允许玩家更多地控制摄像机
  - 上下文敏感视角 – 智能摄像机
- 2D显示
  - 固定单屏幕
  - 横向滚轴
  - 上下滚轴
  - 绘制好的背景

游戏可玩性(gameplay)

- 一系列有趣的选择
- 一系列有趣的交互
- 幻想是可玩性的重要要素
- 玩家为达到游戏目标所必须面对的**挑战**
- 允许玩家采用以应对挑战的**动作**

挑战(challenge)

- 身体协调性挑战 – 驾驶、设计、反应、节奏等
- 模式识别挑战 – 布局、行为模式等
- 时间压力 – 限时、最快完成任务
- 记忆和知识挑战 – 细节、物体或模式的记忆
- 探索挑战 – 空间关系、寻找钥匙、隐秘的通道
- 冲突– 战略、后勤、生存、消灭敌人
- 经济挑战 – 积累资源、建立高效的生产系统
- 概念性推理 – 推理犯罪、理解社会关系、角色动机
- 创造/构建挑战 – 美学要求、有功能目标的建设

动作(action)

- 直接由用户界面解释的玩家输入事件，用于克服挑战并完成游戏任务
- 交互模型很大程度上决定了该模式中可用的动作类型
- 不要期望动作和挑战之间存在一一映射，应当控制动作的数目
- 定义动作时需要考虑：
  - 玩家在游戏中的角色
  - 为可玩性模式设计的挑战
  - 与可玩性不相关的动作

游戏设计分析 – 动作游戏

- 动作游戏中，大多数挑战都是测试玩家的物理技能。但它经常也含有谜题解决、战术冲突和探索挑战。
- 前进：关卡、检测点
- 挑战：障碍和危险、敌人、大boss、锁住的门和钥匙、怪物生成器
- 玩家动作：设计、选择、收集、击打、踢、跳跃、防守等；快速炸弹（气功）、超空间逃离
- 核心机制：生命、能量、收集品、时间限制、分值
- 交互模型：化身
- 用户界面：HUD、图形指示符、微型地图、识别角色、动作控制

游戏设计分析 – 体育游戏

- 模拟真实或虚拟体育运动的某些方面，包括比赛的进程、团队管理和职业管理
- 玩家角色：运动员、教练、经理
- 可玩性：真实运动里的；规则：体育运动规则
- 核心机制：物理、运动员评估、运动员AI、受伤、自动模拟比赛（生成比赛过程、比分、主场优势）、音频解说
- 交互模型：1对1、团队比赛
- 视角：不应选择第一人称

严肃游戏

- 广告游戏 – 产品宣传
- 学习游戏 – 教育目的
- 仿真游戏 – 技能训练
- 新闻游戏 – 事件或社论
- 公共政策游戏– 危机事件（饥饿、恐怖主义）
- 军事游戏 – 军事训练
- 说服游戏 – 说服游戏者
- 艺术游戏 – 美学、艺术观点

严肃游戏的例子

Tactical Language & Culture Training System


MS Flight Simulator


Genomics Digital Lab


Peace Maker


eSmart


Virtual Iraq


WiiRehablitation

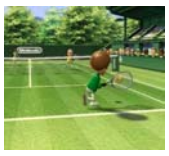
严肃游戏相关资料

- http://en.wikipedia.org/wiki/Serious_game
- http://seriousgamesmarket.blogspot.com/
- http://www.seriousgames.dk/node/511
- http://www.seriousgamessource.com/news.php
- http://www.seriousgames.org/
- http://www.seriousgamesinstitute.co.uk/
- http://social.education.purdue.edu/game/

体感游戏

- 用身体去感受的电子游戏。突破以往单纯以手柄按键输入的操作方式，体感游戏是一种通过肢体动作变化来进行（操作）的新型电子游戏。代表平台有Wii，iPhone，PlayStation Move等
- 在非PC平台上，专业电子游戏机以手柄或操作台的形式进行游戏内容。随着科技进步以及增强玩家游戏体验的需要，相关的游戏公司开发出了专业化，特异化的游戏输入设备以及大型街机游戏项目
- 体感游戏的发展是由电子游戏输入设备地进步展开的，而街机游戏可以说是体感游戏最大的贡献者

体感游戏的例子


Wii Sports


X-Box Joy Ride

iPhone AR Soccer


PS Motion Fighter


跳舞机


ZJU VNM