**中国矿业大学计算机学院**

**2014 级本科生课程报告**

课程名称 游戏设计与开发

报告时间 2017年4月27日

学生姓名 赵勤博

学 号 08143427

专 业 计算机科学与技术

任课教师 张辰

任课教师评语

任课教师评语（①对课程基础理论的掌握；②对课程知识应用能力的评价；③对课程报告相关实验、作品、软件等成果的评价；④课程学习态度和上课纪律；⑤课程成果和报告工作量；⑥总体评价和成绩；⑦存在问题等）：

成 绩： 任课教师签字：

年 月 日

摘 要

随着智能手机终端的普及，手机游戏正走向新的发展热潮，成为人们娱乐生活中不可或缺的一部分其中，酷跑类手游以其益智和轻量级的特色不断吸引着玩家的眼球，成为如今手游市场占有率最高的游戏类型之一。同时，在游戏开发方面，Unity游戏引擎以其便捷的开发模式和完善的生态圈异军突起，逐渐成为手游开发首选。

本文采用Unity引擎开发技术，因为玩家对游戏的整体视听体验及流畅度体验很大程度影响了游戏的成败，故本文以软件工程思想为主线，首先通过用例图分析系统需求，其次用时序图明确各系统模块在实现过程及功能逻辑方法调用关系。游戏控制模块和敌人模块采用有限状态自动机编写。

由于人工智能应用于2D游戏的教程较少，本文的贡献在于实现了人工智能在2D游戏的应用，如靠近算法、追逐算法、避障算法。

**关键字：** 游戏开发；Unity；有限自动状态机；人工智能

ABSTRACT

As high performance smartphone terminal is popularized, mobile games are movingtowards new development upsurge, and have become an indispensable part of entertainment life of people. Among them, mobile Parkour games, characterized by intelligence benefit and lightweight class, constantly catch player’s eyes, and have become one of game types with highest occupancy in current mobile games market. Meanwhile, as game engine indispensable in development of games, Unity, based on its convenient development mode and integrated ecosphere, suddenly rose as a new force, and has gradually becomes the first choice for mobile games development.

In this thesis, the game is based on Unity engine development technology. The success or failure of a game is depended on auditory and visual experience, levels experience and fluency experience of whole game for players basically. This article takes the software engineering idea as the main line by analyzing the system requirements by use case diagram and using the sequence chart to clarify the relationship between the system modules in the implementation process and the functional logic method. The game control module and the enemy module are written in a finite state machine.

Since the application of artificial intelligence to 2D games is less, the contribution of this thesis is to achieve the application of artificial intelligence in 2D games, such as near algorithm, chase algorithm, obstacle avoidance algorithm.

**KEYWORDS:** Game Development; Unity; FSM; Artificial Intelligence

目 录

[1 概述 1](#_Toc481077741)

[1.1游戏简介 1](#_Toc481077742)

[1.2游戏概要设计 1](#_Toc481077743)

[1.2.1故事背景 1](#_Toc481077744)

[1.2.2主要角色 1](#_Toc481077745)

[1.2.3角色遇到的问题 1](#_Toc481077746)

[1.2.4游戏类型 1](#_Toc481077747)

[1.2.5目标游戏人群 1](#_Toc481077748)

[1.2.6 收费设定 1](#_Toc481077749)

[1.2.7 玩家的追求点 2](#_Toc481077750)

[1.2.8 游戏的后期开发 2](#_Toc481077751)

[1.2.9 游戏的玩法 2](#_Toc481077752)

[1.2.10 界面设计 2](#_Toc481077753)

[1.3开发平台 2](#_Toc481077754)

[1.3.1 Unity的特色功能 3](#_Toc481077755)

[1.3.2 Unity的ECS开发模式 3](#_Toc481077756)

[1.3.3 Unity的行为事件执行顺序 4](#_Toc481077757)

[2 游戏软件架构 4](#_Toc481077758)

[2.1文件目录架构 4](#_Toc481077759)

[2.2代码组织架构 5](#_Toc481077760)

[2.3 MonoBehaviour生命周期 5](#_Toc481077761)

[2.4碰撞系统 6](#_Toc481077762)

[2.5粒子系统 7](#_Toc481077763)

[2.6 刚体 8](#_Toc481077764)

[2.7 音频系统 8](#_Toc481077765)

[2.7.1音频管理器 8](#_Toc481077766)

[2.7.2 音频监听器 9](#_Toc481077767)

[2.7.3 音频源 9](#_Toc481077768)

[3 系统建模 9](#_Toc481077769)

[3.1交互模型概述 9](#_Toc481077770)

[3.2用例建模 9](#_Toc481077771)

[3.3时序图 11](#_Toc481077772)

[3.3.1开始游戏系统时序图 11](#_Toc481077773)

[3.3.2获得奖励物系统时序图 11](#_Toc481077774)

[3.3.3敌人追逐系统时序图 12](#_Toc481077775)

[3.4结构模型概述 13](#_Toc481077776)

[3.5 类图 13](#_Toc481077777)

[3.6行为模型概述 15](#_Toc481077778)

[3.7状态图 15](#_Toc481077779)

[4 游戏中的人工智能 17](#_Toc481077780)

[4.1人工智能概述 17](#_Toc481077781)

[4.2控制行为编程的主要基类 17](#_Toc481077782)

[4.2.1概述 17](#_Toc481077783)

[4.2.2将AI角色抽象成一个质点 18](#_Toc481077784)

[4.2.3控制AI角色移动 18](#_Toc481077785)

[4.2.4各种操控行为的基类 18](#_Toc481077786)

[4.2.5靠近行为模型 18](#_Toc481077787)

[4.2.6追逐行为模型 19](#_Toc481077788)

[4.2.7避障模型 20](#_Toc481077789)

[4.2.8算法类图 21](#_Toc481077790)

[4.2.9避障模型时序图 21](#_Toc481077791)

[4.2.10靠近模型时序图 22](#_Toc481077792)

[4.2.11 追逐模型时序图 23](#_Toc481077793)

[4.3有限状态机 24](#_Toc481077794)

[参考文献 26](#_Toc481077795)

# 

# 1 概述

## 1.1游戏简介

《Scream》是一款安卓端单人休闲跑酷类游戏，玩家通过操纵按键控制角色奔跑、跳跃来躲避障碍物和获取奖励物品，玩家通过操作躲避敌人和障碍到达最终的目的地以获取游戏的胜利。

## 1.2游戏概要设计

### 1.2.1故事背景

公元4000年地球资源枯竭，主人公在这个危机的时刻勇敢地站了出来，去未知的外太阳系探索能量的来源，为了使命，主人公不惧外星生物和太空陨石的威胁，义无反顾地踏上了征途。

### 1.2.2主要角色

玩家控制身怀绝技的忍者。

### 1.2.3角色遇到的问题

游戏场景中，角色需要躲避敌人的追逐或打败敌人。角色在运动的过程中要避免落入陷阱。

### 1.2.4游戏类型

《Scream》游戏本质上属于休闲类游戏，借鉴并结合了流行的奖励机制和敌人追逐系统，游戏简单有趣而又给玩家带来一种适度的紧张感。游戏操纵性与策略性并重，只有达到两者的平衡才能获得较高的分数。

### 1.2.5目标游戏人群

《Scream》适合青中年玩家参与，特别适合没有时间尝试长时间游戏的玩家，通过碎片化的游戏时间，减缓生活学习压力。

### 1.2.6 收费设定

设计原则：玩家消费后可以拥有更快的速度和更强的技能，能让玩家在游戏中体验更强的好玩度。让玩家的消费能得到实际的好处，感觉“赚了”而不是“亏了”。

消费引导：分级消费+消费前置。

分级消费：对于缺乏消费能力的玩家，如果一开始就看到太多超出自身能力的消费，便会因门槛太高而放弃游戏，但如果收费的额度根据玩家的 VIP 等级来设定，以层层递进的方式引导消费，玩家就会更容易接受。

消费前置：将欲取之必先与之，对于付费带来的好处，可以让玩家先有所体验，以此来促进消费，比如赠与有时限的角色，皮肤等。

### 1.2.7 玩家的追求点

核心追求：玩家对于探险冒险的追求。

长期追求：拓展剧情、通过精英关卡、竞技场名次。

短期追求：更新的地图玩法、获得的成就、通过关卡等级。

### 1.2.8 游戏的后期开发

增加游戏的地图，开发成就系统，新增更多的技能和收集物品，增强剧情的代入。使得游戏不仅有操作性，还有游戏背景的文化。增加游戏的攻击技能，并有较强的破坏感，使游戏环境更逼真刺激。

### 1.2.9 游戏的玩法

玩家控制主角带着星球人民的托付来到陌生的环境开始新的征程，玩家可以控制主角的行走和跳跃，玩家需要躲避陷阱和敌人的追击，并收集能量。最后玩家到达终点并取得胜利。

### 1.2.10 界面设计

总体游戏分为三块，开始游戏界面，游戏中场景，结束游戏界面。

开始游戏界面中，玩家可以选择开始游戏和设置。

游戏场景中，玩家操作界面有向左移动按键，向右移动按键和跳跃按键。游戏右上角显示玩家的生命数和静音选项，左上角显示玩家设置界面，点开设置界面后，玩家可选择设置而段跳或在PC端游戏时隐藏操作按键。

游戏结束界面显示玩家获得的分数和再来一次按钮。

## 1.3开发平台

Unity3D是由Unity Technologies开发的一个让玩家轻松创建诸如三维视频游戏、建筑可视化、实时三维动画等类型互动内容的多平台的综合型游戏开发工具，是一个全面整合的专业游戏引擎。游戏引擎是指一些已编写好的可编辑电脑游戏系统或者一些交互式实时图像应用程序的核心组件。这些系统为游戏设计者提供各种编写游戏所需的各种工具，其目的在于让游戏设计者能容易和快速地做出游戏程式而不用由零开始。大部分都支持多种操作平台，如Linux、Mac OS X、微软Windows。游戏引擎包含以下系统：渲染引擎（即“渲染器”，含二维图像引擎和三维图像引擎）、物理引擎、碰撞检测系统、音效、脚本引擎、电脑动画、人工智能、网络引擎以及场景管理。本节将从Unity引擎的特色功能、开发模式、行为事件执行顺序、内存管理四个方面对Unity引擎进行概要介绍。

### 1.3.1 Unity的特色功能

Unity是一款分层级的、全面的游戏开发引擎，它加入了父子级Parenting的概念，这意味着所有游戏对象GameObject之间存在从属关系，即下一级对象的位置、缩放比例、等三维数据通过上一级对象来确定，这样的结构模式非常利于游戏开发者管理场景中的对象。其次，对于Unity引擎，一个完整的项目工程是由许多个场景Scene组合而成的，并且，引擎的场景编辑功能具有可视化的特点，可通过直观的场景显示来正确、高效地完成场景编辑；其中，场景中还可包含若干GameObject，包括3D和2D物体、光源、音源、界面、粒子系统、摄像机等等，这些对象又可添加各种组件Component，比如行为脚本、物理、网格、效果、渲染、声音、角色控制等组件，实现各式各样的场景需求。

在物理引擎方面， Unity内置了NVIDIAPhysX的物理引擎，其能较逼真的模拟刚体碰撞、车辆驾驶、布料、重力等这些现实生活中的物理现象；与其相关的碰撞检测组件Collider在游戏开发的过程中用途很广，它通过玩家碰撞游戏物体，调用Unity事件监听机制完成事件的传递和响应。

另一方面， Unity在特效，动画处理上也有独到之处。在特效方面，它拥有柔和光影、炽焙感光的光影渲染系统，以及提供能实现许多细丽特效的粒子系统。在动画方面，其Animation系统非常的灵活强大，支持动画融合、混合、叠加，行走循环的时间同步，动画控制的各个方面，并且支持基于物理效果的布娃娃系统和程序动画。此外，为了让开发者更加方便的开发，Unity还提供了Mesh网格和Shader着色组件，使得对模型的网格化和着色处理得到了改善。

最后， Unity中非常有特色的一类资源叫做预设Prefab，它是一种游戏资源，也是一种可被重复使用的游戏对象。所有Prefab的实例都是Prefab的克隆Clone，在实际运行中因Prefab生成的游戏对象都会具有(Clone)标志。因此，如果要在游戏中创建一些需要重复使用的游戏对象，Prefab是开发的首选。

### 1.3.2 Unity的ECS开发模式

所谓的ECS模式全称为Entity-Component-System模式，即实例、组件、系统的开发模式。简言之，实例就是一个游戏对象实体，一个实体拥有众多的组件，而游戏系统则负责依据组件对实例做出更新。它是为了解决传统的游戏设计思路中导致的"类灾难"而被提出的，目的是使得游戏对象需要什么时，就会给自己添加一个相应组件，并且系统会依据游戏对象增加了哪些组件来做出行为响应。换言之，实例只需要持有必要的数据，由系统负责逻辑就行了，运也就是ECS模式能和数据驱动很好结合的一个原因。

本文中设计的各种墙面，敌人Prefab游戏资源都加载了不同的组件，在被克隆成实体时，系统根据其游戏对象中加载的组件实现墙面显示、获得能量球状态等逻辑。

### 1.3.3 Unity的行为事件执行顺序

Unity不支持多线程，也就是所有操作必须要在主线程中进行，但是可同时创建大量行为脚本，并分别作为组件绑定在不同游戏对象身上，弥补单线程的不足。此外，Unity中，脚本都是从MonoBehaviour所派生，实现上层定义的各种接口，如Start/Update等。

基于Unity单线程运行的模式下， Unity后台对脚本的执行都有一定顺序：Awake-OnEnable-Start-FixedUpdate-Update-OnGUI-OnDisable-OnDestroy。Awake函数在脚本整个生命周期它仅被调用一次。并在所有对象被初始化之后调用；Start函数仅在Update函数第一次被调用前调用。其在行为生命周期中只被调用一次。它和Awake的不同在于Start在脚本实例被启用时调用。FixedUpdate函数用于固定帧更新，Update函数用于正常帧更新。OnGUI函数用于渲染和处理GUI事件时调用并且也是每帧执行一次。OnDestroy函数在MonoBehaviour将被销毁时调用，只在预先被激活的游戏物体上被调用。

此外，当游戏对象是否处于active或者脚本是否为enabled时，函数的调用情况又有所不同。当游戏对象为active但是脚本为disabled时，对象被创建后只有Awake函数会被立刻调用，OnEnable和Start会在enabled脚本后调用，OnEnable在gameObject 的active被设置为true和脚本被设置enable时调用，游戏对象为deactive但是脚本为enabled时，当对象被创建时不会有函数被调用，当active物体之后会按照Awake-OnEnable-Start的顺序调用函数。

# 2 游戏软件架构

## 2.1文件目录架构

Assets:主文件夹，包含所有工程需要用到的资源。

3DModels:用于存放网格文件。

Material:用于存放材质和贴图。

Particles:用于存放粒子效果的相关文件。

PhysMaterials:用于存放物理材质文件。

Plugins:用于存放游戏软件所用的插件。

Prefabs:用于存放预设文件，

Resource:用于存放需要通过代码初始化使用的资源，可以放在工程中的任意文件夹中，在Resources文件夹中的文件可以通过Resources.Load 来读取。

Scences:用于存放场景文件。场景包含游戏的对象。

Scirpts:用于存放脚本文件。

Sound:用于存放声音文件。

Sprites:用于存放精灵相关的文件。

UI: 用于存放界面相关的文件。

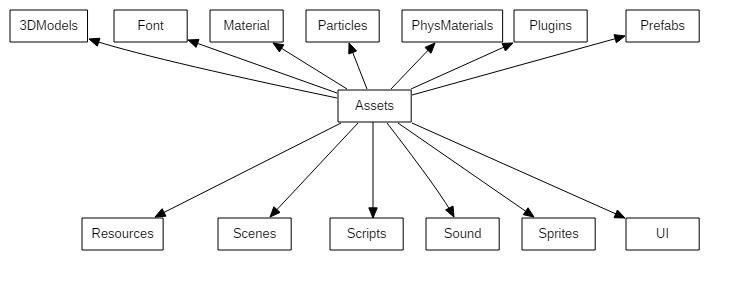


图2.1 文件目录架构

2.2代码组织架构

MVC全名是Model View Controller，是模型(model)－视图(view)－控制器(controller)的缩写，一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。MVC被独特的发展起来用于映射传统的输入、处理和输出功能在一个逻辑的图形化用户界面的结构中。如图

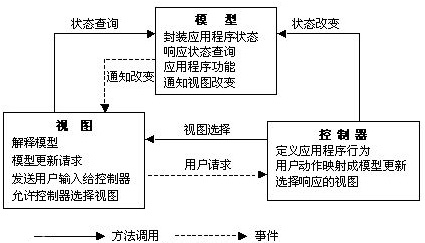


图2.2 MVC组件类型的关系和功能

## 2.3 MonoBehaviour生命周期

脚本自带函数执行顺序如下：将下面脚本挂在任意物体运行即可得到Awake ->OnEable-> Start -> FixedUpdate-> Update -> LateUpdate ->OnGUI ->Reset -> OnDisable ->OnDestroy

Awake用于在游戏开始之前初始化变量或游戏状态。在脚本整个生命周期内它仅被调用一次.Awake在所有对象被初始化之后调用，所以你可以安全的与其他对象对话或用诸如 GameObject.FindWithTag 这样的函数搜索它们。每个游戏物体上的Awke以随机的顺序被调用。因此，你应该用Awake来设置脚本间的引用，并用Start来传递信息Awake总是在Start之前被调用。它不能用来执行协同程序。

Start仅在当第一次脚本启用Update方法被调用之前调用。就像Awake函数，在该脚本的生命周期Start被初始化一次。然而，Awake被调用时在脚本对象初始化时调用，不管脚本是否被启用。在同一帧如果在初始化的时候没有启用脚本，Start不会被调用。在任何对象Start函数被调用之前，Awake函数在场景所有对象被调用。这其实是很有用的，对象A的初始化代码依赖对象B已经被初始化，对象B初始化应该在Awake中，A的代码应该在Start中。凡对象在游戏时实例化，在场景对象的Start的函数已经完成之后，它们的Awake函数自然被调用。

Update：正常帧更新，用于更新逻辑，当MonoBehaviour启用时，其Update在每一帧被调用。每一帧都执行，处理Rigidbody时，需要用FixedUpdate代替Update。例如:给刚体加一个作用力时，你必须应用作用力在FixedUpdate里的固定帧，而不是Update中的帧。(两者帧长不同)FixedUpdate，每固定帧绘制时执行一次，和update不同的是FixedUpdate是渲染帧执行，如果你的渲染效率低下的时候FixedUpdate调用次数就会跟着下降。FixedUpdate比较适用于物理引擎的计算，因为是跟每帧渲染有关。Update就比较适合做控制。

## 2.4碰撞系统

Collider 碰撞器是所有碰撞器的基类。如果一个对象带有碰撞器，需要在游戏中移动，那么你应该添加个Rigidbody组件给它。如果你不想该对象与其他对象有交互，刚体可以设置为运动学刚体（kinematic）。

OnCollisionEnter和OnTriggerEnter2D的比较：

Collider.OnCollisionEnter(Collision)方法，当该collider/rigidbody开始碰到另一个rigidbody/collider时OnCollisionEnter被调用。相比OnTriggerEnter，OnCollisionEnter传递Collision类而不是Collider。Collision类包含接触点，碰撞速度等信息。在这个函数如果你不使用collisionInfo，删去collisionInfo参数以避免不必要的计算。注意，当其中至少一个碰撞器附加非动力学刚体时碰撞事件才会发送。碰撞事件也发送给禁用的MonoBehaviours，允许启用Behaviour来相应碰撞。

Collider2D.OnTriggerEnter2D方法，当另个对象的碰撞器进入到这个对象的触发碰撞器时发送（仅2D物理）。关于碰撞的更多信息在调用过程中通过Collision2D参数报告。

本游戏项目所涉及到的碰撞检测都是进入时的检测，即刚发生碰撞，没有退出和逗留时的碰撞检测，所以另外四个接口函数就不在赘述，添加碰撞检测组件碰撞检测器。

添加方法：

Component->Physics->Collider；Component->Physics->Rigidbody；

Is Trigger属性在勾选状态下，那么该对象就是触发器。下面其它参数则是设置碰撞体的大小、半径。

Mass是质量，Drag是阻力，Angular Drag是角阻力。Use Gravity是否使用重力，如果勾选物体将相对向下做自由落体运动。Is Kinematic是否相对地面静止，如果勾选物体将相对地面静止。Constraints属性块是设置物体的位置和旋转是否静止，勾选Freeze Position Z那么物体在Z轴方向不能移动。

## 2.5粒子系统

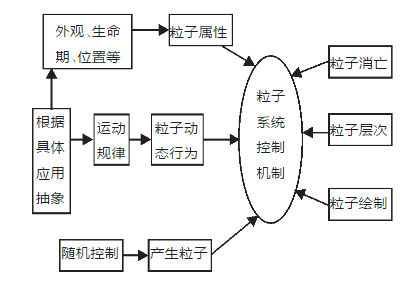


图2.3 粒子系统模型

粒子系统方法就是将大量的粒子单元集合在一起，通过其属性的变化表现物体的物理特性的物体模拟方法。所谓粒子是有着形状、大小、颜色、透明度、位置及速度等属性的几何单元，一个粒子究竟有什么样的属性主要取决于具体的应用。粒子系统并不是一个简单的静态系统，随着时间的推移，系统中已有的粒子不仅不断改变形状，不断运动，而且不断有新的粒子加入，并有旧粒子的消失。为模拟粒子生长和死亡过程，每个粒子被赋予一定的生命周期，它将经历生长、成长、衰老和死亡的过程。同时，为使粒子系统所表示的事物具有良好的随机性，与粒子有关的每一个参数均将受到一个随机过程的控制。概括起来说，粒子系统的基本理论主要有以下六个方面：

（1）物质的粒子组成粒子系统是由具有一定属性的流动粒子所组成的集合，把运动的模糊物体看作由有限的这些粒子以连续或离散的方式充满它所处的空间，并处于不断地运动变化状态，粒子在空间和时间上具有一定的分布范围。

（2）粒子独立关系这里包含两个意思，一是粒子系统中各粒子不与场景中任何其它物体相交，二是粒子之间不存在交互关系，并且粒子是不可穿透的。

（3）粒子的属性系统中的每个粒子并不是抽象的，它们都具有一定的属性，比如粒子的质量，粒子存在的空间，外观属性，例如颜色、亮度、形状、尺寸、透明度等，运动属性，例如速度、加速度等，还有生存属性，其中速度、位置、颜色、亮度等属性随着时间都可以不断地发生变化。

（4）粒子的生命机制粒子系统中的每一粒子都具有一定的生命周期，在一定的时间周期内，粒子经历产生、活动和消亡三个基本生命历程。

（5）粒子的运动机制粒子在存活期间始终是按一定的方式运动的。

（6）粒子的绘制算法 主要有点粒子、面粒子、线 性粒子以及随机形状粒子四种绘制算法。

## 2.6 刚体

对于一个优秀的游戏开发平台来说，除了友好的开发环境， 还必须拥有一个完备的物理引擎系统。现实生活中的所有事物都遵循自然界的物理定律，要想达到模拟现实世界的物理效果，就必须使用同自然物理法则相对应的物理引擎作辅助。

Unity 3D游戏引擎内置了由NVIDIA出品的PhysX物理仿真引擎。该引擎是世界三大引擎之一，具有高效低耗的特点，且仿真程度极高。物理引擎通过为刚性物体赋予真实的物理属性的方式来计算它们的运动、旋转和碰撞反应。在开发过程中只需要简单地操作就可以使物体按照物理运动规律进行运动。

为了便于开发人员控制物理系统，Unity 3D提供了多个属性接口，开发人员可以通过更改这些参数来实现对物体物理状态的控制。在实际的开发过程中，这些参数都被详细地罗列在属性面板中，开发人员可以很方便地对这些属性进行修改。接下来将对本游戏主要用到的属性进行讲解：

（1）质量（Mass)。质量属性表示刚体的质量，其数据类型是float。默认值为1。一般来说， 大部分的物体的Mass属性值应该设置为接近0.1且不超过10.0才符合日常生活中的感官感受，刚体的质量并没有单位，在开发过程中要通过保持物体与物体之间的质量之比来提高其物理仿真度。

（2）阻力（Drag)。这里的阻力指的是物体的移动阻力，物体进行任意方向的移动都会受到 Drag的影响。该属性的数据类型是float，默认值为0。 Drag的方向与物体运动的方向相反，对物体的移动起阻碍作用。通过对Drag设置不同的值，可以分别模拟出羽毛和石头掉落的情景。

（3）旋转阻力（Angular Drag）。Angular Drag与Drag类似，也是阻碍物体运动的一个力。该属性的数据类型是float，默认值是0.05。如果将该属性设置为0，则物体在因受瞬时力而旋转后，将不会停止旋转运动。此属性值越高，物体的角速度衰减就会越严重。

（4）使用重力（Use Gravity )。Use Gravity这个属性是以布尔值的形式存在的，其初始值为true，这一属性设为false时，物体将不受重力的作用，但其他非重力则正常进行计算，可以模拟出物体生外太空等特殊场合的无重力状态，对于某些特殊的场景的物理模拟是非常有用的。

## 2.7 音频系统

### 2.7.1音频管理器

音频管理器（AudioManager）是在宏观上对场景中的声音进行设置，通过点击菜单Edit->Project Settings->Audio打开音频管理器。在实际开发过程中，可以对音频管理器的属性参数进行修改和设置。

### 2.7.2 音频监听器

音频监听器扮演着像麦克风这样的设备。它接收任何在场景输入的音频源（Audio Source），并通过计算机的扬声器播放声音。对于大多数应用程序，经常会把侦听器贴在主摄像头（Main Camera）。音频侦听器（Audio Listener）配合音频源(Audio Sources)，让你可以为你的游戏创建听觉体验。当音频监听器贴在场景中的一个游戏对象，任何音频源，如果足够接近侦听器就会被获取并输出到计算机的扬声器。每个场景只能有1个音频侦听器正常工作。在菜单栏通过Edit->Project Settings->Audio访问项目范围内的音频设置，且每个场景只能有一个音频监听器。

### 2.7.3 音频源

音频源(Audio Source)在场景中播放音频剪辑(Audio Clip)。如果音频剪辑是一个3D剪辑，音频源是在一个给定的位置，并会随距离衰减这样的方式进行播放。音频可以在扬声器之间传播还可以在3D和2D之间进行转换。这是可以控制的随距离衰减曲线。此外，如果侦听器是在1个或多个混响区中，回响将会应用到音频源。从菜单栏选择 GameObject->Create Empty。随着新的游戏对象被选择，选择 Component->Audio->Audio Source，在检视面板中分配音频源组件的音频剪辑属性。

# 3 系统建模

## 3.1交互模型概述

“所有系统都会涉及一些交互。有可能是用户交互，与用户输人输出有关；有可能是正在开发的系统与其他系统之间的或是系统各部分之间的交互。为用户交互建模主要是因为它有助于我们识别用户需求。为系统与系统间的交互建模应将重点放在可能产生的交流问题上。为系统各部分之间的交互建模有助于分析所提出的系统结构能否实现系统所需的功能及其可靠性。”[1]本文采用两种相关的交互建棋的方法，用例建模，该方法主要用来为系统与外部参与者（用户或其他系统）之间的交互建模。时序图，该方法用来为系统各部分之间的交互建模，尽管也包括一些外部因素。

3.2用例建模

用例图是指由参与者（Actor）、用例（Use Case），边界以及它们之间的关系构成的用于描述系统功能的视图。用例图（User Case）是被称为参与者的外部用户所能观察到的系统功能的模型图，呈现了一些参与者和一些用例，以及它们之间的关系，主要用于对系统、子系统或类的功能行为进行建模。本游戏用例视图如下所示

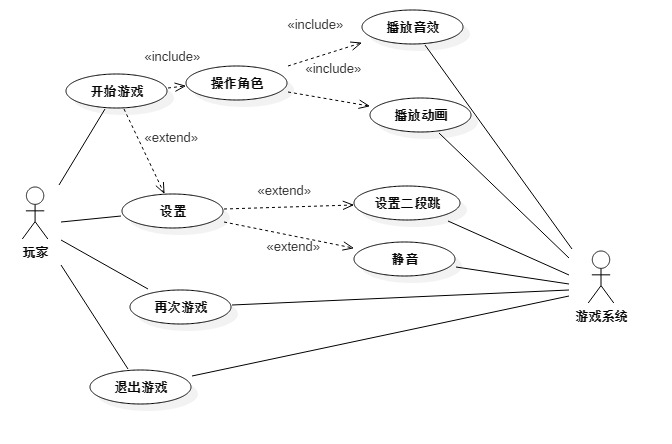


图3.1 游戏用例图

玩家开始游戏用例的表格描述如下：

表3.1 用例“开始游戏”的表格描述

|  |  |
| --- | --- |
| 开始游戏 | |
| 参与者 | 玩家，游戏系统 |
| 描述 | 玩家点击开始游戏按钮，游戏系统有限状态机发送post信息，系统状态转换为游戏状态。 |
| 数据 | 玩家设置的角色信息 |
| 激励 | 由玩家发出的开始游戏指令 |
| 响应 | 向有限状态机发送开始游戏指令 |
| 注释 | 开始游戏后将游戏视图设置为激活状态，并隐藏开始界面 |

玩家操作角色用例的表格描述如下：

表3.2 用例“操作角色”的表格描述

|  |  |
| --- | --- |
| 操作角色 | |
| 参与者 | 玩家，游戏系统 |
| 描述 | 玩家点击向左行走按键、向右行走按键或者跳跃按键，系统响应相应操作并播放动画和音频。 |
| 数据 | 玩家操作的角色的位置信息 |
| 激励 | 由玩家发出的操控按键指令 |
| 响应 | 调用相应方法改变角色的位置 |
| 注释 | 在某些状态下，玩家的操作将无法执行。例如，在玩家未开启二段跳时，玩家在空中禁止跳跃的动作。 |

## 3.3时序图

### 3.3.1开始游戏系统时序图

玩家进入欢迎界面点击开始游戏，向有限状态机发送开始游戏信息。FSM创建事件执行转换状态操作，若没有目标状态则直接返回，若目标状态是当前状态则重置当前状态，若存在目标状态，若不是公有状态则返回，若是公有状态则将状态插入到状态列表中。玩家正常进入游戏的时序图如下所示：

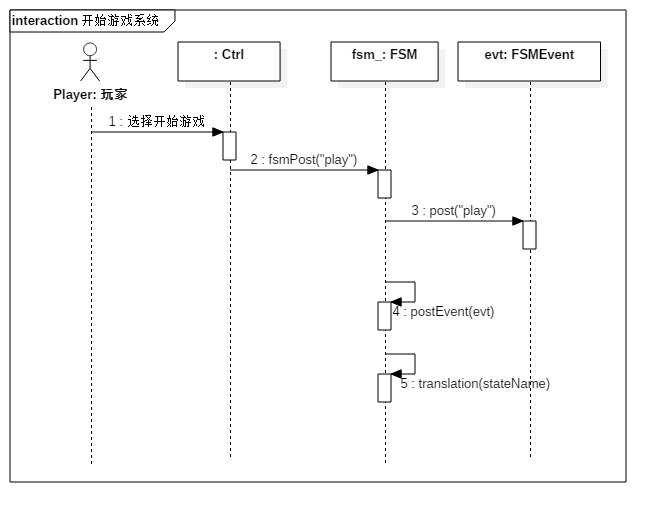


图3.2 开始游戏系统时序图

### 3.3.2获得奖励物系统时序图

玩家获得奖励物，unity通过OnTriggerEnter2D检测两物体碰撞，当前奖励物设置为消失并发散粒子，事件记录对象记录获得奖励物事件

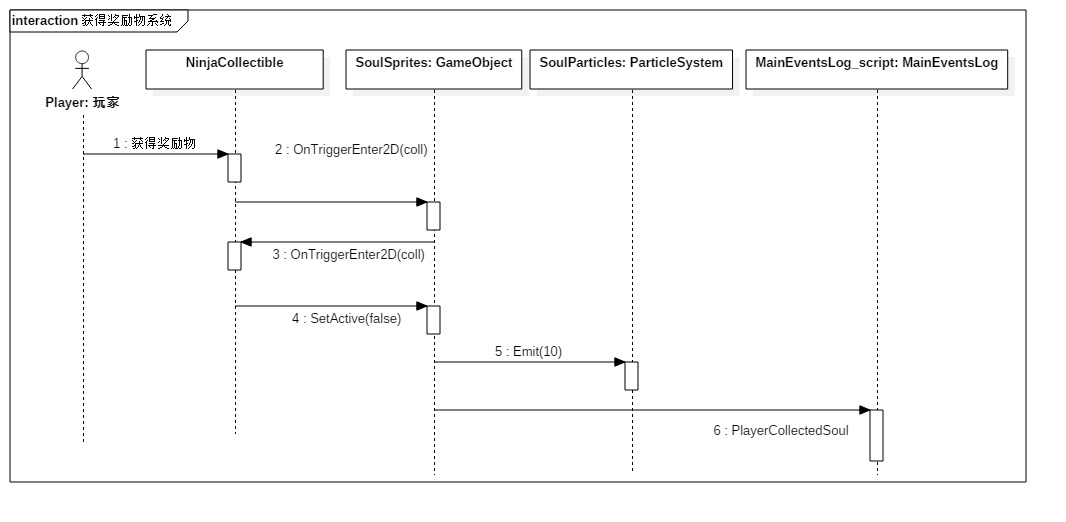


图3.3获得奖励物系统时序图

### 3.3.3敌人追逐系统时序图

敌人检查玩家的位置信息，根据与玩家的距离，敌人表现出不同的状态并更改动画系统设置。

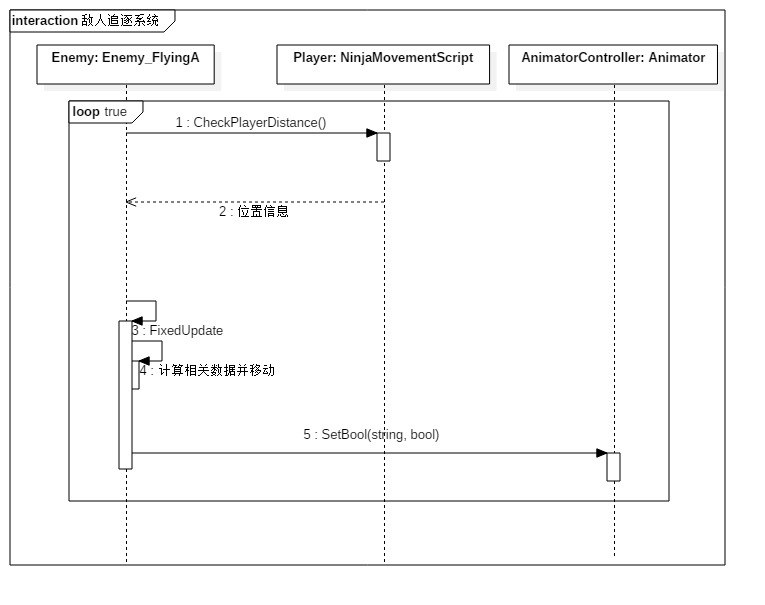


图3.4敌人追逐系统时序图

## 3.4结构模型概述

软件的结构模型表示的是系统的构成，表示为组件构成系统以及组件之间的关系。有的结构是静态模型，表示系统设计的结构；有的是动态模型，表示系统执行时的构成。

## 3.5 类图

游戏整体逻辑采取mvc结构，Ctrl类负责游戏开始界面、游戏界面和结束界面的跳转。类结构如图所示：

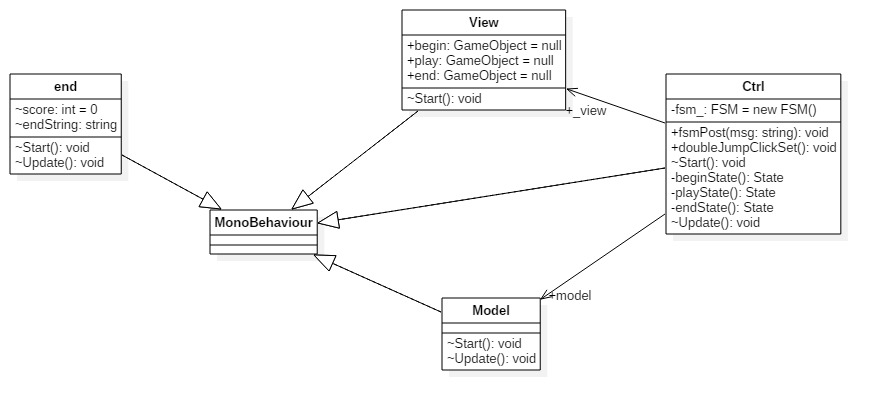


图3.5 mvc结构的类图

游戏采用有限状态机控制游戏逻辑场景的跳转。有限状态机表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。限状态机体现了两点，首先是离散的，然后是有限的。状态（State）存储关于过去的信息，反映从系统开始到现在时刻的输入变化。转换（Actions & Transitions）指示状态变更，并且用必须满足来确使转移发生的条件来描述它。动作是在给定时刻要进行的活动的描述。事件（Event）指对系统重要的某件事情。事件是有生命周期的，它经历被产生(被接受，等待被处理，一般放入事件队列)，被分发(从事件队列取出，分发到响应的状态机处理)，死亡(当状态机处理了该事件，它随之死亡)。有限状态机的类图结构如图所示：

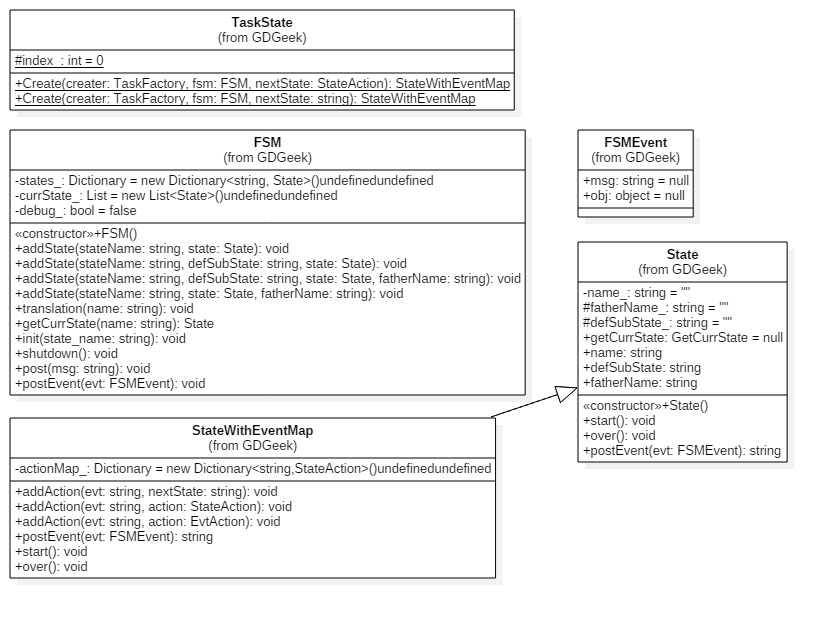


图3.6 有限状态机相关类图

与游戏相关的类，AutoRotator用于控制奖励物品的旋转，Checkpoint用于复活点的设置，MovingPlatform用于移动平台的构建，heart用于显示当前剩余的生命数。相关类如下图所示：

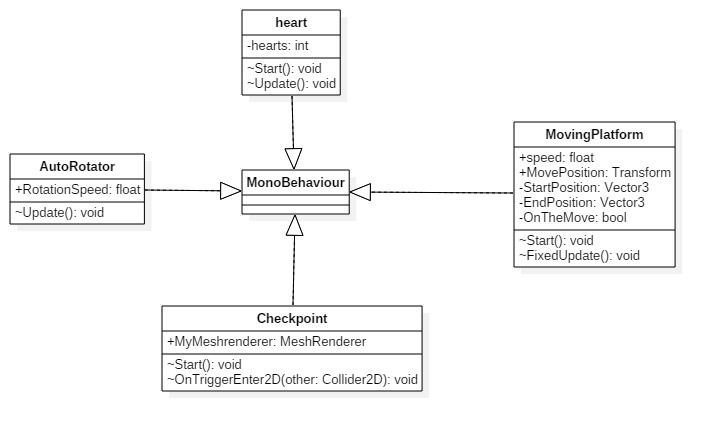


图3.7 非核心游戏类图

CameraFollowTarget类用于摄像头的跟随，MainEventLog用于接收事件，例如用户的点击和玩家收集奖励物品。NinjaCollectible类为收集物品。KillZone类为玩家碰触死亡的对象。NinjaMovementScipt类控制角色的行为和视图效果。类之间的关系如图所示：

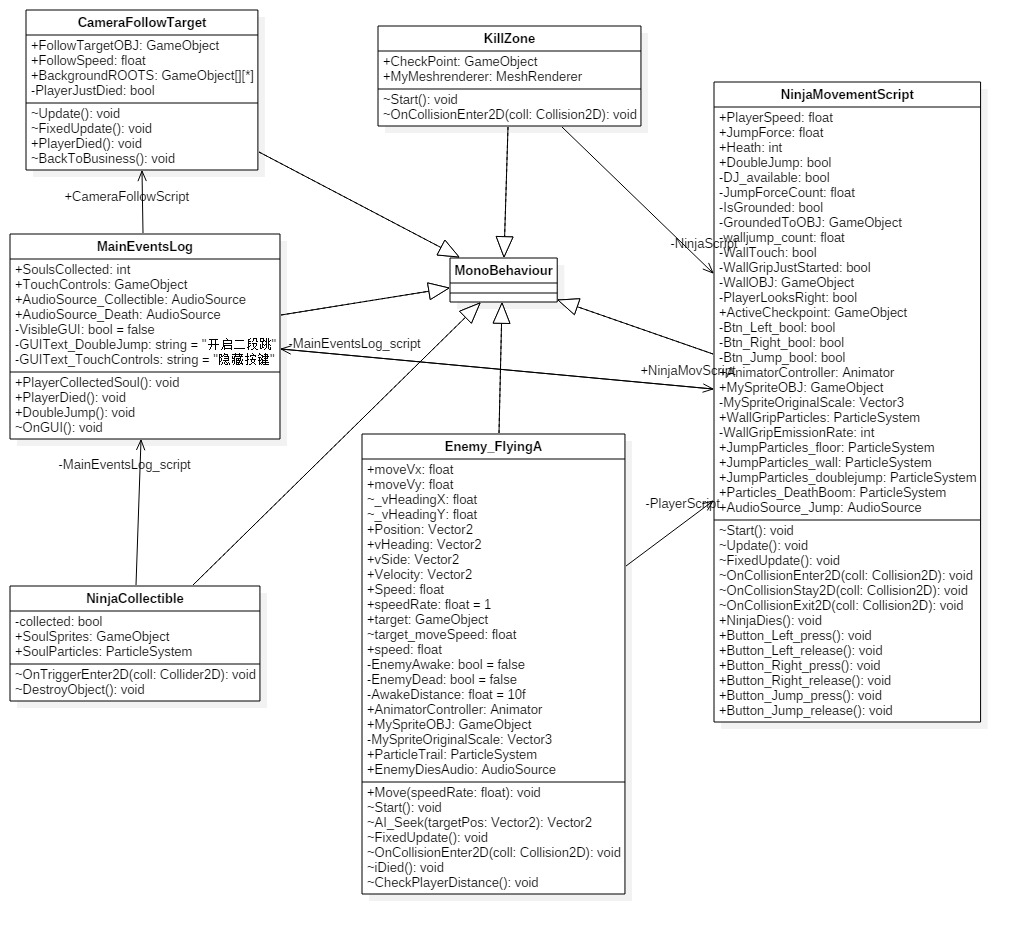


图3.8 核心游戏类图

## 3.6行为模型概述

行为模型是描述系统运行时的动态行为的模型，表示当系统响应来自所处环境的刺激时所发生的或有可能发生的事情。这样的刺激有两种，第一种是数据，一些数据到达必须由系统处理，第二种是事件，某些触发系统处理的事件发生。事件可能会有相关联的数据，但并不总是这样。

## 3.7状态图

状态图(Statechart Diagram)是描述一个实体基于事件反应的动态行为，显示了该实体如何根据当前所处的状态对不同的事件做出反应。通常我们创建一个UML状态图是为了以下的研究目的：研究类、角色、子系统、或组件的复杂行为。

玩家进入游戏后，角色处于空闲（Idle）状态，玩家通过左右键改变角色速度从而改变角色的状态，当角色速度大于某值后，玩家转换为奔跑（Run）状态，玩家按下跳跃键，角色转换为向上跳跃（Jump\_UP）状态，当玩家速度小于0转换为跳跃向下（Jump\_DOWN）状态，当玩家在墙上转换为抓在墙上（WallGrip）状态。当玩家死亡时转换为死亡（Dead）状态，若生命值大于0，则复活转换为空闲状态，若小于0则结束。具体如下图所示：

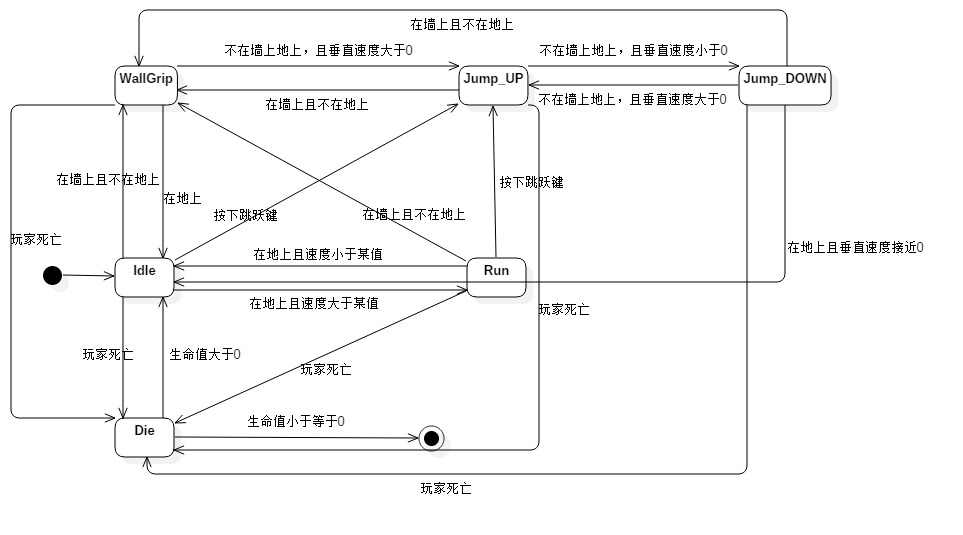


图3.9 玩家状态转换图

敌人默认状态为睡眠（Sleep）状态，当敌人与角色的距离小于某值时转换为唤醒（Awake）状态，若敌人与角色距离大于某值时转换为睡眠状态，若被角色踩死则转换为死亡（Dead）状态。具体如下图所示：

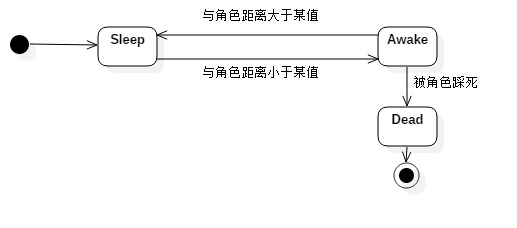


图3.10 玩家状态转换图

奖励物品默认进出创建（Creation）状态，当角色收集奖励物品后转换为已收集（Collection）状态。具体如下图所示：



图3.11 奖励物状态转换图

# 4 游戏中的人工智能

## 4.1人工智能概述

“人工智能（Artificial Intelligence，简称Al)，是指由人工制造出来的系统所表现出来的模拟人类的智能活动，通常也指通过计算机实现的这类智能。在游戏中，对于Al，应该关注的问题是如何让游戏角色能像人或动物那样“感知”、“思考”和“行动”，让游戏中的角色看上去像具有真实的人或动物的反应。” [2]

事实上，对于游戏中的Al角色，可以认为它们一直处于感知 (Sense)，思考（Think）和行动（Act）的循环中。

感知：是Al角色与游戏世界的接口，负责在游戏运行过程中不断感知周围环境，读取游戏状态和数据，为思考和决策收集信息。例如，是否有敌人接近等。

思考：利用感知的结果选择行为，在多种可能性之间切换。例如，战斗还是逃跑？躲到哪里？一般说来，这是决策系统的任务，有时也可能简单地与感知合二为一。

行动：发出命令、更新状态、寻路、播放声音动画也包括生命值减少等。这是运动系统、动画系统和物理系统的任务、而动画和物理系统由游戏引擎提供支持。

尽管每种游戏需要的AI技术都有所不同，但绝大多数现代游戏中对AI的需求都可以用三种基本能力来概括。运动：移动角色的能力；决策：做出决策的能力；战略：战略战术思考的能力。

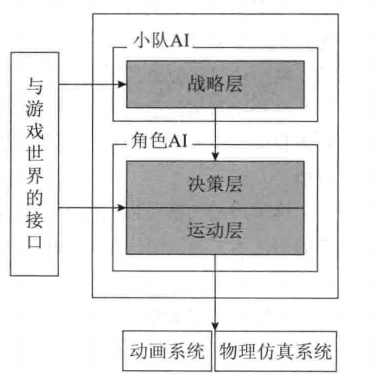


图4.1 通用的AI架构模型

## 4.2控制行为编程的主要基类

### 4.2.1概述

编程中主要涉及到Vehicle类，AILocomotion类和Steering类，它们是实现控制行为的基础，属于运动层。

### 4.2.2将AI角色抽象成一个质点

首先对Al角色做出抽象。假设Al角色的基类名称为“Vehicle”，这个类名的直译为“交 通工具，但实际上它包括了很能自主移动的任何Al角色，如有轮机器、马匹、飞机、 汽车潜水艇动物人类和怪物等。

在Al构架模型中，操控AT角色的基类Vehicle把操作的对象抽象为一个质点，它包含位置、质量、速度等信息而速度随着所施加力的变化而变化。由干速度意味着是实际物理实体，施加在其上的力和能达到的速度都是有限制的，因此还需要最大力和最高速度两个信息。除此之外还要包含一个朝向的信息。 综上这个交通工具位置的计算方法是这样的：

(1)确定每一帧的当前操控力（最大不超过max\_force）；

(2)除以交通工具的质量mass，可以确定一个加速度；

(3)将这个加速度与原来的速度相加得到新的速度（最大不超过max\_speed)；

(4)根据速度和这一帧流逝的时间，计算出位置的变化；

(5)与原来的位置相加得到交通工具的新位置。

### 4.2.3控制AI角色移动

AILocomotion类是Vehicle的派生类，它能真正控制Al角色的移动，包括计算每次移动的距离，播放动画等。

### 4.2.4各种操控行为的基类

Steering类是所有操控行为的基类，包含操控行为共有的变量和方法，操纵AI角色的寻找、逃跑、追逐、躲避、徘徊、分离、队列、聚集等都可由此派生。

### 4.2.5靠近行为模型

操控行为中的靠近是指，指定一个目标位置根据当前的运动速度向量，返回一个操控 Al角色到达该目标位置的操控力，使Al角色自动向该位置移动。

要想让Al角色靠近目标首先需要计算出Al角色在理想情况下到达目标的预期速度。该速度可以看作是从Al角色的当前位置到目标位置的向量。操控向量是预期速度与Al角色当前速度的差，该向量大小随着当前位置变化而变化，从而形成图4.2中该角色的寻找路径。

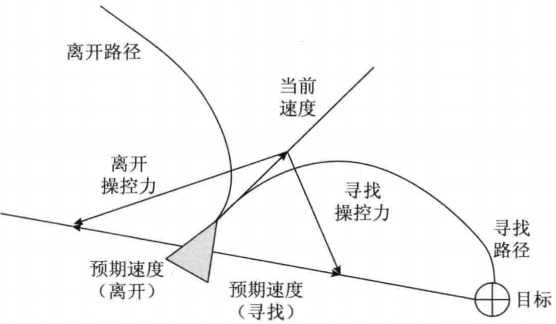


图4.2 靠近模型操控力的计算

图4.2中的小三角表示AI角色，图中画出了它当前的运动速度、预期速度、计算出的操纵向量以及实际的寻找路径。

### 4.2.6追逐行为模型

追逐行为与靠近行为很相似，只不过目标不再是静止不动，而是另一个可移动的角色。 最简单的追逃方式是直接向目标的当前位置靠近，不过这样看上去很不真实。举例来说，当动物追逐猎物的时候．绝不是直接向猎物的当前位置奔跑，而是预测猎物的未来位置，然后向着未来位置的方向追去，这样才能在最短时间内追上猎物。在Al中，把这种操纵行为称为追逐。

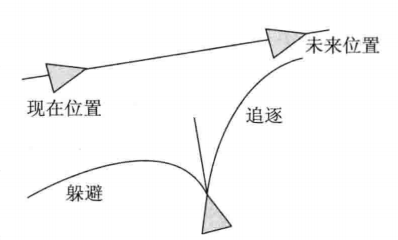


图4.3 追逐行为模型

可以看出，追逐是朝向未来位置，而不是朝向当前位置的。使用一个简单的预测器，在每一帧重新计算它的值。假设采用一个线性预测器，又假设在预测间隔T时间内角色不会转向，角色经过时间T之后的未来位置可以用当前速度乘以T来确定，然后把得到的值加到角色的当前位置上．就可以得到预测位置了。最后，再以预测位置作为目标，应用靠近行为就可以了。

实现追逐行为的一个关键是如何确定预测间隔T。可以把它设为一个常数，也可以当追逐者距离目标较远时设为较大的值，而接近目标时设为较小的值。

这里，设定预测时间和追逐者与逃避者之间的距离成正比，与二者的速度成反比。

一些情况下追逐可能会提前结束。例如，如果逃避者在前面，几乎面对追逐者，那么 追逐者应该直接向逃避者的当前位置移动。二者之间的关系可以通过计算逃避者朝向向量与 Al角色朝向向量的点积得到。

### 4.2.7避障模型

避开障碍行为是指操拉Al角色避开路上的障碍物，例如在动物奔跑时避免与树、端碰 憧。当Al角色的行进路线上发现比较近的障碍时，产生一个排斥力，使Al角色远离这个 障碍物；当前方发现多个障碍物时，只产生躲避最近的障碍物的操控力，如图2.17所示。这样，Al角色就会一个接一个地躲避这些障碍物。

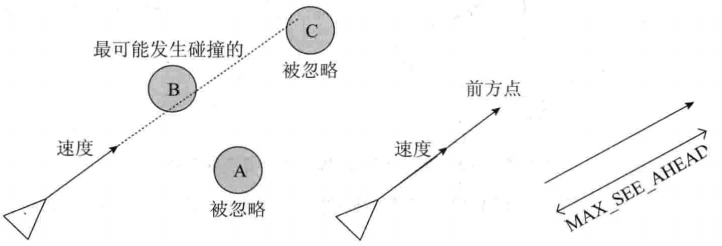


图4.4 躲避障碍物所需控制力计算方法

在这个算法中，首先需要发现障碍物。每个Al角色唯一需要担心的障碍物就是挡在它行进路线前方的那些物体，其他远离的障碍物可以不必考虑。该算法的分析步骤如下。

（1）速度向量代表了Al角色的行进方向，可以利用这个速度向量生成一个新的向量 ahead，它的方向与速度向量一致，但长度有所不同。这个ahead向量代表了Al角色的视线范围。其计算方法为：

ahead = position+normalize( velocity ) \* MAX\_SEE\_AHEAD

ahead向量的长度（MAX\_SEE\_AHEAD）定义了AI角色能看到的距离。MAX\_SEE\_AHEAD 的值越大，Al角色看到障碍的时间就越早，因此，它开始躲避障碍的时间也越早。

（2）每个障碍物都要用一个几何形状来表示，这里采用包围球来标识场景中的每个障碍。

一种可能的方法是检测从Al角色向前延伸的ahead向量与障碍物的包围球是否相交。这种方法当然可以，但这里采用简化的方法，更容易理解，且能够达到相似的效果 。

这里还需要一个向量ahead2这个向量与ahead向量的唯一区别是：ahead2的长度是ahead 的一半，如图4.5所示。

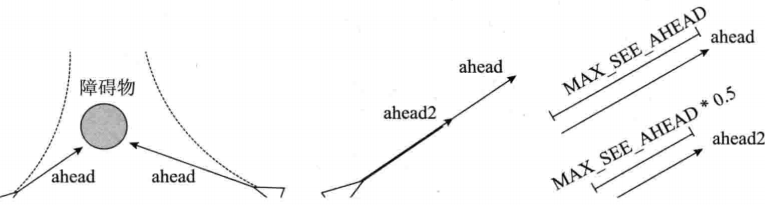


图4.5 躲避障碍物所需控制力的计算方法

### 4.2.8算法类图

编程中主要涉及到Vehicle类，AILocomotion类和Steering类，它们是实现控制行为的基础，属于运动层。操控AT角色的基类Vehicle把操作的对象抽象为一个质点。AILocomotion类是Vehicle的派生类，它能真正控制Al角色的移动，包括计算每次移动的距离，播放动画等。Steering类是所有操控行为的基类，包含操控行为共有的变量和方法，操纵AI角色的寻找、逃跑、追逐、躲避、徘徊、分离、队列、聚集等都可由此派生。SteeringForSeek负责靠近行为模型，SteeringForPursuit负责追逐模型，SteeringForCollisionAvoidance负责避障模型。如图4.6所示：

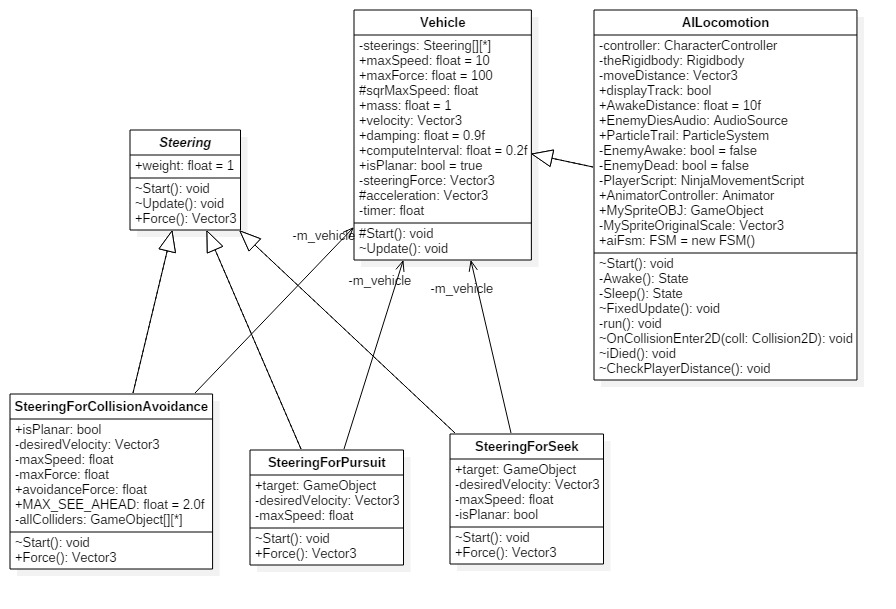


图4.6 敌人AI的人工智能类图

### 4.2.9避障模型时序图

把AI抽象为Vehicle质点，检测质点中绑定的Steering模型中的行为模型，避障模型通过计算得到躲避障碍所需要的操控力，Vehicle获得各个模型的操纵行为表后计算带权操纵力，并计算AI的加速度，AILocomotion类获得计算的加速度并在原有速度基础上计算新的速度矢量，并计算AI的移动距离，AILocomotion类根据AI有限状态机的状态判断是否更新当前AI的位置，如果AI处于唤醒状态，则更新当前AI的位置。如下图所示：

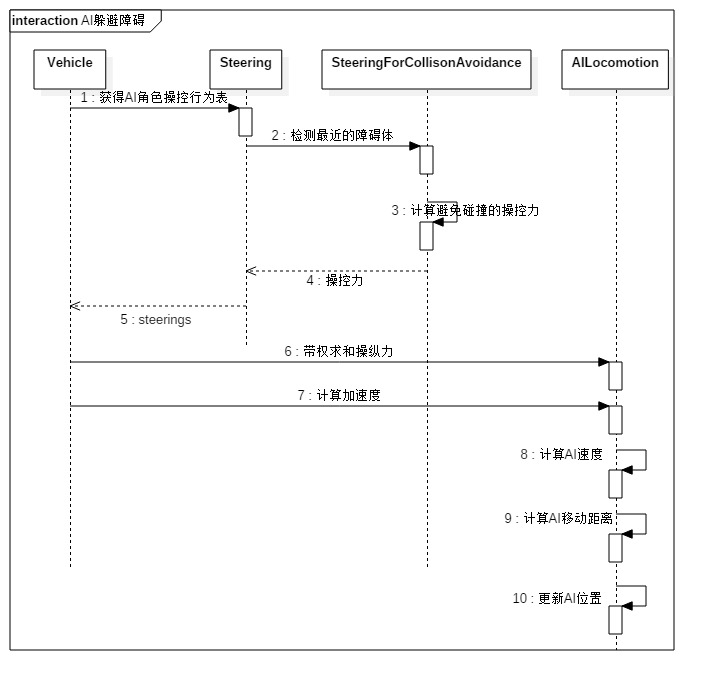


图4.7 AI躲避障碍的时序图

### 4.2.10靠近模型时序图

把AI抽象为Vehicle质点，检测质点中绑定的Steering模型中的行为模型，SteeringForSeek计算AI靠近角色需要的速度矢量并返回操纵力，Vehicle获得各个模型的操纵行为表后计算带权操纵力，并计算AI的加速度，AILocomotion类获得计算的加速度并在原有速度基础上计算新的速度矢量，并计算AI的移动距离，AILocomotion类根据AI有限状态机的状态判断是否更新当前AI的位置，如果AI处于唤醒状态，则更新当前AI的位置。如下图所示：

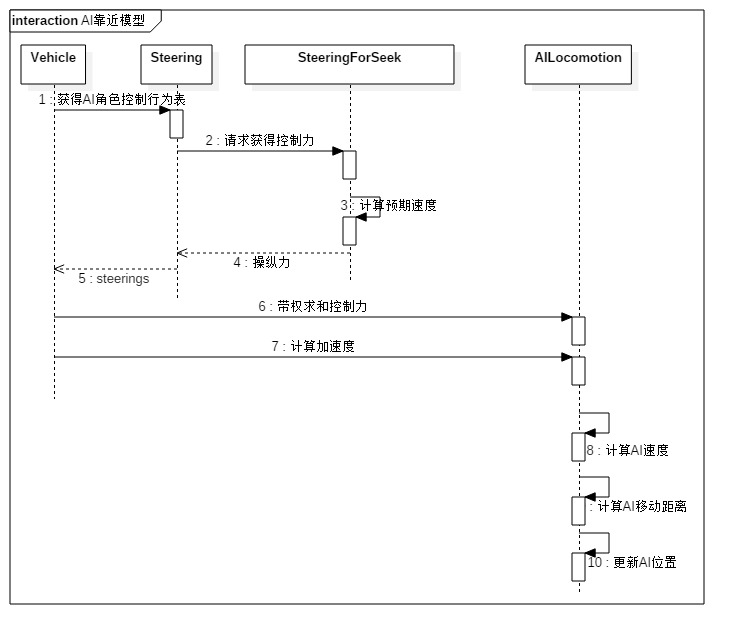


图4.8 AI靠近模型的时序图

### 4.2.11 追逐模型时序图

把AI抽象为Vehicle质点，检测质点中绑定的Steering模型中的行为模型，SteeringForPursuit计算追逐者和逃避者夹角，如果夹角大于0且追逐者基本面对着逃避者，那么直接向逃避者当前位置移动。否则计算预期时间，正比于追逐者与逃避者之间的距离，反比与追逐和和逃避者的速度之和，Vehicle获得各个模型的操纵行为表后计算带权操纵力，并计算AI的加速度，AILocomotion类获得计算的加速度并在原有速度基础上计算新的速度矢量，并计算AI的移动距离，AILocomotion类根据AI有限状态机的状态判断是否更新当前AI的位置，如果AI处于唤醒状态，则更新当前AI的位置。如下图所示：

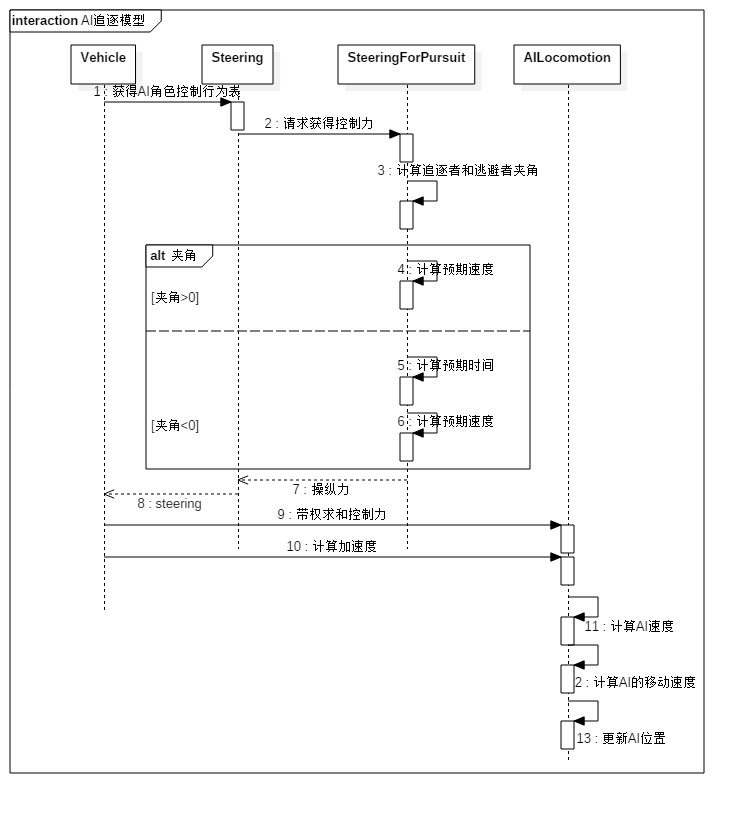


图4.9 AI追逐模型的时序图

## 4.3有限状态机

决策系统的任务是对从游戏世界中收集到的各种信息进行处理（包括内部信息和外部信息）。确定Al角色下一步将要执行的行为。这些行为由两部分组成：一些行为会更改AI角色的外部状态，例如拨动开关、进人房间、开枪等；另一些行为只会引起内部状态的变化，例如改变AI角色的心清状态、改变总体目标等，如图所示：

Al角色通过决策系统来确定下一步的行为，因此决策系统的重要性无需多言。幸运的是，相对于运动系统和感知系统来说游戏设计者并不需要费太多力，利用有限状态机或行为树就可以构造出一个看上去不错的决策系统。

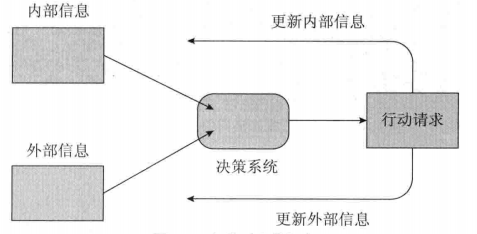


图4.10 决策系统的任务

有限状态机（Finite State Machine: FSM）由一组状态〔包括一个初始状态）、输人和根据输人及现有状态转换为下一个状态的转换函数组成。

有限状态机具有规则的结构，通过采用有限状态机，可以将Al角色的行为模式用一些状态和这些状态之间的转换来表示。

对于游戏Al来说状态的关键意义是：不同的状态对应不同的行为。

状态是由游戏设计者事先设定的。举例来说对于一个具有情绪的Al角色可以斌予它4 种状态：高兴、生气、害泊和悲伤，在游戏中的任一时刻，它都处于这4种状态之一。每种状态都有对应的行为和动作。例如，高兴时它会笑，悲伤时它会哭泣等。

对于游戏中的一个怪物，也可以定义6种状态，分别是：空闲、行走、奔跑、进攻、躲避和死亡。当怪物处于行走状态时，循环播放行走动画，而当它处于奔跑状态时，则循环播放奔跑动画等，每个状态对应的行为也是由游戏设计者定义的。

本游戏中敌人有三种状态，激活（Awake），睡眠（Sleep），死亡（Dead）。FSM图如下：

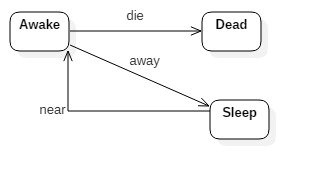


图4.11 敌人AI的FSM图

# 参考文献

[1]Sommerville，程成. 软件工程[M]. 北京：机械工业出版社，2016，51-91

[2]王洪源 陈慕羿.Unity3D人工智能编程精粹[M].北京：清华大学出版社，2014，7-33

[3]吴亚峰，索伊娜.Unity 5.X 3D 游戏开发技术详解与典型案例[M].北京：人民邮电出版社，2016，40-96

[4]徐烈辉. 粒子系统基本理论及其应用[J]. 电脑与信息技术，2009，(03):9-10+39.

王少伟，陈晓青，李颂，等．影视特效粒子系统研究［J］.软件导刊，2012(4)