第一章　计算机游戏：为什么、是什么、怎样写

1.1　为什么要写、正面和反面的意见

大多数人，甚至正规的程序员，对计算机游戏有矛盾的感情。一方面，人们着迷于流行的视频游戏中令人眼花缭乱的图形和效果。另一方面，同样是这些人，认为那些游戏是计算机科学的非重要分支，不值得认真地学习或考虑。在我遇到的程序员中，仅有少数人看了这些游戏后说："噢，希望我也能写这样的游戏！"，正是这些人欣赏用于创建这些游戏的编程技术并且想用自己的作品与之相比。当然，正在读此书的读者就属于这种人。

单纯地沉溺于游戏本身不足以成为编写计算机游戏的原因，下面列出的理由可能会说服读者。

1.1.1　挑战

计算机游戏是所有应用软件编程中最有挑战性的一种，它的挑战性来源于要求程序员具有许多方面的素质：创造性、革新性、耐心、注意细节以及制造乐趣的能力（这里仅列出了一小部分）。为了了解挑战的强度，试试下面的练习：任意挑选一种流行的计算机游戏，列出它所有不同的效果和细节，对其中的每一项考虑一下应该做些什么工作才能办到。我确信甚至读者刚进行了一半，就会对编写该游戏的挑战性有深刻的了解。

1.1.2　推广基本创意

编写一个计算机游戏来说明一个基本创意或想法是推广它的好方法。许多计划在早期只存在于纸面。不幸的是，许多人在讲述仅仅印刷出来的想法时有困难。一个模拟计划的基本创意的计算机程序给人们一个计划者想做什么的直接印象。这有助于他们领会该计划，对它有一种现实的感觉，而只用表格和图形达不到这种效果。这种方法的另一个好处是，它提供给人们的是计划者对基本创意的允诺的持久印象。

1.1.3　提高编程技巧

创造不同的计算机游戏迫使程序员不断地提高他的编程技巧。既然每个游戏都需要新的基本创意和图形效果，程序员必须使自己相应地作调整。这通常意味着编写新工具，开发新算法，尝试新语言的特性。程序员写的游戏越多，他未来编写游戏的工具、技巧、技术就越多。

1.1.4　改善系统知识

许多优秀的计算机游戏程序员认为成功的原因是不但有编程技巧而且熟悉他们的宿主计算机系统。许多在计算机游戏中用到的特殊效果（例如卷动）是通过灵活地使用操作系统和基础硬件而获得的。程序员经常在他们的应用程序里使用低级系统编程技术（例如写用户

中断处理程序)。另外,许多计算机配置了专用的硬件来降低写计算机游戏的难度,比较早期的例子是 Atari 800 中的游戏者/导弹(Player/Missile)图形硬件。成功的计算机游戏程序员明白这些硬件特性,并且最大限度地利用它们。即使是一个生手,在计算机游戏中为制造特殊效果而使用硬件特性也会学到系统的大量知识。

1.1.5 满足感

或许写计算机游戏(或做任何事情)最迫切的原因是个人满足感。在创造这些游戏时最大的动机之一是简单地让它们工作。由于我起步较早,在设计和实现计算机游戏时有许多对创造性和技术性的挑战。战胜这些挑战从而创造一个成功的计算机游戏是个人自豪感的巨大来源。写计算机游戏的另一个愉快的方面是观察人们对它们的反应。多数人(不管懂不懂计算机)通常都会喜爱好的计算机游戏。那些使孩子和我们一齐快乐的计算机游戏有一些基本的东西。在我看来,计算机游戏的这些方面使它们成为任何编程应用中最能获得满足感的一种。

1.1.6 发财

对于少数幸运儿,如果能创造极优秀而能用于出售的计算机游戏,那么就可以发财。推销游戏的一种方式是与专门推销共享计算机游戏的公司联系。如果他们喜欢,就会推销该游戏,并且为卖出的每一份游戏付版税。当然,也可以尝试自己推销游戏,但这样做太复杂、冒险和花钱。通过和一家懂得软件业务的公司合作,就可以利用他们的资源、专业知识和关系。

即使编程者不能以商业性的方式出售程序,还可能通过把它加载到计算机公告牌上而散发出去。不用说,一旦编程者这么做了,该游戏就成为公用域(public domain)的,并且能被任何可访问公告牌的人拷贝。但是,如果编程者要求一些用户付注册费用,某些使用该游戏的人可能会承认并支付这笔钱(尤其是如果他们喜欢该游戏)。如果编程者相应地提供一些对游戏性能的增强,付钱的人可能还会增加。增强的一些例子包括详细的用户指南、未来的升级或附加的游戏脚本文件。如果提供了比已存在于公用域的游戏更新或更好的东西,编程者发财的机会就会增大。

1.2 成功的计算机游戏的特性

什么是成功的计算机游戏的要素?虽然市场上有如此多的游戏,这个问题并不容易回答。但是,经过深思熟虑后,我定义了对大多数计算机游戏都通用的四条特性。下面将逐一介绍。

1.2.1 基本创意

一个游戏的基本创意是对游戏实际做什么的描述。通常,一个游戏的基本创意可以用一个语句总结。例如,游戏 MISSILE COMMAND 的基本创意可以总结为:防卫你的城市免遭敌人导弹的袭击。一旦基本创意定下来,就可以开始制作游戏,在此过程中加入细节和一些细微差别。接着,就可以开始设计实现游戏必需的特色和游戏特性。游戏中使用的每一个特性应该在某种程度上与原始的基本创意有关。一个没有坚实基本创意的游戏就如同无舵的

船，不知道在何处结束。

1.2.2 并行性

并行性是使多个事件出现在计算机游戏中，就好象它们同时发生一样的艺术。例如，如果多个独立物体在游戏中移动，用户不应当觉察到一个物体的移动延迟了其它的物体，相反，每个独立物的运动应当看起来天衣无缝，并且与其它物体无关。并行性是一个实时游戏质量的关键指标。

1.2.3 注意细节

注意细节存在于设计和实现复杂的现象图标、背景和特殊效果。计算机游戏的这个特性通常是专业程序员和业余爱好者之间的区别。商业化的视频游戏中的许多特殊效果是特别地注重细节方面的高超结果。如今出售的大多数计算机游戏中的细节的水平不是偶然的；它是通过使用完善的编程工具集得到的。这些工具集是由软件公司为家庭使用而开发的，并且通常是专利软件。与此相对的是，许多爱好者几乎没有任何工具来获得这些效果，没有这些工具就很难（如果不是不可能的话）使一般爱好者达到细节的专业水平。

1.2.4 实时用户交互

实时用户交互是在和用户的所有交互中游戏维持现实的响应时间，这个性质（比其它任何性质）影响游戏的可用性。一个对用户输入反应迟钝的游戏很快就会被抛弃；同样，对用户响应要求太快的游戏也不行。用户对于和他们速度不配合的游戏没有忍耐性。结果是，计算机游戏维护可接受的响应级别的能力是计算机游戏性质中最重要的一个。

1.3 三步法

作出具有这些品质的计算机游戏不是一件容易的工作。本书的目的是使读者获得开发与图形紧密相关的计算机游戏所需要的技巧和知识。为了达到此目的，本书给出了三步法的要点，介绍计算机游戏开发的基本创意。这些基本创意在本书中通过特定计算机游戏工具、技术和应用程序的例子而反复强调。

1.3.1 步骤

本书的第二章定义了制造计算机游戏的整个步骤。有关游戏构造步骤的描述给出了开发的原则，读者可以把这些原则应用到后面章节描述的工具和技术上。读者也可以把这个步骤应用到开发自己的计算机游戏上。

1.3.2 工具和技术

本书的第二部分讨论在 PC 上开发计算机游戏要使用的工具和技术。将会描述计算机游戏的基本成分（例如游戏者、背景等等）。另外，还描述了对每个成分（例如游戏者动画）都可适用的编程技巧并给出了例子。本书中使用的设计工具也在这一部分介绍。每个讨论的工具都提供了详细的指导。这些指导是以计划文件和例子输出形式摘要给出，以给读者每个

工具怎样应用的现成例子。

1.3.3 应用例子

本书以几个完整的计算机游戏应用实例结束。对用于创造每个游戏的基本创意、技术和工具都详细地进行了讨论。这部分的目的是使读者明白怎样结合步骤和技术，以创造完整的计算机游戏。

第二章　游戏开发步骤

我们在玩计算机游戏时有如此多乐趣，以至于很容易忘记使它得以运行的软件。游戏是由计算机程序组成的，它与其他（不太有吸引力的）类型的应用程序一样适用相同的规则。因此，需要对游戏开发加入一些原则措施。保持这种原则的方法是遵循开发游戏应用程序的特定步骤。游戏开发步骤包括定义、设计、编码和测试几步。使用特定步骤并不意味着失去了写游戏的乐趣；相反，它提供了一个框架以保证游戏写得很合适。毕竟事情做得正确比把它做得过火更容易和更有乐趣，本章将深入探讨游戏是怎样开发的。

2.1　定义游戏基本创意

开发游戏（或做任何事情）的第一步是对其进行构思。灵感的来源无限的，我先假定读者已经有了一个游戏的创意。游戏的初始创意通常非常简单；在很多情况下可以用一个单句总结（例如，进行对抗恶魔 Galactic Empire 的战争！）。不管有什么创意，在开始开发游戏前先弄清楚自己是否的确对它感兴趣。

既然游戏的大多数创意开始时都太模糊了，在开始编码前扩展思路通常是必要的。游戏的精髓应该被总结为基本创意。基本创意应该让用户对游戏如何工作有个基本了解。为了详细地描述游戏，基本创意应该包括以下信息：

- **目标**——目标定义的是通过玩游戏用户可以完成什么。在大多数情况下，目标描述游戏者如何赢得游戏。例如，战争模拟游戏的目标可能是占领敌人的首都。游戏的所有的其它方面应该支持（或阻碍）用户完成目标。
- **角色**——角色是在游戏中各个实体的职务。基本创意应该描述在游戏中的每个角色。如果没有实在的角色，则描述应该集中于玩游戏者扮演的角色。飞行模拟是一个面向经历（用户扮演飞行员的角色）而不是面向角色的游戏的例子。描述用户的角色是紧要的，但不要忘记其它角色；它们对游戏的成功也是一样重要的。对每个角色的描述应该包含足够的细节，以便其它人能准确地编程。
- **规则**——规则是描述怎样玩游戏的指导方针。虽然没有游戏应有多少规则的固定公式，但下面几种规则应该包含在游戏的基本创意中：

  - 管理每个角色行为的规则。
  - 描述游戏角色如何互相交互作用的规则。
  - 描述如何符合游戏目标的规则。

在写基本创意时，集中精神于想让游戏做什么而不是怎样写游戏。原因很简单，基本创意应该是游戏能成为什么样的目标。如果太早尝试指出怎样实现基本创意，就会无意中去掉认为不能实现的特性。如果发生了这种事，就会以削弱游戏的潜力为代价简化了开发。先独立地发展一个基本创意而不去考虑是否可能完成可能更好。一旦知道了想在游戏里包含什

么，就可以一步一步试着开发它。结果读者将会惊奇地发现，那些一开始认为绝不可能完成的部分都实现了。

对某些人来说，给出游戏基本创意的文档看起来是不必要的。许多程序员相信在开始编码以前任何工作都没有意义。这样一来，立刻开始写游戏就有巨大的诱惑力。这些人在编码时创建基本创意。这种方法的问题是通常导致浪费力气（这些程序员经常在搞清楚想要什么以前扔掉一半的代码）。没有理解自己目标的编程就象在不熟悉的地区开车，而手上还没有地图——最终得停在某个地方，但不一定是想去的地方。这不是说在编码时不能产生出好念头（在写游戏时一些最好的创意可能会产生出来）。但是，在一开始就计划好想实现什么，就能集中力量并大幅度提高效率。

2.1.1 找出开发瓶颈

现在已经完成了游戏的基本创意，是准备编写的时候了。主要精力应集中在开发中最困难的地方。通过一开始找出潜在的开发瓶颈，就可以较早地解决它们。这些初始的投资将帮助避免开发和集成中潜伏的大的困难。下面将对一些开发中最普通的瓶颈来源进行讨论。

2.1.2 新技术和算法

如前面说过的那样，在不考虑如何具体实现游戏的前提下构造基本创意是一种好的方法。这将会产生更好、更详细的基本创意。但是，最终还是不得不坐下来决定怎样把基本创意变成现实。这时候可能会发现基本创意中描述的特性目前不知道如何实现。在大多数情况下就不得不开发新的技术和/或算法以实现这些特性。开发这种功能的工作要求具有极大的创造性和革新性。在开发阶段找出这些特性就可以在游戏的设计完全建立以前给出几种不同方法的原型（prototype）。

2.1.3 繁琐的工作

如今的大多数游戏的质量是如此的高，以至于很容易把其中的工作视为当然。游戏中的许多工作都包含细节，例如背景构造和角色设计。虽然不是特别困难，但这种工作通常很乏味，并且要花大量时间和精力。结果是，很容易将这种编程视为繁琐的工作。

一种减少繁琐工作的方法是开发自动进行设计步骤的工具。目标应该是创造允许开发者更专心于设计的工具，而屏蔽掉工作中辛苦而又令人厌烦的部分。既然繁琐的工作通常是由简单、但是重复操作的数据组成，于是设计步骤自动化通常很容易。几个这种工具的例子在第四、五、六和九章都有。虽然开发一个工具集可能看起来要付出许多努力，但是收益将是编程效率的极大提高。

2.1.4 用户接口

用户接口控制游戏者与游戏的交互。由于它的重要性，与用户接口相关的问题必须在开发过程的早期找出来并解决。典型的用户接口问题包括显示属性（要用的色彩、显示分辨率、显示页数量等等）和I/O（输入/输出）方面的考虑（例如I/O设备的类型）。同时，还得确定游戏是否有某些特殊的接口要求超越了编译器能支持的范围。例如，如果要在游戏中使用菜单，就可能不得不让编译器支持合适的函数库。一定要提前发现用户接口需求，让游戏第一

次就正确还要做很多工作。

2.1.5 运行性能

游戏的运行性能是由它执行的效率来表明的。例如，有较好运行性能的游戏运动较快并且反应迅速。与此相反，运行时性能较差的游戏通常行动迟缓。虽然性能问题通常直到集成时才显示出来，但还是有可能在早期阶段发现引起它们的瓶颈。潜在的性能瓶颈的某些症状包括：

- 效果的出现要花很长时间(1 秒或更长)。
- 算法是算术型、图形密集型或递归的。
- 大的例程经常执行。
- 循环次数很大。
- 例程轮循用户输入。

一旦发现了设计中潜在的瓶颈，就可以在开展设计时灵活地处理它。一些避免运行时性能问题的设计策略将在第十三章中讨论

2.2 编写游戏

必要的准备工作完成以后，下一步就是开始编写游戏。可以把这个开发步骤分为三个不同部分：设计场景、建立支撑例程、构造关键循环。

2.2.1 设计场景

在剧本中，场景是由背景布置、道具和角色服饰组成的。游戏场景基本上由相同的成份构成。在设计游戏场景时，应主要关心于画出背景和角色的外观。虽然这个工作很长而且乏味，但是非常重要，因为游戏通常是以场景的质量来评价的。所以，本书中的许多部分都是关于场景建造的。这些材料包括背景构造的编程工具和技术(第三章到六章)和角色设计(第七章和第九章)。

2.2.2 原型化支撑例程

支撑例程包含给游戏注入活力的函数。支撑例程基本上是操纵或控制游戏中的背景和/或角色的任何例程。包含在支撑例程中的一些函数例子如下：

- 角色动画。
- 角色移动。
- 检测和处理碰撞。
- 输入处理。
- 动作模拟。
- 卷动。

支撑例程操纵游戏的背景和角色。在设计场景时编写这些函数是有意义的。这保证了在场景特征和其相关操作之间不会有较大的不连贯。第八章到第十二章介绍了读者在开发

自己游戏的支撑例程时可以用到的工具和技术。

2.2.3 构造关键循环

虽然游戏处理的大部分内容存在于支撑例程中，但还是需要逻辑以使游戏执行连贯——这个逻辑就在关键循环中。关键循环可以看作是把游戏粘在一起的胶，它负责调度使游戏能执行的支撑任务。所以，关键循环在开发的集成和测试阶段起着关键作用。在第十三章中，将详细讨论实现关键循环的方法。

2.3 集成和测试

集成和测试(I&T)是把单独的程序成份组成连贯的应用程序的开发阶段。I&T 的大部分工作是确保达到了游戏的功能和性能目标。在 I&T 时，关键循环应被作为测试新写成的游戏例程的框架。当新的功能完成时，它应当被集成到框架中，这种集成的产物被称为建筑(build)。通过使用这个过程，框架就会逐渐延伸为完整的应用程序。这就是集成和测试的"建筑一点，测试一点"的方法。

不论何时进行建筑，必须执行三个基本行动以确保新软件被适当地集成。每一集成行动将在下面讨论。

2.3.1 测试新功能

虽然测试新功能的需要看起来很明显，但通常不知道怎么做。大多数错误(bug)出现在"已完成"代码中，因为在测试中没有检测出来。检测不出错误大多是因为测试人员的态度。不幸的是，许多开发人员热心于证明他们的软件能工作而不是发现潜在的错误。虽然没有"魔法子弹"这种能消除程序中所有错误的东西，但仍然还有许多能提高测试的错误检测率的通用策略。这些策略包括：

- **使用独立的人员测试软件**——这些人应当是没有参与编码的。由于独立的测试人员对处理有不同的观察角度，他们通常能发现开发人员忽略的错误。
- **尽量测试新功能的所有排列组合**——测试人员通常假定如果一个特性在一两种条件下能工作，那么它在任何条件下都能工作。不幸的是，它不总是那样。一个特性在以非预料的方式使用时通常出错。例如，考虑一个必须处理几种不同类型角色之间的碰撞的游戏。测试两个物体之间的碰撞可能证明功能运行在预料之中，但是可能还不足够。测试人员必须确定每一种排列的碰撞(例如三个不同角色同时相撞)都工作得很好。
- **使测试可以重复**——当不能重新产生曾经观察到的异常时，就出现了一个很常见的测试问题。可以通过记录每次测试用到的条件和步骤来解决这个问题。一旦能够分离出引起异常的原因，那么修复它的工作就完成了大半。
- **跟踪所有观察到的错误**——如果一个错误只出现了一次，忽略掉它就变得很有吸引力了。一个程序员通常会说服自己：这个错误并未真正发生或者它自己恢复了。不要欺骗自己——如果发现了错误，那么它就确实存在！防止错误导致崩溃的一种方法是记录所有观察到的故障。记录应该包括错误的描述和可能导致

错误的所有程序条件。仅仅在已经改正了某一错误或富于逻辑性地说明了这一错误不可能再次发生之后，再把这一错误从记录中注销。

- **在较长时间里测试软件的稳定性**——大多数软件错误在短时间内就会出现，而某一些却要用更多时间后才出现。对那些使用随机产生的数据的游戏更是这样。在某些情况下，错误要在游戏运行一段时间后才会出现。为了防止这种情况，测试很长一段时间以发现这些错误就很重要了。

2.3.2 回归测试

执行回归测试的目的是发现建筑之间不可预测的改变。由于使用了“建筑一点，测试一点”的方法，游戏在完成前要经历许多次建筑。既然每个建筑加入新软件，回归测试就保证变化不会影响以前建筑的能力。为了进行回归测试，请使用编排好的脚本来测试每个建筑。每个脚本必须运用以前系列建筑的功能和/或性能。随着新软件的加入，更新测试脚本以反映当前建筑的能力。虽然脚本应该足以检测能力的显著缺陷，但它们没有必要重复已经执行过的彻底测试。

2.3.3 协调性能

集成的目标之一是在建筑中加入新代码时维持可接受的性能水平。为了达到这个目标，在加入新软件时通常有必要协调应用程序。这看起来似乎容易，但优化游戏的性能是非常复杂的任务。一种简化问题的方法是找出游戏中直接影响处理负荷的变量。一旦分离出这些变量，就可以用常量来代替它们。如果代码是以这种方式开发的，处理瓶颈就可以通过调整这些常量而解决。协调以提高应用程序的性能是集成的一个重要部分。第十三章中有这方面的深入讨论以及其它提高运行性能的方法。

2.4 小　结

本章讨论了游戏开发的步骤。第一步是创造游戏的基本创意。由于基本创意是游戏的蓝本，使用它就可以找出实现游戏所需要的功能。在进行分析时，尤其要注意标出潜在的性能瓶颈。完成了开发的准备工作以后，就可以开始编码了。游戏的设计可以分为三个主要的部分：场景、支撑例程和关键循环。场景的代码用于实现背景和角色。支撑例程根据基本创意中定义的规则操纵场景。关键循环实现执行中控制游戏需要的逻辑。随着场景和支撑例程中的各部分的完成，就可以逐渐把它们和关键循环集成在一起。这种“建筑一点，测试一点”的集成方法可以让程序员在开发游戏时递增地测试它。每个增加的建筑在把另加的软件加在它上面以前都应该进行完全的测试。最终的建筑是完整的游戏应用程序，遵循这些步骤可以保证最终产品的质量。

第三章　背景定义

现在我已经介绍过了开发游戏的步骤，现在可以进入到建筑游戏的细节问题了。计算机游戏的最重要部分之一是它的背景。背景告知游戏者游戏动作发生的环境。游戏的背景可以以多种不同方式使用。它可以和游戏没有任何关系（如空白），或者对游戏仅有消极的影响，例如显示的道具或固定物。背景也可能是游戏不可缺少的一部分。最后这句话的一个好例子是跳棋盘（Checkboard）背景。跳棋盘的背景设计很简单，而游戏的规则取决于跳棋盘的布置。不管怎样使用背景，游戏通常是以背景的质量来评价的。所以，把想法和细节放入适合于游戏的范围和目标的背景是必需的。为了帮助读者作到这一点，我们将给出怎样给应用程序配置读者设计的背景类型的基础。这包括浏览设置显示所必需的图形函数。首先，我们将讨论视频模式的选择，包括与选择有关的折衷关系。接下来，我们将讨论怎样使用图形函数来在每种视频模式下选择背景和前景颜色。最后，我们将讨论在计算机游戏中通用的一些背景类型。

3.1　显示器配置基础知识

为了创建高质量的背景，首先必须正确地配置显示器。由于图形函数库是C语言最不标准化的方面之一，用来执行显示器配置的函数取决于所用的编译器。自然而然地，熟悉程序员手册中有关这些函数的部分就非常重要了。读者将会发现，在一开始就掌握这些函数的用法，最后在效率和软件质量方面将得到慷慨的回报。

3.1.1　视频模式属性

配置显示器最重要的决定是选择视频模式。Microsoft Visual C++中可用的图形模式在表3.1中给出。图形模式的选择影响许多关键的显示属性，如下所述：

- **模式类型**——有两种基本的视频模式：文本模式和图形模式。文本模式只能显示文本，图形模式下文本和图形都能显示。图形原语函数，例如_lineto( )，只能在图形视频模式下使用。模式类型可以继续按是单色还是彩色分类。
- **尺寸**——这个属性描述显示的范围。在文本模式下，这个属性是以显示文本字符的行列数描述的。在图形模式下，尺寸描述了水平方向（X）和垂直方向（Y）上的像素数。有时候也称作显示分辨率。
- **页面**——指在特定视频模式下可用的显示页面。每个显示页面可被当作独立的屏幕。虽然所有的页面同时存在于内存中，通常在某一时刻仅有一个页面能被显示。
- **颜色**——颜色的数量和种类也取决于选择的视频模式。对于单色图形模式，颜色的数量指的是灰度的数量。老的黄褐色或绿色单显在此不进行讨论。

3.1.2 视频模式折衷关系

在为应用程序选择视频模式时，了解显示器配置属性之间的关系是很重要的。下面给出了这些关系及其折衷关系。

- **显示分辨率和可用颜色**——在视频图形模式下(例如 CGA 和 EGA)，可用的颜色数随着显示分辨率的提高而减少。这个折衷关系是因为视频卡的内存限制。由于高分辨率显示需要较多的内存，能用于描述颜色的位就少了。所以，可用的颜色数减少了。
- **显示分辨率和可用页面的数量**——如同前面的例子，每次显示增加的内存需求减少了可用的页面数量。这成为提供多页显示的 EGA 和 VGA 图形模式的限制因素。这个折衷关系在应用程序需要在不同的复杂图像间快速切换时成为一个因素。这种类型的应用程序的一个例子是游戏 COBRA(第十五章)。由于它要求笨拙地频繁重绘整个屏幕，最好通过维持足够数量的显示页来避免这一情况。如果主要关心屏幕上项目的密集度，就应当减少应用程序需要的页面数以提高分辨率。当然，如果应用程序仅需要一个显示页，使用最高的分辨率是需要三思而行的。

作为新游戏的开发前期计划的一部分，读者应当考虑所有这些属性以决定游戏的真正需求。一旦完成了这个工作，选择适当的视频模式就相对简单了。

3.1.3 设置视频模式

一旦决定了合适的视频模式，下一步就是编写得到该模式的代码。在 Microsoft C 中，选择视频模式的函数是_ setvideomode( )。

```
#include<graph.h>
short _setvideomode(short mode);
```

注：mode 是在表 3-1 中定义的视频模式和初始化视频模式。

表 3-1 _SETVIDEOMODE( )的参数表

常 量	模 式	类型[1]	适配器[2]	尺 寸	颜 色[3]	页 面
_TEXTBW40	0	M/T	CGA	40 列 25 行	32	8
_TEXTC40	1	C/T	CGA	40 列 25 行	32	8
_TEXTBW80	2	M/T	CGA	80 列 25 行	32	8
_TEXTC80	3	C/T	CGA	80 列 25 行	32	8
_MRES4COLOR	4	C/G	CGA	320(x) 200(y)	4	1
_MRESNOCOLOR	5	C/G	CGA	320(x) 200(y)	4	1
_HRESBW	6	M/G	CGA	640(x)200(y)	2	1
_TEXTMONO	7	M/T	MDPA	80 列 25 行	32	8
_HERCMONO	8	M/T	HGC	80 列 25 行	32	8
_MRES16COLOR	13	C/G	EGA	320(x)200(y)	16	8
_HERES16COLOR	14	C/G	EGA	640(x)200(y)	16	4
_ERESNOCOLOR	15	C/G	EGA	640(x)350(y)	4	2

续表

常　量	模　式	类型[1]	适配器[2]	尺　寸	颜　色[3]	页　面
_ERESCOLOR	16	C/G	EGA	640(x)350(y)	16	2
_VRES2COLOR	17	C/G	VGA	640(x)480(y)	2	1
_VRES16COLOR	18	C/G	VGA	640(x)480(y)	16	1
_MRES256COLOR	19	C/G	VGA	320(x)200(y)	256	1

1) 类型:M/T——单色文本模式;
C/T——彩色文本模式;
C/G——彩色图形模式;
M/G——单色图形模式。
2) 适配器:CGA——彩色图形适配器;
EGA——增强图形适配器;
HGC——大力神图形兼容适配器;
VGA——视频图形适配器;
MDPA——单色适配器。
3) 单色模式下的彩色指的是灰度。

_setvideomode()函数通过 mode 型参数初始化视频模式。如果函数调用成功,返回值等于文本行数;返回值为 0 表明失败。这里必须要注意到使用各种视频模式的能力取决于计算机的图形适配卡类型,它跟具体的计算机属于哪一代产品有较大关系。在比较新的计算机中,SVGA 是很普遍的,而 VGA 通常是标准配置。在比较老的 PC 中,CGA 和 EGA 较常见。老的 PC 可以通过购买 VGA 卡和监视器而升级,大概需要 250 美元。不论系统的配置是什么,在开始策划游戏时彻底了解它的限制是很重要的。

3.1.4 分割显示器

设置好视频模式以后,就可以通过把显示器分成不同的观察点或窗口而进一步配置它。如果需要频繁地清除显示器的一部分或改变显示器的一个子区域的坐标系统,这个特性就很方便了。可用于分割显示器的两个函数是_setviewport( )和_setwindow( )。

```
#include <graph.h>
void_setviewport(short x1,short y1,short x2,short y2);
注:x1,y1 定义视口的左上角。
   x2,y2 定义视口的右下角。
```

_setviewport( )函数创建一个由参数 x1、y2、x2、y2 定义的物理坐标限制的逻辑视口。视口的原点(0,0)在视口的左上角(位置 x1、y1)。在这个功能调用后,后继的图形函数调用

使用视口原点为参照。

```
#include <graph.h>
    short _setwindows (short finvert,double wx1,double wy1,double wx2,double wy2);
```

注： finvert 是个布尔参数(TRUE,FALSE),它指出窗口原点的位置。

wx1,wy1 定义视口的左上角。

wx2,wy2 定义视口的右下角。

_setwindow( )函数和_setviewport( )函数相似,但有一个重要的不同之处。_setwindow( )定义的坐标范围被逻辑地映射到当前选择的视口的物理坐标空间。所以,就有可能定义一个逻辑坐标系统,它的范围和那些使用物理显示范围的坐标系统完全不同。另外,finvert 参数指明窗口中原点的位置。如果 finvert 为 TRUE,原点就在 wx1,wy2。这意味着 y 轴增加的方向是从屏幕的底部到顶部。如果 finvert 为 FALSE,原点就在 wx1,wy1。

程序 3-1 描绘了怎样使用_setviewport( )和_setwindow( )来分割显示器。在这个例子中,建立了三个使用不同坐标系统的显示分区。一个是缺省视口,一个是使用_setviewport( )设置的定制视口。最后一个使用_setwindow( )。在每种情况下都从屏幕分区的一端到另一端画一条线。在运行这个程序时,读者将会发现(图 3-1)线的位置各不相同,原因是从逻辑坐标系统(由函数调用指定)到物理坐标系统(实际显示在屏幕上的)的映射各不相同。在用_setviewport( )创建的视口中,线在 X 轴上与物理原点有 1/3 显示范围的偏移。在用_setwindow( )创建的视口中,线的方向变了,因为 finvert 参数被设为真。另外,这个分区的逻辑坐标超出了视频模式下可用的物理坐标。

程序 3-1　显示分割技术

```
#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <conio.h>

#define SETUP_DISPLAY                                    \
  {                                                      \
        /* Set the video mode.                          */\
                                                         \
        _setvideomode(_MRES16COLOR);                     \
        _displaycursor(_GCURSOROFF);                     \
                                                         \
        /* Set the colors.                              */\
                                                         \
        _setbkcolor(_LIGHTBLUE);                         \
        _setcolor(WHITE);                                \
  }
#define PRINT_HEADER                                     \
  {                                                      \
        _settextposition(20,6);                          \
```

```
        printf("PRESS ANY KEY TO EXIT TO DOS");                    \
    }

#define EXIT_TO_DOS                                                \
    {                                                              \
        getch();                                                   \
        _setvideomode(_DEFAULTMODE);                               \
    }

#define EXTENT_X            319 /* X-axis size in pixels.        */
#define EXTENT_Y            199 /* Y-axis size in pixels.        */
#define WHITE 7 /* Color index value for WHITE.                   */
/*                                                                 */
/* Example 3-1                                                     */
/* This program demonstrates the use of the functions:            */
/*          _setviewport();                                        */
/*          _setwindow();                                          */
/*                                                                 */

main ()
{
    SETUP_DISPLAY

    PRINT_HEADER
    /* Draw the line in the default viewport.                      */
    _rectangle(_GBORDER,0,0,EXTENT_X/3,199);
    _moveto(0,0);
    _lineto(EXTENT_X/3,199);
    /* Draw the line in the customized viewport.                   */
    _setviewport(EXTENT_X/3+1,0,EXTENT_X/3*2,199);
    _rectangle(_GBORDER,0,0,EXTENT_X/3,199);
    _moveto(0,0);
    _lineto(EXTENT_X/3,199);
    /* Create a window inside a new viewport.                      */
    /* Note that while the window occupies a subset of the         */
    /* display, its coordinates are scaled to occupy the full      */
    /* display.                                                    */

    _setviewport(EXTENT_X/3*2+1,0,EXTENT_X/3*3,199);
    _setwindow(1,-500.0,-500.0,500.0,500.0);

    /* Draw a line inside a rectangle to show the window's new     */
    /* coordinate system relative to the viewport's.               */

    _rectangle_w(_GBORDER,-500,-500,500,500);
    _moveto_w(-500,-500);
    _lineto_w(500,500);


    /* Display until a key is pressed.                             */
```

```
    EXIT _ TO _ DOS
}
```

使用_setviewport( )或_setwindow( )来分割显示器的一个优点是显示器的某个指定部分可被清除而不会影响剩余部分。通常,视频模式在应用程序中只选择两次,一次在开始而另一次在结束(把它恢复到执行程序前的值)。频繁地改变视频模式会产生令人厌烦的结果——清除掉软件以前创建的图形。如果需要简单地清除一个(或一部分)单独的显示页,_clearscreen( )函数是比setvideomode( )更好的选择。_clearscreen( )格式的例子如下所述:

```
#include <graph.h>
void _far _clearscreen(short area);
```

注: area 必须是以下常量之一:

_GCLEARSCREEN——清除活动显示页

_GVIEWPORT——清除当前视口

_GWINDOW——清除当前窗口

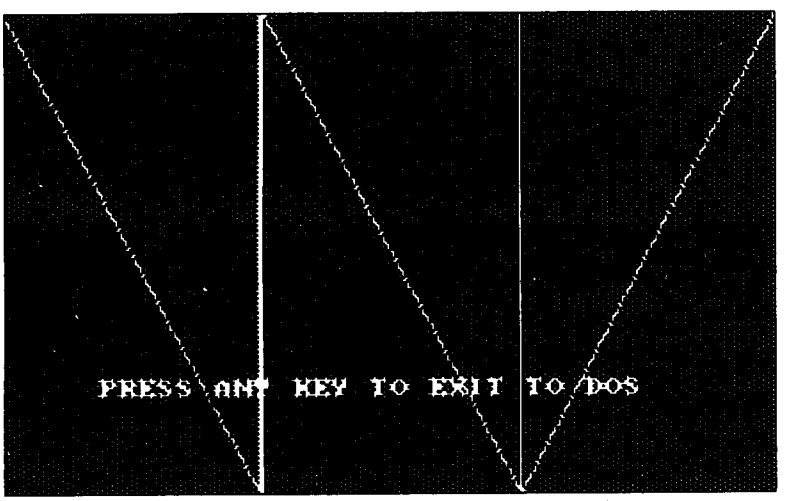

图 3-1 分割显示器

_clearscreen( )函数将清除选择的显示区(通过在其中填充当前背景色),并把光标位置复位到显示器的左上角(0,0)。

3.1.5 背景色选择

配置好显示器后,下一个逻辑步骤是选择色彩。基本上有两种色彩:背景和前景。首先看看怎样使用背景色。这可以使用_setbkcolor( )函数设置。

```
#include <graph.h>
long _far _setbkcolor(long color);
```

注:color 指出表 3-2 中的某种用户需要的颜色。

表 3-2　颜色参数表

颜　色	前景索引	背景常量
黑色	0	_BLACK[1]
蓝色	1	_BLUE
绿色	2	_GREEN
青色	3	_cYAN
红色	4	_RED
洋红色	5	_MAGENTA
褐色	6	_BROWN
白色	7	_WHITE
灰色	8	_GRAY
淡蓝色	9	_LIGHTBLUE
淡绿色	10	_LIGHTGREEN
淡青色	11	_LIGHTCYAN
淡红色	12	_LIGHTRED
淡洋红色	13	_LIGHTMAGENTA
黄色	14	_LIGHTYELLOW
亮白色	15	_BRIGHTWHITE

1)黑色是缺省背景色;这个缺省值可以用_setbkcolor( )函数改变。

这个函数使所有的背景点变为选择的颜色,而前景中画的任何东西都不受影响。该函数返回以前的背景色。_setbkcolor( )的作用在程序 3-2 中演示。执行这个例子时,每次击键都会引起背景色改变。这时,读者将会看到背景色的改变绝不会影响前景色。在使用_setbkcolor( )函数时,一定要记住使用后面的表 3-2 中给出的常量,因为这个函数要求输入一个长型(long)参数。与此不同的是,前景色使用索引值选择颜色。因此,背景色和前景色的输入参数是不兼容的。

程序 3-2　背景色变化

```
#include <conio.h>
#include <graph.h>

#define SETUP_DISPLAY                                    \
  {                                                      \
        _setvideomode(_HRES16COLOR);                     \
        _displaycursor(_GCURSOROFF);                     \
  }
#define PRINT_HEADER                                     \
  {                                                      \
        _settextposition(2,25);                          \
        _outtext("PRESS <ENTER> TO ADVANCE COLOR");      \
```