游戏概述

我想模仿的是FC游戏《火箭车》，不过实现的仅仅是最基本的游戏模式，完整的关卡和这个游戏最精髓的撞车后车辆失控的机制并没有做出来。

架构设计

我基本上使用了实验指南中给的主函数框架，另外直接使用了原来的示例代码中的函数`draw_tile()`。

其他自定义的函数功能如下：

global_initial()	完成io设备，分辨率，绘制游戏开始界面的工作
draw_bdr/line/car()	绘制双黄线/虚线/汽车
crash()	判断两辆车是否相撞

部分实现

1. 关于给整个屏幕上色，如果直接使用在(0,0)处绘制一个长为w,宽为h的区域会因内存不够而失败。我的改进是用循环不停绘制一个10*10的区域直至覆盖整个屏幕。
2. 关于适应不同分辨率：因为我觉得对于每种分辨率画一系列不同的像素图工作量较大，所以采取了更为直接的方法：直接在画面中间定死一个给定大小的方框作为游戏的边界。

2. 所谓的汽车是由五个矩形构成的。
3. 我设计中汽车有6个档位（包括停车），为了使得加速和减速的过程自然而不至于突变，每次按下（按住）前进/后退键会让计数器值增加1，而每20个计数器值对应一档的切换。
4. 为了实现车辆前进的效果，我的实际实现其实是让道路虚线和其他车辆后退，这两部分的实现还不太一样。对于道路虚线，我让它们的显示位置每次加上一个与速度相关的偏移量（当然是模意义下的），这样就不用对于每根线维护一个位置了。这样带来的后果是，如果程序运行的足够长（当然不太可能），偏移量会溢出。
5. 这样的实现会有另一个问题：因为画矩形函数的参数是矩形的起点，当起点处于下边界的边缘附近时，画出的线就会溢出到边界之外。当然可以用一个判断来调整此时的矩形长度，但我用了一个更直接的办法，就是每轮都把下边界以下的一定大小区域涂黑。
6. 对于其他车辆，因为可能会有横向移动，我跟踪了每辆车的位置并在每次迭代中更新。
7. 关于车辆位置的更新：我原本想只记录每辆车在每轮循环中的位置，并且在画面更新时，先把屏幕全部清空，再画上新的图像。但是因为1中所述原因，清屏是一件代价很大的事情。因此我后来采取的实现是记录每辆车的现在位置和上一轮时的位置，并在每轮循环中在上一轮位置画一辆黑色的车来实现更新位置的效果。
8. 其实可以使用链表的数据结构跟踪每一辆车，但是鉴于编程复杂度较高，以及尚未实现的`malloc()`和`free()`函数所迫，我最后采用了定长数组，即限制最多出现的车数。车的位置是否为(0,0)则恰好可以作为有效位进行判断。

遇到的问题

1. 我原本直接调用`rand()`函数来产生随机数，但是这样每局游戏的结果都是一样的。后来我发现需要用`srand()`设置随机数种子，而提供的`get_timeofday()`函数刚好提供了一个很好的种子。