Realtouch 例程·贪吃蛇

例程目的

熟悉 RTGUI 的使用，展现 realtouch 平台上的魅力。

例程简介

在我很小很小，小到我只知道 bb 机和大哥大才是我追求的目标那时，我有一个掌上游戏机。严格来说那个游戏机不是我的，因为我那时候根本就没有零花钱，是我堂哥买花了 20RMB 在俺们的镇子上买来的。那时候对这种‘高科技’很稀罕，很痴迷。经常玩的就是码砖块——更确切点讲是俄罗斯方块。在网上偶然看到一个外国人用吉他演奏了那首金典的背景音乐。童年的时光仿佛冲走一遍，我分享给了 aozima 神牛，然而接下来的并不是不是俄罗斯方块的再现版，而是本主题的开场。

神牛看到 realtouch 平台的资源魅力，果断要在 realtouch 平台上重现他那段儿时的神作——贪吃蛇，据说 aozima 在课堂上玩贪吃蛇被老师抓现行，罚站了两节课。不过当然的是，在 realtouch 上跑个贪吃蛇那确实是杀鸡用牛刀啊！不过咱还得说句实话，想到广大爱好者对 rt-thread 的支持，咱们也得回报一点力所能及的事。

内容介绍

贪吃蛇相信大家都玩过，版本万千，本例程并不打算去复制某个版本的游戏场景，完全凭那点脑海的记忆去自由发挥。

游戏元素包括：地图，蛇，食物。

游戏思路主要是在地图中随着蛇吃到的食物越多而增长的越长。

软件实现中主要使用到了 RTGUI 组件，为了在刷新每一帧图像的时候降低系统开销，软件会考虑对关键部分做出一些优化。设计之时为了更好的体现结构美，软件分为两个部分，一个部分负责 GUI 绘图(map)，另一部分主要为了提供 GUI 绘图的数据(snake)。当然 GUI 还同时接管了输入状态的处理。

注意本例程侧重的是主题，脱离主题的元素不在此介绍，例如 LCD 驱动等将不做介绍，请参考其他例程。

GUI/Map:

Map（地图）是整个游戏的接口，在软件中主要负责刷新游戏的动态。比如蛇的移动，食物的变动等。这个实现在 snake_gui.c 文件中。

因为是使用 RT-Thread module 的方式加载程序，所以整个程序的入口是 main 函数。在这里创建了一个 win，所以内容都是在这个 win 上面显示的。同时创建了一个定时器，以实现蛇的爬行，

event_handler 处理所有的事件，其中主要执行两个操作：

1. snake_draw()

   本函数用于窗口建立或从隐藏显示出来时绘制整个画面。

2. rtgui_timer_start()
   本操作用于启动定时器，让蛇不断地爬行。
3. rtgui_timer_stop()
   本操作用于停止定时器，让蛇停止爬行。用于结束或暂停游戏。
4. snake_handler()
   最主要的是本函数，用于接收所有外部输入，以控制蛇前进的方向。待下一次定时器到点后，蛇将按新输入的方向前进。

当定时器到点后，将执行定时器的超时回调函数 timeout()。本函数中，会根据新的状态更新画面，snake 核心中做了优化，直接提供了此次更新的部分，这样，将只更新改变过的地方，以提高性能。

Snake:

这一部分主要实现游戏的业务逻辑，并输出游戏动态数据提供给 GUI，例如蛇的移动，食物的生成等。这部分实现在 snake.c/.h 文件中。

snake.c/.h

这节内容主要围绕函数来讲解，以游戏中的逻辑顺序依次来分析。个人文笔有限，所以一直坚持 read the fucking source code!

```
map_t* map_init(rt_uint32_t width, rt_uint32_t heigth);
```

此函数初始化创建一个指定大小的掩码地图。分析这个函数必须要了解 map_t 这个结构的每一个字段的含义。

```
typedef struct
{
   rt_int32_t width;      // max x
   rt_int32_t height;     // max y
   rt_uint8_t *range;      // map, map->range[y * map->width + x]
   point_t snake_flush[2];
   point_t food_flush[1];
}map_t;
```

`Width/height` 为当前地图的大小。

`Range` 为掩码地图的句柄，记录当前地图中每一个区域的状态（蛇，食物，空区）。

`Snake_flush/food_flush` 记录当前的动态点，既被更新了的点。

函数的功能也就是建立一份包括以上信息的数据结构。`Range` 字段采用一维主要是为了开辟内存的方便。虽然带来了一点难以接受的麻烦，但是也并非不可。在 `GUI` 中的绘图中将会用到这个掩码地图的信息进行针对性的绘制，对于没有必要更新绘制的地方将节省开销。在 `range` 中会有三种状态值，分别为 `NORMAL`（空区）、`FOOD`（食物）、`OVER`（蛇）。在初始化时全部是 `NORMAL`。

```
rt_bool_t snake_init(const point_t *start, const int length, const SNAKE_DIR dir, map_t *map);
```

这个函数主要任务是在 map 中指定的点构造一条指定长度的蛇。理解这个函数我想只介绍这几个函数的参数即可。

Start 蛇的起始位置坐标。

Length 蛇的长度，将要求小于地图在 dir 指定的方向上的跨度，例如上下运动的话将会要求小于地图的高度，左右运动的话将要求小于地图的宽度。

Dir 指定蛇的初始运动方向——上下左右。

Map 也即 map_init 初始化出来的掩码地图句柄。在此函数中将会把蛇所在的区域填上 OVER 值。

```
rt_bool_t food_init(map_t *map, rt_uint32_t max_num);
```

这个函数主要是在地图中创建出食物。在食物所在的点上会将 map 的值修改为 FOOD。

```
SYS_STE snake_step(SNAKE_DIR dir, map_t *map);
```

这个函数主要是游戏的主循环。每隔一定时间调用一次这个函数来获取当前游戏的进度。游戏中蛇的移动信息就是通过这个函数来实现的更新。这是这个 snake 模块实现的核心部分。包括对方向的判断，运动的判断，穿墙的判断等。函数的返回值更说明了当前游戏的进展状态，例如是咬到了自身还是吃到了食物。Dir 参数指定了这一步蛇的运动方向，如果 dir 的方向和上次的方向相反，则不会改变蛇的运动方向，因为蛇不会从向前突然改变为向后运动。关于这个方向的判断是 dir_adjust 函数实现的。dir_adjust 函数很简单，很容易理解。

Snake 模块中有一个记录蛇身的列表 snake_head，关于蛇的信息就记录在这个列表中，蛇会是由很多个节点组成的列表，这样做很形象，也符合设计的需要。

由于蛇每次会移动一步，所以在地图中蛇会可能改变的是蛇头和蛇尾。

```
tail = rt_list_entry(snake_head.prev, snake_t, list);
head = rt_list_entry(snake_head.next, snake_t, list);
```

取到蛇头和蛇尾，以操作这两个节点的更新。通常情况下如果下一步没有遇到食物则将尾节点取出来修改坐标后用作蛇头节点。这样既避免了动态开辟内存，又能快速利用现有的资源而提高效率。

```
switch (dir)
{
case SNAKE_DIR_UP:
case SNAKE_DIR_DOWN:
    node.y = head->body.y + (dir == SNAKE_DIR_DOWN ? -1 : 1);
    break;
case SNAKE_DIR_LEFT:
case SNAKE_DIR_RIGHT:
    node.x = head->body.x + (dir == SNAKE_DIR_RIGHT ? 1 : -1);
    break;
}
```

以上代码片段将会更具 dir 方向来算出下一步的坐标。这里需要明白一点是计算方法和屏幕坐标要配合，一开始我把坐标原点定在了左上角，所以就出现了 90° 的旋转偏差，哈哈。咱们的屏幕坐标和数学中的坐标系相同，原点在左下角。

```
across_XY(&node, map);
```

这个函数将对坐标做进一步的处理，主要是处理穿墙的效果。这个函数容易理解，不做多解释。

```
static SYS_STE node_update(snake_t *tail, const point_t *node, map_t *map);
```

这个函数将利用新的坐标点来更新下一步的状态，是迟到的食物还是咬到了自身或者正常前行？如果吃到了食物，那么就得将当前食物的点变成蛇的蛇头使蛇增长。

```
if (FOOD == map->range[node->y * map->width + node->x])
{
    // 吃一个食物增加一个节点
    snake_t *new = (snake_t*)rt_malloc(sizeof(snake_t));
    if (!new)
        return NORMAL;

    pos[0] = *node;
    new->body = *node;
    rt_list_insert_after(&snake_head, &new->list);
}
```

同时这个函数还记录了被更新了的坐标点。如果是正常运动那么将会像前面所述的，将尾节点修改后放到列表前作为蛇头节点。

```
else if (NORMAL == map->range[node->y * map->width + node->x])
{
    // 将尾巴修改后拿到头部，其他不变
    rt_list_remove(&tail->list);
    map->range[tail->body.y * map->width + tail->body.x] = NORMAL;

    pos[0] = *node;
    pos[1] = tail->body;

    tail->body = *node;
    rt_list_insert_after(&snake_head, &tail->list);
}
```

如果是咬到了蛇的身体则什么也不处理直接返回当前的运行状态。更新到的点要标记好当前的状态 NORMAL(走过的点)或者 OVER(走到的点)。

文中还有其他一些函数，主要是一些析构资源的函数，不做过多解释。其中有一个函数 snake_restart 是当蛇到达一定长度时重新开始游戏。