UEFI Development Exploration 78- YIE001PCIe Development Board (11 Snake)

原创 luobing4365 Posted on 2021-02-14 12:19:11 Views: 684 Collection 3 Likes 4

Category Column: UEFI Development Article Tags: UEFI Low-level application development PCI-E Snake Option ROM

YIE001 PCIe development board Snake

It's time to implement an interesting project. I chose to implement the Snake game under UEFI.

It would be difficult to debug if it is implemented directly on Option ROM. Therefore, I first implemented the UEFI application of Snake. After successful debugging, I ported it to YIE001.

1. Snake framework design

Considering that the code will eventually be ported to YIE001, I try to reduce the number of protocols used. This is mainly because I find that different hosts support different protocols when running Option ROM during normal development. For example, the platform I am currently testing does not support SimplePointer Protocol (mouse).

To simplify the thinking about the program, we designed the snake to be composed of rectangular blocks. The main problems that need to be solved include: (1) map design; (2) data structure
design of the snake ; (3) how to judge whether the snake hits the wall; (4) how to judge whether the snake bites itself; (5) random appearance of food for the snake to eat.

In fact, after solving problems 1 and 2, basically all the subsequent problems can be easily solved.

We designed the map, snake and food as shown in Figure 1:

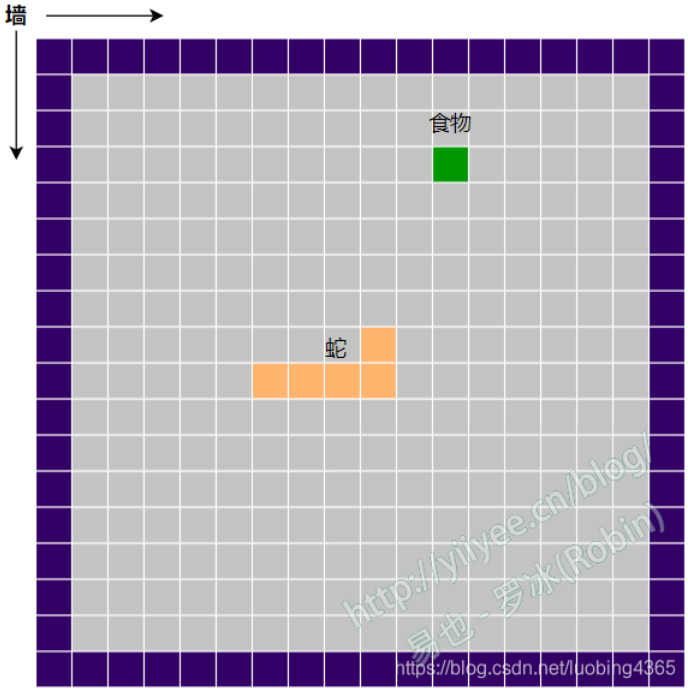

Figure 1 Concept map of the greedy snake

The snake itself is made up of rectangular blocks, and the entire map is also made up of rectangular blocks. Therefore, during the movement of the snake, we only need to deal with the color change of the rectangular blocks.

In addition, this design makes it easy to determine whether the snake has hit a wall, eaten food, or eaten itself. This can be done by simply determining whether the next rectangular block in its direction of travel is a wall, food, or its own rectangular block.

2 Code Implementation

The program is basically written according to the conceptual diagram in Figure 1. We number the rectangular blocks of the entire map from left to right and from top to bottom, starting from 0.

Therefore, we can use the rectangular block number or the first pixel coordinate (upper left corner) of the rectangular block to determine whether the snake hits the wall or bites itself. In fact, knowing the rectangular block number, we can calculate the coordinate of the first pixel in its upper left corner. These two methods are used in the program in a mixed way, and there is no difference.

The following is a detailed explanation of the code implementation process.

2.1 Data structures, global variables and macro definitions

The data structures, global variables and macro definitions used are shown below.

```
#define X_MAP 50      //容纳多少个SnakeBlock
#define Y_MAP 50
#define SNAKEBLOCK 8
#define SNAKEBLANK 2

#define CROSSWALL 1
#define BITESELF  2
#define USEREXIT  3

#define SnakeUP     1
#define SnakeDOWN   2
#define SnakeLEFT   3
#define SnakeRIGHT  4
typedef struct GREEDSNAKE //贪吃蛇的数据结构
{
    INT32 x;
    INT32 y;
    INT32 BlockNumber; //总共X_MAP*Y_MAP个SnakeBlock
    struct GREEDSNAKE *next;
}greedsnake;
twen
twen greedsnake *head,*food;  //蛇头指针，食物指针
twen greedsnake *pSnake; //遍历所用指针
twen UINT8 foodColor = BLACK;
UINT8 snakeColor = BLACK;
INT32 FoodBlocks[X_MAP*Y_MAP];  //保存可用的SnakeBlock
INT32 FoodBlockCounts;

```

```
INT32 SnakeStatus,SleepTime = 200; //运行的延时时间，ms为单位
INT32 Score=0;       //成绩，吃到一个食物则加10分
```

Among them, X_MAP and Y_MAP represent the number of rectangular blocks on the X coordinate and Y coordinate. SNAKEBLOCK represents the side length of the square rectangular block, and SNAKEBLANK represents the outside of the rectangular block. In actual drawing, the rectangular block is drawn like this:

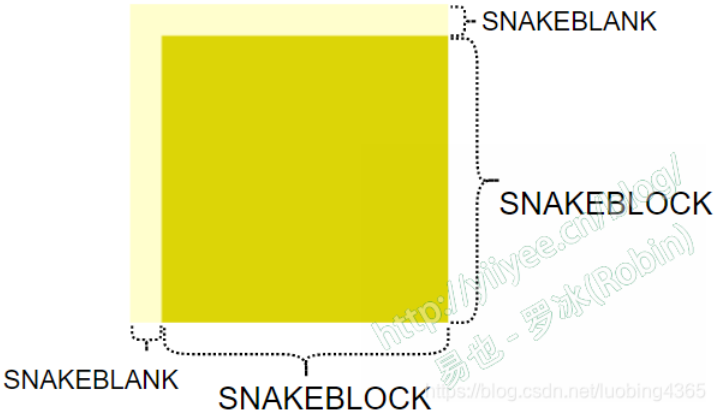

Figure 2 Drawing of rectangular blocks

When calculating, it is treated as a whole. The x coordinate and y coordinate in the greedsnake data structure refer to the coordinates of the first pixel in the upper left corner of the entire rectangular block.

The BlockNumber and x, y coordinates in the greedsnake data structure can be converted to each other. The conversion formula is:

```
x = ((BlockNumber) % X_MAP)  *(SNAKEBLOCK + SNAKEBLANK);
y = ((BlockNumber) / X_MAP)  *(SNAKEBLOCK + SNAKEBLANK);
```

2.2 Drawing the map and initializing the snake

The drawing of a rectangular block can be realized by the rectblock() function provided in Graphic.c . This function is encapsulated in the SnakeElement() function in the code.

The functions for implementing map drawing and snake initialization are as follows:

```
/**
  绘制地图

  @param  VOID
  @retval VOID
**/
VOID CreateMap(VOID)
{
  INT32 xnum;
  INT32 ynum;
  INT32 counter=0;

  for(xnum=0;xnum<X_MAP;xnum++)
  {
    SnakeElement(xnum*(SNAKEBLOCK+SNAKEBLANK),0,BLACK);
    SnakeElement(xnum*(SNAKEBLOCK+SNAKEBLANK),(Y_MAP-1)*(SNAKEBLOCK+SNAKEBLANK),BLACK);
  }

  for (ynum = 0; ynum < Y_MAP; ynum++)
  {
    SnakeElement(0, ynum*(SNAKEBLOCK + SNAKEBLANK), BLACK);
    SnakeElement((X_MAP - 1)*(SNAKEBLOCK + SNAKEBLANK), ynum*(SNAKEBLOCK + SNAKEBLANK), BLACK);
  }
  for(ynum=1;ynum<Y_MAP-1;ynum++)
    for(xnum=1;xnum<X_MAP-1;xnum++)
    {
      FoodBlocks[counter]=ynum*X_MAP + xnum;
      counter++;
    }
  FoodBlockCounts=counter;
}

/**
  初始化蛇身

  @param  VOID
  @retval VOID
**/

```

```
VOID InitSnake(VOID)
{
    greedsnake *tail;
    INT32 i;
    tail = (greedsnake*)AllocateZeroPool(sizeof(greedsnake));//从蛇尾开始，头插法，以x,y设定开始的位置//
    tail->x = (X_MAP/2)*(SNAKEBLOCK + SNAKEBLANK);
    tail->y = (Y_MAP/2)*(SNAKEBLOCK + SNAKEBLANK);
    tail->BlockNumber = (Y_MAP/2) * X_MAP + (X_MAP/2);
    tail->next = NULL;

    for (i = 1; i <= 4; i++)  //4个SnakeBlock
    {
        head = (greedsnake*)AllocateZeroPool(sizeof(greedsnake));
        head->next = tail;
        head->x = (X_MAP/2)*(SNAKEBLOCK + SNAKEBLANK) + i*(SNAKEBLOCK + SNAKEBLANK);
        head->y = (Y_MAP/2)*(SNAKEBLOCK + SNAKEBLANK);
    head->BlockNumber = tail->BlockNumber + 1;
        tail = head;
    }
    while (tail != NULL)//从头到尾，输出蛇身
    {
    SnakeElement(tail->x, tail->y,snakeColor);
        tail = tail->next;
    }
}
```

The snake itself is stored in a linked list. Every time a snake body is added, a part of the memory is requested from the system. Therefore, in subsequent programming, it is necessary to pay attention to the release of linked list memory at any time.

2.3 Hitting the wall or biting oneself

As mentioned before, this can be determined by the x, y coordinates or BlockNumber in the data structure. The code is as follows:

```
/**
  判断是否咬到自己

  @param  VOID
  @retval 1     咬到自己
          0     没有咬到
**/
UINT8 BiteSelf()
{
    greedsnake *self;
    self = head->next;
    while (self != NULL)
    {
        if (self->x == head->x && self->y == head->y)
        {
            return 1;
        }
        self = self->next;
    }
    return 0;
}
/**
  不能穿墙

  @param  VOID
  @retval 1   穿墙了
          0   没有穿墙
**/
UINT8 NotCrossWall(VOID)
{
  UINT32 BlockX,BlockY;
  BlockX = ((head->BlockNumber) % X_MAP);
  BlockY = ((head->BlockNumber) / X_MAP);
  if((BlockX==0) || (BlockX==(X_MAP-1)) || (BlockY==0) || (BlockY==(Y_MAP-1)))
  {
    EndGameFlag =  CROSSWALL;
    return 1;
  }
  return 0;
}
```

Self-biting is determined by x and y coordinates, while wall collision is determined by BlockNumber.

2.4 Random Food

The global variables FoodBlocks, FoodBlockCounts, and foodColor are all used to generate random food.

The FoodBlocks array contains the serial numbers of all SnakeBlocks except the wall. Due to the existence of the wall, the SnakeBlock used to produce food is not continuous, so this little trick is used.

And foodColor is used to indicate the color of food. In the actual program, I added a small function that when the snake eats the food, its own color will become the same as the food color.

The function that generates random numbers is robin_rand(), which has been introduced in previous blogs. It is a pseudo-random number generator .

The code for generating random food is as follows:

```
/**
  随机出现食物

  @param  VOID
  @retval VOID
**/
VOID RandomFood(VOID)
{
  greedsnake *tempfood;
  INT32 randNum;

  randNum = robin_rand() % FoodBlockCounts;  //不能超过食物可出现位置的总数
  tempfood = (greedsnake*)AllocateZeroPool(sizeof(greedsnake));
  tempfood->BlockNumber = FoodBlocks[randNum];
  //递归判断蛇身与食物是否重合
  pSnake=head;
  while(pSnake == NULL)
  {
    if(pSnake->BlockNumber == FoodBlocks[randNum])  //重合了
    {
      FreePool(tempfood);  //释放内存
      RandomFood();     //递归产生食物
    }
    pSnake = pSnake->next;
  }

  tempfood->x = ((tempfood->BlockNumber) % X_MAP)  *(SNAKEBLOCK + SNAKEBLANK);
  tempfood->y = ((tempfood->BlockNumber) / X_MAP)  *(SNAKEBLOCK + SNAKEBLANK);
  tempfood->next = NULL;

  food = tempfood;
  foodColor = (UINT8)(robin_rand() % 10);  //共10个颜色可选
  if(foodColor == DEEPBLUE)
    foodColor = BLACK;
  SnakeElement(food->x,food->y,foodColor);
}
```

2.5 Snake Movement

The snake can only move in one direction, and the next BlockSnake of its head can be food, a wall, or an empty BlockSnake. The implementation code of its movement is:

```
/**
  蛇的移动

  @param  VOID
  @retval 1     撞到自己或墙了
          0     啥事没有
**/
UINT8 SnakeMove(VOID)
{
  greedsnake *nexthead;

  if(NotCrossWall())
    return 1;

  nexthead = (greedsnake*)AllocateZeroPool(sizeof(greedsnake));

  switch(SnakeStatus)
  {
    case SnakeUP:
      nexthead->BlockNumber = head->BlockNumber - X_MAP;
      break;
    case SnakeDOWN:
      nexthead->BlockNumber = head->BlockNumber + X_MAP;
      break;
```

```
    case SnakeLEFT:
      nexthead->BlockNumber = head->BlockNumber -1;
      break;
    case SnakeRIGHT:
      nexthead->BlockNumber = head->BlockNumber +1;
      break;
    default:
      break;
  }
  if(nexthead->BlockNumber == food->BlockNumber)  //找到食物!
  {
    nexthead->x = food->x;
    nexthead->y = food->y;
    nexthead->next=head;
    head=nexthead;
    pSnake = head; //准备遍历
    snakeColor = foodColor;
    while(pSnake != NULL)
    {
      SnakeElement(pSnake->x,pSnake->y,snakeColor);  //变成食物的颜色
      Delayms(50);
      pSnake=pSnake->next;
    }
    Score+=10;
    RandomFood();
  }
  else
  {
    nexthead->x = ((nexthead->BlockNumber) % X_MAP)  *(SNAKEBLOCK + SNAKEBLANK);
    nexthead->y = ((nexthead->BlockNumber) / X_MAP)  *(SNAKEBLOCK + SNAKEBLANK);
    nexthead->next = head;
      head = nexthead;

    pSnake = head; //准备遍历
    while(pSnake->next->next !=NULL)
    {
      SnakeElement(pSnake->x,pSnake->y,snakeColor);
      pSnake=pSnake->next;
    }
    SnakeElement(pSnake->next->x,pSnake->next->y,DEEPBLUE);  //消除，即变成背景色
    FreePool(pSnake->next);  //释放内存
    pSnake->next=NULL;
  }

  if(BiteSelf() == 1)
  {
    EndGameFlag = BITESELF;
    return 1;
  }
  return 0;
}
```

2.6 Game operation and control

Considering that Option ROM may not support Event well, the program framework does not introduce Event mechanism. The program checks whether the keyboard is pressed, obtains the corresponding key code, and controls the direction of the snake. The implementation code is as follows:

```
/**
  运行游戏

  @param  VOID
  @retval VOID
**/
VOID GameRun(VOID)
{
  EFI_INPUT_KEY key={0,0};

  SnakeStatus = SnakeRIGHT;
  while(1)
  {
    CheckKey(&key);
    if((key.ScanCode==0x01) && (SnakeStatus!=SnakeDOWN)) //UP key
    {
      SnakeStatus=SnakeUP;
    }
    else if((key.ScanCode==0x02) && (SnakeStatus!=SnakeUP))   //DOWN key
    {
      SnakeStatus=SnakeDOWN;
twen
twen
```

```
twen      }
twen      else if((key.ScanCode==0x03) && (SnakeStatus!=SnakeLEFT))   //RIGHT key
  25      {
  26        SnakeStatus=SnakeRIGHT;
  27      }
  28      else if((key.ScanCode==0x04) && (SnakeStatus!=SnakeRIGHT))   //LEFT key
  29      {
  30        SnakeStatus=SnakeLEFT;
  31      }
  32      else if(key.ScanCode==0x17)   //ESC
  33      {
  34        EndGameFlag = USEREXIT;
  35        break;
  36      }
  37      Delayms(SleepTime);
  38      if(SnakeMove())
  39        break;
  40    }
  41    EndGame();
}
```

The EndGame() function is used to end the game and display the user's score. I won't post the specific implementation here, but the download link for the code is given at the end of the blog.

3 Testing

Code compilation:

```
1 C:\UEFIWorkspace>build -t VS2015x86 -p RobinPkg\RobinPkg.dsc \
2 -m RobinPkg\Applications\Snake\Snake.inf -a IA32
```

Tested in UEFI Shell, the results are as follows:

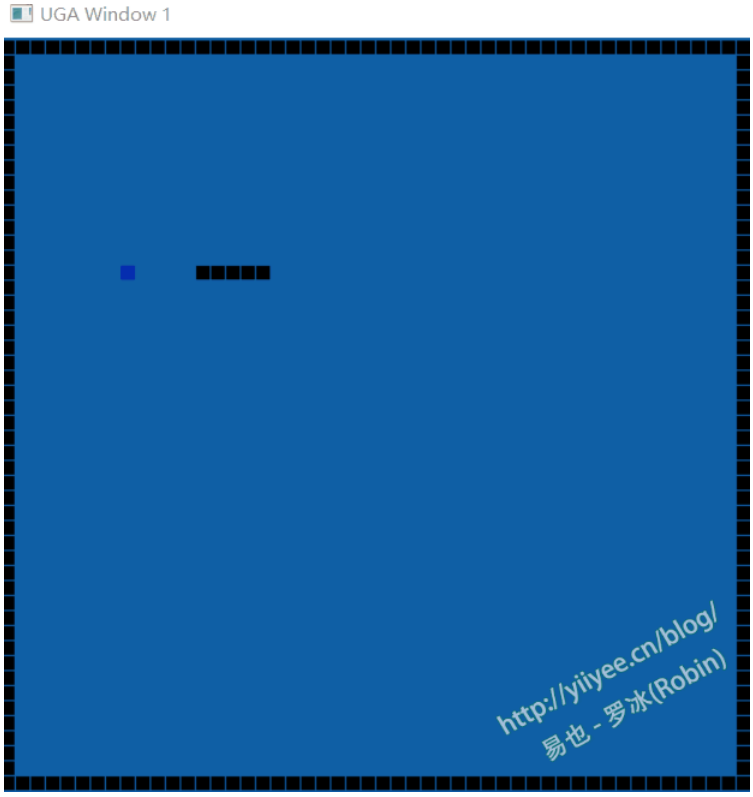

Figure 3 Running the Snake game

Gitee address: https://gitee.com/luobing4365/uefi-explorer

Project code is located in: /FF RobinPkg/ RobinPkg /Applications/Snake