# 遇到的问题以及解决方法

|  |  |
| --- | --- |
| 问题 | 如何让飞机在键盘事件的控制下一帧一帧地去左右移动？ |
| 解决方法 | 1. 需要独立出来一个“键盘对象”，并且监听 document 对象的键盘事件：keyup 以及 keydown 2. 一帧一帧地去移动：**通过更新变量，然后在动画执行函数里面再去移动飞机**。   即：按下左箭头，就设定 pressedLeft = true; 然后在动画的执行函数里面，判断是不是pressLeft，如果是则飞机向左绘制。当抬起左箭头，就要将pressedLeft = false，阻止飞机继续向左移动 |
| 中途遇到的问题以及解决方法 | 其实子弹一帧一帧发射的原理与飞机一样，但是如果我按以下代码写的话，会出现一个bug：我只按一下空格键就抬起来了，正常来说，应该只发射一枚子弹，但是，却发射了3枚子弹。  if (board.pressedSpace) {  plane.shoot(); }  一开始的想法是：在每一次创建子弹前将子弹数组 bullets 清空，解决了上述问题，但又出现了另外一个bug：不抬起空格键，只能发射一枚子弹，不符合项目要求（飞机能连续发射子弹）  以下是正确思路：  if (board.pressedSpace) {  board.pressedSpace = false;  plane.shoot(); }  原因：【仍待解决】 |
| 代码 |  |

|  |  |  |
| --- | --- | --- |
| 问题 | 如何按一定间隔绘制多行敌人？ | |
| 解决方法 | 思路：每一关卡的敌人的个数 = 7n（n是第几关），敌人的排列布局就是矩阵。因此，敌人的行数由“游戏的关数”决定  就像排序一样，利用两层循环，外循环控制有多少行敌人，内循环控制有多少列敌人，这样就形成了敌人的矩阵布局。  【注：每一关卡创建敌人前，都需要将原来的敌人数据清空】 | |
| 中途遇到的问题以及解决方法 | | 无 |
| 代码 | // 每次创建敌人前，先将原来的敌人清空 this.enemies = []; // 共6关，每一关以 7n 创建敌人 // 外循环控制行数，内循环控制每一行有多少个敌人 for (var i = 0; i < level; i++) {  for(var j = 0; j < enemyNumPerLine; j++) {  var enemy = {  x: padding + j \* (enemySize + enemyPerGap),  y: padding + i \* (enemySize + enemyPerGap),  size: enemySize,  speed: speed  };  this.enemies.push(new *Enemy*(enemy));  } } | |

|  |  |
| --- | --- |
| 问题 | 如何让一堆的敌人做反弹运动？ |
| 解决方法 | 思路：希望敌人在画布的左/右边界处反弹回来，就好像边界处有一堵墙。因此，需要检查敌人是否超过了画布的左/右边界，如果一到达边界处，则反向改变敌人的运动方向。  需要知道左右边界 + 判断逻辑  // 飞机的左边边界：画布的左padding this.planeMinX = cP; // 飞机的右边边界：整个画布的宽 - 画布的右padding - 飞机本身的宽度 this.planeMaxX = this.canvasWidth - cP - option.planeSize.width; |
| 中途遇到的问题以及解决方法 | 因为有很多个敌人，如何做到同时控制随意一个敌人到达边界，就所有敌人同时向下移动一格？  这里也是采用 更新变量 的方法：基于选择排序的思想，从数组的开头开始，将第一个元素和其他元素作比较，找到最大/小 x 坐标。当然，这里并不需要排序，只要一层遍历即可。  【一次遍历，两种比较，找到最大/小 x 坐标 并存到相应变量中】  function *getBoundary*(*elems*) {  var currentX1,  currentX2;  *elems*.forEach(function (*item*) {  if (!currentX1 && !currentX2) {  currentX1 = *item*.x;  currentX2 = *item*.x;  } else {  // 寻找最小值  if (*item*.x < currentX1) {  currentX1 = *item*.x;  }   //寻找最大值  if (*item*.x > currentX2) {  currentX2 = *item*.x;  }  // console.log(item.x, currentX1, currentX2);  }  });   return {  minX: currentX1,  maxX: currentX2  }; }  在 game.updateEnemies()中：  变量isDown控制敌人能否向下移动一格的标识，通过边界的比较，如果敌人超过左右边界，则将isDown = true，表示敌人可以向下移动一格。  / 通过边界的比较，修改敌人水平移动的方向  if (boundary.minX < this.enemyMinX || boundary.maxX > this.enemyMaxX) {  this.option.enemyDirection = this.option.enemyDirection === 'right' ? 'left' : 'right';  // 只要敌人水平移动的方向改变，就使每一个敌人都向下移动  isDown = true;  } |
| 代码 |  |

|  |  |
| --- | --- |
| 问题 | 如何实现子弹与敌人的碰撞检测？ |
| 解决方法 | 基于敌人状态去判断是否需要碰撞检测  判断条件：  1、x方向：  子弹的x 坐标 >= 敌人的 x 坐标 && 子弹的x 坐标 <= 敌人的x坐标+敌人的宽度  2、y方向：  子弹的y 坐标 >= 敌人的 y 坐标 && 子弹的y 坐标 <= 敌人的y坐标+敌人的高度  （白色的子弹永远会在每一个红色的框框之内）  这两个条件保证了子弹射到的是一个敌人的范围，不会误伤到其他敌人. |
| 中途遇到的问题以及解决方法 |  |
| 代码 | // 敌人的状态改变  switch (item.status) {  case 'exist':  if (plane.hasHit(item)) {  item.eliminate();  item.draw(\_self.context);  }  break;  case 'booming':  item.eliminate();  break;  case 'eliminated':  enemies.splice(len, 1);  \_self.score += 1;  break; }  *Plane*.prototype.hasHit = function (*enemy*) {  var bullets = this.bullets,  len = bullets.length;   // 正常从 0 开始循环遍历数组有可能会出错，因为循环过程中调用 splice 方法删除数组的项，  // 这样会影响数组后面项的序号，所以应该从数组后面开始遍历，这样就不会影响前面项  for (; len--;) {  var bullet = bullets[len],  isHitX = bullet.x >= *enemy*.x && bullet.x <= (*enemy*.x +*enemy*.size),  isHitY = bullet.y >= *enemy*.y && bullet.y <= (*enemy*.y +*enemy*.size);   // 如果子弹与敌人碰撞了，就将子弹删去  if (isHitX && isHitY) {  bullets.splice(len, 1);  return true;  }  }   return false; }; |

|  |  |
| --- | --- |
| 问题 | 那么多个敌人，如何知道敌人移动到了下边界？ |
| 解决方法 | 思路：其实只要有一个敌人到了下边界，就证明敌人没有被飞机全部消灭，那么此时游戏结束 |
| 中途遇到的问题以及解决方法 |  |
| 代码 | var lastEnemy = this.enemies[this.enemies.length - 1]; if (lastEnemy.y >= this.enemyDownLimit) {  this.endGame('failed');  // 获取最终得分  this.getFinalScore();  return; } |