**软件设计与实践II结题报告**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **一、学生姓名（列出所有组员的名字）** | | | | |
| 于雅冬 | | 李雷 | | 叶鋆郴 |
| **二、软件设计与实践II的题目** | | | | |
| Pthread多线程游戏 | | | | |
| **三、开题报告中列出的结题目标（列出标题即可）** | | | | |
| 1. 完成多线程游戏。 2. 可外接键盘，进行多人对战。 | | | | |
| **四、完成的结题目标及完成情况（要详细展开）** | | | | |
| 1. 结题目标一：完成多线程游戏。   程序编译：make all  运行：./game  编译前，建议先make clean  游戏介绍：游戏命名为多人打飞碟。游戏开始进入选择页面，按数字键1或2可选择单人或双人游戏。按键’a’,’d’,’w’ 为1号玩家左移、右移、发射子弹。按键’j’, ‘l’, ‘i’ 为2号玩家左移、右移、发射子弹。按键’Q’终止游戏。游戏过程中飞碟会按照随机的速度出现，玩家要发射子弹打消飞碟。一定时刻会出现大Boss，小心不要被它的子弹击中啦，然后打倒它吧！子弹数目为0或者飞碟逃跑数达到上限，或者被Boss子弹击中，游戏结束。和小伙伴一起玩吧~  关于线程：  主线程Main：获取键盘输入，判断玩家移动。  子弹线程rocketThreads：每次玩家发射出的子弹是由单独的线程来控制。每个子弹信息由一个rocket的数据结构来存储，该线程用于对该数据结构进行修改（或说移动），在画面上刷新并显示子弹。  飞碟线程sThread：每个出现的飞碟有单独的线程控制。飞碟信息由saucer的数据结构来存储。该线程用于对数据结构进行初始化，判断飞碟的位置并在画面刷新并显示飞碟。  创造飞碟线程saucerThread：飞碟会随时出现，由该线程来控制。该线程随时休眠并唤醒，每次唤醒就产生飞碟。  破坏者线程destroyerThread：控制Boss的线程。和飞碟不同，Boss会定时出现，该Boss若不消灭就不会从游戏画面中消失。类似于sThread，该线程用来初始化destroyer这个数据结构并在画面刷新并显示破坏者。  破坏者子弹线程destroyerRocketThread：控制破坏者发射子弹的线程。类似于rocketThreads，破坏者会不定时发射一排密集子弹，该线程用来初始化destroyerRocket数据结构，在画面刷新并显示子弹。  冲突线程collisionThread：判断子弹是否击中飞碟的线程。很明显只需在任意给定的时刻判断是否有子弹和飞碟是否在同一坐标。若有这该飞碟数据结构isAlive置为-1表飞碟消灭。  破坏者冲突线程destCollisionThread：判断破坏者子弹是否击中玩家控制的飞机。同样通过坐标来判断，若在同一坐标则发出游戏结束信息。  结束条件线程gameMonitor：时刻判断游戏结束条件，当飞碟逃离数超过上限或子弹用完，结束游戏并打印游戏结束信息。  重要临界信号量：  Mx--curses界面。由上面的线程可以看出，每次画面刷新并在屏幕上显示时都需要lock mx。否则会产生信号冲突。  Escaped--for escaping saucers。由于飞碟逃脱是共享数据，被用在许多地方，防止冲突需要信号量控制。  具体实现（个别重要函数）：   1. 主函数功能：打印选择界面，建立以下各个线程：   pthread\_create(&saucerSetup, NULL, setupSaucer, NULL);  pthread\_create(&destroyerThread, NULL, sendDestroyer, &destShip);  pthread\_create(&collisionThread, NULL, collisionDetection, NULL);  pthread\_create(&destCollisionThread, NULL, checkDestRocketCollision, NULL);  pthread\_create(&gameMonitor, NULL, checkEndConditions, NULL);  循环接受字符输入，调动相应函数执行。   1. Setup：   void setup(): 初始化curse界面。  void setRocketsToDead(struct rocket \*r):初始化飞碟数据结构。   1. Displaying Info & Menu Output:   void printStartMessage(): 打印开始界面，选择1、2号玩家，以及介绍相应按键。  void printEndGameMessage(char \*message): 打印游戏结束界面。  void displayInfo(): 显示共享数据，实时刷新。  void eraseInfo(): 擦除屏幕信息。   1. Collision Detection:   void \*collisionDetection(): 线程函数，调动compareCoords()函数检测子弹与飞碟坐标，判断是否碰撞。  void compareCoords(int i, int j): 判断存活的的saucer[i]是否和rocket[j]是否碰撞。  void \*checkDestRocketCollision(): 判断破坏者的子弹是否会打到玩家飞机的线程函数。   1. Cannon:   void setupCannon(int players): 输入玩家个数，初始化玩家飞机位置。  int getCannonCol(int player): 输入第几号玩家，得到其纵坐标。  int getCannonHit(int player): 返回玩家是否被破坏者击中。  void setCannonHit(int player): 指示玩家是否打中破坏者。  void moveCannon(int dir, int player): 根据方向dir和玩家号在画面上移动相应玩家飞机位置。  void displayCannon(int player): 画面上显示飞机。  6、Saucers (Spawning & lifecycle):  void \*setupSaucer(): 线程函数，随时产生飞碟。  void initSaucer(struct saucer \*ship, int row, int delay): 初始化飞碟信息，如出现行数，速度等。  void \*animateSaucer(void \*arg): 移动飞碟并画面展现。   1. Rockets (Spawning & lifecycle):   void shootRocket(int player): 产生子弹，即启动一个子弹线程。  void \*setupRocket(void \*arg): 初始化子弹数据结构。调动initRocket()和animateRocket()。  void animateRocket(struct rocket \*r)： 屏幕显示子弹轨迹，若打中或没打中则从屏幕中擦除。   1. Destroyer (Spawning & lifecycle):   void \*sendDestroyer(void \*arg): 产生破坏者的线程函数。  void initDestroyer(struct destroyer \*ship, int row, int delay): 初始化破坏者Boss.  animateDestroyer(pthread\_t thread, struct destroyer \*ship): 显示Boss轨迹。   1. Destroyer Rocket:   void \*shootDestRocket(void \*args)，void initDestRocket(struct destroyer \*ship)，void animateDestRocket()， void eraseDestroyerRocket()：大Boss发射的子弹函数线程，初始化，以及屏幕显示和擦除。  游戏截图：   1. 结题目标二：外接键盘，完成多人游戏。   与结题目标一不同，目标二更接近操作系统，尤其是IO模块。  完成成果：能正确识别外接键盘并获取键盘信息。可惜无法对外接键盘的输入字符进行区分，不能进行多键盘同一区域不同玩家的多人游戏。  成果说明：通过查阅外文资料，终于从茫茫大海中找到了获取操作系统IO信息并通信的libusb开源库。在libusb中，每个USB设备通过libusb\_device和libusb\_device\_handle对象操作。libusb API 连接一个打开的设备至特定的接口。这意味着如果在设备上请求多个接口，必须同样多次打开设备来接受一个libusb\_dev\_handle，对应每个想进行通讯的接口。不要忘记调用libusb\_dev\_handle。  编译程序：g++ -o usb usb.cpp -lusb-1.0  //需链接-lusb-1.0  Usb.cpp完成以下功能：  1.通过调用libusb\_init来初始化库，同时创建一个对话；  2.调用libusb\_get\_device\_list来获得已经连接的设备的队列。这会创建一个libusb\_device的数组，包含了所有连接到系统上的usb设备；  3.循环遍历所有的设备来检查他们的选项；  4.发现其中需要的一个，使用libusb\_open或者libusb\_open\_device\_with\_vid\_pid（已知这个设备vendor id和product id）来打开设备；  5.使用libusb\_free\_device\_list清除使用libusb\_get\_device\_list获得的队列；  6.通过libusb\_claim\_interface请求接口（需要你知道设备的接口数值）；  7.操作想得到的I/O;  8.通过libusb\_release\_interface释放设备；  9.通过libusb\_close将之前打开的设备关闭；  10.通过libusb\_exit来关闭对话；  Usb信息截图如下：  未完成部分说明：  获取了usb的外接键盘Vid和Pid后，碰到了一个问题。当调用dev\_handle = libusb\_open\_device\_with\_vid\_pid(ctx, VID, PID)时，程序运行停止，该函数作用是打开相应外设，并从外设判断并获取字符，产生error如下：  libusbx: error [\_get\_usbfs\_fd] libusbx couldn't open USB device /dev/bus/usb/006/004: Permission denied  libusbx: error [\_get\_usbfs\_fd] libusbx requires write access to USB device nodes.  又是一番查阅资料，发现permission denied是因为外接键盘原打开设定为root模式，即程序开启权限不足。此处停留，还需一番改进。  收获：目标二虽然未完全实现，但是收获丰厚。还记得到处查阅区别外接键盘输入方法的无奈，偶然在一个外文资料上看到这个，喜不自胜，废寝忘食终于懵懵懂懂，虽然未实现外接键盘的多人游戏，但是还是觉得收获很多，尤其是对于操作系统IO部分USB接口的理解更深一层。总之，值了！  总体程序架构如下： | | | | |
| **五、组员分工（详细的分工）** | | | | |
| 于雅冬：  游戏界面美化。Qt研究设计。  飞碟线程创建。  涉及文件：saucer.c,saucer.h， main.c，score.c，score.h等。 | 李雷：  Qt研究设计。游戏界面美化。  结束条件线程完善。  涉及文件：main.c，rocket.c,rocket.h，score.c,score.h等。 | | 叶鋆郴：  总体游戏设计，线程规划，信号量同步，usb接口完善。  涉及文件：Makefile，usb.cpp，destroyer.c,destroyer.h,lock.c,lock.h等。 | |