|  |  |  |  |
| --- | --- | --- | --- |
| **学生学号** | 0121610870126 | **实验课成绩** |  |

**学 生 实 验 报 告 书**

|  |  |
| --- | --- |
| **实验课程名称** | 数据结构与算法综合实验 |
| **开课学院** | 计算机科学与技术学院 |
| **指导教师姓名** | 李晓红 |
| **学生姓名** | 严伟滔 |
| **学生专业班级** | 软件工程1604 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2017 | -- | 2018 | 学年 | 第 | 2 | 学期 |

实验课程名称： 数据结构与算法综合实验

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验项目名称 | 线性结构与连连看游戏 | | | 报告成绩 |  |
| 实验者 | 严伟滔 | 专业班级 | 软件1602 | 组别 |  |
| 同组者 |  | | | 完成日期 | 2018年6月18日 |
| 第一部分：实验分析与设计（可加页）   1. 实验目的和具体内容   1.实验目的  通过连连看项目，达到如下目标：  (1)了解业务背景，调研与连连看同类型游戏，了解连连看游戏的功能和规则等。  (2)掌握C++开发工具和集成开发环境（Microsoft Visual Studio 2015）  (3)掌握C++面向对象的编程思想和C++的基础编程。  (4)了解MFC基本框架，包括MFC Dialog应用程序和GDI编程。  (5)了解线性结构，重点掌握数组和栈操作，数组遍历、消子和胜负判断等算法。  (6)了解项目开发流程，了解系统需求分析和设计，应用迭代开发进行项目开发。  (7)养成良好的编码习惯和培养软件工程化思维，综合应用“C++编程、MFC Diaolog、算法、线性结构”等知识，开发“连连看游戏”桌面应用程序，达到掌握和应用线性结构核心知识的目的。  2.实验内容  实现基本功能：开始游戏、暂停游戏、消子、判断胜负、提示、重排、计时等。  （1）主界面：设计“欢乐连连看”项目的主界面，在主界面上添加一个背景图片，并在适当的地方添加“基本模式”、“休闲模式”、“关卡模式”、“帮助”、“设置”、“排行榜”按钮。  （2）开始游戏：当玩家在主界面选择“基本模式”时，出现基本游戏界面，并隐藏主界面，玩家点击“开始游戏”按钮，生成游戏地图。  （3）消子：对玩家选中的两张图片进行判断，判断是否符合消除规则。符合一条直线连通、两条直线连通、三条直线连通这三种情况之一就可以消除。如果可以消除，从游戏地图中提示连接路径，然后消除这两张图片。如果不能消除，则保持原来的游戏地图。    消子规则  （4）判断胜负：在基本模式下如果将游戏地图中的所有的图片都消除，则提示玩家获胜，并且可以重新开始新游戏。  （5）提示：可以提示界面上能够消除的一对图片。  （6）重排：根据随机数，重新排列游戏地图上的图片。  （7）计时：设定一定的时间来辅助游戏是否结束。  （8）暂停游戏：游戏过程中可以暂停计时，并且将游戏地图遮盖，按钮显示为继续游戏。选择继续游戏，计时继续。     1. 分析与设计   欢乐连连看项目采用MFC框架，软件采用三层结构。使用二维数组来保存游戏地图中的数据，基本实现了连连看的核心功能。  整体采用MVC架构,将游戏核心模型,UI界面,游戏控制器进行分离,降低耦合性,方便修改  1.数据结构的设计  游戏核心逻辑/模型:  class LLKModel  {  public:  LLKModel();  ~LLKModel();  ......  private:  ......  };  基础存储结构:  /\*  \* 模板类Array2D  \* 采用了连续的内存空间并按行优先存储的方式存储元素  \* 重载了 [] 运算符,可以方便的通过"array[i][j]"形式索引元素  \* 也可以直接获取内存地址,对元素存储内存空间进行直接操作,像一维数组一样直接索引  \* 使用时注意传入参数行与列的范围不能过大,最大分配空间为4GB \*/  template <typename T>  class Array2D{  T\* Array2D<T>::operator[](size\_t \_x);  }  单个方格结构体:  typedef struct GirdBoxItem  {  uint16\_t type;  uint16\_t exist;  }GirdBoxItem, GirdBox;  其中type用来记录此方格的类型  exist用来记录此方格是否存在  2.界面设计  基础控件部分:  这里我使用了WindowsAPI与GDI+,重写了基础控件:  1.文本类,支持自定义字体,字号,格式,对齐方式,位置等  2.边框类,支持更改边框粗细,颜色,位置,大小,还有形状:矩形,椭圆形,圆角矩形  3.按钮类,由边框类与文本类还有背景色组合而成,支持动画效果,可以对鼠标操作作出响应  4.进度条类,基础实现与按钮类大致相同,在此基础上增加根据百分比显示进度功能,由触发器更新进度并显示,可根据不同进度显示不同颜色  整体UI部分:  分为加载界面,主界面,游戏界面三大部分:  加载界面: 显示启动图并停留5秒,然后跳转主界面  主界面:由背景,菜单按钮控件等部分组成  游戏界面:连连看Map部分,,辅助功能部分,时间模式下的进度条部分,背景等  3.类设计  1.Array2D模板类,  template <typename T>  class Array2D  {  public:  /\*构造函数,传入数组行与列大小\*/  Array2D(size\_t \_x, size\_t \_y);  /\*析构函数\*/  ~Array2D();  /\*重载运算符\*/  T\* operator[](size\_t \_x);  /\*Dump内存\*/  void dumpMemData(char\* fileName);  /\*获取分配得到的内存首地址\*/  const T\* getMemBlockPtr();  /\*获取数组行与列大小\*/  void getXY(int32\_t& \_x, int32\_t& \_y);  private:  /\*对象状态\*/  int status;  /\*数组行与列大小\*/  size\_t x, y;  /\*T大小(字节)\*/  size\_t TSize;  /\*内存区域指针\*/  void\* memblockptr;  /\*数组元素总数\*/  size\_t arraytotalnum;  /\*内存区域大小\*/  size\_t memblocksize;  /\*内存分配函数\*/  void tdaMalloc();  /\*内存回收函数\*/  void tdaRecycle();  };  2. LLKModel(连连看核心游戏模型):  class LLKModel  {  public:  LLKModel();  ~LLKModel();  /\*初始model,参数为map宽高以及方格种类数目\*/  void init(int32\_t \_mapWidth, int32\_t \_mapHeight, int32\_t \_girdTypeNum);  /\*按照默认模式生成带随机地图的model,且保证配对数为偶数\*/  bool generateModelByDefault();  /\*按照最外圈留白模式生成带随机地图的model,且保证配对数为偶数\*/  bool generateModelWithBlankAround();  /\*随机打乱map函数,map上每个方格几何位置以及存在状态不会受影响,打乱的是每个方格type值\*/  void disruptMap();  /\*点击(\_tappedPosiX, \_tappedPosiY)位置的方格作出响应函数\*/  int32\_t processTappedGirdBox(int32\_t \_tappedPosiX, int32\_t \_tappedPosiY, Paths\* paths);  /\*导出map数据到文件,方便调试时查看\*/  void dumpMemData(char\* fileName);  /\*获取Array2D模板类对象map\*/  Array2D<GirdBox>\* getMap();  /\*获取模型状态\*/  LLKModelStatus getStatus();  /\*获取map剩余方格数目\*/  int32\_t getRemainingGirdNum();  /\*获取map原有方格总数\*/  int32\_t getTotalNum();  /\*获取map上当前被选中方格坐标值\*/  void getSelectedPosi(int32\_t\* \_sx, int32\_t\* \_sy);  private:  /\*模型状态\*/  LLKModelStatus modelStatus;  /\*map宽与高\*/  int32\_t mapWidth;  int32\_t mapHeight;  /\*map中方格种类数目\*/  int32\_t girdTypeNum;  /\*map剩余方格数目\*/  int32\_t remainingGirdNum;  /\*map原有方格总数\*/  int32\_t totalGirdNum;  /\*Array2D模板类对象map\*/  Array2D<GirdBox>\* map;  /\*map上当前被选中方格坐标值\*/  int32\_t selectedGirdPosiX;  int32\_t selectedGirdPosiY;  /\*设置map宽高与种类\*/  void setMapWidth(int32\_t \_mapWidth);  void setMapHeight(int32\_t \_mapHeight);  void setGirdTypeNum(int32\_t \_girdTypeNum);  /\*检查数据有效性,合理性\*/  bool checkRationality();  /\*核心算法: ab两方格进行消子判断并获取消除路径\*/  bool checkElimination(int32\_t aX, int32\_t aY, int32\_t bX, int32\_t bY, Paths\* paths);  };  3:UI控件,这里以Button为例:  /\*回调函数指针类型\*/  typedef void (\*BCallBack)();  class Button  {  public:    Button();  Button(Text& \_text);  virtual ~Button();  /\*绘制函数\*/  void draw();  void draw(int \_x, int \_y);  void draw(int \_x, int \_y, int \_width, int \_height);  void draw(Gdiplus::Graphics\* \_graphics);  void draw(Gdiplus::Graphics\* \_graphics, int \_x, int \_y);  void draw(Gdiplus::Graphics\* \_graphics, int \_x, int \_y, int \_width, int \_height);  /\*触发器\*/  void trigger();  /\*处理鼠标消息函数\*/  int onMouseEventProcess(MouseLRBtnGeoStatus\* \_mlrbgs);  /\*设置点击回调函数\*/  void setOnClickCallBack(BCallBack \_callBack);  /\*开启/关闭背景显示\*/  void enableDisplayBackground(bool \_b);  /\*设置鼠标按下颜色\*/  void setPressedColor(int \_A, int \_R, int \_G, int \_B);  /\*设置鼠标划过颜色\*/  void setFlittedColor(int \_A, int \_R, int \_G, int \_B);  /\*设置背景颜色\*/  void setBackgroundColor(int \_A, int \_R, int \_G, int \_B);  /\*开启/关闭边框显示\*/  void enableDisplayBorder(bool \_b);  /\*设置边框形状\*/  void setShapeType(BorderType \_borderType);  /\*设置GDI+绘图引擎\*/  void setGDIGraphics(Gdiplus::Graphics\* \_graphics);  /\*设置位置\*/  void setPosi(int \_x, int \_y);  /\*设置宽高\*/  void setSize(int \_width, int \_height);  /\*获取相关属性\*/  int getPosiX();  int getPosiY();  int getPosiWidth();  int getPosiHeight();  BorderType getShapeType();  Gdiplus::Graphics\* getGDIGraphics();  /\*设置文本\*/  void setText(char\* \_srcA);  void setText(wchar\_t\* \_srcW);  void setText(std::string& \_srcStringA);  void setText(std::wstring& \_srcStringW);  /\*设置字体\*/  void setFontName(char\* \_fontNameA);  void setFontName(wchar\_t\* \_fontNameW);  void setFontName(std::string& \_fontNameA);  void setFontName(std::wstring& \_fontNameW);  /\*设置字号\*/  void setFontSize(int \_fontSize);  /\*设置字体格式,加粗,倾斜,下划线等\*/  void setFontStyle(Gdiplus::FontStyle \_fontStyle);  /\*设置文本对齐方式\*/  void setStringAlignment(Gdiplus::StringAlignment \_stringAlignment);  /\*设置文本颜色\*/  void setTextColor(Gdiplus::Color& \_color);  void setTextColor(int \_R, int \_G, int \_B);  void setTextColor(int \_A, int \_R, int \_G, int \_B);  /\*设置鼠标按下文本颜色\*/  void setPressedTextColor(int \_A, int \_R, int \_G, int \_B);  /\*设置鼠标划过文本颜色\*/  void setFlittedTextColor(int \_A, int \_R, int \_G, int \_B);  /\*获得文本内容\*/  std::string getTextA();  std::wstring getTextW();  /\*获得相关属性\*/  int getTextPosiX();  int getTextPosiY();  int getFontSize();  Gdiplus::Color getTextColor();  /\*设置边框粗细\*/  void setBorderStroke(float \_f);  /\*设置边框为圆角矩形时的圆角半径\*/  void setBorderRectRadius(int \_rectRadius);  /\*设置边框颜色\*/  void setBorderColor(Gdiplus::Color& \_color);  void setBorderColor(int \_R, int \_G, int \_B);  void setBorderColor(int \_A, int \_R, int \_G, int \_B);  /\*获得边框粗细\*/  float getBorderStroke();  /\*获得边框为圆角矩形时的圆角半径\*/  int getBorderRectRadius();  /\*获得边框颜色\*/  Gdiplus::Color getBorderColor(Gdiplus::Color& \_color);  protected:  Gdiplus::Graphics\* graphics;  Gdiplus::RectF rect;  /\*点击事件回调函数指针\*/  BCallBack callBack;  int x, y;  int width, height;  Text text;  /\*int tarA, tarR, tarG, tarB;  int curA, curR, curG, curB;\*/  bool displayBorder;  bool displayBackground;    Gdiplus::Color pressedColor;  Gdiplus::Color flittedColor;  Gdiplus::Color normalColor;  Gdiplus::Color backgroundColor;  Gdiplus::Color pressedTextColor;  Gdiplus::Color flittedTextColor;  Gdiplus::Color normalTextColor;  Gdiplus::Color textColor;  Border border;  BorderType borderType;  /\*初始化\*/  void init();  /\*绘制过程\*/  void drawProcess();  /\*计算文本位置(用来使文本刚好处于按钮正中央)\*/  void calcTextPosi();  /\*计算边框位置\*/  void calcBorderPosi();  };  4.游戏控制器模型GameController:  class GameController  {  public:  GameController();  ~GameController();  /\*带参数初始化控制器\*/  bool init(Gdiplus::Graphics\* \_graphics, int32\_t \_mapWidth, int32\_t \_mapHeight, int32\_t \_girdTypeNum, int32\_t \_posiX, int32\_t \_posiY);  Gdiplus::Graphics\* getGDIGraphics();  /\*输入设备消息响应\*/  void actionProc(int, void\*, void\*);  /\*绘制函数\*/  void draw();  /\*重排地图\*/  void rearrangeMap();  /\*获取游戏模型\*/  LLKModel\* getModel();  /\*获得本次游戏模型基本信息,宽高左上角坐标等\*/  void getBaseInfo(int32\_t&, int32\_t&, int32\_t&, int32\_t&, int32\_t&);  /\*获取本次游戏模型剩余方格数\*/  int32\_t getRemainingNum();  /\*获取消子路径(如果有)\*/  Paths\* getPaths();  private:  Gdiplus::Graphics\* graphics;  LLKModel\* model;  Paths path;  /\*游戏模型基础信息\*/  int32\_t mapWidth;  int32\_t mapHeight;  int32\_t mapWidthPix;  int32\_t mapHeightPix;  int32\_t girdTypeNum;  int32\_t posiX, posiY;  /\*这里要求每个方格贴图为正方形,单位边长像素值\*/  int32\_t girdBoxImageSize;  /\*方格贴图\*/  std::vector<Gdiplus::Image\*> girdImages;  /\*处于选中状态的方格特效图片(最外圈有一层白色光圈)\*/  Gdiplus::Image\* selectedSpecEffectImage;  /\*背景小方格图片\*/  Gdiplus::Image\* oddNumImage;  Gdiplus::Image\* evnNumImage;  /\*整体背景图片\*/  Gdiplus::Image\* gameUIBackgroundImage;  /\*载入资源\*/  void loadRes();  /\*回收资源\*/  void recycleRes();  /\*判断点击位置是否在游戏模型Map内\*/  bool GameController::isInGameMapArea(int x, int y);  /\*通过点击位置计算得到被点击方格在游戏模型Map中的逻辑位置(行与列)\*/  bool GameController::calculateGirdBoxPosi(int mbupx, int mbupy, int\* girdx, int\* girdy);  };  三、主要仪器设备及耗材   1. 安装了Windows XP或Windows 7或其它版本的Windows操作系统的PC机1台 2. PC机系统上安装了Microsoft Visual Studio开发环境   第二部分：实验过程和结果（可加页）   1. 实现说明   **1.游戏核心模型基础逻辑1-消子判断:**  **消子判断:我们在游戏时,观察一下连连看游戏的消子规则,可以发现对于Map上的任意AB两点,如果可以进行消子,则路径拐点不会超过2次,也就是消子路径最多有三段折线,分析后,可以得知,消子路径只会出现在下图中的区域1与区域2中.(从左到右为X轴正方向,从上到下为Y轴正方向)**  C:\Users\Asus\Desktop\$icon\LLK说明.jpg  **我们在查找消子路径时按照如下两种情况考虑:**  **1.点B在A点上方,记为情况A,在上图所示的区域1(y∈[bY,aY])中查找**  **2.点B在A点左侧,记为情况B,在上图所示的区域2(x∈[bX,aX])中查找**  **A情况(我们将下方,左侧,右侧分别记为0,1,-1)(共计九种情况)**  **从B下方开始查找,有如下三种情况:**  C:\Users\Asus\Desktop\$icon\LLK说明A_0_0.jpgC:\Users\Asus\Desktop\$icon\LLK说明A_0_1.jpgC:\Users\Asus\Desktop\$icon\LLK说明A_0_-1.jpg  **从B左侧开始查找,有如下三种情况:**    **从B右侧开始查找,有如下三种情况:**    **B情况(我们将右侧,上方,下方分别记为0,1,-1)(共计九种情况)**  **从B右侧开始查找,有如下三种情况:**    **从B上方开始查找,有如下三种情况:**    **从B下方开始查找,有如下三种情况:**    **AB两类共计3\*3\*2=18种情况,按照此思路编写程序即可**  **下面是以A情况的B点右侧查找为例的算法代码,(对应Case A\_-1\_0,** **Case A\_-1\_1,** **Case A\_-1\_-1):**  /\*寻路方式2:从B点左侧开始遍历,直到x为0\*/  for (int32\_t xi = bX - 1; xi >= 0;xi--) {  /\*bX为0时,已为最左侧,不进行遍历\*/  if (bX == 0) {  break;  }  /\*此位置有方格,则停止向左遍历\*/  if ((\*map)[xi][bY].exist == 1) {  break;  }  /\*检查第二条路径是否连通,注意这里不必检查到y = aY这一行\*/  if (checkVerticalConnectivity(map, xi, bY, aY - 1)) {  /\*第二条路径尽头为A点的情况\*/  if (xi == aX) {  /\*出口A\_-1\_0\*/  setPathNum(paths, 2);  setP1(paths, bX, bY);  setP2(paths, aX, bY);  setP3(paths, aX, aY);  return true;  }  /\*A点在第二条路径尽头左侧的情况\*/  else if (aX < xi) {  /\*检查第三条路径是否连通,注意这里不必检查x = aX这一列\*/  if (checkHorizontalConnectivity(map, aY, aX + 1, xi)) {  /\*出口A\_-1\_-1\*/  setPathNum(paths, 3);  setP1(paths, bX, bY);  setP2(paths, xi, bY);  setP3(paths, xi, aY);  setP4(paths, aX, aY);  return true;  }  else {  /\*以(aX + 1, aY)为起点,(xi, aY)为终点的水平路径不连通,检查下一个xi\*/  continue;  }  }  /\*A点在第二条路径尽头右侧的情况\*/  else {  /\*检查第三条路径是否连通,注意这里不必检查x = aX这一列\*/  if (checkHorizontalConnectivity(map, aY, xi, aX - 1)) {  /\*出口A\_-1\_1\*/  setPathNum(paths, 3);  setP1(paths, bX, bY);  setP2(paths, xi, bY);  setP3(paths, xi, aY);  setP4(paths, aX, aY);  return true;  }  else {  /\*以(xi, aY)为起点,(aX - 1, aY)为终点的水平路径不连通,检查下一个xi\*/  continue;  }  }  }  else {  /\*以(xi, bY)为起点,(xi, aY - 1)为终点的垂直路径不连通,检查下一个xi\*/  continue;  }  }  **2.游戏核心模型基础逻辑2-方格点击响应函数:**  /\*点击(\_tappedPosiX, \_tappedPosiY)位置的方格作出响应函数\*/  int32\_t LLKModel::processTappedGirdBox(int32\_t \_tappedPosiX, int32\_t \_tappedPosiY, Paths\* paths) {  /\*位置非法则记录错误信息,直接返回\*/  if (\_tappedPosiX < 0 || \_tappedPosiY < 0 || \_tappedPosiX >= mapWidth || \_tappedPosiY >= mapHeight) {  LLKErrLog("TappedGBPositionERR: X: %03d, Y: %03d\n", \_tappedPosiX, \_tappedPosiY);  /\*将paths置为无效状态\*/  invalidatePaths(paths);  /\*返回状态为点击位置参数非法(负数,超过map宽高等非法状态)\*/  return LLKMLTP\_ERROR\_POSITION;  }  /\*本次点击位置方格状态为不存在时返回,同时将之前的已选中方格记录清空\*/  if ((\*map)[\_tappedPosiX][\_tappedPosiY].exist == 0) {  selectedGirdPosiX = -1;  selectedGirdPosiY = -1;  //LLKInfLog("TappedGBPositionEmpty: X: %03d, Y: %03d\n", \_tappedPosiX, \_tappedPosiY);  /\*将paths置为无效状态\*/  invalidatePaths(paths);  /\*返回状态为所选方格不存在\*/  return LLKMLTP\_EMPTY;  }  /\*本次点击位置与已选中方格相同时返回,同时将之前的已选中方格记录清空\*/  if (\_tappedPosiX == selectedGirdPosiX && \_tappedPosiY == selectedGirdPosiY) {  selectedGirdPosiX = -1;  selectedGirdPosiY = -1;  //LLKInfLog("TappedGBPositionIsConsistentWithLastTime : X: %03d, Y: %03d\n", \_tappedPosiX, \_tappedPosiY);  /\*将paths置为无效状态\*/  invalidatePaths(paths);  /\*返回状态为所选方格与上一次重复\*/  return LLKMLTP\_CONSISTENT;  }  /\*处理当前模型中没有已选中方格的情况,将本次点击的方格记为待配对状态\*/  if (selectedGirdPosiX == -1 && selectedGirdPosiY == -1) {  selectedGirdPosiX = \_tappedPosiX;  selectedGirdPosiY = \_tappedPosiY;  /\*将paths置为无效状态\*/  invalidatePaths(paths);  /\*返回状态为选中某方格\*/  return LLKMLTP\_SELECTED;  }  /\*处理当前模型中有已选中方格的情况,进行消子判断\*/  else{  /\*判断选中方格与点击方格是否为同一类\*/  bool b0 = ((\*map)[selectedGirdPosiX][selectedGirdPosiY].type == (\*map)[\_tappedPosiX][\_tappedPosiY].type);  /\*两方格可消\*/  if (b0 && checkElimination(selectedGirdPosiX, selectedGirdPosiY, \_tappedPosiX, \_tappedPosiY, paths)) {  /\*两子存在状态置为否\*/  (\*map)[selectedGirdPosiX][selectedGirdPosiY].exist = 0;  (\*map)[\_tappedPosiX][\_tappedPosiY].exist = 0;  /\*已选中方格记录清空\*/  selectedGirdPosiX = -1;  selectedGirdPosiY = -1;  /\*剩余未消除数量减2\*/  this->remainingGirdNum -= 2;  /\*将paths置为有效状态,其中存储了消除路径信息\*/  validatePaths(paths);  /\*返回消除成功\*/  return LLKMLTP\_ELIMINATE\_SUCCESS;  }  /\*两方格不可消\*/  else {  /\*已选中方格记录转换到\_tapped位置\*/  selectedGirdPosiX = \_tappedPosiX;  selectedGirdPosiY = \_tappedPosiY;  /\*将paths置为无效状态\*/  invalidatePaths(paths);  /\*消除失败,返回状态为选中方格转移到tapped位置\*/  return LLKMLTP\_SWITCH\_SELECTED;  }  }  /\*正常状态不可能到达这里\*/  LLKErrLog("FunctionERR -ImpossibleProcess | X: %03d, Y: %03d\n", \_tappedPosiX, \_tappedPosiY);  return LLKMLTP\_ERROR\_STATUS;  }  **3.游戏核心模型基础逻辑3-随机打乱重排:**  /\*  \*打乱map  \*map上每个方格几何位置以及存在状态不会受影响,打乱的是每个方格type值  \*/  void LLKModel::disruptMap() {  if (map == NULL) {  LLKErrLog("MapPtrNULL!\n");  return;  }  if (remainingGirdNum < 2 || remainingGirdNum > LLKML\_MAXGBNUM) {  LLKInfLog("remainingGirdNum Illegal! value: %d\n", remainingGirdNum);  return;  }  /\*获取map上方格总数\*/  int32\_t girdBoxNum = this->mapWidth \* this->mapHeight;  /\*为数组rawData分配内存空间\*/  uint16\_t\* rawData = new uint16\_t[remainingGirdNum];  /\*初始化数组rawData为全0\*/  memset(rawData, 0, sizeof(uint16\_t) \* remainingGirdNum);  /\*获取Array2D类的实例map中底层连续内存空间首地址\*/  GirdBox\* p = (GirdBox\*)map->getMemBlockPtr();  /\*遍历map中所有方格,将exist属性为1的方格的type值存入数组rawData中\*/  for (int32\_t i = 0, j = 0; i < girdBoxNum && j < remainingGirdNum; i++) {  if (p[i].exist == 1) {  rawData[j] = p[i].type;  j++;  }  }  /\*随机打乱rawData数组\*/  shuffle(rawData, remainingGirdNum);  /\*遍历map中所有方格,将打乱后的rawData数组中的元素重新写入exist属性为1的方格\*/  for (int32\_t i = 0, j = 0; i < girdBoxNum && j < remainingGirdNum; i++) {  if (p[i].exist == 1) {  p[i].type = rawData[j];  j++;  }  }  /\*回收内存\*/  delete[] rawData;  }   1. 源代码   **UI视图(View)**  **游戏模型(Model)**    **控制器(Controller)**   1. 调试说明（调试手段、过程及结果分析）   主界面:    *计时模式:*    每次游戏前生成模型阶段会将Map地图dump到文件中,方遍调试:    AI模式自动完成:    胜利提示与失败提示:    第三部分：实验小结、收获与体会  实验小结:  本游戏主要使用MVC架构,将相关模块解耦合进行设计,Model为核心模型与游戏基础算法通过分类讨论解决消子判断,做到了较好的可移植性,可移植到Linux,Android等平台,View部分主要使用WindowsAPI与GDI+接口,摒弃老旧的MFC,重写按钮,边框,进度条等基础控件,实现动画等高级特效,使用双缓冲技术提升GDI+贴图效率防止闪屏,Controller部分将Model与View有机的结合在一起,响应时间,外设消息来控制游戏与界面的更新.除此以外还有AI模式等,可以做到自动完成游戏的操作  图片等资源均为通过Photoshop设计,进行了相应的美化  双缓冲绘制原理:  **附:Win32中，实现双缓冲的步骤如下:(这里以客户区绘图为例, hdc、hdcBuffer、hdcBmp均是HDC类型变量名)**  (1)首先获取客户区DC——hdc  (2)获取关于DC的内存兼容DC——hdcBuffer、获取关于DC的兼容内存位图并选入hdcBuffer中  (3)先在hdcBuffer中绘制所需的图(例如很多条直线、图形等等)  (4)如果你想一次性贴很多位图 , 那么你还应该获取一个关于DC的内存DC——hdcBmp , 将位图依次选入hdcBmp中，然后将位图从hdcBmp贴到hdcBuffer中  (5)最后将hdcBuffer(也就是内存中)中绘制好的位图贴到原客户区DC中  收获与体会:  1.善于运用this,在get/set类方法中遇到局部变量与类成员变量同名时可通过this来解决冲突问题  2.面向对象的思想在编程中有着重要的运用,实际是计算机发展过程中对大自然各种现象的模拟,深入理解对编程的学习很有帮助  3.编程过程中,对于边界条件需要仔细考虑,比如数组下标越界,特殊情况未考虑到等,本实验中主要体现在对不足8bit的部分需要补0处理,以及记录不足8bit的位数,方便以及提示用户输入有误,这些都是很重要的细节,在产品设计中也有着很重要的运用,做到良好的人机交互性.  4.一个工程项目中层次应该一目了然,目录以及文件名要有意义,对功能要做好恰当的划分,做好代码的可维护性,低耦合性,以及健壮性,这样也方便功能的迭代以及日后的维护  5.遇到复杂算法问题时,可以通过分情况讨论,分类解决,本实验中的消子判断属于情况较多的算法问题,编写程序前应仔细思考,将情况分类清楚后,再编写代码  6.熟练掌握VisualStudio的调试功能,在出现问题时,设置断点,单步调试,实时查看内存中的各个变量的值以及指针所指的内存地址的内容,来解决设计时产生的Bug,对于设计一个完整的工程性C++程序是强有力的工具.我在设计和操作链表时,由于没有注意对头节点和尾节点的考虑,出现了很多奇怪的问题,以及进行多次对链表节点进行添加或者删除操作时,会发生地址异常的问题,使用调试后发现,操作两个指针时有一个没有发生相应的移动,经过排查发现,这是某函数中缺失了一条语句而造成的.调试使得开发者更容易查找到问题所在,从而快速修复Bug,进行下一项功能的设计.  7. 最后还有细节与心态.细节比如代码风格,缩进,注释,养成良好的习惯是很必要的,好的缩进有利于查看,方便Debug,注释则是解释相关的函数功能结构,防止经过较长时间后忘记所带来的麻烦.另一个就是心态,对链表的相关操作以及对文件的输入输出是最为繁琐和最需要小心谨慎的,在这里出了很多问题,但告诉自己要冷静,不要着急着写,应该仔细思考后在动手,这样才能一气呵成.  经过本次C++实验开发,我从中学到了很多原来所不了解的知识,对C++这门语言也有了更加深刻的认识,也学到了一个优秀的软件工程师应有的品质与心态,希望自己以后在计算机的世界中汲取更多知识,更多技能,最终成为一个优秀的软件工程师. | | | | | |