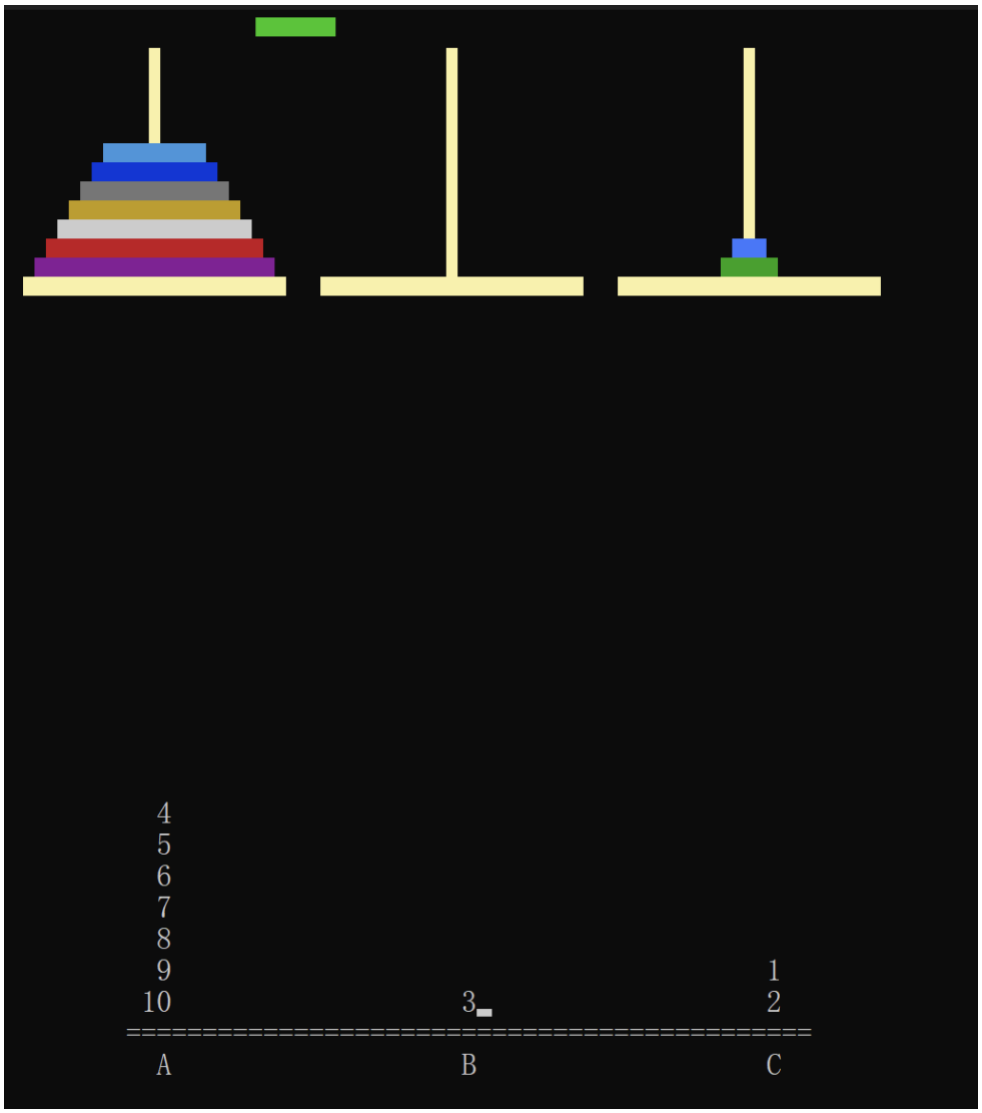
汉诺塔综合演示实验报告



姓名：汤皓宇
班级：5000244001608
学号：2454307
完成日期：2025年5月15日

1. 题目

设计并实现一个集成的汉诺塔综合演示程序，有菜单选择功能模块，通过键入数字来选择9种演示模式或者退出：

1.基本解：给出汉诺塔的解的每一步，包括该步操作的被操作盘，移动的起始基座与目标基座；

2.基本解（步数记录）：在给出基本解的基础上记录步数，每移动一次加一步，并输出在每行的最前端；

3.内部数组显示（横向）：在1、2同时给出的同时，在1、2输出的末尾追加对三个基座上盘子数对应的三个栈数组内部数据的显示，要求有足够的空间显示所有的盘子；

4.内部数组显示（横向+纵向）：首先按照3的要求输出横向信息，其位置需要按照文件hanoi_const_value.h所给的起始位置为起点进行输出。同时，按照头文件给出的另一起始位置信息输出用字符展示的汉诺塔字符图像，要求每次移动只能对目标盘进行覆盖不能重新输出。同时对用户选择的延时选项进行状态展示，状态栏位置也为头文件预设值；

5.图形解-预备-画三个圆柱：使用伪图形化函数画出三个托盘，每次作画之间有头文件预设的延时；

6.图形解-预备-在起始柱上画n个盘子：在5画完的基础上在用户选择的起始盘上画n个在头文件中预设好颜色的大小递减的圆盘，作画中间也有头文件预设的延时；

7.图形解-预备-第一次移动：在6的基础上进行盘子的第一次移动，选中起始柱上最小的盘子进行第一次移动，该过程通过动画展现，分为上移，平移，下移三部分，使用头文件给定的预设步长和上移的最高高度来进行演示；

8.图形解-自动移动版本：以汉诺塔的递归解为基础，利用5～7所构建的函数进行解的自动图形化演示，演示的同时也要按给出的位置显示4所包含的内容。

2. 整体设计思路

主要分为主控部分、菜单交互部分和函数具体实现三部分。主控部份负责循环调用菜单函数，根据用户选择来执行对应的9个模式的函数或退出程序。菜单部分负责显示功能菜单，捕获用户输入并返回选项编号。函数具体实现部分主要实现递归算法、图形绘制、动画移动、输入验证等核心功能，通过全局变量“stack[3][11]”记录三根圆柱的盘子状态，“top[3]”记录每根柱子的当前高度。通过对不同函数参数的设定来实现多个功能函数共用同一函数的要求。

九个主要模式函数中会调用其他函数进行一些递归前和递归后的必要操作，包括对option[]进行预设、读入必要的值包括总盘数、起始柱、终点柱、延迟时间、提前画好初始图像以及递归结束后的收尾处理即等待回车清屏回到主菜单。在进入递归后，hanoi函数会进行递归操作对其求解，其中包含出栈入栈等操作来表示圆盘的移动，这些夹在两次自我调用中间，此时也是输出相关信息的最佳时

机，通过判断option[]参数每一项的真假，来判断应该输出什么。

前三个水平方式的输出可以通过传递option[]参数统一在同一个输出函数中,这些和输入输出信息相关的函数都被以info为开头的函数名命名，第四个竖直方式输出的用字符构建的类图形化对汉诺塔的模拟，其相关函数则被以char开头的函数名命名。而剩余要用到的伪图形界面的函数则被用gui开头的函数名命名。对于“char”和“gui”类型界面的实现方式，都是先在模式函数绘制好基本界面，在进入递归后，每次出栈入栈后对操作的盘子进行擦除再绘制达到移动的效果。

3. 主要功能的实现

3.1. 主函数的构建

以一个无限执行下去的while(true)循环为主体，在其中先用Menu()函数来显示菜单并返回用户的选择，对用户的选择进行switch判断，case1~9分别执行九个模式函数来满足题目的九个要求，0则直接return 0退出主函数结束程序。

3.2. 菜单函数的构建

菜单函数主要包括显示菜单和选择模式两部分组成。第一部分显示菜单需要先cct_cls()清理屏幕，然后输出各菜单选项。第二部分选择模式则由一个while循环主导，使用_getch()对用户键入进行判断，包含错误检测，如果输入合理则回车至下一行便于后续操作，同时把用户选择的值返回main函数便于模式函数的调用。

3.3. 九个模式函数的构建

九个模式函数的主要用途是根据题目所给出的九种功能对应的要求定义初始的option[]值，并将option作为参数传递给其所调用的功能函数。作为功能开关数组，option控制是否显示步数（option[0]）、横向状态（option[1]）、纵向状态（option[2]）、图形动画（option[3]）四项，true为开false为关。为了方便，将他们分别define为showStep、showHor、showVer、showGraph。

以函数hanoi_graphics_automove为例，开头定义option[]={true, true, true, true},代表四个项目都要显示。然后定义n，src，tmp，dst代表总盘子数、起始柱、中间柱、终点柱，通过调用函数info_input(n, src, tmp, dst, 20)来进行读入，其中20为最大延迟时间，结果出存在全局变量Delay中。接着是info_init(n, src)，对信息初始化，包括全局变量信息的重置以及n个盘子的预先入栈操作。然后进入演示界面，调用cct_cls与hdc_cls执行清屏、画布初始化的操作。接着调用info_printInfo、info_showStatus、char_drawDisks、gui_drawCylinders、gui_drawDisks分别来打印初始步骤信息、显示当前模式选择的状态、画出数值字符汉诺塔初始状态、画出伪图形化界面的

基座和立柱、画出初始的10个盘子。在这些准备工作完成后便hanoi(n, src, tmp, dst, option)进入递归求解，包括移动的演示,最终使用func_ending(option)函数来输出结束语句等待回车返回初始菜单页面。其他模式函数的形式基本相同，总结下来就是定义初始参数、读入处理信息、界面初始化、进入递归并演示、结束界面、返回的流程。

3.4. 递归函数hanoi的结构

递归函数void hanoi(int n, char src, char tmp, char dst, bool option[])是程序的核心。第一句判断n是否为0用来判断当前的深度是否已经涵盖所有盘子。后面有两次不同次序对更深一层汉诺塔函数的调用，夹在中间的是出栈入栈以及通过判断option来对当前界面进行动态修改。这样的结构代表了先将n-1个盘子从源柱（src）移动到中间柱（tmp），再将第n个盘子从src移动到目标柱（dst）接着将n-1个盘子从tmp移动到dst的递推流程。

函数中间会根据盘子的移动执行相应的入栈出栈操作。操作后，调用info_printInfo函数传递option打印信息。然后对option选项进行判断，若需要竖直方式显示、伪图形化界面显示则分别调用char_moveDisks和gui_moveDisks来对当前界面的盘子进行操作位移。在修改显示之后会调用func_Delay函数根据用户在一开始的选择进行延时处理来达到停顿的效果。

3.5. 步骤等信息的打印——Info为前缀的函数

在项目中，info为前缀的函数用来处理和信息相关的时间。例如info_printInfo负责输出移动的状态信息，info_showStauts负责显示当前的选项状态，info_Input负责信息、选项的输入功能，info_readCommand负责第9项中命令的读取与错误判断等。这些函数通过cct_gotoxy和使用头文件中的参数来确定输出的位置。例如info_printInfo在开头通过对showVer和showGraph的判断来选择输出的起始位置。通过对option中内容的判断来选择不同的输出形式。例如通过判断showHor来判断是否具体显示三个栈数组的具体内容，如果为真则调用info_watchDisks函数来输出具体信息。

3.6. 垂直显示——Char为前缀的函数

在项目中，Char为前缀的函数用来实现通过字符显示汉诺塔移动过程。char_drawDisks函数根据预设坐标画好底盘框架以及初始的盘子，然后在hanoi中通过调用char_moveDisks来实现盘子的移动。每次调用char_moveDisks时，函数会根据src、dst以及对应的top值计算移动盘开始与终止的坐标，通过输出空格进行擦除，再用cct_gotoxy走到目标位置打印移动后的盘子。从而实现垂直方式的展示。

3.7. 伪图形化界面展示——Gui为前缀的函数

首先是gui_drawCylinders函数，开始时根据给出的起始坐标将光标移动到对应位置，接着以给出的盘子间距和宽度，以给定的间隔时间绘制三个底盘。接着计算盘子中间横坐标，其值减去立柱宽度的一半，高度减去立柱高度，得到三个立柱绘制的起始坐标，从而达成模式5的要求。

然后是gui_drawDisks函数，先从下往上记录出每个圆盘的宽度，编号为n的盘子的宽度为(2n+1)倍的HDC_Base_Width,通过计算立柱的中间横坐标减去盘子宽度的一半来得到绘制盘子的起始坐标，在根据盘子所处栈的位置来计算盘子绘制起始的纵坐标，再循环中一个个将盘子绘制出来，从而达成模式6的要求。

最后是gui_moveDisks函数对盘子移动动画的实现。动画分上移、平移、下移三阶段实现，三个阶段分别由三个while循环来实现。上移的循环中，先擦除盘子底部高为单位高度，长为盘子宽度的一条矩形（看上去是一条细直线），然后补画被擦去的立柱，宽度为立柱宽度，高度为单位高度，最后在盘子上端补画一条相同的细直线，从而得以实现移动的效果。移动时需判断盘子底端是否离开立柱，从而停止对立柱的补画；以及盘子顶端是否到达高度限制，从而停止移动进入下一阶段。平移的循环中，则是在循环中画朝目标立柱方向的细线，擦除相反方向的细线，达成移动的效果，直到移动到目标柱上房后停止。下降的时候实现逻辑与上升一直，只是逻辑判断略有不同，判断是否需要处理被擦除的立柱需要以盘子的上线为基准。

3.8. 交互游戏的实现

在第九个模式中，程序进入循环等待用户输入两个字母（例如 “AC”），分别表示源柱子（src）和目标柱子（dst）。每次输入前通过 info_readCommand(from, to) 获取用户命令，并将输入转换为大写形式以规范格式。程序首先会验证输入的合法性，例如判断源柱子是否存在盘子、目标柱子是否允许接收该盘等。如果输入合法，则进行相应的入栈和出栈操作：通过 pop(from) 从源柱子弹出顶部盘子，并通过 push(to, fromT) 将该盘子压入目标柱子。接下来的流程和hanoi递归函数里的相似，通过info_printInfo输出本次操作的相关信息，包括当前移动的盘子编号及其源目标柱子。接着进行伪图形化界面的同步更新，根据预设的option来判断是否需要更新字符模拟界面和伪图形化界面，通过直接调用char_moveDisks和gui_moveDisks即可实现这样的功能。

在每次盘子移动之后，检查目标柱子（dst）上是否已经堆满了 n 个盘子（即原始的全部盘子数量）。通过判断dst对应的top是否到达n来控制循环是否继续，一旦目标柱子填满，说明玩家成功完成了汉诺塔游戏，循环终止。最终，程序调用 func_Ending(option) 来展示游戏结束界面，返回主菜单等待下一次用户操作。

4. 调试过程碰到的问题

4.1. 图形移动时的错位问题

在编写图形化自动移动盘子的功能过程中，遇到了一个较为棘手的问题——移动过程中偶尔会出现盘子错位，具体表现为动画执行时并没有选中当前柱子最顶端的盘子，而是错误地显示为一个黑色方块，并且在移动落下后，盘子还会“穿透”目标柱子的递补，最终和底盘保持同一高度。

经过细致分析和多次调试，最终发现问题出在函数调用的顺序上。原本 gui_moveDisk(src, dst) 函数放置在递归移动函数中执行盘子的出栈和入栈操作之后。因此，在图形动画绘制的时候，栈的状态已经被修改，栈顶的盘子编号已经更新，不再是用户原本意图移动的那个盘子，导致图形显示和数据结构不同步，从而出现错位。

经过思考，找到了解决问题的方法：在调用 gui_moveDisk(src, dst) 函数之前，暂时保存并恢复栈的 top 标记数组至出入栈操作之前的状态。这样一来，图形绘制函数能够正确读取原本的栈顶信息，确保移动的是原本的盘子而非更新后的盘子，从而彻底解决了移动错位的显示问题，提升了动画的准确性和连贯性。

4.2. 图形盘子移动时的方块闪烁问题

在完成基本的移动逻辑之后，对比测试运行结果和参考demo程序的表现后发现，当前程序在盘子移动的过程中存在较为明显的闪烁感，影响了动画的流畅性和美观性。

通过深入分析gui_moveDisk(src, dst)函数的实现过程，发现闪烁问题的根本原因在于绘图机制的处理方式：每次盘子移动时，函数会擦除整个目标区域原有的盘子方块，再重新绘制新的盘子位置。这种完全擦除再重画的方式会在图像缓冲未能及时刷新时在观感上产生瞬时的黑色，从而形成明显的闪烁现象。针对这一问题，改进的策略是尽可能减少不必要的擦除操作，将每次动画绘制的粒度缩小，从“整块重绘”优化为“局部改写”。具体做法是：每次仅擦除并重绘一条水平线，即当前盘子所在行，而不是整块方形区域。通过这个方式，闪烁现象得到了明显缓解，整个移动过程看起来更加平滑自然，视觉效果大大提升。

5. 心得体会

5.1. 完成本次作业得到的一些心得体会、经验教训

在这次图形汉诺塔项目的编写过程中，我体会到了将复杂问题拆解为多个模块的重要性。通过将主函数、菜单处理、图形显示等分别封装进不同的cpp文件中，使得程序结构更加清晰、逻辑更加分明。过程中也意识到，合理使用头文件可以有效地集中管理参数和函数声明，提高程序的可维护性。例如，在头文件中统一定义显示配置参数后，仅需修改一处，或者更换一个预设的头文件，即可实现整个项目的样式变更。此外，图形部分的开发让我更加注重细节处理，如盘子错位和闪烁问题，都是在深入理解函数调用顺序和绘图机制后，逐步定位并解决的。这些问题不仅锻炼了我的分析能力，也

让我意识到再小的细节也可能对整体用户体验产生巨大影响

5.2. 做一些复杂程序，分为若干小题还是直接一道大题的形式更好？

我认为将复杂程序划分为若干小题是更为科学有效的做法。每小题聚焦解决一个具体功能，例如字符界面绘制、盘子移动动画、用户命令输入等。降低了每一部分的开发难度，也帮助我在完成一个模块时获得即时反馈，增强信心。如果一开始就构建完整的体系，容易逻辑混乱，管理不来。

5.3. 如何才能更好地重用代码

完成多个汉诺塔相关题目的过程中，我逐渐意识到复用函数的重要性。例如，早期实现的字符界面显示函数 char_drawDisks()、动画绘制 gui_moveDisk() 等功能在后续编写中被直接调用并加以扩展，使得代码编写更有效率。同时，函数如 info_readCommand() 和 info_printInfo() 也被多次复用，用于处理命令输入和信息显示，这种“写一次、多处调用”的方式减少了重复率，降低了代码出错的风险。为此，在代码中我把不同模式调用的函数尽可能融合到一个函数中，通过 option 调控，达成代码复用的效果。

5.4. 如何才能更好地利用函数来编写复杂的程序

以本次汉诺塔作业为例，要编写复杂程序，函数的设计与复用是关键。应尽可能将每个功能模块封装为单一职责的函数，如输入处理、图形绘制、信息打印等，做到功能明确、接口清晰。同时，函数命名上要语义清晰，如处理伪图形化界面的函数都被命民为以 gui 为前缀。同时，对于重复使用的逻辑应尽可能融合到一个函数中，例如盘子的入栈出栈逻辑，可以统一调用 push 和 pop，从而减少冗余代码。通过主函数调用其他模块，使各模块协同运作，形成清晰、可控的程序结构。

6. 附件：源程序

```
 1. //汤皓宇 计算机 2454307
 2. void char_moveDisks(bool option[], int n,
char src, char dst) {
 3.     int srcT = top[tag(src)] + 1;
 4.     int dstT = top[tag(dst)] - 1;
 5.     int initx, inity;
 6.     if (showGraph) {
 7.         initx = MenuItem9_Start_X +
Underpan_A_X_OFFSET - 1;
 8.         inity = MenuItem9_Start_Y +
Underpan_A_Y_OFFSET - 1;
 9.     }
10.     else {
11.         initx = MenuItem4_Start_X +
Underpan_A_X_OFFSET - 1;
12.         inity = MenuItem4_Start_Y +
Underpan_A_Y_OFFSET - 1;
13.     }
14.     cct_gotoxy(initx + Underpan_Distance
* tag(src), inity - srcT);
15.     cout << "  ";
16.     cct_gotoxy(initx + Underpan_Distance
* tag(dst), inity - dstT - 1);
17.     cout << setw(2) << n;
18. }
19. void char_drawDisks(bool option[])
20. {
21.     int initx, inity;
22.     if (showGraph) {
```

```
 23.         initx = MenuItem9_Start_X +
Underpan_A_X_OFFSET;
 24.         inity = MenuItem9_Start_Y +
Underpan_A_Y_OFFSET;
 25.     }
 26.     else {
 27.         initx = MenuItem4_Start_X +
Underpan_A_X_OFFSET;
 28.         inity = MenuItem4_Start_Y +
Underpan_A_Y_OFFSET;
 29.     }
 30.     cct_gotoxy(initx, inity);
 31.     cout << 'A' <<
setw(Underpan_Distance) << 'B' <<
setw(Underpan_Distance) << 'C';
 32.     cct_gotoxy(initx - 2, inity - 1);
 33.     cout << setfill('=') <<
setw(Underpan_Distance * 2 + 5) << '=' <<
setfill(' ');
 34.     initx -= 1;
 35.     inity -= 2;
 36.     for (int t = 0; t < 3; t++) {
 37.         for (int i = 0; i < top[t]; i++) {
 38.             cct_gotoxy(initx + t *
Underpan_Distance, inity - i);
 39.             cout << setw(2) <<
stack[t][i];
 40.         }
 41.     }
 42. }
 43. void info_printInfo(bool option[], int n
= 0, char src = 0, char dst = 0) {
 44.     if (showVer) {
 45.         if (showGraph)
 46.             cct_gotoxy(MenuItem9_Start_X,
MenuItem9_Start_Y);
 47.         else
 48.             cct_gotoxy(MenuItem4_Start_X,
MenuItem4_Start_Y);
 49.     }
 50.     if (cntAll) {
 51.         if (showStep)
 52.             cout << "第" << setw(4) <<
cntAll << " 步( ";
 53.         cout << n << "#: " << src << "-->"
<< dst;
 54.         if (showStep)
 55.             cout << ") ";
 56.     }
 57.     else
 58.         cout << "初始: ";
 59.     if (showHor)
 60.         info_watchDisks();
 61.     cout << endl;
 62. }
 63. void info_showStauts(int n, char src,
char dst, bool showDelays = true) {
 64.     cct_gotoxy(Status_Line_X,
Status_Line_Y);
 65.     cout << "从 " << src << " 移动到 " <<
dst;
 66.     cout << ", 共 " << n << " 层";
 67.     if (!showDelays)
 68.         return;
 69.     cout << ", ";
 70.     if (delay == 0) {
 71.         cout << "按回车单步演示";
 72.         return;
 73.     }
 74.     cout << "延时设置为 ";
 75.     if (delay == -1)
 76.         cout << 0;
 77.     else
 78.         cout << delay;
 79.     cout << "ms";
 80. }
 81. void gui_drawCylinders() {
 82.     int initx = HDC_Start_X;
 83.     int inity = HDC_Start_Y;
 84.     int width = HDC_Base_Width * 23;
 85.     int high = HDC_Base_High * (10 + 2);
 86.     int disNext = width +
HDC_Underpan_Distance;
 87.     for (int i = 0; i < 3; i++) {
 88.         hdc_rectangle(initx + i * disNext,
inity, width, HDC_Base_High, HDC_COLOR[11]);
 89.         Sleep(HDC_Init_Delay);
 90.     }
 91.     for (int i = 0; i < 3; i++) {
 92.         int mid = initx + width / 2 + i *
disNext;
 93.         hdc_rectangle(mid - HDC_Base_Width
/ 2, inity - high, HDC_Base_Width, high,
HDC_COLOR[11]);
 94.         Sleep(HDC_Init_Delay);
 95.     }
 96. }
 97. void gui_drawDisks() {
 98.     int widthPlate = HDC_Base_Width * 23;
 99.     int initx = HDC_Start_X + widthPlate
/ 2 - HDC_Base_Width / 2;
100.     int inity = HDC_Start_Y;
101.     int disNext = widthPlate +
HDC_Underpan_Distance;
102.     for (int t = 0; t < 3; t++) {
103.         for (int i = 0; i < top[t]; i++) {
104.             int width = (2 * stack[t][i] +
1) * HDC_Base_Width;
105.             int X = initx - stack[t][i] *
HDC_Base_Width;
106.             int Y = inity - (i + 1) *
HDC_Base_High;
107.             hdc_rectangle(X, Y, width,
HDC_Base_High, HDC_COLOR[stack[t][i]]);
108.             Sleep(HDC_Init_Delay);
109.         }
110.         initx += disNext;
111.     }
112. }
113. void gui_moveDisk(char src, char dst) {
114.     int srcT = top[tag(src)];
115.     int dstT = top[tag(dst)] - 1;
```

```
116.     int srcN = stack[tag(src)][srcT];
117.     int width = (2 * srcN + 1) *
HDC_Base_Width;
118.     int high = HDC_Base_High;
119.     int widthPlate = HDC_Base_Width * 23;
120.     int highPlate = HDC_Base_High * (10 +
2);
121.     int disNext = widthPlate +
HDC_Underpan_Distance;
122.     int baseX = HDC_Start_X + widthPlate
/ 2 - HDC_Base_Width / 2;
123.     int baseY = HDC_Start_Y;
124.     int srcX = baseX + tag(src) * disNext
- srcN * HDC_Base_Width;
125.     int srcY = baseY - (srcT + 1) *
HDC_Base_High;
126.     int dstX = baseX + tag(dst) * disNext
- srcN * HDC_Base_Width;
127.     int dstY = baseY - (dstT + 1) *
HDC_Base_High;
128.     int srcPillarX = baseX + tag(src) *
disNext;
129.     int dstPillarX = baseX + tag(dst) *
disNext;
130.     int PillarY = baseY - HDC_Base_High *
(10 + 2);
131.
132.     int posX = srcX, posY = srcY;
133.     while (posY > HDC_Top_Y) {
134.         posY -= HDC_Step_Y;
135.         hdc_rectangle(posX, posY + high,
width, HDC_Step_Y, HDC_COLOR[0]);
136.         if (posY >= PillarY -
HDC_Base_High)
137.             hdc_rectangle(srcPillarX, posY
+ high, HDC_Base_Width, HDC_Step_Y,
HDC_COLOR[11]);
138.         hdc_rectangle(posX, posY, width,
HDC_Step_Y, HDC_COLOR[srcN]);
139.         func_Delay();
140.     }
141.     while (posX != dstX) {
142.         if (srcX < dstX) {
143.             hdc_rectangle(posX, posY,
HDC_Step_X, high, HDC_COLOR[0]);
144.             hdc_rectangle(posX + width,
posY, HDC_Step_X, high, HDC_COLOR[srcN]);
145.             posX += HDC_Step_X;
146.         }
147.         else {
148.             posX -= HDC_Step_X;
149.             hdc_rectangle(posX + width,
posY, HDC_Step_X, high, HDC_COLOR[0]);
150.             hdc_rectangle(posX, posY,
HDC_Step_X, high, HDC_COLOR[srcN]);
151.         }
152.         func_Delay();
153.     }
154.     while (posY < dstY) {
155.         hdc_rectangle(posX, posY, width,
HDC_Step_Y, HDC_COLOR[0]);
156.         if (posY >= PillarY)
157.             hdc_rectangle(dstPillarX,
posY, HDC_Base_Width, HDC_Step_Y,
HDC_COLOR[11]);
158.         hdc_rectangle(posX, posY + high,
width, HDC_Step_Y, HDC_COLOR[srcN]);
159.         posY += HDC_Step_Y;
160.         func_Delay();
161.     }
162. }
163. void hanoi(int n, char src, char tmp,
char dst, bool option[], bool demo = false)
164. {
165.     if (n == 0)
166.         return;
167.     hanoi(n - 1, src, dst, tmp, option);
168.     pop(src);
169.     push(dst, n);
170.     cntAll++;
171.     info_printInfo(option, n, src, dst);
172.     if(showVer)
173.         char_moveDisks(option, n, src,
dst);
174.     if(showGraph)
175.         gui_moveDisk(src, dst);
176.     func_Delay();
177.     hanoi(n - 1, tmp, src, dst, option);
178. }
179. void hanoi_basic() {
180.     bool option[4] = { false, false,
false, false };
181.     int n;
182.     char src, tmp, dst;
183.     info_Input(n, src, tmp, dst);
184.     cntAll = 0;
185.     hanoi(n, src, tmp, dst, option);
186.     func_Ending(option);
187. }
188. void hanoi_basic_step_record() {
189.     bool option[4] = { true, false,
false, false };
190.     int n;
191.     char src, tmp, dst;
192.     info_Input(n, src, tmp, dst);
193.     info_Init(n, src);
194.     hanoi(n, src, tmp, dst, option);
195.     func_Ending(option);
196. }
197. void hanoi_array_display_horizontal() {
198.     bool option[4] = { true, true, false,
false };
199.     int n;
200.     char src, tmp, dst;
201.     info_Input(n, src, tmp, dst);
202.     info_Init(n, src);
203.     hanoi(n, src, tmp, dst, option);
204.     func_Ending(option);
205. }
206. void
hanoi_array_display_vertical_horizontal() {
```

```
207.     bool option[4] = { true, true, true,
false };
208.     int n;
209.     char src, tmp, dst;
210.     info_Input(n, src, tmp, dst, 200);
211.     info_Init(n, src);
212.     cct_cls();
213.     char_drawDisks(option);
214.     info_printInfo(option);
215.     info_showStauts(n, src, dst);
216.     func_Delay();
217.     hanoi(n, src, tmp, dst, option);
218.     func_Ending(option);
219. }
220. void
hanoi_graphics_prepare_draw_cylinders() {
221.     bool option[4] = { false, false,
false, true };
222.     cct_cls();
223.     hdc_cls();
224.     hdc_init();
225.     gui_drawCylinders();
226.     func_Ending(option);
227. }
228. void hanoi_graphics_prepare_draw_disks()
{
229.     bool option[4] = { false, false,
false, true };
230.     int n;
231.     char src, tmp, dst;
232.     info_Input(n, src, tmp, dst);
233.     info_Init(n, src);
234.     cct_cls();
235.     hdc_cls();
236.     hdc_init();
237.     info_showStauts(n, src, dst, false);
238.     gui_drawCylinders();
239.     gui_drawDisks();
240.     func_Ending(option);
241. }
242. void hanoi_graphics_first_move() {
243.     bool option[4] = { false, false,
false, true };
244.     int n;
245.     char src, tmp, dst;
246.     info_Input(n, src, tmp, dst, 20);
247.     info_Init(n, src);
248.     cct_cls();
249.     hdc_cls();
250.     hdc_init();
251.     info_showStauts(n, src, dst, false);
252.     gui_drawCylinders();
253.     gui_drawDisks();
254.     if (n % 2 == 0) {
255.         pop(src);
256.         push(tmp, n);
257.         gui_moveDisk(src, tmp);
258.     }
259.     else {
260.         pop(src);
261.         push(dst, n);
262.         gui_moveDisk(src, dst);
263.     }
264.     func_Ending(option);
265. }
266. void hanoi_graphics_auto_move() {
267.     bool option[4] = { true, true, true,
true };
268.     int n;
269.     char src, tmp, dst;
270.     info_Input(n, src, tmp, dst, 20);
271.     info_Init(n, src);
272.     cct_cls();
273.     hdc_cls();
274.     hdc_init();
275.     info_printInfo(option);
276.     info_showStauts(n, src, dst, false);
277.     char_drawDisks(option);
278.     gui_drawCylinders();
279.     gui_drawDisks();
280.     hanoi(n, src, tmp, dst, option);
281.     func_Ending(option);
282. }
283. void hanoi_graphics_game() {
284.     bool option[4] = { true, true, true,
true };
285.     int n;
286.     char src, tmp, dst;
287.     info_Input(n, src, tmp, dst, 20);
288.     info_Init(n, src);
289.     cct_cls();
290.     hdc_cls();
291.     hdc_init();
292.     info_printInfo(option);
293.     info_showStauts(n, src, dst, false);
294.     char_drawDisks(option);
295.     gui_drawCylinders();
296.     gui_drawDisks();
297.     while (top[tag(dst)] < n) {
298.         char from, to;
299.         info_readCommand(from, to);
300.         if (from == 'Q')
301.             break;
302.         int fromT =
stack[tag(from)][top[tag(from)] - 1];
303.         push(to, fromT);
304.         pop(from);
305.         cntAll++;
306.         info_printInfo(option, fromT,
from, to);
307.         if (showVer)
308.             char_moveDisks(option, fromT,
from, to);
309.         if (showGraph)
310.             gui_moveDisk(from, to);
311.     }
312.     func_Ending(option);
313. }
```