ОКОННАЯ ГРАФИКА ПОЗИЦИОННЫХ ИГР

ЗАДАНИЕ

Разработать графическую программу для игры Норткотта, в которой два противника должны поочередно передвигать свои фишки по рядам клеток прямоугольного игрового поля. В каждом его ряду игроки имеют по одной своей фишке, которая может занимать любую клетку от своего края до фишки противника. На каждом ходе игрок должен переставить свою фишку в одном из рядов на любую клетку в указанных пределах.

Цель работы

Создание патовой позиции для противника, где нет доступных клеток, чтобы сделать ответный ход.

Графический интерфейс игры

Игровой интерфейс должен быть программно реализован в графическом окне, фиксированный габарит которого заполнен горизонтальными рядами одинаковых клеток. Число рядов и клеток в каждом из них должно задаваться геометрическим аргументом командной строки вызова программы вместе с начальным расположением ее окна (16x4+0+0 по умолчанию).

Стратегия игры

- Фишки игроков должны обозначаться символами 0 и X, которые выровнены по центру в занятых ими клетках. Сначала все метки фишек должны располагаться в клетках левого (0) и правого (X) краев игрового поля.
- На каждом ходе должна производиться перестановка одной фишки 0 или X щелчком, соответственно, левой или правой кнопки мыши по любой доступной клетке ее ряда между фишками.
- Если указана клетка слева от фишки 0 или справа от фишки X, то в нее должна быть переставлена ближайшая фишки ряда по щелчку любой кнопки мыши. При любом выборе в ответ должно последовать автоматическое перемещение фишки противника, ряд и ход которой определяется по выигрышной стратегии программы.
- За конечное число ходов фишек партия игры должна завершаться патовой позицией с левой или правой стороны игрового поля. Чтобы начать новую партию игры, в программе должен быть реализован возврат фишек в исходную позицию при нажатии клавиши ESC на клавиатуре.

Стратегия игры

- Кроме того, должна быть предусмотрена установка начальной позиции с произвольным расположением фишек на игровом поле и корректировка текущей позиции.
- При этом принудительная перестановка любой фишки 0 или X должна производиться как ход по щелчку, соответственно, левой или правой кнопки мыши, но при нажатой клавише CTRL на клавиатуре и без автоответа противника.
- После получения желаемой расстановки фишек игра может быть продолжена. Корректное завершение программы должно обеспечивать нажатие комбинации клавиш CTRL-0 или CTRL-X на клавиатуре.
- При разработке программы должна применяться полиоконная графическая технология, в соответствии с которой клетки и ряды игрового поля должны быть реализованы массивами графических окон с обработкой событий и изображений в них библиотечными функциями базового программного интерфейса X Window System.

пример возможного расположения фишек на игровом поле с размерами (16x4)

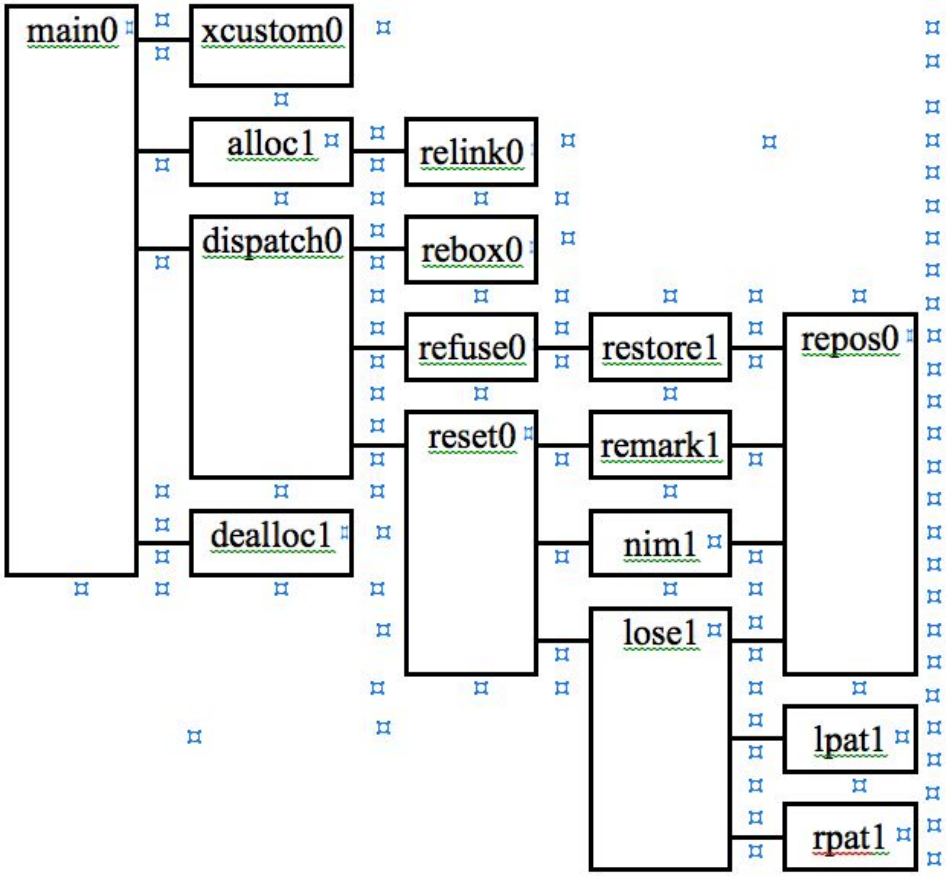
Указанные размеры таблицы устанавливаются по умолчанию

И	Г	Р	О	В	О	Е		П	О	Л	Е				
				O	X										
						O							X		
O	X														
														O	X

Структура программы

- Исходный код программы Норкотта (xpat) разделяется на два модуля для реализации графического интерфейса игры (xpat0) и игровых действий (xpat1).
- В каждом модуле специфицировано 8 прикладных функций.
- В графический модуль входят следующие функции: main (основная функция), relink (адресация разделяемых данных), xcustom (настройка параметров), dispatch (диспетчер событий), refuse (прервать партию), reset (выбор хода и корректировка позиции), repos (позиционирование фишки), rebox (перерисовка фишек).
- Игровой модуль состоит из следующих функций: alloc (распределение разделяемых данных), dealloc (очистка памяти), restore (восстановление исходной позиции), remark (перестановка фишки), nim (выигрышный ход), lose (вынужденный ответ), lpat и rpat (проверка пата слева и справа).

Иерархическая структура функций управления



Графический модуль (xpat0)

Исходный текст графического модуля (xpat0) начинает подключение стандартных заголовков базовой X графики, графических утилит для взаимодействия с оконным менеджером и для макроопределения логических кодов клавиш клавиатуры. Указанные заголовки подключаются следующими директивами:

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <X11/keysymdef.h>
```

статические переменные

Для передачи графических данных между прикладными функциями модуля вводятся следующие внешние статические переменные:

```
static Display *dpy;        /* Адрес дисплейной структуры */
static Window desk;     /* Окно игрового поля программы */
static GC gc[2];      /* Графические контексты для клеток */
static Xrectangle cell;     /*Ячейка литеры фишки в клетке */
static char* mark[]={"0", "X"};              /* Литерыфишек */
static int X0=0;              /* Начальные координаты окна */
static int Y0=0;                    /* программы на экране */
```

Внешние статические переменные

Кроме того, вводятся следующие внешние статические переменные для разделяемых игровых данных:

```
static unsigned NY=4;              /* Число рядов клеток */
static unsigned NX=16;             /* Число клеток ряда */
static Window* row;              /* Адрес массива NY рядов */
static Window** box;     /* Адрес массива NYxNX клеток */
static unsigned** pos;     /* Адрес массива NYx2 позиций
```

Эти игровые данные используются функциями обоих модулей, образуя общее адресное пространство, которое разделяется ими. Габариты игрового поля (NXxNY) определяются в функции main из аргумента командной строки вызова программы, а функции игрового модуля распределяют соответствующего размера разделяемые массивы для окон рядов (row), клеток (box) и позиций фишек (pos).

Передача их адресов в графический модуль

Передача их адресов в графический модуль осуществляется через параметры вызова его прикладной функции relink, исходный код которой приведен ниже.

```
/* Адресация разделяемых данных */
 int relink(void** p, void* r, void** b) {
pos = (unsigned**) p;       /* Адрес массива фишек */
row = (Window* ) r;         /* Адрес массива рядов */
box = (Window**) b;         /* Адрес массива клеток */
return(0);
} /* relink */
```

Инициализация графических и разделяемых игровых данных

- Для инициализации графических и разделяемых игровых данных вызывается функция настройки xcustom. Эта функция создает иерархию графических окон для игрового поля (desk), рядов (row) и клеток (box), фиксируя их идентификаторы, геометрию и обработку событий.
- Окно игрового поля снабжено обрамлением оконного менеджера, который устанавливает его размер и положением на экране по аргументам командной строки вызова программы. При этом ряды создаются как визуально неразличимые подокна окна игрового поля без рамок, которые должны получать события при выборе их клеток.
- Клетки создаются как подокна рядов, которые регулярно заполняют их пространство с фиксированными промежутками. Для окон клеток предусматривается перерисовка литер фишек при потере изображения. Кроме создания окон, функция xcustom загружает графический шрифт (9x15) литер фишек и фиксирует их положение в окнах клеток. Также создается массив из пары графических контекстов для рисования gc[0] и стирания gc[1] литер фишек этим шрифтом в окнах клеток.

Исходный код функции xcustom

Исходный код рассмотренной функции xcustom для перечисленных настроек имеет следующий вид:

```
 /* Настройка графических параметров */
int xcustom() {
static unsigned XSP=4;     /* зазор между клетками ряда */
static unsigned YSP=4;              /* зазор между рядами */
static unsigned BW=32;                    /* ширина клетки */
static unsigned BH=32;                    /* высота клетки */
int x, y;                                 /* позиции окон */
unsigned w, h;                            /* габариты окон */
int dx, dy;                          /* зазоры между окнами */
XGCValues gval;   /* структура графического контекста*/
int depth = DefaultDepth(dpy, 0);      /* глубина экрана 0 */
Window root;                         /* корневое окно экрана */
XSetWindowAttributes attr;                /* атрибуты окон */
unsigned long amask;          /* маска оконных атрибутов */
XSizeHints hint;     /* геометрия оконного менеджмента */
XFontStruct* fn;      /* параметры шрифта литер фишек */
int i, j;                                 /* индексы окон */
```

Модуль расчета геометрии ячеек

```
/* Расчет геометрии ячеек литер фишек */

fn = XLoadQueryFont(dpy, "9x15"); /* Загрузка шрифта */
cell.width = fn->max_bounds.width;
cell.height = fn->max_bounds.ascent + fn->max_bounds.descent;
cell.x = (BW − fn->max_bounds.width)/2;
dy = (fn->max_bounds.ascent − fn->max_bounds.descent);
cell.y = BH/2 + dy / 2;
```

Исходный код настройки графических контекстов

```
/* Настройка графических контекстов */

root = DefaultRootWindow(dpy);   /* корневое окно экрана */
gval.font = fn->fid;                    /* идентификатор шрифта */
gval.foreground = 0;                  /* цвет изображения (black) */
gc[0] = XCreateGC(dpy, root, GCFont | GCForeground, &gval);
gval.foreground = 0x00FFFF;         /* cyan = box background */
gc[1] = XCreateGC(dpy, root, GCFont | GCForeground, &gval);
```

Исходный код настройки игрового окна

```
/* Настройка игрового окна программы */

attr.override_redirect = False;    /* WM обрамление окна */
attr.background_pixel = 0xFFFFFF;  /* white */
amask = (CWOverrideRedirect | CWBackPixel);
w = NX*BW + (NX + 1)*XSP + 2*NX + 2; /* Габариты */
h = NY*BH + (NY + 1)*YSP + 4*NY; /* игрового окна */
x = X0; y = Y0;    /* Начальные координаты окна игры */
desk = XCreateWindow(dpy, root, x, y, w, h,1, depth,
 InputOutput, CopyFromParent, amask, &attr);
```

Параметры оконного менеджера

```
/* Геометрические рекомендации оконного менеджера */

hint.flags = (PMinSize | PMaxSize | PPosition);
hint.min_width = hint.max_width = w;     /* Фиксировать */
hint.min_height = hint.max_height = h;       /* габариты и */
hint.x = x; hint.y = y;      /* позицию окна игрового поля */
XSetNormalHints(dpy, desk, &hint);    /* в свойстве WM */
XStoreName(dpy, desk, "xpat");           /* Заголовок окна */
```

Параметры подокон рядов клеток

```
/* Настройка подокон рядов клеток */

amask = CWOverrideRedirect | CWBackPixel | CWEventMask;
attr.override_redirect = True;     /* Отмена обрамления окна */
attr.background_pixel = 0xFFFFFF;                /* Белый фон */
attr.event_mask = (ButtonPressMask | KeyPressMask);
w = NX*BW + (NX − 1)*XSP + 2*NX + 2; /* Ширина ряда */
h = BH + 2 + 2;              /* Высота клетки ряда + 2 рамки */
x = XSP;                                  /* Горизонтальный отступ */
dy = h + YSP;          /* Высота ряда + Вертикальный зазор */
for(i = 0, y = YSP; i < NY; i++, y += dy)   /* Цикл по рядам */
  row[i] = XCreateWindow(dpy, desk, x, y, w, h, 0, depth,
              InputOutput, CopyFromParent, amask, &attr);
```

Параметры окон клеток

```
/* Настройка окон клеток */

amask = CWOverrideRedirect | CWBackPixel | CWEventMask;
attr.override_redirect = True;      /* Отмена обрамления окна */
attr.background_pixel = 0x00FFFF; /* cyan = gc1 background*/
attr.event_mask = (KeyPressMask | ExposureMask);
w = BW; h = BH;                     /* Габариты окна клетки */
dx = w + XSP + 2;  /* Ширина окна клетки + рамка + зазор */
for(i = 0, y = 0; i < NY; i++)      /* Цикл по рядам клеток */
  for(j = 0, x=0; j < NX; j++, x += dx) /* Создать окна клеток */
    box[i][j] = XCreateWindow(dpy, row[i], x, y, w, h, 1, depth,
                    InputOutput, CopyFromParent, amask, &attr);
```

Отображение окон на экране

```
/* Отображение всех окон на экране */

XMapWindow(dpy, desk);
XMapSubwindows(dpy, desk);
for(i = 0; i < NY; i++)
  XMapSubwindows(dpy, row[i]);
return(0);
} /* xcustom */
```

Функция циклической обработки событий

- После начальных установок, распределений и настроек вызывается прикладная функция dispatch для обслуживания игровых партий.
- Она реализует циклическую обработку событий, которые передаются из очереди X-сервера окнам программы по запросу XNextEvent.
- Маски событий, установленные для окон программы функцией настройки xcustom, предусматривают обработку событий при потере изображения (Expose) окон литерных клеток, а также нажатия кнопок мыши (ButtonPress) и клавиш клавиатуры (KeyPress) в рядах клеток.
- Функция dispatch дифференцирует обработку событий по указанным типам, адресуя их структуры прикладным функциям rebox, reset и refuse

Исходный код функции dishatch()

```
int dispatch() {    /* Диспетчер событий */
XEvent event;      /* структура событий */
int done = 0;                  /* флаг выхода */
while(done == 0) {    /* цикл обработки событий */
  XNextEvent(dpy, &event);     /* чтение событий */
  switch(event.type) {
    case Expose: rebox(&event);      /* перерисовка */
                    break;
    case ButtonPress: reset(&event);
                        break;  /* позиционирование */
    case KeyPress: done = refuse(&event);
                      break;              /* прерывание */
    default: break;
  } /* switch */
} /* while */
return(0);
} /* dispatch */
```

Функция перерисовки литерных клеток

Функция rebox вызывается диспетчером событий dispatch для перерисовки литер фишек в окнах клеток, чтобы восстановить изображение при получении события типа Expose любым из них. Идентификация таких окон осуществляется по полю window адресованного события при просмотре массивов клеток и позиций фишек, как показано в следующем исходном коде функции rebox.

```
/* Перерисовка литерных клеток */
int rebox(XEvent* ev) {
int i, j;                          /* индексы рядов и клеток */
for(i = 0; i < NY; i++)                    /* Цикл по рядам */
  for(j = 0; j < 2; j++)   /* Контроль пары фишек ряда */
    if(ev->xexpose.window == box[i][pos[i][j]])
      XDrawString(dpy, box[i][pos[i][j]], gc[0],
                        cell.x, cell.y, mark[j], 1);
return(0);
} /* rebox */
```

Графическое обслуживание ходов партии

Функция reset обеспечивает графическое обслуживание ходов партии или корректировку позиции фишек по событию от нажатия кнопок мыши в окнах клеток игрового поля. Индексы выбранной клетки ряда определяются по полям window (ряд) и subwindow (клетка) структуры XEvent адресованного мышиного события при просмотре массивов окон рядов (row) и клеток (box). Переставляемая фишка ряда сначала выбирается по номеру нажатой кнопки мыши в поле button структуры XEvent того же события. Функция remark меняет ее на другую фишку ряда, если она не может быть переставлена в указанную клетку по правилам игры. В любом случае функция remark вызывает функцию repos для позиционирования выбранной фишки в указанную клетку. Дальнейшая обработка зависит от состояния клавиши CTRL на клавиатуре, которое контролируется по полю state структуры XEvent мышиного события. Если клавиша CTRL нажата, то передвижение фишки считается корректировкой позиции и функция reset завершается без ответного хода. В противном случае, если клавиша CTRL не нажата вместе с кнопкой мыши, интерактивное перемещение фишки считается ходом игрока и требуется ответ. Для этого по позиции автоматически выбирается выигрышный или вынужденный ход, который реализует вызов игровой функции nim или lose, соответственно, для перестановки фишки противника. Рассмотренная функция reset является наиболее логически сложной в графическом модуле. Ее разнообразные возможности реализует следующий исходный код.

Исходный код функции игры

```
/* Ход игры или корректировка позиций */
int reset(XEvent* ev) {
int m;        /* индекс фишки в ряду */
int i, j;     /* индексы ряда и клетки */
for(i = 0; i < NY; i++)              /* идентификация ряда */
  if(ev->xbutton.window == row[i])
    break;
if(i == NY) return(NY);
for(j = 0; j < NX; j++)     /* идентификация клетки ряда */
  if(ev->xbutton.subwindow == box[i][j])
     break;
if(j == NX) return(NY);
m = (ev->xbutton.button == Button1) ? 0 : 1; /* O или X ?*/
if(remark(i, j, &m) == NY)        /* перестановка фишки */
  return(NY);
if(ev->xbutton.state & ControlMask)   /* Корректировка */
  return(i);                          /* позиции без ответа */
m = (m == 0) ? 1 : 0;   /* идентификация фишки ответа */
```

Прерывание партии игры

- Функцию refuse вызывает диспетчер событий dispatch для обработки клавиатурных событий от нажатия клавиш ESCAPE и CTRL-O(0) или CTRL-X при отказе продолжать партию игры.
- Для идентификации логических кодов клавиш по адресуемой структуре клавиатурного события применяется запрос XLookupKeysym. Полученные логические коды используются для выбора альтернатив функциональной обработки.
- В частности по клавише ESCAPE вызывается функция restore, чтобы восстановить исходную позицию фишек для новой партии. Нажатие клавиши 0, O(o) или X(x) в сочетании с клавишей CTRL, состояние которой контролируется по полю state в структуре XEvent клавиатурного события, обеспечивает возврат ненулевого кода в функцию диспетчера событий dispatch.
- Представленный ниже исходный код функции refuse реализует рассмотренную схему обработки клавиатурных событий.

Исходный код функции refuse()

```
/* Прерывание партии игры */
int refuse(XEvent* ev) {
KeySym ks = XLookupKeysym((XKeyEvent *) ev, 0);
switch(ks) {
  case XK_Escape: restore();     /* начать новую партию */
                          break;
  case XK_0:    /* 0x30 ASCII */
  case XK_o:    /* 0x6F или 0x4F ASCII */
  case XK_x: if(ev->xkey.state & ControlMask)
                    return(1);          /* код конца игры */
                  break;
  default: break; } /* switch */
return(0);       /* код продолжения игры */
} /* refuse */
```

Реализация перестановки литеры фишки в требуемую позицию

- Различные игровые функции, которые вызываются в альтернативах цикла обработки событий, обращаются к функции repos для реализации перестановки литеры фишки в требуемую позицию.

- При обращении ей передается номер ряда, индекс фишки и номер клетки, куда ее переставить. Номер прежней клетки фишки может быть легко получен из массива позиций по ее индексу и номеру ряда.
- При перестановке заданной фишки в массиве позиций фиксируется номер ее новой клетки.

- Для отображения результата перестановки литера фишки стирается в окне своей прежней клетки и рисуется в окне новой клетки ряда по запросу XDrawString с соответствующими графическими контекстами.

Исходный код функции repos()

Исходный код рассмотренной перестановки и перерисовки фишек имеет следующий вид:

```
int repos(int i, int j, int m) {     /* Переставить фишку ряда */
int p = pos[i][m];       /* прежняя позиция фишки m ряда i */
XDrawString(dpy, box[i][p], gc[1], cell.x, cell.y, mark[m], 1);
pos[i][m] = j;  /* зафиксировать новую позицию j фишки */
XDrawString(dpy, box[i][j], gc[0], cell.x, cell.y, mark[m], 1);
XFlush(dpy);
return(i);                /* возврат ряда перестановки фишки */
} /* repos */
```

Основная функция main()

Спецификации графического модуля завершает исходный код основной функции main. В его начале по геометрическому запросу XParseGeometry производится разбор аргумента командной строки программы, чтобы определить габариты игрового поля и начальные координаты его окна на экране. Недостающие параметры задаются значениями по умолчанию. В любом случае полученные габариты передаются функции alloc для доступа игрового модуля и распределения массивов разделяемых данных.

После этого по запросу XOpenDisplay устанавливается контакт с X-сервером, затем вызывается функция xcustom для инициализации данных, наконец, реализуется цикл обработки событий игровых партий в функции dispatch. Для корректного завершения программы в функции main предусмотрено закрытие всех окон по запросам XDestroy(Sub)Window(s), разрыв связи с X-сервером по запросу XCloseDisplay и освобождение памяти разделяемых игровых массивов функцией dealloc. Следующий исходный код содержит перечисленные функциональные вызовы основной функции main.

Исходный код основной функции

```
/* Основная функция */
int main(int argc, char* argv[]) {
if(argc < 2)
   fprintf(stderr, "Default: xpat 16x4+0+0\n");
XParseGeometry(argv[1], &X0, &Y0, &NX, &NY);
alloc(NX, NY);
dpy = XOpenDisplay(NULL);
xcustom();
dispatch();
XDestroySubwindows(dpy, root);
XDestroyWindow(dpy, root);
XCloseDisplay(dpy);
dealloc(pos, box, row);
return(0); } /* main */
```

Функции игрового модуля для реализации игровых действий без графических запросов

Исходный текст игрового модуля (xpat1) начинается подключением заголовка стандартной библиотеки системы программирования С следующей директивой:

```
#include <stdlib.h>
```

Для удобства спецификации разделяемых данных прикладных функций игрового модуля (пере)определяются следующие внешние статические переменные:

```
static unsigned NY;      /* число рядов клеток */
static unsigned NX;      /* число клеток ряда */
static unsigned **pos;   /* адрес массива позиций фишек */
```

Эти внешние переменные являются одноименными эквивалентами тех же разделяемых данных графического модуля. Их передачу и распределение обеспечивает вызов прикладной функции alloc из основной функции main.

При вызове ей передаются габариты игрового поля из командной строки, чтобы фиксировать размеры и распределить адресное пространство разделяемых игровых массивов рядов, клеток и фишек с помощью стандартной функции calloc.

Исходный код перечисленных передаточных и распределительных действий функции alloc

Чтобы адресовать перечисленные массивы в графический модуль, предусмотрен вызов функции relink. Кроме того, адрес массива фишек (pos) фиксируется вместе с габаритами игрового поля (NX, NY) для удобства доступа функций игрового модуля.

Исходный код функции alloc имеет следующий вид.

```
int alloc(int _nx, int _ny) {/* Рас(пре)деление данных */
void** p;                    /* адрес массива фишек */
void* r;                     /* адрес массива рядов клеток */
void** b;                    /* адрес массива клеток */
int i;                       /* индекс ряда клеток */
NX = _nx; NY = _ny;   /* Фиксировать игровые габариты */
p = calloc(NY, sizeof(unsigned*));     /* Адреса пар фишек */
r = calloc(NY, sizeof(unsigned long));       /* Массив рядов */
b = calloc(NY, sizeof(void*));     /* Массив адресов клеток */
```

Продолжение исходного кода функции alloc

```
for(i = 0; i < NY; i++) {        /* Распределение памяти для */
  b[i] = calloc(NX, sizeof(unsigned long));     /* клетки ряда */
  p[i] = calloc(2, sizeof(unsigned));       /* пары фишек ряда */
} /* for */
relink(p, r, b);            /* Адресация в графический модуль */
for(i = 0, pos = (unsigned**) p; i < NY; i++) { /* Фиксация */
  pos[i][0] = 0; pos[i][1] = NX - 1;                /* начальной */
} ./* for */                                      /* позиции фишек */
return(0);
} /* alloc */
```

Освобождение распределенной памяти разделяемых массивов рядов

Чтобы освободить распределенную память разделяемых массивов рядов, клеток и фишек в конце программы, предусмотрен вызов прикладной функции dealloc из основной функции main. При вызове ей передаются адреса указанных массивов для корректной очистки памяти стандартной функцией free, как в следующем исходном коде.

```
int dealloc(void** p, void** b, void* r) { /* Чистка памяти */
int i;        /* индекс ряда */
for(i = 0; i < NY; i++) {                  /* цикл по рядам */
  free(b[i]);                          /* чистка памяти клетки */
  free(p[i]);                  /* чистка памяти пары фишек */} /* for */
free(p);             /* чистка массива адресов пар фишек */
free(b);                  /* чистка массива адресов клеток */
free(r);                    /* чистка массива рядов клеток */
return(0); } /* dealloc */
```

Функции модуля, непосредственно обеспечивающие реализацию игровых партий

- Функции alloc и dealloc обслуживают доступ к игровым данным, однако, не выполняют никаких игровых действий. Они включены в игровой модуль, чтобы уровнять функциональный состав модулей. Остальные функции этого модуля непосредственно обеспечивают реализацию игровых партий.
- Самая простая из них функция restore применяется для установки фишек в исходную позицию по противоположным краям игрового поля. Перестановку фишек каждого ряда обеспечивают два последовательных вызова функции repos (из графического модуля) с соответствующими индексами рядов клеток и фишек, как показано в следующем фрагменте исходного кода:

```
int restore() {   /* Установка начальной позиции */
int i;              /* индекс ряда */
for(i = 0; i < NY; i++) {    /* цикл по рядам игрового поля */
  repos(i, 0, 0);              /* левый край поля для фишки 0 */
  repos(i, (NX − 1), 1);  /* правый край поля для фишки X */} /* for */
return(0); } /* restore */
```

Графические функции для ходов партии или корректировки позиции

- Функцию restore вызывает функция refuse графического модуля по нажатию клавиши ESCAPE, чтобы начать новую партию.
- Остальные игровые функции явно или косвенно вызывает графическая функция reset для ходов партии или корректировки позиции.
- В частности, функция remark вызывается для контролируемой перестановки одной из фишек заданного ряда в указанную клетку по щелчку левой или правой кнопки мыши. При вызове в функцию remark передаются соответствующие индексы клетки игрового поля, а также адресуется номер фишки ряда, чтобы вернуть результат ее выбора. Выбор фишки в функции remark зависит от позиции указанной клетки. Если она находится между фишками, то в нее должна быть передвинута фишка, адресованная аргументом функции. Иначе указанную клетку должна занять ближайшая фишка ряда, а ее индекс фиксируется и адресуется аргументом функции remark. В любом случае ее выполнение завершает вызов функции repos для графического позиционирования передвигаемой фишки.

Исходный код, реализующий действие функции remark

Следующий исходный код реализует действие функции remark.

```
int remark(int i, int j, int* m) {/* Перестановка фишки */
if(j < pos[i][0])          /* перестановка фишки 0 влево */
  (*m) = 0;                /* фиксировать индекс фишки 0 */
if(j > pos[i][1])       /* перестановка фишки X вправо */
  (*m) = 1;                /* фиксировать индекс фишки X */
if(j != pos[i][(*m)])  /* перестановка фишки к центру */
  return(repos(i, j, *m));  /* позиционирование фишки */
return(NY);
} /* remark */
```

Реализация выигрышного ответа фишкой

Самой важной для организации игры является функция nim. Она реализует выигрышный ответ фишкой, которую индексирует аргумент ее вызова из функции reset. Выбор выигрышного хода основан на вычислении НИМ-суммы по модулю 2 для разностей номеров фишек по всем рядам игрового поля.

Если НИМ-сумма оказалась равна 0, то выигрышный ход невозможен, и следует немедленный возврат числа рядов.

Если НИМ-сумма не равна 0, нужно найти ее максимальный значащий бит (MSB), который равен 1. Для ответа выбирается первый по порядку ряд, где разность номеров клеток фишек имеет 1 в разряде MSB. После этого инвертируются все биты разности ряда ответа (0<->1), которые не старше MSB. Результат инверсии дает разность позиций между фишками в этом ряду после выигрышного хода. По этой разности легко вычислить номер новой клетки передвигаемой фишки, относительно позиции парной фишки в том же ряду. Требуемое позиционирование фишки обеспечивается функцией repos с возвратом ряда выигрышного ответа. Выигрышную стратегию функции nim реализует исходный код.

Выигрышный ход для фишки m по стратегии NIMM

```
/* Исходный код выигрышного хода для фишки m по стратегии NIMM */
 int nim(int m) {
unsigned b = 8*sizeof(unsigned);          /* старший 1 бит */
unsigned d=0;                          /* разность позиций (X-0) */
unsigned s=0;                                  /* NIMM-сумма */
int i;                                         /* индекс ряда */
for(i = 0; i < NY; i++)       /* Вычисление NIMM-суммы */
  s ^= (pos[i][1] − pos[i][0] − 1); /* по (X-0) для всех рядов */
if(s == 0)                             /* Нет выигрышного хода */
  return(NY);   /* Идентификация проигрышной позиции */
for( ; b > 0; b >>= 1)                          /* Поиск MSB */
  if(b & s)                 /* (наибольшего значащего бита 1) */
    break;                                     /* NIMM-суммы */
```

Продолжение исходного кода выигрышного хода для фишки m по стратегии NIMM

```
for(i = 0; i < NY; i++)                    /* Выбрать ряд ответа */
  if((d = (pos[i][1] − pos[i][0] − 1)) & b)       /* по разности */
    break;                                 /* позиций фишек и MSB */
for( ; b > 0; b >>= 1)                                 /* Инверсия */
  if(b & s)                                          /* младших битов */
    d = (d & b) ? (d ^ b) : (d | b);       /* разности ряда ответа */
d = (m > 0) ? (d + 1) : −(d + 1); /* Величина сдвига фишки */
repos(i, pos[i][(m+1)%2] + d, m);     /* Позиционирование */
return(i);                  /* Возврат ряда выигрышного ответа */
} /* nim */
```

Если в позиции партии нет выигрышного ответа фишкой с заданным индексом, вызывается функция lose, чтобы сделать вынужденный ход.

Аналогично функции nim, ее вызывает функция reset графического модуля, а ее аргументом является индекс фишки, которая делает ответный ход.

Сначала делается попытка выбрать ряд, где можно сдвинуть фишку с этим индексом навстречу парной фишке ряда на одну клетку, используя функцию repos.

Если это не может быть реализовано, выполняется проверка пата функциями lpat или rpat в зависимости от индекса фишки ответа.

Если нет пата, то их код возврата идентифицирует ряд, где фишка ответа может отступить на одну клетку к своему краю игрового поля. Передвижение фишки как обычно должно осуществляться функцией repos. При пате функция lose завершается без ответа. Рассмотренный выбор вынужденного хода реализует следующий исходный код функции lose.

Вынужденный ответ фишкой m

```
/* Исходный код реализации вынужденного ответа фишкой m */
int lose(int m) {
int i;                                          /* индекс ряда */
int d = (m == 0) ? 1 : −1;  /* оценка направления 1 сдвига */
for(i = 0; i < NY; i++)              /* Выбор ряда для сдвига */
  if((pos[i][1] − pos[i][0]) > 1)            /* по направлению */
    break;                                      /* к парной фишке */
if(i < NY)                          /* Сдвиг вперед на 1 клетку к */
  return(repos(i, pos[i][m] + d, m));  /* парной фишке ряда */
i = (m == 0) ? lpat() : rpat();                  /* контроль пата */
if(i < NY)                                  /* выбор ряда отступа */
  repos(i, pos[i][m] − d, m);   /* отступ на 1 клетку к краю */
return(i);                /* возврат ряда вынужденного ответа */
} /* lose */
```

Функции проверки патовой позиции lpat и rpat

- Спецификации игрового модуля завершает исходный код функций проверки патовой позиции lpat и rpat, которые вызываются функцией lose при выборе вынужденного ответного хода.
- В частности, функция lpat применяется для проверки патовой позиции у левых фишек (0).
- При левом пате все фишки X стоят в клетках с индексом 1 (считая с 0) своих рядов, что можно быстро установить по массиву позиций для индекса 1.
- В этом случае код возврата функции lpat равен числу рядов игрового поля.
- Если пата нет, возвращается номер ряда, где можно сделать ход фишкой 0.

Проверка левого пата в функции lpat

Следующий исходный код реализует рассмотренную проверку левого пата в функции lpat:

```
/* Контроль пата слева для фишек 0 */

 int lpat() {
int i;                          /* индекс ряда */
for(i = 0; i < NY; i++)
  if(pos[i][1] > 1)
    break;
return(i);
} /* leftpat */
```

Проверка правого пата в функции rpat

Функция rpat для проверки пата правых фишек (X) устроена аналогично функции lpat. Только при правом пате все фишки 0 находятся в предпоследней клетке (NX−2, считая с 0) своих рядов, что легко проверяется по массиву позиций для индекса 0. Очевидно, следующий исходный код функции rpat почти совпадает с lpat.

```
/* Контроль пата справа для фишек X */

int rpat() {
int i;                    /* индекс ряда */
for(i=0; i < NY; i++)
  if(pos[i][0] < (NX-1-1))
    break;
return(i);
} /* rightpat */
```

Получение исполняемого файла xpat

Если исходный код модулей программы игры Норкотта сохранен в файлах xpat0.c и xpat1.c, то для их компиляции и компоновки с графической библиотекой нужно выполнить следующую команду:

```
$ cc -o xpat xpat0.c xpat1.c –lX11
```

В результате должен быть получен выполняемый файл xpat. Он может быть вызван с любым реалистичным аргументом в полном или частичном формате спецификации геометрических ресурсов X-клиентов, а также вообще без аргумента. В частности, следующая командная строка образует в правом верхнем углу экрана графическое окно игрового поля с заголовком "xpat", которое имеет 4 ряда по 32 клетки (цвета циан):

```
$ xpat 32x4−0+0 &
```

При вызове программы с аргументом в любом сокращенном формате, например, когда указаны только габариты игрового поля без координатных смещений окна, или только 1 из габаритов, значение недостающих параметров выбираются по умолчанию из строки "16x4+0+0" (4 ряда по 16 клеток в левом верхнем углу +0+0 экрана). Вызов программы без аргумента сопровождает диагностический вывод формата по умолчанию в окне эмулятора терминала.