TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN
MÔN: **LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

HƯỚNG DẪN ĐỒ ÁN

CARO

TP.HCM, ngày 23 tháng 09 năm 2018

MỤC LỤC

1 Giới thiệu

Trong phần đồ án này ta sẽ phối hợp các kĩ thuật và cấu trúc dữ liệu cơ bản để xây dựng một trò chơi đơn giản cờ caro.

Để thực hiện được đồ án này ta cần các kiến thức cơ bản như: thiết kế hướng đối tượng, xử lý tập tin, con trỏ hai chiều…

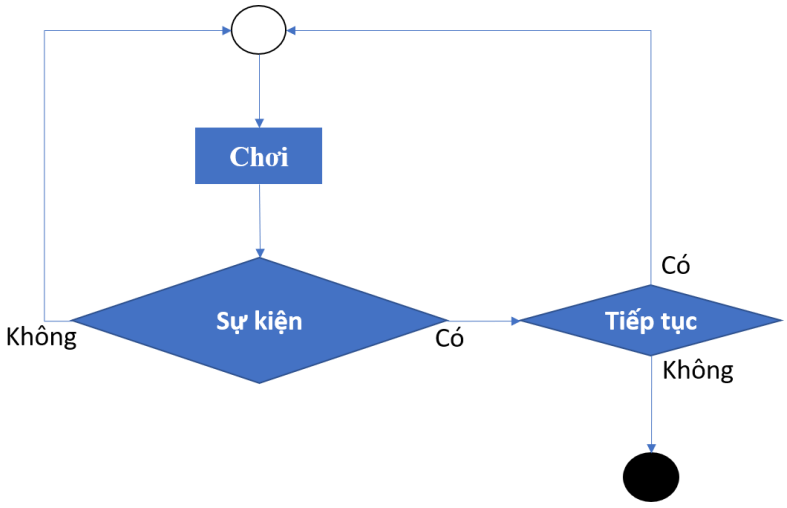
Phần hướng dẫn giúp sinh viên xây dựng trò chơi ở mức độ cơ bản, các em tự nghiên cứu để hoàn thiện một cách tốt nhất có thể.

2 Kịch bản trò chơi

Lúc đầu khi vào game sẽ xuất hiện bảng cờ caro, người chơi sẽ dùng các phím ‘W’, ‘A’, ‘S’, ‘D’ để điều chỉnh hướng di chuyển. Khi người chơi nhấn phím ‘enter’ thì sẽ xuất hiện dấu ‘X’ hoặc ‘O’ tùy vào lượt.

Khi một trong hai người chiến thắng theo luật caro thì màn hình xuất hiện dòng chữ chúc mừng người chiến thắng. Sau đó sẽ hỏi người dùng muốn tiếp tục chơi hay không, nếu chọn phím “y” thì chương trình khởi động lại dữ liệu từ đầu, còn nhấn phím bất kì thì thoát chương trình.

Trường hợp khi vị trí bàn cờ đã kín chỗ thì màn hình xuất hiện dòng chữ ‘Hai ben hoa nhau’. Sau đó hỏi người dùng có muốn thoát hay chơi tiếp tương tự như trên.



Hình 1: Sơ đồ kịch bản trò chơi

3 Các bước xây dựng trò chơi

Trong phần này ta sẽ lần lượt đi qua các bước xây dựng trò chơi. Lưu ý đây chỉ là một gợi ý thiết kế, sinh viên có thể tự thiết kế mẫu phù hợp trong quá trình làm đồ án.

Bước 1: Trong bước này ta xây dựng một lớp dùng chung có tên là _Common. Thực chất giống như lớp giả bao bọc các hàm ta hay dùng. Hàm fixConsoleWindow cố định màn hình với kích thước thích hợp, làm điều này giúp tránh trường hợp người dùng tự co giãn màn hình sẽ gây khó khăn trong quá trình xử lý. Ngoài ra còn có hàm gotoXY để di chuyển dấu nháy chuột tới vị trí thích hợp trên bàn cờ.

Dòng	
1	`class _Common{`
2	`   public:`
3	`      static void fixConsoleWindow();`
4	`      static void gotoXY(int, int);`
5	`};`
6	`void _Common::gotoXY(int pX, int pY) {`
7	`   COORD coord;`
8	`   coord.X = pX;`
9	`   coord.Y = pY;`
10	`   SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);`
11	`}`
12	`void _Common::fixConsoleWindow() {`
13	`   HWND consoleWindow = GetConsoleWindow();`
14	`   LONG style = GetWindowLong(consoleWindow, GWL_STYLE);`
15	`   style = style & ~(WS_MAXIMIZEBOX) & ~(WS_THICKFRAME);`
16	`   SetWindowLong(consoleWindow, GWL_STYLE, style);`
17	`}`

Trong đoạn mã trên, kiểu HWND là một con trỏ trỏ tới chính cửa sổ Console. Để làm việc với các đối tượng đồ họa này, ta cần có những kiểu như thế. Cờ GWL_STYLE được xem là dấu hiệu để hàm GetWindowLong lấy các đặc tính mà cửa sổ Console đang có. Kết quả trả về của hàm GetWindowLong là một số kiểu long, ta sẽ hiệu chỉnh tại dòng số 4. Ý nghĩa là để làm mờ đi nút maximize và không cho người dùng thay đổi kích thước cửa sổ hiện hành. Sau khi đã hiệu chỉnh xong, ta dùng hàm SetWindowLong để gán kết quả hiệu chỉnh trở lại. Ta có thể thử nghiệm hàm trên và tự xem kết quả. Trong trò chơi sẽ có rất nhiều vị trí mà ta muốn in tại đó, vì vậy ta cần có khả năng di chuyển tới tất cả các vị trí trong màn hình console. Trong đoạn mã này ta sử dụng struct _COORD (COORD), đây là một cấu trúc dành xử lý cho tọa độ trên màn hình console. Ta gán hoành độ và tung độ cho biến coord sau đó thiết lập vị trí lên màn hình bằng hàm SetConsoleCursorPosition. Lưu ý: hàm này cần một đối tượng chính là màn hình console

(màn hình đen), vì vậy ta cũng cần có một con trỏ trỏ tới đối tượng này (HANDLE thực chất là void*). Ta có được bằng cách gọi hàm GetStdHandle với tham số là một cờ STD_OUTPUT_HANDLE.

Bước 2: Tiếp theo ta cần xây dựng lớp đại diện cho việc người dùng đánh vào bàn cờ. Ta xem mỗi vị trí người dùng đánh là một điểm gồm tung độ và hoành độ. Ngoài ra, để phân biệt được 'X' hay 'O' ta them một biến để làm dấu, ví dụ biến _check, với _check = -1 sẽ in ra 'X' và _check = 1 sẽ in ra 'O'.

Dòng	
1	`class _Point{`
2	`    int _x, _y, _check;`
3	`    public:`
4	`        _Point();`
5	`        _Point(int, int);`
6	`        bool setCheck(int);`
7	`        int getX();`
8	`        int getY();`
9	`        int getCheck();`
10	`        void setX(int);`
11	`        void setY(int);`
12	`};`
13	`_Point::_Point(){_x = _y = _check = 0;}`
14	`_Point::_Point(int pX, int pY){`
15	`    _x = pX; _y = pY;`
16	`    _check = 0;`
17	`}`
18	`int _Point::getX(){return _x;}`
19	`int _Point::getY(){return _y;}`
20	`int _Point::getCheck(){return _check;}`
21	`void _Point::setX(int pX) {_x = pX;}`
22	`void _Point::setY(int pY) {_y = pY;}`
23	`bool _Point::setCheck(int pCheck){`
24	`    if(pCheck == -1 || pCheck == 1 || pCheck == 0) {`
25	`        _check = pCheck;`
26	`        return true;`
27	`    }`
28	`    return false;`
29	`}`

Bước 3: Bước tiếp theo ta sẽ xây dựng lớp bàn cờ (_Board). Trong lớp này ta sẽ có các thuộc tính cần thiết như kích thước bàn cờ (_size), góc trên trái bắt đầu vẽ bàn cờ (_left &

_top). Cuối cùng là một **con trỏ hai chiều** _pArr kiểu _Point. Để tiện trong quá trình phát triển trò chơi, ta xây dựng một tập các hàm lấy dữ liệu (hàm dạng get…), ví dụ getSize, getLeft, getTop, getXAt(dòng, cột) và getYAt(dòng, cột).

Dòng	
1	`class _Board{`
2	`    private:`
3	`        int _size;`
4	`        int _left, _top;`
5	`        _Point** _pArr;`
6	`    public:`
7	`        int getSize();`
8	`        int getLeft();`
9	`        int getTop();`
10	`        int getXAt(int, int);`
11	`        int getYAt(int, int);`
12	`};`
13	`int _Board::getSize(){return _size;}`
14	`int _Board::getLeft(){return _left;}`
15	`int _Board::getTop(){return _top;}`
16	`int _Board::getXAt(int i, int j){`
17	`    return _pArr[i][j].getX();`
18	`}`
19	`int _Board::getYAt(int i, int j){`
20	`    return _pArr[i][j].getY();`
21	`}`

Bước 4: Tiếp theo ta cần bổ sung vào lớp _Board hàm dựng và hàm hủy. Công dụng hàm dựng là khởi tạo các giá trị ban đầu cho các thuộc tính cũng như khởi tạo vùng nhớ cho biến con trỏ. Công dụng hàm hủy là dọn dẹp tài nguyên đã xin cấp phát trong hàm dựng.

Dòng	
1	`class _Board{`
2	`   _Board(int, int, int);`
3	`   ~_Board();`
4	`   //…`
5	`}`
6	`_Board::_Board(int pSize, int pX, int pY){`
7	`    _size = pSize;`
8	`    _left = pX;`
9	`    _top = pY;`
10	`    _pArr = new _Point*[pSize];`

```
11      for(int i = 0; i < pSize; i++) _pArr[i] = new _Point[pSize];
12  }
13  _Board::~_Board(){
14      for(int i = 0; i < _size; i++) delete[] _pArr[i];
15      delete[] _pArr;
16  }
```

Bước 5: Tiếp theo ta bổ sung cho lớp _Board hàm resetData để khởi động dữ liệu cho ma trận bàn cờ, và hàm drawBoard để vẽ bàn cờ ra màn hình

```
Dòng
1   class _Board{
2     //…
3     void resetData();
4     void drawBoard();
5   };
6   void _Board::resetData() {
7   if(_size == 0) return; // Phải gọi constructor trước khi resetData
8     for(int i = 0 ; i < _size ; i++){
9       for(int j = 0 ; j < _size ; j++){
10        _pArr[i][j].setX(4 * j + _left + 2); // Trùng với hoành độ màn hình bàn cờ
11        _pArr[i][j].setY(2 * i + _top + 1); // Trùng với tung độ màn hình bàn cờ
12        _pArr[i][j].setCheck(0);
13      }
14    }
15  }
16  void _Board::drawBoard(){
17    if(_pArr == NULL) return; // phải gọi constructor trước
18    for(int i = 0; i <= _size ; i++){
19      for(int j = 0; j <= _size ; j++){
20        _Common::gotoXY(_left + 4 * i, _top + 2 * j);
21        printf(".");
22      }
23    }
24    _Common::gotoXY(_pArr[0][0].getX(), _pArr[0][0].getY());// di chuyển vào ô đầu
25  }
```

Bước 6: Cuối cùng, ta bổ sung hàm checkboard để thực hiện cập nhật thuộc tính _check tại vị trí mà người dùng gõ phím ‘enter’ để đánh cờ, tùy vào lượt mà _check sẽ nhận 1 hay -1. Ngoài ra ta them hàm testBoard kiểm tra thắng thua theo luật caro. Đoạn mã bên dưới chỉ là minh họa, sinh viên tự cài đặt sao cho hàm trả về -1 (người thứ 1 thắng), 0 (hòa), 1 (người thứ 2 thắng) hay 2 (chưa ai thắng).

Dòng	
1	`class _Board{`
2	`    //…`
3	`    int checkboard(int, int, bool);`
4	`    int testBoard();`
5	`};`
6	`int _Board::checkBoard(int pX, int pY, bool pTurn){`
7	`  for(int i = 0; i < _size; i++){`
8	`    for(int j = 0; j < _size; j++){`
9	`      if(_pArr[i][j].getX() == pX && _pArr[i][j].getY() == pY && _pArr[i][j].getCheck() == 0){`
10	`        if(pTurn) _pArr[i][j].setCheck(-1);// Nếu lượt hiện hành là true: c = -1`
11	`        else _pArr[i][j].setCheck(1);// Nếu lượt hiện hành là false: c = 1`
12	`        return _pArr[i][j].getCheck();`
13	`      }`
14	`    }`
15	`  }`
16	`  return 0;`
17	`}`
18	`int _Board::testBoard(){return 0;} // Trả mặc định là hòa`

Bước 7: Tiếp theo ta xây dựng lớp _Game để vận hành trò chơi. Trong lớp _Game sẽ chứa một bàn cờ kiểu _Board, biến _turn kiểu bool biểu diễn hai lượt chơi, tọa độ (_x, _y) hiện hành của dấu nháy chuột, biến _command nhận phím từ người dùng, và biến _loop kiểm tra việc kết thúc trò chơi.

Dòng	
1	`class _Game{`
2	`    _Board* _b;// một bàn cờ`
3	`    bool _turn;// lượt chơi: true lượt người một & false lượt người thứ hai`
4	`    int _x, _y; // Tọa độ hiện hành của nháy chuột`
5	`    int _command; // phím gõ từ người dùng`
6	`    bool _loop; // Biến quyết định thoát game hay không`
7	`    public:`
8	`        _Game(int, int, int);`
9	`        ~_Game();`
10	`};`
11	`Game::_Game(int pSize, int pLeft, int pTop){`
12	`    _b = new _Board(pSize, pLeft, pTop);`
13	`    _loop = _turn = true;`
14	`    _command = -1; // Gán lượt và phím mặc định`
15	`    _x = pLeft; _y = pTop;`
16	`}`

17	`_Game::~_Game(){delete _b;}`

Hàm dựng sẽ lần lượt khởi tạo các biến đơn giản cũng như gọi hàm dựng của đối tượng thuộc lớp _Board để xin cấp phát vùng nhớ cho ma trận. Hàm hủy sẽ thực hiện hủy đối tượng _b (vì ta khai báo con trỏ nên phải gọi tường minh toán tử delete).

Bước 8: Ta sẽ lần lượt bổ sung các hàm cung cấp thông tin về phím người dùng vừa nhập (biến _command) hay kiểm tra xem người dùng có thoát chương trình hay chưa (biến _loop).

Dòng	
1	`class _Game{`
2	`    // …`
3	`    public:`
4	`        _Game(int, int, int);`
5	`        ~_Game();`
6	`        int getCommand();`
7	`        bool isContinue();`
8	`        char waitKeyBoard(); // Hàm nhận phím từ người dùng`
9	`        char askContinue();`
10	`};`
11	`int _Game::getCommand(){ return _command;}`
12	`bool _Game::isContinue(){ return _loop;}`
13	`char _Game::waitKeyBoard(){`
14	`  _command = toupper(getch());`
15	`  return _command;`
16	`}`
17	`char _Game::askContinue(){`
18	`_Common::gotoXY(0, _b->getYAt(_b->getSize() - 1, _b->getSize() - 1) + 4);`
19	`  return waitKeyBoard();`
20	`}`

Ngoài hai hàm getCommand và isContinue, ta cài đặt them hai hàm tiện ích là waitKeyBoard để nhận phím khi người dùng gõ vào và askContinue để hỏi người dùng có muốn thoát chương trình khi ván cờ kết thúc hoặc khi người dùng nhấn phím ‘esc’. Lưu ý: thực ra hàm askContinue và waitKeyBoard khác nhau ở chỗ trong hàm askContinue ta cần di chuyển tới vị trí thích hợp để người dùng gõ phím, còn hàm waitKeyBoard thì người dùng gõ phím trong phạm vi bàn cờ.

Bước 10: Ta cần xây dựng hàm startGame và exitGame để bắt đầu và kết thúc trò chơi

Dòng	
1	`class _Game{`

```
2     // …
3     public:
4         //…
5         void startGame(); // Hàm bắt đầu game
6         void exitGame(); // Hàm thoát game
7  };
8  void _Game::startGame() {
9    system("cls");
10   _b->resetData(); // Khởi tạo dữ liệu gốc
11   _b->drawBoard(); // Vẽ màn hình game
12   _x = _b->getXAt(0, 0);
13    _y = _b->getYAt(0, 0);
14 }
15 void _Game::exitGame() {
16   system("cls");
17   //Có thể lưu game trước khi exit
18   _loop = false;
19 }
```

Bước 11: Trong trò caro, ta cần xử lý khi người dùng nhấn phím ‘enter’ tại vị trí của bàn cờ, sau đó ta phải kiểm tra kết quả nước cờ đó có dẫn tới thắng/thua/hòa/không có gì hay không. Ta cũng cần xử lý việc di chuyển lên/xuống/trái/phải khi người dùng nhấn phím di chuyển.

```
Dòng
1  class _Game{
2     // …
3     public:
4         //…
5         int processFinish();
7         bool processCheckBoard();
8         void moveRight();
9         void moveLeft();
10        void moveUp();
11        void moveDown();
12 };
14 bool _Game::processCheckBoard() {
15   switch(_b->checkBoard(_x, _y, _turn)){
16     case -1:
17       printf("X");
18       break;
19     case 1:
```

```
20        printf("O");
22        break;
23      case 0: return false; // Khi đánh vào ô đã đánh rồi
24    }
25    return true;
26  }
27  int _Game::processFinish() {
29    // Nhảy tới vị trí thích hợp để in chuỗi thắng/thua/hòa
30    _Common::gotoXY(0, _b->getYAt(_b->getSize() - 1, _b->getSize() - 1) + 2);
31    int pWhoWin = _b->testBoard();
32    switch(pWhoWin){
33      case -1:
34        printf("Nguoi choi %d da thang va nguoi choi %d da thua\n", true, false);
35        break;
36      case 1:
37        printf("Nguoi choi %d da thang va nguoi choi %d da thua\n", false, true);
38        break;
39      case 0:
40        printf("Nguoi choi %d da hoa nguoi choi %d\n", false, true);
41        break;
42      case 2:
43        _turn = !_turn; // Đổi lượt nếu không có gì xảy ra
44    }
45    _Common::gotoXY(_x, _y);// Trả về vị trí hiện hành của con trỏ màn hình bàn cờ
46    return pWhoWin;
47  }
48  void _Game::moveRight() {
49    if (_x < _b->getXAt(_b->getSize() - 1, _b->getSize() - 1)){
50      _x += 4;
51      _Common::gotoXY(_x, _y);
52    }
53  }
54  void _Game::moveLeft() {
55    if (_x > _b->getXAt(0, 0)) {
56      _x -= 4;
57      _Common::gotoXY(_x, _y);
58    }
59  }
60  void _Game::moveDown() {
61    if (_y < _b->getYAt(_b->getSize() - 1, _b->getSize() - 1)){
62      _y += 2;
```

```
63      _Common::gotoXY(_x, _y);
64    }
65  }
66  void _Game::moveUp() {
67    if (_y > _b->getYAt(0, 0)) {
68      _y -= 2;
69      _Common::gotoXY(_x, _y);
70    }
71  }
```

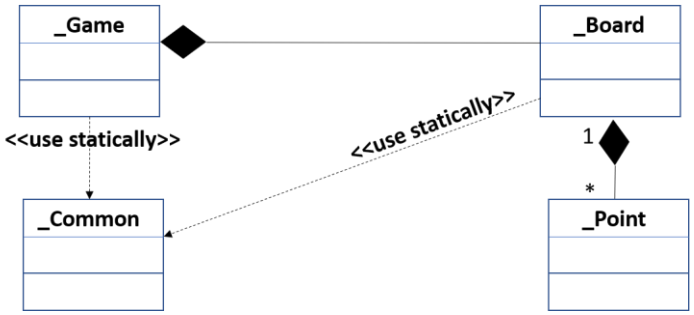
Trong quá trình di chuyển trên màn hình bàn cờ, nếu vượt quá phạm vi thì ta không xử lý (lưu ý: ta truy xuất các vị trí thông qua ma trận bàn cờ của đối tượng _b), ngược lại ta di chuyển dấu nháy chuột tới vị trí mới.

Bước 12: Cuối cùng ta xây dựng hàm main để thực hiện tiếp nhận phím từ người dùng.

```
Dòng
1   void main(){
2     _Common::fixConsoleWindow();
3     _Game g(BOARD_SIZE, LEFT, TOP); // Sinh viên tự định nghĩa các hằng số
4     g.startGame();
5     while(g.isContinue()){
6       g.waitKeyBoard();
7       if (g.getCommand() == 27) g.exitGame();
8       else {
9         switch(g.getCommand()){
10          case 'A':
11            g.moveLeft();
12            break;
13          case 'W':
14            g.moveUp();
15            break;
16          case 'S':
17            g.moveDown();
18            break;
19          case 'D':
20            g.moveRight();
21            break;
22          case 13:
23            //Đánh dấu bàn cờ, sau đó kiểm tra và xử lý thắng/thua/hòa/tiếp tục
24            if(g.processCheckBoard()){
25              switch(g.processFinish()){
26                case -1: case 1: case 0:
```

```
27                  if(g.askContinue() != 'Y') g.exitGame();
28                  else g.startGame();
29              }
30          }
31      }
32  }
33 }
34 }
```

Trong hàm main, bước đầu tiên là ta cố định màn hình ngăn ngừa người chơi thay đổi kích thước sẽ gây vỡ giao diện chương trình. Sau đó ta khởi tạo đối tượng _Game và gọi startGame() để thực hiện chuẩn bị dữ liệu cho màn chơi. Hàm main() **liên tục** chờ phím từ người chơi, sau đó tùy vào phím người dùng chọn chương trình sẽ có phản ứng thích hợp. Tất cả đều thông qua đối tượng _Game điều hành.



Hình 1: Sơ đồ các lớp trong trò chơi

4 YÊU CẦU ĐỒ ÁN

Trong phần hướng dẫn trên ta còn thiếu một vài chức năng cơ bản

4.1 Xử lý lưu/tải trò chơi (save/load) - 2đ

Trong hướng dẫn chưa xử lý việc lưu/tải trò chơi. Ta cần cài đặt thêm tính năng này. Khi người dùng nhấn phím 'L' thì sẽ hiện dòng chữ yêu cầu người dùng nhập tên tập tin muốn lưu trạng thái hiện hành của trò chơi. Khi người dùng nhấn phím 'T' thì sẽ hiện dòng chữ yêu cầu người dùng nhập tên tập tin muốn tải lại.

4.2 Nhận biết thắng/thua/hòa - 2đ

Ta cần bổ sung tính năng kiểm tra quy luật thắng/thua /hòa trong caro. Từ đó sẽ kiểm tra sau mỗi bước đi của người chơi

4.3 Xử lý hiệu ứng thắng/thua/hòa - 2đ

Trong hướng dẫn, khi thắng/thua/hòa chỉ hiển thị dòng chữ báo hiệu đơn giản. Ta hãy cài đặt hiệu ứng giúp sinh động hơn

4.4 Xử lý giao diện màn hình khi chơi – 1.5đ

Trong quá trình chơi, cho hiển thị các thông số của hai người chơi, ví dụ người chơi thứ nhất đã đánh bao nhiêu bước, người chơi thứ hai đã thua mấy ván… Sinh viên tự tổ chức giao diện màn hình sao cho rõ rang, sinh động.

4.5 Xử lý màn hình chính – 1.5đ

Trước khi vào trò chơi, hiển thị danh sách menu, ví dụ như “New Game”, “Load Game”, “Settings”, … Như vậy sẽ giúp chương trình caro hoàn thiện và giống thực tế một trò chơi hơn.

4.6 Xử lý chơi với máy (giải thuật cắt tỉa alpha – beta) – 1đ

Cho người chơi chọn chế độ chơi với máy thay vì chơi hai người với nhau, có thể cho người dùng lựa chọn cấp độ, nếu dễ thì máy sẽ random vị trí, còn cấp độ khó sẽ áp dụng giải thuật alpha – beta.