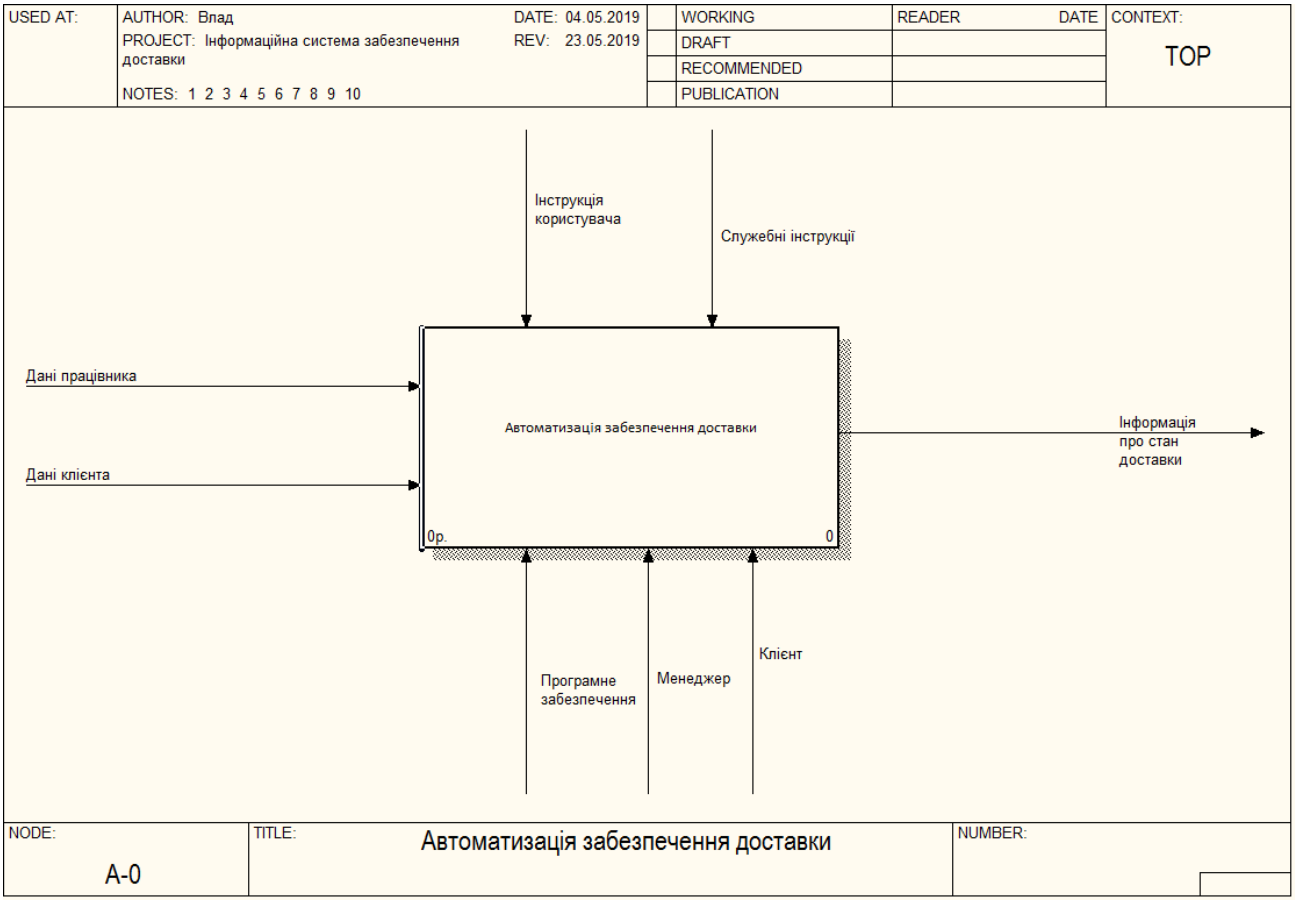
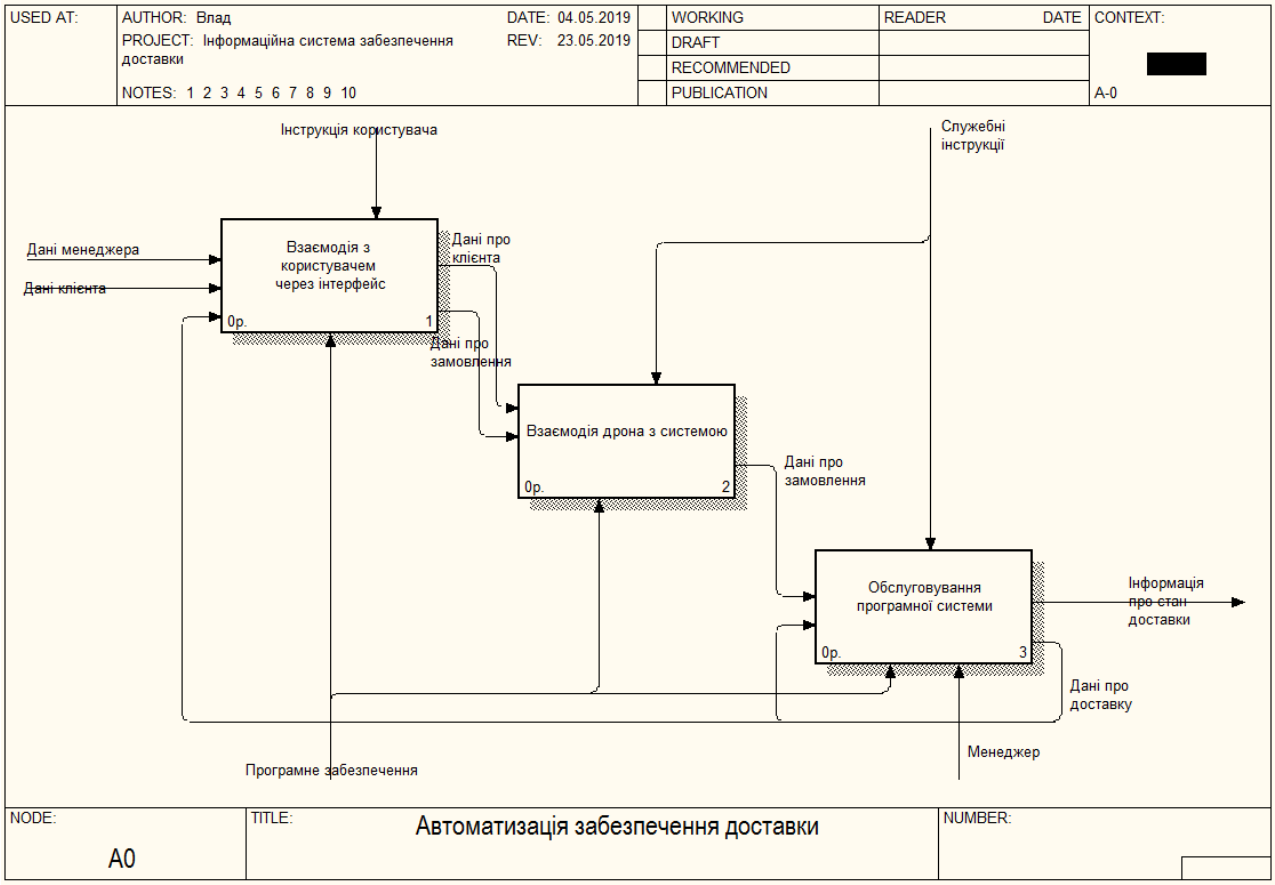
ДОДАТОК А
Графічні матеріали

ГЮИК.502528.006



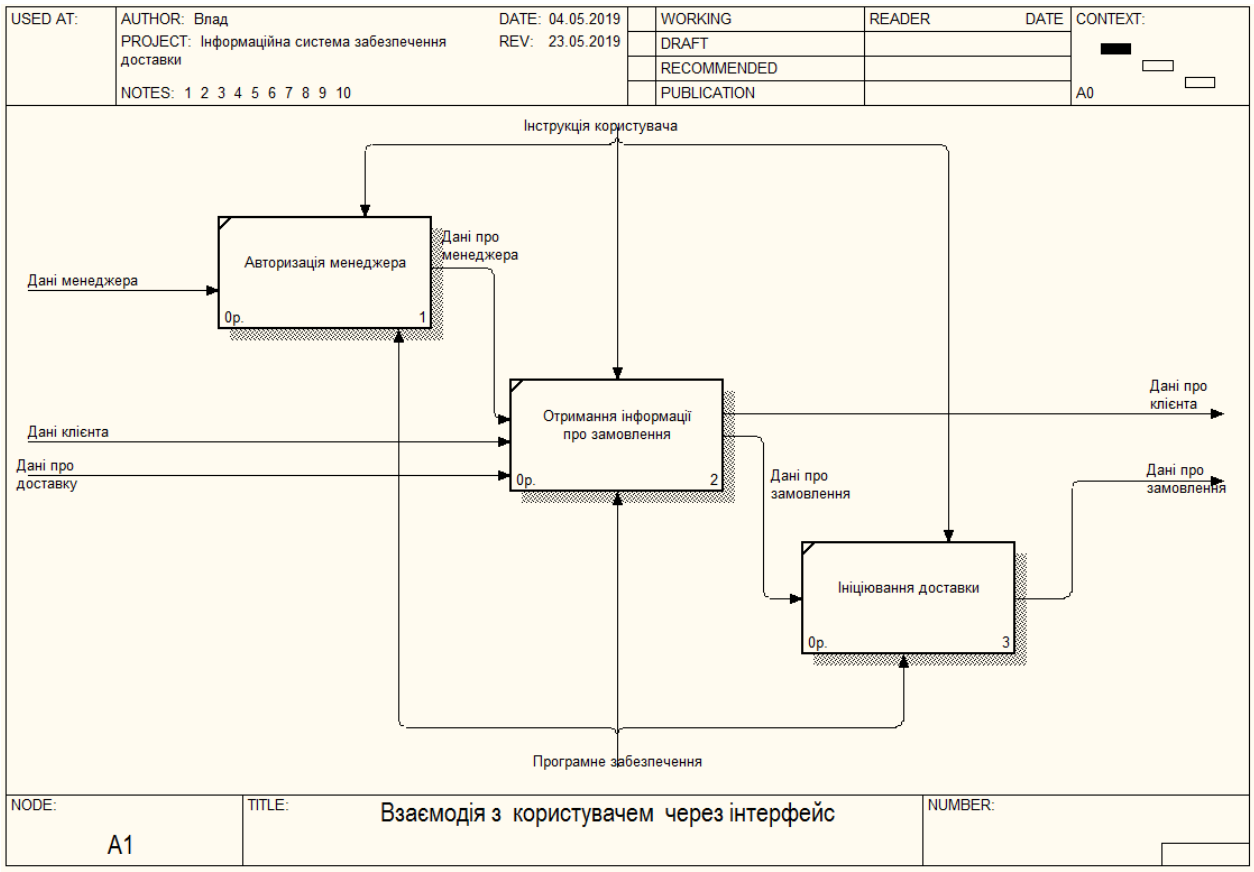
					ГЮИК.502528.006			
					Концептуальна діаграма компоненту системи	Літ.	Маса	Масштаб
Змін.	Лист	№ докум.	Підпис	Дата				
Розроб.		Костюкевич В.В.				Лист 1	Листів 10	
Перевір.		Решетнік В.М..						
Т. Контр.						ХНУРЕ Кафедра СТ		
Реценз.								
Н. Контр.								
Затверд.		Гребеннік І.В.						

ГЮИК.502528.006



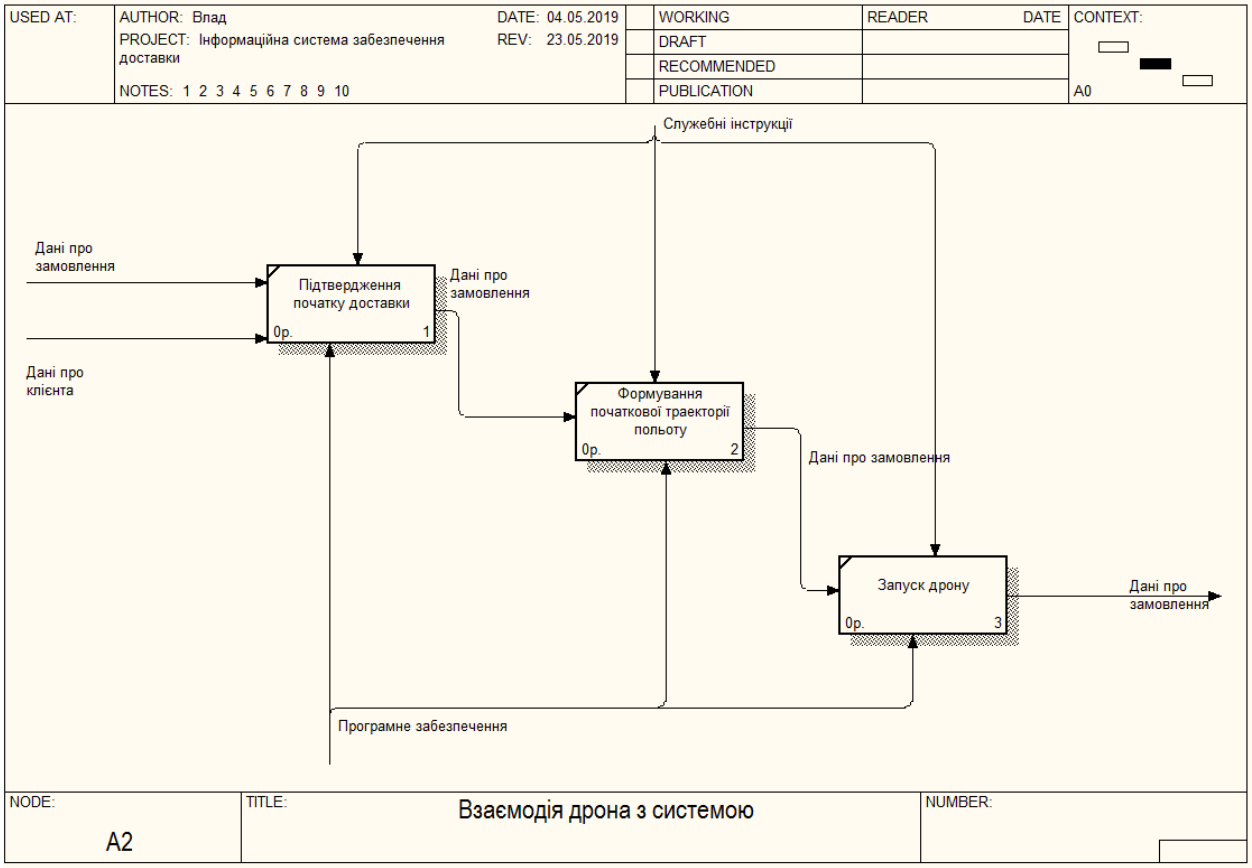
					ГЮИК.502528.006			
					Декомпозиція діаграми компоненту системи	Літ.	Маса	Масштаб
Змін.	Лист	№ докум.	Підпис	Дата				
Розроб.		Костюкевич В.В.				Лист 2	Листів 10	
Перевір.		Решетнік В.М.						
Т. Контр.						ХНУРЕ Кафедра СТ		
Реценз.								
Н. Контр.								
Затверд.		Гребеннік І.В.						

ГЮИК.502528.006



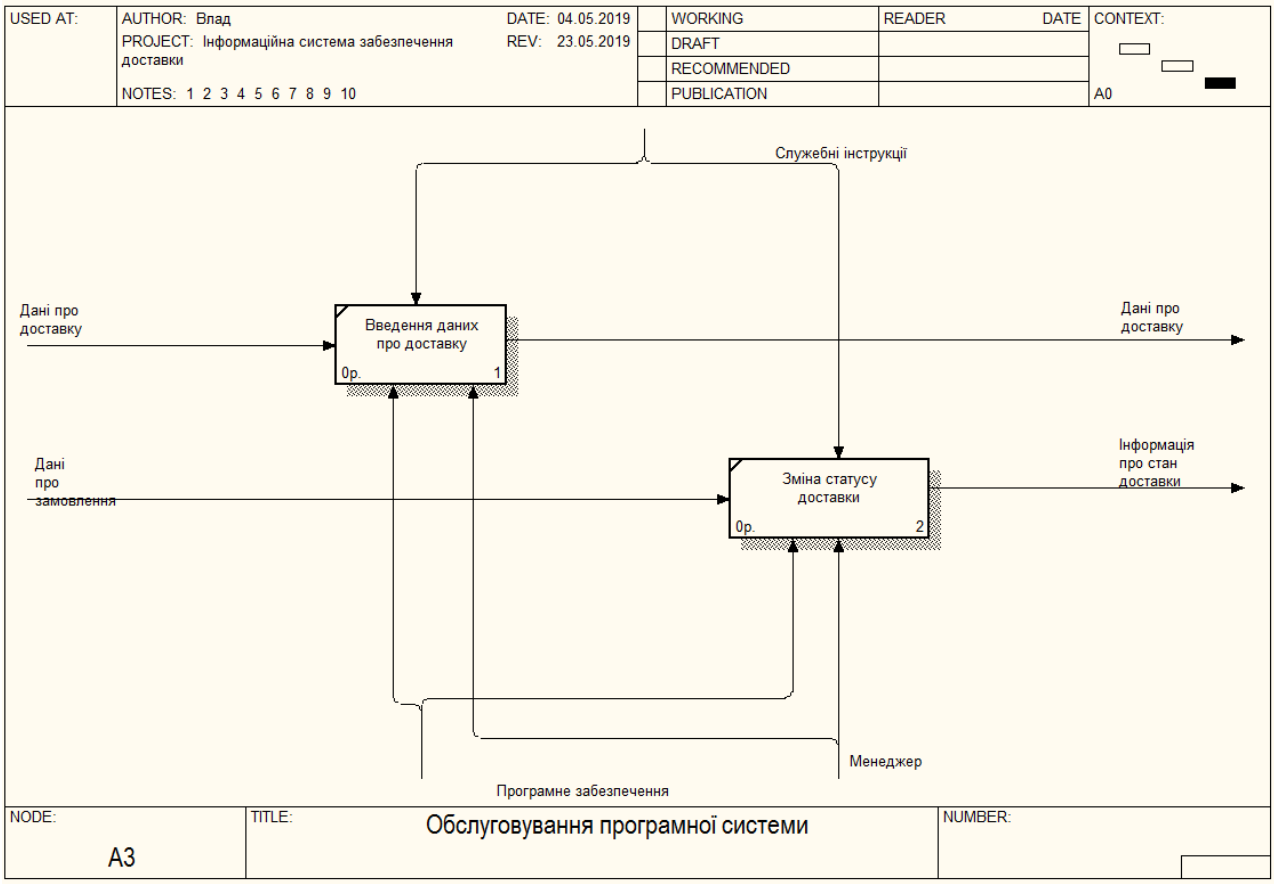
					ГЮИК.502528.006			
Змін.	Лист	№ докум.	Підпис	Дата	Декомпозиція функції взаємодії користувача через інтерфейс	Літ.	Маса	Масштаб
Розроб.		Костюкевич В.В.						
Перевір.		Решетнік В.М.				Лист 3		Листів 10
Т. Контр.								
Реценз.						ХНУРЕ Кафедра СТ		
Н. Контр.								
Затверд.		Гребеннік І.В.						

ГЮИК.502528.006



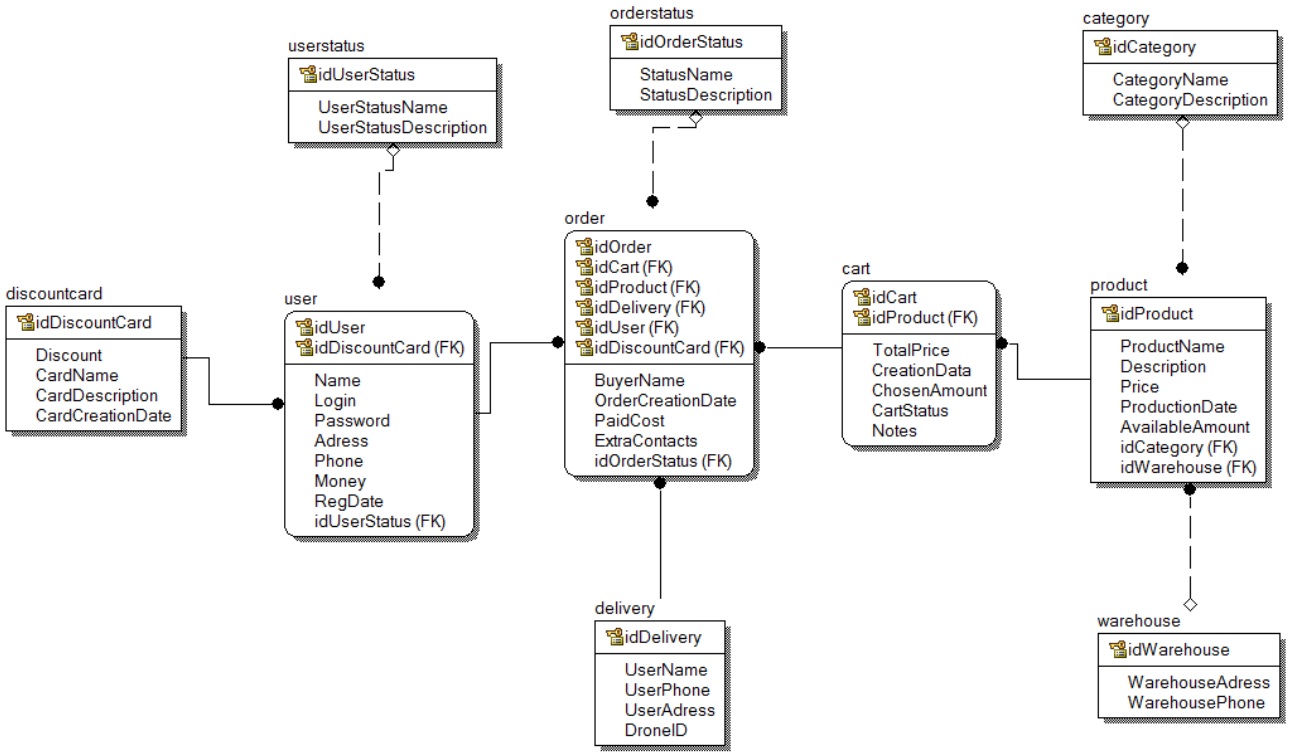
					ГЮИК.502528.006			
					Декомпозиція функції взаємодії безпілотного літального апарату з системою	Літ.	Маса	Масштаб
Змін.	Лист	№ докум.	Підпис	Дата				
Розроб.		Костюкевич В.В.				Лист 4	Листів 10	
Перевір.		Решетнік В.М.						
Т. Контр.								
Реценз.						ХНУРЕ Кафедра СТ		
Н. Контр.								
Затверд.		Гребеннік І.В.						

ГЮИК.502528.006



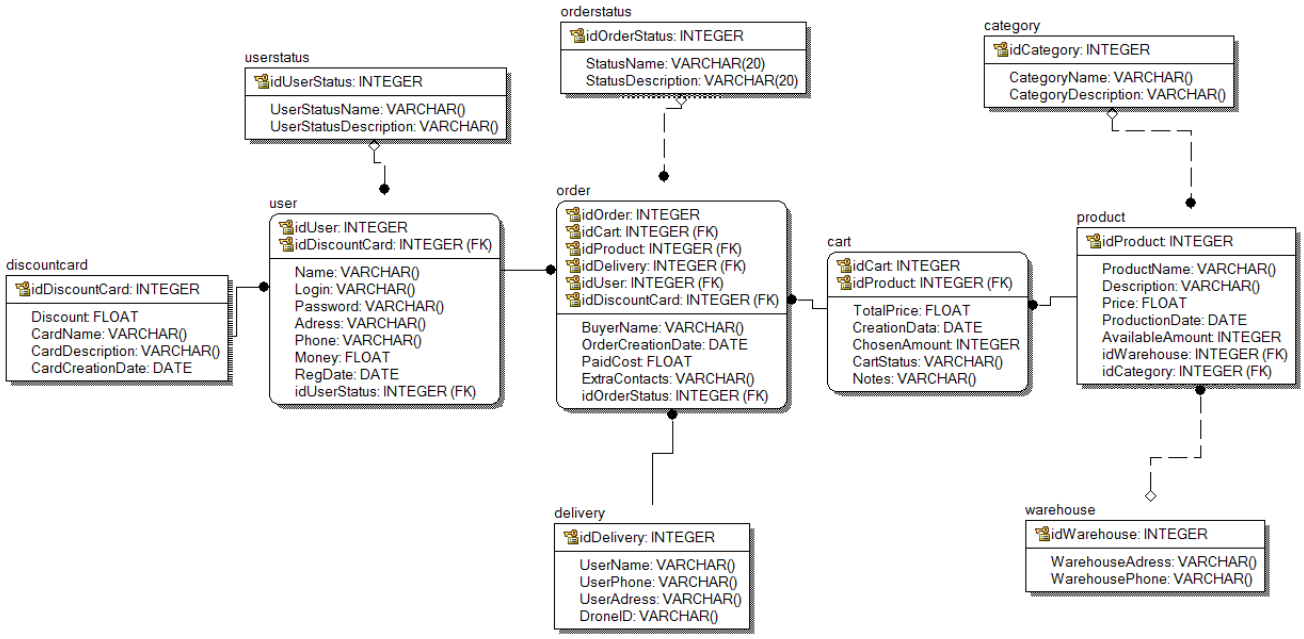
					ГЮИК.502528.006			
Змін.	Лист	№ докум.	Підпис	Дата				
Розроб.		Костюкевич В.В.			Декомпозиція функції обслуговування програмної системи	Літ.	Маса	Масштаб
Перевір.		Решетнік В.М.						
Т. Контр.						Лист5	Листів 10	
Реценз.						ХНУРЕ Кафедра СТ		
Н. Контр.								
Затверд.		Гребеннік І.В.						

ГЮИК.502528.006



					ГЮИК.502528.006			
Змін.	Лист	№ докум.	Підпис	Дата	Логічна схема бази даних	Літ.	Маса	Масштаб
Розроб.		Костюкевич В.В.						
Перевір.		Решетнік В.М.				Лист 6		Листів 10
Т. Контр.								
Реценз.						ХНУРЕ Кафедра СТ		
Н. Контр.								
Затверд.		Гребеннік І.В.						

ГЮИК.502528.006



					ГЮИК.502528.006	
					Фізична схема бази даних	Літ. / Маса / Масштаб
Змін.	Лист	№ докум.	Підпис	Дата		
Розроб.		Костюкевич В.В.				Лист 7 / Листів 10
Перевір.		Решетнік В.М.				
Т. Контр.						ХНУРЕ Кафедра СТ
Реценз.						
Н. Контр.						
Затверд.		Гребеннік І.В.				

ДОДАТОК Б
Текст програми

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

«ЗАТВЕРДЖУЮ»
Керівник атестаційної роботи
_____ доц. Решетнік В.М.
(підпис)

ДОСЛІДЖЕННЯ МЕТОДІВ ПОШУКУ ОПТИМАЛЬНИХ ШЛЯХІВ РУХУ В ТРЬОХВИМІРНОМУ ПРОСТОРІ ДЛЯ ІНФОРМАЦІЙНИХ СИСТЕМ ОРГАНІЗАЦІЇ ПОВІТРЯНОЇ ДОСТАВКИ КОРИСНИХ ВАНТАЖІВ

Текст програми

ЛИСТЗАТВЕРДЖЕННЯ
ГЮИК.502528.006 01 12 01 – ЛЗ

РОЗРОБИВ
ст. гр. СПРм-19-1
Костюкевич В.В.

2020

ЗАТВЕРДЖЕНО
ГЮИК.506120.07 34 01 - 01 12 01

ДОСЛІДЖЕННЯ МЕТОДІВ ПОШУКУ ОПТИМАЛЬНИХ ШЛЯХІВ РУХУ В ТРЬОХВИМІРНОМУ ПРОСТОРІ ДЛЯ ІНФОРМАЦІЙНИХ СИСТЕМ ОРГАНІЗАЦІЇ ПОВІТРЯНОЇ ДОСТАВКИ КОРИСНИХ ВАНТАЖІВ

Текст программы

ГЮИК.502528.006 – 01 12 01
Листів 14

2020

A_D_Star.cpp

```
#include <Windows.h>
#include <ctime>
#include <iostream>
#include <math.h>
#include <algorithm>
#include <vector>
#include <list>
using namespace std;
//static const int width = 10, height = 10, depth = 10;
//static const int width = 25, height = 25, depth = 25;
//static const int width = 50, height = 50, depth = 50;
static const int width = 100, height = 100, depth = 100;
float delta = 1;
struct Point
{     float pointX;
      float pointY;
      float pointZ; };
struct CubeData
{     float cost, diagCost;};
class Cube
{public:
      Point center;
      CubeData data;
      float arcCost, pathCost;
      bool isOpen;
      Cube *previous;
      Cube()
      {           isOpen = false; }
      ~Cube()
      {     previous = NULL;
            delete previous;}
      bool operator < (const Cube& rCube) const
      {     return pathCost < rCube.pathCost;     }
      Point getCenter() const
      {     return center;  }
      friend bool operator == (const Cube& lCube, const Cube& rCube)
      {     if  ((lCube.getCenter().pointX  ==  rCube.getCenter().pointX)  &&
(lCube.getCenter().pointY       ==       rCube.getCenter().pointY)       &&
(lCube.getCenter().pointZ == rCube.getCenter().pointZ))
            {     return true;          }
            else
            {     return false;         }     }};
Cube ***cube;
vector<Cube*> getNeighbours(int i, int j, int k)
{     vector<Cube*> neighbours;
      for (int x = -1; x <= 1; x++)
      {     for (int y = -1; y <= 1; y++)
            {     for (int z = -1; z <= 1; z++)
                  {     if (((i + x)>-1) && ((j + y) > -1) && ((k + z) > -1)
```

```
&& ((i + x)<width) && ((j + y) < height) && ((k + z) < depth))
                    {    if    ((cube[i    +    x][j    +    y][k    +
z].getCenter().pointX  !=  cube[i][j][k].getCenter().pointX)  ||  (cube[i  +
x][j + y][k + z].getCenter().pointY != cube[i][j][k].getCenter().pointY) ||
(cube[i     +     x][j     +     y][k     +     z].getCenter().pointZ     !=
cube[i][j][k].getCenter().pointZ))
                         {    if   ((abs(cube[i   +   x][j   +   y][k   +
z].getCenter().pointX  -  cube[i][j][k].getCenter().pointX)  <=  delta)  &&
(abs(cube[i    +    x][j    +    y][k    +    z].getCenter().pointY    -
cube[i][j][k].getCenter().pointY) <= delta) && (abs(cube[i + x][j + y][k +
z].getCenter().pointZ - cube[i][j][k].getCenter().pointZ) <= delta))
                              {    neighbours.push_back(&cube[i + x][j +
y][k + z]);}}}}}}
     return neighbours;}
struct CubeFind
{    CubeFind(Cube* rCube) : c(rCube) {}
     Cube *c;
     bool operator()(Cube* lCube) const
     {    return lCube == c;    }};
class CubeBuilder
{    int cWidth, cHeight, cDepth;
public:
     float cDelta;
     CubeBuilder(int width, int height, int depth, float delta)
     {    cDelta = delta;
          cWidth = width;
          cHeight = height;
          cDepth = depth; }
     Cube createCube(int i, int j, int k, int cost)
     {    Cube result;
          float z = i;
          float y = j;
          float x = k;
          result.center = { x, y, z };
          if (cost > 6)
          {    cost = 1;  }
          else
          {    cost = 10; }
          result.data.cost = cost;
          result.data.diagCost = cost + 0.414;
          return result;  }};
float cDistance(Cube *start, Cube *finish)
{    float dist;
     dist = powf((start->getCenter().pointX - finish->getCenter().pointX),
2) + powf((start->getCenter().pointY - finish->getCenter().pointY), 2) +
powf((start->getCenter().pointZ - finish->getCenter().pointZ), 2);
     dist = sqrtf(dist);
     return dist;}
void setPath(Cube *current, list<Cube*> *closed)
{    vector<Cube*>  neighb  =  getNeighbours(current->getCenter().pointX,
current->getCenter().pointY, current->getCenter().pointZ);
```

```
     for (int i = 0; i < neighb.size(); i++)
     {    if (neighb[i]->data.diagCost < 9)
          {    if (cDistance(current, neighb[i]) > delta)
               {    neighb[i]->arcCost = current->data.cost + neighb[i]-
>data.diagCost; }
               else
               {    neighb[i]->arcCost = current->data.cost + neighb[i]-
>data.cost;}
               if        ((find_if(closed->begin(),         closed->end(),
CubeFind(neighb[i])) != closed->end()) == false)
               {if     ((neighb[i]->getCenter().pointX     !=     current-
>getCenter().pointX)   ||   (neighb[i]->getCenter().pointY   !=   current-
>getCenter().pointY)   ||   (neighb[i]->getCenter().pointZ   !=   current-
>getCenter().pointZ))
                    {    neighb[i]->previous = current;  }}}}}
vector<Cube*> make_path(Cube *start, Cube *cube)
{    vector<Cube*> path;
     int v=0;
     path.push_back(cube);
     while  ((cube->previous)  &&  (cube  !=  start)  &&  (cube  !=  cube-
>previous->previous))
     {    cube = cube->previous;
          path.push_back(cube);
          v++;
          printf("%i)  %.1f,  %.1f,  %.1f\n",  v,  cube->getCenter().pointX,
cube->getCenter().pointY, cube->getCenter().pointZ);  }
     return path;}
vector<Cube*> aStar(Cube *start, Cube *finish)
{    int n=0;
     bool isFirst = true;
     vector<Cube*> neighb;
     list<Cube*> open;
     open.push_back(finish);
     start->isOpen = true;
     list<Cube*> closed;
     start->data.cost = 0;
     start->data.diagCost = 0;
     finish->data.cost = 0;
     finish->data.diagCost = 0;
     start->pathCost = start->data.cost + cDistance(start, finish);
     Cube *current, *next;
     while (!open.empty())
     {    open.sort();
          current = open.front();
          n++;
          system("CLS");
          printf("%i\n", n);
          if  ((find_if(closed.begin(),  closed.end(),  CubeFind(start))  !=
closed.end()) == true)
          {    open.clear();
               closed.clear();
```

```
                return make_path(finish, current);    }
          open.pop_front();
          current->isOpen = false;
          setPath(current, &closed);
          closed.push_back(current);
          list<Cube*> openNeighb;
          neighb   =   getNeighbours(current->getCenter().pointX,   current-
>getCenter().pointY, current->getCenter().pointZ);
          for (int i = 0; i < neighb.size(); i++)
          {    if          ((find_if(closed.begin(),          closed.end(),
CubeFind(neighb[i])) != closed.end()) == false)
               {    if (neighb[i]->data.diagCost < 9)
                    {    openNeighb.push_back(neighb[i]);
                         neighb[i]->pathCost   =   neighb[i]->arcCost   +
cDistance(neighb[i], finish);
                         if      ((find_if(open.begin(),      open.end(),
CubeFind(neighb[i])) != open.end()) == false)
                         {    neighb[i]->isOpen = true;
                              open.push_back(neighb[i]);      }
                         else
                         {    openNeighb.sort();
                              next = openNeighb.front();
                              if     (neighb[i]->data.cost     <     next-
>data.cost)
                              {    next = neighb[i];
                                   next->previous = current;  }}}
                    else
                    {    neighb[i]->isOpen = false;
                         closed.push_back(neighb[i]);}}}
          openNeighb.clear();   }
     open.clear();
     closed.clear();
     printf("no path\n");
     return make_path(start, start);}
Cube*  make_path_dStar(Cube  *start,  Cube  *finish,  list<Cube*>  *open,
list<Cube*> *closed)
{    Cube *current, *next;
     vector<Cube*> neighb;
     open->sort();
     if (open->empty())
     {    printf("no path\n");
          return start;   }
     current = open->front();
     open->pop_front();
     current->isOpen = false;
     setPath(current, closed);
     closed->push_back(current);
     list<Cube*> openNeighb;
     neighb    =    getNeighbours(current->getCenter().pointX,    current-
>getCenter().pointY, current->getCenter().pointZ);
     for (int i = 0; i < neighb.size(); i++)
```

```
     {    if           ((find_if(closed->begin(),           closed->end(),
CubeFind(neighb[i])) != closed->end()) == false)
          {    if (neighb[i]->data.diagCost < 9)
               {    openNeighb.push_back(neighb[i]);
                    neighb[i]->pathCost     =     neighb[i]->arcCost     +
cDistance(neighb[i], finish);
                    if        ((find_if(open->begin(),        open->end(),
CubeFind(neighb[i])) != open->end()) == false)
                    {    neighb[i]->isOpen = true;
                         open->push_back(neighb[i]);     }
                    else
                    {    openNeighb.sort();
                         next = openNeighb.front();
                         if (neighb[i]->data.cost < next->data.cost)
                         {    next = neighb[i];
                              next->previous = current;} }}
               else
               {    neighb[i]->isOpen = false;
                    closed->push_back(neighb[i]);   }}}
     openNeighb.clear();
     if (open->empty())
     {    printf("no path\n");
          return start;   }
     open->sort();
     current = open->front();
     return current;}
Cube* changeCost(Cube *move, Cube *neighb, float newCost, list<Cube*> open,
list<Cube*> closed)
{    neighb->data.cost = newCost;
     if   ((find_if(closed.begin(),   closed.end(),   CubeFind(neighb))   !=
closed.end() == true))
     {    open.push_back(move); }
     open.sort();
     return open.front();}
int newCost()
{    if ((rand() % 10 + 1) <= 3)
     {    return rand() % 19 - 9;    }
     else
     {    return 0;  }}
Cube* dStar(Cube *start, Cube *finish)
{    int n=0;
     Cube *move, *min;
     vector<Cube*> neighb;
     list<Cube*> open;
     open.push_back(finish);
     finish->isOpen = true;
     list<Cube*> closed;
     start->data.cost = 0;
     start->data.diagCost = 0;
     finish->data.cost = 0;
     finish->data.diagCost = 0;
```

```
     while (((find_if(closed.begin(), closed.end(), CubeFind(start)) !=
closed.end()) == false) || (open.empty()))
     {    min = make_path_dStar(start, finish, &open, &closed);
          n++;
          system("CLS");
          printf("%i\n", n);    }
     move = start;
     int v = 0;
     while (move != finish)
     {    neighb.clear();
          neighb    =    getNeighbours(move->getCenter().pointX,    move-
>getCenter().pointY, move->getCenter().pointZ);
          n++;
          printf("%i   ", n);
          for (int i = 0; i < neighb.size(); i++)
          {    int x = 0;
               if ((neighb[i]->data.cost + (x = newCost())) != neighb[i]-
>data.cost)
               {    min    =    changeCost(move,    neighb[i],    neighb[i]-
>data.cost + x, open, closed);  }    }
          while ((min->data.cost < move->data.cost) && (!open.empty()))
          {    min = make_path_dStar(min, finish, &open, &closed);
               n++;
               printf("\r%i   ", n); }
          if (open.empty())
          {    open.clear();
               closed.clear();
               printf("no path\n");
               return start;   }
          move = move->previous;
          v++;
          printf("%i)  %.1f,  %.1f,  %.1f\n",  v,  move->getCenter().pointX,
move->getCenter().pointY, move->getCenter().pointZ);  }
     open.clear();
     closed.clear();
     return move;}
void main()
{    CubeBuilder cubes(width, height, depth, delta);
     cube = new Cube**[width];
     for (int i = 0; i < width; i++)
     {    cube[i] = new Cube*[height];
          for (int j = 0; j < height; j++)
          {    cube[i][j] = new Cube[depth];    }    }
     vector<Point> cNeighbours;
     srand(time(0));
     for (int i = 0; i<width; i++)
     {    for (int j = 0; j<height; j++)
          {    for (int k = 0; k<depth; k++)
               {    cube[i][j][k]    =    cubes.createCube(i,    j,    k,
rand()%10+1);}} }
     printf("gen done\n");
```

```
	clock_t st = clock();
	float et;
	//aStar(&cube[0][0][0], &cube[9][9][9]);
	//aStar(&cube[0][0][0], &cube[24][24][24]);
	//aStar(&cube[0][0][0], &cube[49][49][49]);
	//aStar(&cube[0][0][0], &cube[99][99][99]);
	//dStar(&cube[0][0][0], &cube[9][9][9]);
	//dStar(&cube[0][0][0], &cube[24][24][24]);
	//dStar(&cube[0][0][0], &cube[49][49][49]);
	dStar(&cube[0][0][0], &cube[99][99][99]);
	printf("done\n");
	et = (clock() - st) / CLOCKS_PER_SEC + 0.001 * ((clock() - st) %
CLOCKS_PER_SEC);
	printf("%f\n", et);
	for (int i = 0; i < width; i++)
	{	for (int j = 0; j < height; j++)
		{	delete[] cube[i][j];  }
		delete[] cube[i];}
	delete[] cube;
	system("pause");}

Cube.h

#pragma once
struct Coords
{	float x;
	float y;
	float z;
	Coords(){};
	Coords(float x, float y, float z);};
const float INF_COST = 10000000;
class Cube
{public:
	Cube()
	{	diagCost = 0;
		pathCost = 0;
		arcCost = 0;
		previous = 0;};
	Cube(float x, float y, float z, float cost);
	Cube(const Coords& coords, float cost);
	~Cube();
	void setCoords(const Coords& new_coords) { coords = new_coords; };
	void setCost(const float& new_cost)
	{	if (new_cost == 0)
		{	cost = 10; }
		else
		{	cost = 1;  }};
	Coords getCoords() const { return coords; };
	float getCost() const
	{ if (cost == 1)
		{	return 1;  }
```

```
            else
            {     return 0;  }};
      void Draw(/*Some handle to draw*/);
private:
      Coords coords;
      float cost, diagCost, pathCost, arcCost;
      Cube *previous;};

Grid.h

#pragma once
#include "Cube.h"
typedef unsigned int uint;
class Grid
{public:
      Grid(int width, int height, int depth);
      ~Grid();
      int getWidth() const { return width; };
      int getHeight() const { return height; };
      int getDepth() const { return depth; };
      float   Available(int   i,   int   j,   int   k)   const   {   return
vertices[i][j][k].getCost(); };
      void  setAvailable(int  i,  int  j,  int  k,  const  bool&  available)  {
vertices[i][j][k].setCost(available); };
      void SaveToFile(const char *filename);
      void ReadFromFile(const char *filename);
      void Draw(/*Some handle to draw*/);
private:
      int width;
      int height;
      int depth;
      Cube ***vertices;
      void Init();
      void Release();};

Grid.cpp

#include "Grid.h"
#include <fstream>
#include <iostream>
Grid::Grid(int i_width, int i_height, int i_depth)
{     width = i_width;
      height = i_height;
      depth = i_depth;
      Init();}
void Grid::Init()
{     vertices = new Cube**[width];
      for (int i = 0; i < width; i++)
      {     vertices[i] = new Cube*[height];
            for (int j = 0; j < height; j++)
            {     vertices[i][j] = new Cube[depth];     }     }
```

```
     int cost = 0;
     for (int i = 0; i < width; i++)
     {    for (int j = 0; j < width; j++)
          {    for (int k = 0; k < width; k++)
               {    cost = rand() % 10 + 1;
                    if (cost>6)
                    {    setAvailable(i, j, k, 1);  }
                    else
                    {    setAvailable(i, j, k, 0);  }}}}}
Grid::~Grid()
{    Release();}
void Grid::Release()
{    for (int i = 0; i < width; i++)
     {    for (int j = 0; j < height; j++)
          {    delete[]vertices[i][j];    }
          delete[]vertices[i];  }
     delete[]vertices;}
void Grid::SaveToFile(const char *filename)
{    using namespace std;
     fstream file;
     file.open(filename, ios::out);
     file << width << " " << height << " " << depth << endl;
     for (int i = 0; i < width; i++)
     {    for (int j = 0; j < height; j++)
          {    for (int k = 0; k < height; k++)
               {    file << vertices[i][j][k].getCost() << " ";}
               file << endl;   }}
     file.close();}
void Grid::ReadFromFile(const char *filename)
{    Release();
     using namespace std;
     fstream file;
     file.open(filename, ios::in);
     file >> width >> height >> depth;
     Init();
     for (int i = 0; i < width; i++)
     {    for (int j = 0; j < height; j++)
          {    for (int k = 0; k < height; k++)
               {    float c;
                    file >> c;
                    vertices[i][j][k].setCost(c);   }}}
     file.close();}

Lee.h

#pragma once
#include "PathFinder.h"
class Lee : public PathFinder
{public:
     Lee(int start_x, int start_y, int start_z,
          int end_x, int end_y, int end_z);
```

```
      virtual ~Lee();
      virtual void Do(const Grid& grid);
private:
      int Wave(int n);
      int FindPath(int n);
      static const int EMPTY = -1;
      static const int WALL = -2;
      int ***field;};

Lee.cpp

#include "Lee.h"
#include <Windows.h>
#include "Grid.h"
#include <queue>
Lee::Lee(int x_start, int y_start, int z_start, int x_end, int y_end, int
z_end) :
      PathFinder(x_start, y_start, z_start, x_end, y_end, z_end)
{     title = "Lee";}
Lee::~Lee(){}
void Lee::Do(const Grid& grid)
{     int i, j, k, n=0;
      field = new int**[FIELD_W];
      for (i = 0; i < FIELD_W; i++)
      {     field[i] = new int*[FIELD_H];
            for (j = 0; j < FIELD_H; j++)
            {     field[i][j] = new int[FIELD_D]{-1};
                  for (k = 0; k < FIELD_D; k++)
                  {     field[i][j][k] = (grid.Available(i, j, k)==1) ? EMPTY
: WALL;    }}}
      n = Wave(n);
      n = FindPath(n);
      printf("%i \n", n);
      for (i = 0; i < FIELD_W; i++)
      {     for (j = 0; j < FIELD_H; j++)
            {     delete[]field[i][j];  }
            delete[]field[i];     }
      delete[]field;}
int Lee::Wave(int n)
{     std::queue<int> need_visit;
      int i, j, k;
      int x, y, z;
      int d_hw = FIELD_H * FIELD_W;
      field[start_x][start_y][start_z] = 0;
      need_visit.push(start_z * d_hw + start_y * FIELD_W + start_x);
      do
      {     x = (need_visit.front() % d_hw) % FIELD_W;
            y = (need_visit.front() % d_hw) / FIELD_W;
            z = need_visit.front() / (d_hw);
            need_visit.pop();
            for (i = -1; i <= 1; i++)
```

```
          {    if ((x + i >= 0) &&
                    (x + i < FIELD_W))
                    for (j = -1; j <= 1; j++)
                    {    if ((y + j >= 0) &&
                              (y + j < FIELD_H))
                              for (k = -1; k <= 1; k++)
                              {    if ((z + k >= 0) &&
                                        (z + k < FIELD_D))
                                   {    if (field[x + i][y + j][z + k]
== EMPTY)
                                        {    field[x + i][y + j][z + k]
= field[x][y][z] + 1;
                                             if (((i + x) == end_x) &&
                                                  ((j + y) == end_y) &&
                                                  ((k + z) == end_z))
                                                  return n;
                                             need_visit.push((z        +
k)*(d_hw) + (y + j)*FIELD_W + (x + i));    }}}}}
          n++;
     } while (!need_visit.empty());
     return n;}
int Lee::FindPath(int n)
{    if (field[end_x][end_y][end_z] > 0)
     {    int d_hw = FIELD_H * FIELD_W;
          int i, j, k;
          int x, y, z;
          int p;
          x = end_x;
          y = end_y;
          z = end_z;
          path[0] = end_z * d_hw + end_y * FIELD_W + end_x;
          p = 1;
          do
          {    for (i = -1; i <= 1; i++)
               {    if ((x + i >= 0) &&
                         (x + i < FIELD_W))
                    {    for (j = -1; j <= 1; j++)
                         {    if ((y + j >= 0) &&
                                   (y + j < FIELD_H))
                              {    for (k = -1; k <= 1; k++)
                                   {    if ((z + k >= 0) &&
                                             (z + k < FIELD_D))
                                        {    if (field[x + i][y + j][z
+ k] == (field[x][y][z] - 1))
                                                  goto FIND; }}}}}}
          FIND:
               x = x + i;
               y = y + j;
               z = z + k;
               path[p] = z*d_hw + y*FIELD_W + x;
               p++;
```

```
                Sleep(100);
                n++;
          } while ((x != start_x) || (y != start_y) || (z != start_z));
          path_result = P_WAS_BUILD; }
     else
          path_result = P_CAN_NOT_BUILD;
     return n;}
```

Main.cpp

```
#include "PathFinder.h"
#include "AStar.h"
#include "Deikstra.h"
#include "Lee.h"
#include "Grid.h"
#include <ctime>
#include <iostream>
#include <memory>
const int GRID_WIDTH = 100;
const int GRID_HEIGHT = 100;
const int GRID_DEPTH = 100;
void Test(std::unique_ptr<PathFinder> test, const Grid& grid);
void main()
{    //srand(time(0));
     Grid myGrid(GRID_WIDTH, GRID_HEIGHT, GRID_DEPTH);
     //myGrid.SaveToFile("filename100.txt");
     //myGrid.ReadFromFile("filename100.txt");
     Test(std::unique_ptr<Lee>(new  Lee(0,  0,  0,  GRID_WIDTH  -  1,
GRID_HEIGHT - 1, GRID_DEPTH - 1)), myGrid);}
void Test(std::unique_ptr<PathFinder> test, const Grid& grid)
{    test->Algorithm(grid);
     test->ShowResults();
     system("pause");}
```

Pathfinder.h

```
#pragma once
#include "MiniTimer.h"
class Grid;
class PathFinder
{public:
     PathFinder(
          int start_x, int start_y, int start_z,
          int end_x, int end_y, int end_z);
     virtual ~PathFinder();
     void SetBegin(int x, int y, int z) { start_x = x; start_y = y;
start_z = z; };
     void SetEnd(int x, int y, int z) { end_x = x; end_y = y; end_z = z;
};
     double GetResultTime() const { return timer.getTime(); };
//   int * GetResultPath() const {};
```

```
      virtual void Algorithm(const Grid& grid);
      void ShowResults();
      void ShowTitle();
      void ShowPath();
      void ShowTime();
protected:
      virtual void Do(const Grid& grid) = 0;
      int start_x;
      int start_y;
      int start_z;
      int end_x;
      int end_y;
      int end_z;
      int FIELD_W;
      int FIELD_H;
      int FIELD_D;
      int *path;
      int path_length;
      enum
      {     P_WAS_NOT_BUILD,
            P_WAS_BUILD,
            P_CAN_NOT_BUILD,
            P_EQUAL
      } path_result;
      MiniTimer timer;
      const char *title;
      void CreatePath(int length);};

Pathfinder.cpp

#include "PathFinder.h"
#include "Grid.h"
#include <stdio.h>
PathFinder::PathFinder(
      int x_start, int y_start, int z_start,
      int x_end, int y_end, int z_end) :
      start_x(x_start), start_y(y_start), start_z(z_start),
      end_x(x_end), end_y(y_end), end_z(z_end),
      FIELD_W(0), FIELD_H(0), FIELD_D(0)
{     path = nullptr;
      path_result = P_WAS_NOT_BUILD;}
PathFinder::~PathFinder()
{     delete[]path;}
void PathFinder::CreatePath(int length)
{     delete[]path;
      path_length = length;
      path = new int[path_length];
      for (int i = 0; i < path_length; i++)
            path[i] = -1;}
void PathFinder::Algorithm(const Grid& grid)
{     FIELD_W = grid.getWidth();
```

```
      FIELD_H = grid.getHeight();
      FIELD_D = grid.getDepth();
      if ((start_x == end_x) &&
            (start_y == end_y) &&
            (start_z == end_z))
      {     path_result = P_EQUAL;     }
      else
      {     timer.Start();
            CreatePath(FIELD_W*FIELD_H*FIELD_D);
            Do(grid);
            timer.Finish(); }}
void PathFinder::ShowTitle()
{     printf("%s Algorithm\n", title);}
void PathFinder::ShowResults()
{     ShowTitle();
      ShowPath();
      ShowTime();}
void PathFinder::ShowPath()
{     switch (path_result)
      {case P_WAS_NOT_BUILD: puts("Path was not build!"); break;
      case P_CAN_NOT_BUILD: puts("Path was not build!"); break;
      case P_EQUAL:
            puts("Start and end points are equal");
            printf("Start   coords   (%d,   %d,   %d)\n",   start_x,   start_y,
start_z);
            printf("End coords (%d, %d, %d)\n", end_x, end_y, end_z);
            break;
      case P_WAS_BUILD:
            int x, y, z;
            int d_hw = FIELD_H * FIELD_W;
            puts("Path: ");
            for (int i = 0; (i < path_length) && (path[i] != -1); i++)
            {     x = (path[i] % d_hw) % FIELD_W;
                  y = (path[i] % d_hw) / FIELD_W;
                  z = path[i] / d_hw;
                  printf("%02d. (%d, %d, %d)\n", (i + 1), x, y, z);}
            break;}}
void PathFinder::ShowTime()
{     printf("Time of algorithm execution = %f\n", timer.getTime());}
```

Додаток В

Відомість атестаційної роботи

	Позначення		Назва	Дод. відомості
			Текстові документи	
1	ГЮИК. 502528.006 ПЗ		Пояснювальна записка	74 листів
2	ГЮИК. 502528.006 01-12-01		Текст програми	14 листів
	ГЮИК. 502528.006		Графічні матеріали	
3			Концептуальна діаграма компоненту системи	1 лист
4			Декомпозиція діаграми компоненту системи	1 лист
5			Декомпозиція функції взаємодії користувача через інтерфейс	1 лист
6			Декомпозиція функції взаємодії безпілотного літального апарата з системою	1 лист
7			Декомпозиція функції обслуговування програмної системи	1 лист
8			Логічна схема бази данних	1 лист
9			Фізична схема бази данних	1 лист

					ГЮИК. 502528.006 ДЗ		
Изм.	*Лист*	*№ докум.*	*Підпис*	*Дата*			
Розробив		*Костюкевич В.В.*			*Дослідження методів пошуку оптимальних шляхів руху в трьохвимірному просторі для інформаційних систем організації повітряної доставки корисних вантажів*	Лист	Листів
Перевірив.		*Решетнік В.М.*				*1*	*1*
Н. Контр.		*Решетнік В.М.*				*ХНУРЕ Кафедра СТ*	
Затвердив.		Гребеннік І.В.					