ДОДАТОК А

ЛІСТИНГ КОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Точка запуска програми:

```
static void Main(string[] args)
        {
            var settings = new NativeWindowSettings() { Size = new Vector2i(1200, 700),
Title = "Route DMS" };
            using (var window = new DMSWindow(GameWindowSettings.Default, settings))
            {
                // configure coordinates
                NGAlgorithm algorithm = new NGAlgorithm();
                algorithm.SetData(Maps.CubePositions);
                Vector3[] points = algorithm.GetRoute(new Vector3(-2, 0, 5), new
Vector3(3.0f, 0.0f, 4.0f));
                window.SetMap(Maps.CubePositions);
                window.SetPoints(points);
                window.Run();
            }
        }
```

Алгоритм модифікованого методу НГ

```
internal enum Pos
    {
        Top = 1,
        Bottom = 2,
        Left = 4,
        Right = 8,
        Front = 16,
        Back = 32,
        X = Right | Left,
        Y = Top | Bottom,
        Z = Front | Back,
    }


public class NGAlgorithm
    {

        private IList<BoundingBox> initBoxes;
        private IList<BoundingBox> usingBoxes;

        public void SetData(Vector3[] positions)
        {
            this.initBoxes = positions.ToNavGraphCube();
            this.usingBoxes = new List<BoundingBox>();
            this.calcIntersectionBoxes();
        }
```

```
        private void calcIntersectionBoxes()
        {
            IList<BoundingBox> intersected = new List<BoundingBox>();

            foreach (var box in this.initBoxes)
            {
                if (intersected.Contains(box)) continue;

                var intersectedBox = this.searchIntersectionBox(box, intersected);
                if (intersectedBox.HasValue)
                {
                    this.unionIntersectionBoxes(box, intersectedBox.Value, intersected);
                } else
                {
                    this.usingBoxes.Add(box);
                }
            }
        }

        private BoundingBox? searchIntersectionBox(BoundingBox box, IList<BoundingBox> skip)
        {
            foreach (var compareBox in this.initBoxes.Except(skip).Except(this.usingBoxes))
            {
                if (box.Equals(compareBox)) continue;

                if (box.Contains(compareBox.Min) || box.Contains(compareBox.Max))
                {
                    return compareBox;
                }
            }

            return null;
        }

        private void unionIntersectionBoxes(BoundingBox box, BoundingBox intersectedBox,
IList<BoundingBox> intersected)
        {
            intersected.Add(box);
            intersected.Add(intersectedBox);

            BoundingBox usingBox = BoundingBox.Union(box, intersectedBox);

            bool restart;
            do
            {
                restart = false;
                var newIntersectedBox = this.searchIntersectionBox(usingBox, intersected);
                if (newIntersectedBox.HasValue)
                {
                    intersected.Add(newIntersectedBox.Value);
                    usingBox.Union(newIntersectedBox.Value);
                    restart = true;
                }
            } while (restart);

            this.usingBoxes.Add(usingBox);
        }

        private Point3d[] getAdditionalPoints(BoundingBox bbox, Point3d point1, Point3d
point2)
```

```
        {
            Pos s1 = this.checkSide(bbox.Min, bbox.Max, point1);
            Pos s2 = this.checkSide(bbox.Min, bbox.Max, point2);

            if (this.isParallel(s1, s2))
            {
                var p1 = point1;
                var p2 = point2;

                // X, Y
                var zf = (s1 == (s1 & Pos.Z)) ? 100 : point1.Z - bbox.Min.Z + point2.Z -
bbox.Min.Z;
                var zb = (s1 == (s1 & Pos.Z)) ? 100 : bbox.Max.Z - point1.Z + bbox.Max.Z -
point2.Z;

                // X, Z
                var yt = (s1 == (s1 & Pos.Y)) ? 100 : bbox.Max.Y - point1.Y + bbox.Max.Y -
point2.Y;
                var yb = 100; // point1.Y - bbox.Min.Y + point2.Y - bbox.Min.Y;

                // Y, Z
                var xl = (s1 == (s1 & Pos.X)) ? 100 : point1.X - bbox.Min.X + point2.X -
bbox.Min.X;
                var xr = (s1 == (s1 & Pos.X)) ? 100 : bbox.Max.X - point1.X + bbox.Max.X -
point2.X;

                var res = new double[] { zf, zb, yt, yb, xl, xr };

                if (res.Min() == zb)
                {
                    p1.Z = bbox.Max.Z;
                    p2.Z = bbox.Max.Z;
                    if (s1 == (s1 & Pos.X)) // X
                    {
                        var d = (point1.Y - point2.Y) / 3;
                        p1.Y = point1.Y - d;
                        p2.Y = point2.Y + d;
                    }
                    else // Y
                    {
                        var d = (point1.X - point2.X) / 3;
                        p1.X = point1.X - d;
                        p2.X = point2.X + d;
                    }
                }
                else if(res.Min() == zf)
                {
                    p1.Z = bbox.Min.Z;
                    p2.Z = bbox.Min.Z;
                    if (s1 == (s1 & Pos.X)) // X
                    {
                        var d = (point1.Y - point2.Y) / 3;
                        p1.Y = point1.Y - d;
                        p2.Y = point2.Y + d;
                    }
                    else // Y
                    {
                        var d = (point1.X - point2.X) / 3;
                        p1.X = point1.X - d;
                        p2.X = point2.X + d;
                    }
```

```
}
else if (res.Min() == yt)
{
    p1.Y = bbox.Max.Y;
    p2.Y = bbox.Max.Y;
    if (s1 == (s1 & Pos.X))
    {
        var d = (point1.Z - point2.Z) / 3;
        p1.Z = point1.Z - d;
        p2.Z = point2.Z + d;
    }
    else // Z
    {
        var d = (point1.X - point2.X) / 3;
        p1.X = point1.X - d;
        p2.X = point2.X + d;
    }
}
else if (res.Min() == yb)
{
    p1.Y = bbox.Min.Y;
    p2.Y = bbox.Min.Y;
    if (s1 == (s1 & Pos.X))
    {
        var d = (point1.Z - point2.Z) / 3;
        p1.Z = point1.Z - d;
        p2.Z = point2.Z + d;
    }
    else // Z
    {
        var d = (point1.X - point2.X) / 3;
        p1.X = point1.X - d;
        p2.X = point2.X + d;
    }
}
else if (res.Min() == xl)
{
    p1.X = bbox.Min.X;
    p2.X = bbox.Min.X;
    if (s1 == (s1 & Pos.Y))
    {
        var d = (point1.Z - point2.Z) / 3;
        p1.Z = point1.Z - d;
        p2.Z = point2.Z + d;
    }
    else // Z
    {
        var d = (point1.Y - point2.Y) / 3;
        p1.Y = point1.Y - d;
        p2.Y = point2.Y + d;
    }
}
else
{
    p1.X = bbox.Min.X;
    p2.X = bbox.Min.X;
    if (s1 == (s1 & Pos.Y))
    {
        var d = (point1.Z - point2.Z) / 3;
        p1.Z = point1.Z - d;
        p2.Z = point2.Z + d;
```

```
            }
            else // Z
            {
                var d = (point1.Y - point2.Y) / 3;
                p1.Y = point1.Y - d;
                p2.Y = point2.Y + d;
            }

        }

        return new Point3d[] { p1, p2 };
    } else
    {
        var p = Point3d.Origin;

        if (s1 == (s1 & Pos.X))
        {
            p.X = point1.X;
        }
        else if (s2 == (s2 & Pos.X))
        {
            p.X = point2.X;
        }
        else
        {
            p.X = point1.X + (point2.X - point1.X) / 2;
        }

        if (s1 == (s1 & Pos.Y))
        {
            p.Y = point1.Y;
        }
        else if (s2 == (s2 & Pos.Y))
        {
            p.Y = point2.Y;
        }
        else
        {
            p.Y = point1.Y + (point2.Y - point1.Y) / 2;
        }

        if (s1 == (s1 & Pos.Z))
        {
            p.Z = point1.Z;
        }
        else if (s2 == (s2 & Pos.Z))
        {
            p.Z = point2.Z;
        }
        else
        {
            p.Z = point1.Z + (point2.Z - point1.Z) / 2;
        }

        return new Point3d[] { p };
    }
}


private bool isParallel(Pos s1, Pos s2)
{
```

```
            if ((s1 == (s1 & Pos.X) && s2 == (s2 & Pos.X)) ||
                (s1 == (s1 & Pos.Y) && s2 == (s2 & Pos.Y)) ||
                (s1 == (s1 & Pos.Z) && s2 == (s2 & Pos.Z)))
            {
                return true;
            }
            return false;
        }

        private Pos checkSide(Point3d min, Point3d max, Point3d point)
        {
            Pos pos = 0;
            if (point.X == min.X) pos = Pos.Left | pos;
            if (point.X == max.X) pos = Pos.Right | pos;
            if (point.Y == min.Y) pos = Pos.Bottom | pos;
            if (point.Y == max.Y) pos = Pos.Top | pos;
            if (point.Z == min.Z) pos = Pos.Front | pos;
            if (point.Z == max.Z) pos = Pos.Back | pos;

            return pos;
        }

        public Vector3[] GetRoute(Vector3 start, Vector3 end)
        {
            return this.searchIntersectionPoints(new Line(start.X, start.Y, start.Z, end.X,
end.Y, end.Z)).ToArray();
        }
    }
```

Прорисовка (OpenGL, OpenTK)

```
public class DMSWindow : GameWindow
    {
        private Camera camera;
        private readonly float sensitivity = 0.2f;
        private readonly float cameraSpeed = 3f;

        private int barrierVAO;
        private int routerVAO;
        private int lineVAO;
        private int cylinderVAO;

        private int cubeVBO;
        private int lineVBO;
        private int cylinderVBO;
        private int cylinderEBO;

        private Shader defaultShaderT;
        private Shader defaultShader;

        private Texture barierTexture;
        private Texture planeTexture;

        private Vector3[] barrierPositions;
        private Vector3[] routePoints;
```

```
        // moving
        private int currentPoint;
        private Vector3 direction;
        private float offset;

        // MP
        private float len_z13 = 3f;
        private float len_z2 = 4f;
        private float heightMP = 4f;

        private float rotateY;
        private float rotateZ1;
        private float rotateZ2;
        private float rotateZ3;

        public DMSWindow(GameWindowSettings gameWindowSettings, NativeWindowSettings
nativeWindowSettings) : base(gameWindowSettings, nativeWindowSettings)
        {
        }

        protected override void OnLoad()
        {
            GL.ClearColor(Color4.AliceBlue);
            GL.Enable(EnableCap.DepthTest);

            // Setup TEXTURES
            this.barierTexture = new Texture("Resources/barier.png");
            this.planeTexture = new Texture("Resources/plane.png");

            // Setup SHADERS
            this.defaultShaderT = new Shader("Shaders/defaultT.vert",
"Shaders/defaultT.frag");
            this.defaultShader = new Shader("Shaders/default.vert", "Shaders/default.frag");

            // Setup BUFFER OBJECTS
            this.cubeVBO = GL.GenBuffer();
            GL.BindBuffer(BufferTarget.ArrayBuffer, this.cubeVBO);
            GL.BufferData(BufferTarget.ArrayBuffer, Shapes.Quad.Length * sizeof(float),
Shapes.Quad, BufferUsageHint.StaticDraw);

            this.lineVBO = GL.GenBuffer();
            GL.BindBuffer(BufferTarget.ArrayBuffer, this.lineVBO);
            GL.BufferData(BufferTarget.ArrayBuffer, Shapes.MP.Length * sizeof(float),
Shapes.MP, BufferUsageHint.StaticDraw);

            this.cylinderVBO = GL.GenBuffer();
            GL.BindBuffer(BufferTarget.ArrayBuffer, this.cylinderVBO);
            var cylinderV = Shapes.GenerateCylinderV(0.5f, 0.5f);
            GL.BufferData(BufferTarget.ArrayBuffer, cylinderV.Length * sizeof(float),
cylinderV, BufferUsageHint.StaticDraw);

            this.cylinderEBO = GL.GenBuffer();
            GL.BindBuffer(BufferTarget.ElementArrayBuffer, this.cylinderEBO);
            var cylinderI = Shapes.GenerateCylinderI();
            GL.BufferData(BufferTarget.ElementArrayBuffer, cylinderI.Length * sizeof(uint),
cylinderI, BufferUsageHint.StaticDraw);

            // Setup Barriers VAO
            {
                this.barrierVAO = GL.GenVertexArray();
                GL.BindVertexArray(this.barrierVAO);
```

```
                GL.BindBuffer(BufferTarget.ArrayBuffer, this.cubeVBO);

                var positionLocation = this.defaultShaderT.GetAttribLocation("aPos");
                GL.EnableVertexAttribArray(positionLocation);
                GL.VertexAttribPointer(positionLocation, 3, VertexAttribPointerType.Float,
false, 5 * sizeof(float), 0);

                var texCoordLocation = this.defaultShaderT.GetAttribLocation("aTexCoords");
                GL.EnableVertexAttribArray(texCoordLocation);
                GL.VertexAttribPointer(texCoordLocation, 2, VertexAttribPointerType.Float,
false, 5 * sizeof(float), 3 * sizeof(float));
            }
            // SETUP Routers VAO
            {
                this.routerVAO = GL.GenVertexArray();
                GL.BindVertexArray(this.routerVAO);
                GL.BindBuffer(BufferTarget.ArrayBuffer, this.cubeVBO);

                var positionLocation = this.defaultShader.GetAttribLocation("aPos");
                GL.EnableVertexAttribArray(positionLocation);
                GL.VertexAttribPointer(positionLocation, 3, VertexAttribPointerType.Float,
false, 5 * sizeof(float), 0);
            }

            // SETUP Cylinder VAO
            {
                this.cylinderVAO = GL.GenVertexArray();
                GL.BindVertexArray(this.cylinderVAO);
                GL.BindBuffer(BufferTarget.ArrayBuffer, this.cylinderVBO);
                GL.BindBuffer(BufferTarget.ElementArrayBuffer, this.cylinderEBO);

                var positionLocation = this.defaultShader.GetAttribLocation("aPos");
                GL.EnableVertexAttribArray(positionLocation);
                GL.VertexAttribPointer(positionLocation, 3, VertexAttribPointerType.Float,
false, 3 * sizeof(float), 0);
            }

            // SETUP Line VAO
            {
                this.lineVAO = GL.GenVertexArray();
                GL.BindVertexArray(this.lineVAO);
                GL.BindBuffer(BufferTarget.ArrayBuffer, this.lineVBO);

                var positionLocation = this.defaultShader.GetAttribLocation("aPos");
                GL.EnableVertexAttribArray(positionLocation);
                GL.VertexAttribPointer(positionLocation, 3, VertexAttribPointerType.Float,
false, 3 * sizeof(float), 0);
            }

            this.camera = new Camera(Vector3.UnitZ * 3, this.Size.X / (float)this.Size.Y);

            this.CursorVisible = false;
            this.CursorGrabbed = true;

            base.OnLoad();
        }

        protected override void OnRenderFrame(FrameEventArgs args)
        {
            GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);
```

```
            var invZ = Matrix4.CreateScale(1, 1, -1);

            // Drawing Barries
            GL.BindVertexArray(this.barrierVAO);
            this.barierTexture.Use();
            this.defaultShaderT.Use();

            this.defaultShaderT.SetMatrix("view", this.camera.GetViewMatrix());
            this.defaultShaderT.SetMatrix("projection", this.camera.GetProjectionMatrix());

            foreach (var position in this.barrierPositions)
            {
                Matrix4 model = Matrix4.CreateTranslation(position) * invZ;
                this.defaultShaderT.SetMatrix("model", model);
                GL.DrawArrays(PrimitiveType.Triangles, 0, 36);
            }

            // + Plane
            this.planeTexture.Use();
            {
                Matrix4 model = Matrix4.CreateScale(20, 0, 20) *
Matrix4.CreateTranslation(3, -0.5f, -4f);
                this.defaultShaderT.SetMatrix("model", model);
                GL.DrawArrays(PrimitiveType.Triangles, 24, 6);
            }

            // Drawing route
            GL.BindVertexArray(this.routerVAO);
            this.defaultShader.Use();
            this.defaultShader.SetVector3("objectColor", new Vector3(1.0f, 0.0f, 0.0f));

            this.defaultShader.SetMatrix("view", this.camera.GetViewMatrix());
            this.defaultShader.SetMatrix("projection", this.camera.GetProjectionMatrix());

            {
                Matrix4 model = Matrix4.CreateScale(0.5f) *
Matrix4.CreateTranslation(this.routePoints[this.currentPoint]) *
Matrix4.CreateTranslation(this.offset * this.direction) * invZ;
                this.defaultShader.SetMatrix("model", model);
                GL.DrawArrays(PrimitiveType.Triangles, 0, 36);
            }

            // + base MP
            GL.BindVertexArray(this.cylinderVAO);

            this.defaultShader.SetVector3("objectColor", new Vector3(0.0f, 0.0f, 1.0f));
            for (int i = 0; i <= this.heightMP; i ++) {
                Matrix4 model = Matrix4.Identity * Matrix4.CreateTranslation(0, i, 0);
                this.defaultShader.SetMatrix("model", model);
                GL.DrawElements(PrimitiveType.Triangles, 372, DrawElementsType.UnsignedInt,
0);
            }

            // Drawing line
            GL.BindVertexArray(this.lineVAO);
            this.defaultShader.SetVector3("objectColor", new Vector3(0.0f, 1.0f, 0.0f));
            GL.LineWidth(10);
            {
                Matrix4 model = Matrix4.CreateScale(this.len_z13, 0, 0)
                    * Matrix4.CreateRotationZ(this.rotateZ1)
                    * Matrix4.CreateTranslation(0, this.heightMP, 0)
```

```
                    * Matrix4.CreateRotationY(this.rotateY);
                this.defaultShader.SetMatrix("model", model);
                GL.DrawArrays(PrimitiveType.Lines, 0, 2);
            }
            {
                Matrix4 model = Matrix4.CreateScale(this.len_z2, 0, 0)
                    * Matrix4.CreateRotationZ(this.rotateZ2 + MathHelper.PiOver2 -
this.rotateZ1)
                    * Matrix4.CreateTranslation(0, this.len_z13, 0)
                    * Matrix4.CreateRotationZ((MathHelper.PiOver2 - this.rotateZ1) * -1)
                    * Matrix4.CreateTranslation(0, this.heightMP, 0)
                    * Matrix4.CreateRotationY(this.rotateY);
                this.defaultShader.SetMatrix("model", model);
                GL.DrawArrays(PrimitiveType.Lines, 0, 2);
            }
            {
                Matrix4 model = Matrix4.CreateScale(this.len_z13, 0, 0)
                    * Matrix4.CreateRotationZ(this.rotateZ3 - this.rotateZ2)
                    * Matrix4.CreateTranslation(this.len_z2, 0, 0)
                    * Matrix4.CreateRotationZ(this.rotateZ2 + MathHelper.PiOver2 -
this.rotateZ1)
                    * Matrix4.CreateTranslation(0, this.len_z13, 0)
                    * Matrix4.CreateRotationZ((MathHelper.PiOver2 - this.rotateZ1) * -1)
                    * Matrix4.CreateTranslation(0, this.heightMP, 0)
                    * Matrix4.CreateRotationY(this.rotateY); ; //
                this.defaultShader.SetMatrix("model", model);
                GL.DrawArrays(PrimitiveType.Lines, 0, 2);
            }

            this.SwapBuffers();
            base.OnRenderFrame(args);
        }


        private void calcIKP()
        {
            Vector4 curPos = new Vector4(this.routePoints[this.currentPoint], 1)
                        * Matrix4.CreateTranslation(this.offset * this.direction);

            if (curPos.X >= 0)
            {
                this.rotateY = MathF.Atan(curPos.Z / curPos.X);
            } else
            {
                this.rotateY = MathF.Atan(curPos.Z / curPos.X) + MathHelper.Pi;
            }


            Vector3 curPL = new Vector3(curPos.X, curPos.Y - this.heightMP, curPos.Z);

            var len = (curPL - Vector3.Zero).Length;
            var A = Vector3.CalculateAngle(new Vector3(curPL.X, 0, curPL.Z), curPL);
            A *= (curPL.Y < 0) ? -1 : 1;
            var B = MathF.Acos((len - this.len_z2) / 2 / this.len_z13);

            this.rotateZ1 = A + B;
            this.rotateZ2 = A;
            this.rotateZ3 = A - B;
        }

        protected override void OnUpdateFrame(FrameEventArgs args)
```

```
        {
            if (!this.IsFocused)
            {
                return;
            }

            if (this.currentPoint < this.routePoints.Length - 1) {
                var lenght = (this.routePoints[this.currentPoint + 1] -
this.routePoints[this.currentPoint]).Length;
                if (lenght <= this.offset)
                {
                    this.currentPoint++;
                    this.offset = 0;

                    this.direction = this.currentPoint < this.routePoints.Length - 1 ?
Vector3.Normalize(routePoints[this.currentPoint + 1] - routePoints[this.currentPoint]) :
Vector3.Zero;

                } else
                {
                    this.offset += (float)args.Time / 0.5f;

                    this.calcIKP();
                }
            }

            if (this.IsKeyDown(Keys.Escape))
            {
                this.Close();
            }

            if (this.IsKeyDown(Keys.Q))
            {
                this.resetMoving();
            }

            if (this.IsKeyDown(Keys.W))
            {
                this.camera.Position += this.camera.Front * this.cameraSpeed *
(float)args.Time; // Forward
            }
            if (this.IsKeyDown(Keys.S))
            {
                this.camera.Position -= this.camera.Front * this.cameraSpeed *
(float)args.Time; // Backwards
            }
            if (this.IsKeyDown(Keys.A))
            {
                this.camera.Position -= this.camera.Right * this.cameraSpeed *
(float)args.Time; // Left
            }
            if (this.IsKeyDown(Keys.D))
            {
                this.camera.Position += this.camera.Right * this.cameraSpeed *
(float)args.Time; // Right
            }
            if (this.IsKeyDown(Keys.Space))
            {
                this.camera.Position += this.camera.Up * this.cameraSpeed *
(float)args.Time; // Up
            }
```

```
            if (this.IsKeyDown(Keys.LeftShift))
            {
                this.camera.Position -= this.camera.Up * this.cameraSpeed *
(float)args.Time; // Down
            }

            base.OnUpdateFrame(args);
        }

        protected override void OnMouseMove(MouseMoveEventArgs e)
        {
            this.camera.Pitch -= e.DeltaY * sensitivity;
            this.camera.Yaw += e.DeltaX * sensitivity;

            base.OnMouseMove(e);
        }

        protected override void OnMouseWheel(MouseWheelEventArgs e)
        {
            this.camera.Fov -= e.OffsetX;

            base.OnMouseWheel(e);
        }

        protected override void OnResize(ResizeEventArgs e)
        {
            GL.Viewport(0, 0, e.Size.X, e.Size.Y);
            this.camera.AspectRatio = this.Size.X / (float)this.Size.Y;

            base.OnResize(e);
        }

        protected override void OnUnload()
        {
            GL.BindVertexArray(0);
            GL.BindBuffer(BufferTarget.ArrayBuffer, 0);
            GL.UseProgram(0);

            GL.DeleteBuffer(this.cubeVBO);
            GL.DeleteVertexArray(this.barrierVAO);
            GL.DeleteVertexArray(this.routerVAO);

            base.OnUnload();
        }

        public void SetMap(Vector3[] cubePositions)
        {
            this.barrierPositions = cubePositions;
        }

        public void SetPoints(Vector3[] routePoints)
        {
            this.routePoints = routePoints;
            this.resetMoving();
        }

        private void resetMoving()
        {
            this.currentPoint = 0;
            this.offset = 0;
```

```
            this.direction = this.routePoints.Length > 1 ?
Vector3.Normalize(routePoints[this.currentPoint + 1] - routePoints[this.currentPoint]) :
Vector3.Zero;
        }
    }
```

ДОДАТОК Б
ДЕМОНСТРАЦІЙНИЙ МАТЕРІАЛ У ВИГЛЯДІ ПРЕЗЕНТАЦІЇ

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮИК. 503320.03 ПЗ	Пояснювальна записка	90 с., ф. А4
		Графічні документи	
2	ГЮИК 503320.03.ПР	Презентація	у форматі
			*.ppt

					ГЮИК. 503320.03 ВД			
Змін.	Арк.	№ документа	Підп.	Дата				
Розробив		Дієсперов А. В.			Інтелектуальна система прийняття рішень для маніпуляційного робота	Літ.	Аркуш	Аркушів
Перевірив		Цимбал О. М.						
Н. контр.		Демська Н. П.				ХНУРЕ Кафедра КІТАМ		
Затвердив		Невлюдов І. Ш.						