

Házi feladat Terv

Programozás alapjai 2.

Feladat leírás

A program egy konzolos applikáció, amely képes bármilyen kétdimenziós konvex síkidom ábrázolására. Ezeket grafikus úton is képes megjeleníteni. Fő funkciója ezen síkidomok érintkezésének vizsgálata. A felhasználó parancssoros utasításokkal hozhat létre síkidomokat megadva azok típusát és paramétereit, és egyedi névvel különbözteti meg ezeket. A létrehozott síkidomokat mozgathatja, elforgathatja, és nagyíthatja. A program lehetőséget ad a létrehozott síkidomok érintkezésének vizsgálatára a GJK algoritmus segítségével. A program használható önmagában is, de könyvtár verzióban is elérhető, így a felhasználó a parancsokat és a síkidomtípusokat bővítheti.

Síkidomok kezelése

A program támogatni fog alapvető síkidom típusokat. A könyvtár verzióban ezeket a felhasználó saját típusaival bővítheti úgy, hogy azokkal a parancsok és az érintkezés vizsgálata automatikusan működjön.

A felhasználó a megadott típusokból választva parancssoros utasítással példányosíthat síkidomokat, amelyeket egyedi névvel lát el, és paraméterez az adott síkidom típusának megfelelően. Nem megfelelő paraméterezés esetén a program hibaüzenetet ír a standard kimenetre és a példányosítás sikertelen lesz. Ha a felhasználó egy síkidomnak egy, már létező síkidommal azonos nevet ad, a példányosítás sikertelen lesz. Sikeres példányosítás esetén a síkidomra a felhasználó a továbbiakban a neve alapján hivatkozhat.

Az alábbiakból támogatott síkidomok és paramétereik:

- Pont - pont x, y koordinátája
- Kör – sugár hossza
- Ellipszis – nagy tengely felének hossza, kis tengely felének hossza
- Sokszög – csúcsok száma, csúcsonként x, y koordináta
- Szabályos sokszög – csúcsok száma, egy oldalának hossza
- Margós sokszög – csúcsok száma, margó mérete, csúcsonként x, y koordináta
- Bézier-görbe – szegmensek száma, szegmensenként bal, közép és jobb pont x, y komponense

A program az alaptípusokra ellenőrzi a konvexitást, és hibaüzenetet ír a standard kimenetre, ha a megadott síkidom konkáv (például sokszög esetében). A felhasználó által készített típusok esetében a konvexitás ellenőrzése a felhasználó feladata.

A program a síkidom példányokat képes *.shps* formátumú fájlba menteni és beolvasni azokat. Az ilyen fájlok felépítése a következő:

Egy síkidom adatainak kezdetét a *new* token jelzi, majd a síkidom típusa következik és a síkidom paramétereit. Ez a paraméterlista nem egyezik a parancssoros *create* parancs esetén megadandó paraméterlistával.

A formátum nem követeli meg hogy a síkidomok különböző sorokban szerepeljenek, de a program esztétikai okokból ezeket automatikusan sorokba rendezi.

Parancsok

A program képes a standard bemenetről szöveges utasításokat fogadni. A könyvtár verzióban ezeket a felhasználó saját parancsaival bővítheti.

A program ezekkel a parancsokkal rendelkezik:

- **help** – Kilistázza a végrehajtható parancsokat, és leírásaikat.
- **shapetype** – Kilistázza a példányosítható síkidom típusokat.
- **shapes** – Kilistázza a létrehozott síkidomokat és azok típusát.
- **create** – Létrehoz egy síkidomot a megadott névvel és paraméterekkel.
- **destroy** – Törli a megnevezett síkidomot.
- **move** – Elmozgatja a megnevezett síkidomot egy megadott vektorral.
- **rotate** – Elforgatja a megnevezett síkidomot egy megadott szöggel.
- **scale** – Tengelyenként nagyítja a megnevezett síkidomot megadott x, y komponenssel.
- **contacts** – Kilistázza az érintkező síkidomok nevét.
- **contact** – Megvizsgálja, hogy két megnevezett síkidom érintkezik e.
- **save** – A létrehozott síkidomokat elmenti a betöltött fájlba.
- **saveas** – A létrehozott síkidomokat egy új fájlba menti. A nem fogad el már létező fájlt.
- **load** – Beolvassa a megadott fájlban tárolt síkidomokat.
- **merge** – Beolvassa a megadott fájlban tárolt síkidomokat, és a már meglévőkhöz adja.
- **openwin** – Új ablakot nyit, ahol a síkidomok grafikusán ábrázolva tekinthetők meg.
- **exit** – Kilép a programból.

Egy parancsot meghívni a helyes kulcsszóval és helyes paraméterezéssel lehet. Ha a megadott kulcsszó nem felismerhető, vagy a felhasználó nem adott meg elég paramétert, a program hibaüzenetet ír a standard kimenetre, és a végrehajtani kívánt parancsnak nem lesz hatása. Ha túl sok paramétert ad a felhasználó az nem befolyásolja a parancs végrehajtását. A nem felhasznált paraméterekről a felhasználó szöveges értesítést kap a standard kimeneten.

Az al tárolók helyettesítésére elkészítendő a glib(general library) névtérben a:

- list
- array
- registry(nem teljes map) osztályok
- string

A könnyebb olvashatóságért a programban definíciók specializált template típusokat, melyeket az alábbi konvenciók alapján nevezek el:

```
*List = glib::list<T>
*Array = glib::array<T>
*Reg = glib::registry<T, ?>
```

```

classDiagram
    class Shape {
        ~_position : vec2d
        ~_scale : vec2d
        ~_rotation : double
        ~transform_update : bool
        ~transform : Transform
        rshape : VertexArray
        rdisplayColor : Color
        rname : string
        +WorldObj()
        ~virtual ~WorldObj()
        ~virtual write(stream&) : void
        ~virtual read(stream&) : void
        ~virtual fromConsole(stringstream&) : void
        +setPosition(const vec2d) : WorldObj&
        +setPosition(int, int) : WorldObj&
        +setScale(const vec2d) : WorldObj&
        +setScale(int, int) : WorldObj&
        +setRotation(double) : WorldObj&
        +move(const vec2d) : WorldObj&
        +move(int, int) : WorldObj&
        +scale(const vec2d) : WorldObj&
        +scale(int, int) : WorldObj&
        +rotate(double) : WorldObj&
        +getPosition() : const vec2d&
        +getScale() : const vec2d&
        +getRotation() : bool
        +setColor(Color) : void
        +getNamed() : const string&
        ~recalculateTransform() : void
        ~virtual draw(RenderTargets, RenderStates) : void
    }

    class ConvexShape {
        #my_type : string
        +ConvexShape(const string&)
        ~virtual ~ConvexShape()
        ~virtual write(stream&) : void
        ~virtual read(stream&) : void
        ~virtual fromConsole(stringstream&) : void
        +support(const vec2d) : vec2d
        ~virtual objSpaceSupport(const vec2d direction) : vec2d
        +getType() : const string&
    }

    class Point {
        +Point(string, double, double)
        +Point(string, vec2d)
        ~virtual write(stream&) : void
        ~virtual read(stream&) : void
        ~virtual fromConsole(stringstream&) : void
        ~virtual objSpaceSupport(const vec2d direction) : vec2d
        ~virtual draw(RenderTargets, RenderStates) : void
    }

    class Circle {
        ~r : double
        +Circle(string, double)
        ~build() : void
        ~virtual write(stream&) : void
        ~virtual read(stream&) : void
        ~virtual fromConsole(stringstream&) : void
        ~virtual objSpaceSupport(const vec2d direction) : vec2d
    }

    class Ellipse {
        ~a : double
        ~b : double
        +Ellipse(string, double, double)
        ~build() : void
        ~virtual write(stream&) : void
        ~virtual read(stream&) : void
        ~virtual fromConsole(stringstream&) : void
        ~virtual objSpaceSupport(const vec2d direction) : vec2d
    }

    class Polygon {
        +Polygon(string)
        +Polygon(string, initializer_list)
        ~build(initializer_list)
        ~virtual write(stream&) : void
        ~virtual read(stream&) : void
        ~virtual fromConsole(stringstream&) : void
        ~virtual objSpaceSupport(const vec2d direction) : vec2d
    }

    class BezierCurve {
        ~segments : SegmentArray
        +BezierCurve(string)
        +getPoint(Segment, double)
        ~virtual write(stream&) : void
        ~virtual read(stream&) : void
        ~virtual fromConsole(stringstream&) : void
        ~virtual objSpaceSupport(const vec2d direction) : vec2d
    }

    class Segment {
        +a : vec2d
        +b : vec2d
        +c : vec2d
        +d : vec2d
        Segment(vec2d, vec2d, vec2d, vec2d)
    }

    class Sandbox {
        ~cmd_registry : CmdReg
        ~_registry : ShapeReg
        ~_shapes : ShapeList
        ~<disp : ConsoleDisplay
        ~<disp : WindowDisplay
        ~is_running : bool
        ~myfile : string
        +Sandbox()
        ~+Sandbox()
        ~run() : void
        ~stop() : void
        ~openWindow() : void
        ~closeWindow() : void
        +getCmdReg() : CmdReg&
        +getShapeReg() : ShapeReg&
        +getShapeList() : ShapeList&
        +getMyFile() const string&
        +setMyFile(const string&)
    }

    class ConsoleDisplay {
        ~onUpdate() : void
    }

    class WindowDisplay {
        ~selected : ShapeListIterator
        ~window : RenderWindow
        ~setView() : void
        ~<colorShapes() : void
        ~handleEvent() : void
        ~render() : void
        ~virtual onActivation() : void
        ~virtual onDeactivation() : void
        ~virtual onUpdate() : void
    }

    class GJKSolver {
        ~s1 : ConvexShape&
        ~s2 : ConvexShape&
        ~is_contact : bool
        ~simplex : VecresArray
        +GJKSolver(ConvexShape&, ConvexShape&)
        +isContact() : bool
        ~getSupportPoint(const vec2d&, vec2d&) : bool
        ~findFirstSimplex() : bool
        ~findNextPoint(vec2d&) : bool
        ~checkOverlap() : void
    }

    class Command {
        ~receiver : Sandbox&
        ~desc : string
        ~params : string
        +Command(Sandbox&)
        ~virtual ~Command()
        ~virtual execute(stringstream&)
    }

    Shape <|-- ConvexShape
    ConvexShape <|-- Point
    ConvexShape <|-- Circle
    ConvexShape <|-- Ellipse
    ConvexShape <|-- Polygon
    ConvexShape <|-- BezierCurve
    ConvexShape <|-- Segment
    Sandbox *--> ConsoleDisplay
    Sandbox *--> WindowDisplay
    ConsoleDisplay --> WindowDisplay
    WindowDisplay --> GJKSolver
    GJKSolver --> Command
    Command --> Sandbox : felhívó parancsok végrehajtása
    Polygon --> RegularPolygon
    class RegularPolygon {
        +RegularPolygon(string)
        ~build(size_t, double)
        ~virtual fromConsole(stringstream&) : void
    }
  
```

A GJK algoritmussal eldönthető, hogy két konvex síkidom érintkezik e. Az algoritmus szükséglete, hogy legyen olyan függvény minden síkidomra, ami megadja egy bizonyos irányban a síkidom legmesszebbi pontját. Ezt a függvényt support függvénynek hívák.

- simplex: két elemű `vec2d` tömb
- `s1`, `s2`: a két tesztelt síkidom
- `is_contact`: `bool`

```
függvény getSupportpoint(direction: vec2d, point: vec2d):
    point = s1.support(direction) – s2.support(direction)
    visszatér point * direction > 0
```

```
függvény findFirstSimplex:
    direction := s1.getPosition() – s2.getPosition()
    Ha getSupportPoint(direction, simplex[0]):
        visszatér hamissal
    direction = -simplex[0]
    Ha getSupportPoint(direction, simplex[1]:
        visszatér hamissal
    visszatér igazgal
```

```
függvény findNextPoint(point: vec2d):
    edge := simplex[0] – simplex[1]
    to_origo := -simplex[1]
    normal := getNormal(edge, to_origo)
    visszatér getSupportPoint(normal, point)
```

```
eljárás checkOverlap():
    Ha nem findFirstSimplex():
        visszatér
    next_point: vec2d
    foundRtriangle = hamis
    foundValidPoint = igaz
    ciklus:
        foundValidPoint = findNextPoint(next_point)
        Ha foundValidPoint:
            normal1 := getNormal(simplex[0]–next_point, next_point–simplex[1])
            normal2 := getNormal(simplex[1]–next_point, next_poin –simplex[0])
            np_to_origin = -next_point
            Ha normal1 * np_to_origin > 0:
                simplex[1] = next_point
            Különben ha normal2 *np_to_origin > 0:
                simplex[0] = next_point
            Különben:
                foundTriangle = igaz
    Amíg foundValidPoint és nem foundTriangle
    is_contact = foundTriangle
```