Shape Overlaps Detection Algorithm Implementation

1. 프로젝트 개요

1.1 프로젝트 목적 및 기능

본 프로젝트는 웹 환경에서 다양한 형태의 **도형 간 겹침 여부를 자동으로 판별하고, 시각적으로 명확히 표현하는 것**을 목적으로 한다. 특히 기하학적 알고리즘(SAT 등)을 활용해 다각형과 원의 정확한 충돌 감지 메커니즘을 구현하고, 다수의 도형이 복합적으로 겹쳐 있는 상황에서도 연쇄적인 그룹화 처리를 통해 **겹침 구조를 직관적으로 이해**할 수 있도록 돕는다.

본 프로젝트를 통해 다음과 같은 목적을 달성하고자 한다.

- **정확한 기하학적 겹침 검출 알고리즘을** 구현하여 기술적 역량 강화
- Spring Boot 기반의 REST API 개발과 **객체지향 프로그래밍(OOP) 설계 능력** 함양
- **웹 환경에서 시각적 데이터를 처리하고 표현**하는 방법 습득 및 응용

본 프로젝트에서 구현하는 핵심 기능은 다음과 같다.

(1)다양한 도형 생성

- 원(Circle), 정다각형(Regular Polygon), 일반 다각형(Irregular Polygon) 자동 생성
- 사용자 입력(도형 개수, 최대 반경, 다각형 최대 변의 개수)에 따라 매번 다른 형태와 크기의 도형 생성

(2)정확한 겹침 감지 알고리즘

- **원-원 충돌 검사**: 중심 간 거리와 반지름을 비교하는 간단하고 효율적인 알고리즘
- **다각형-다각형 충돌 검사**: SAT(Separating Axis Theorem, 분리 축 정리) 알고리즘을 사용하여 볼록 다각형 간 정확한 교차 여부 확인
- **원-다각형 충돌 검사**: 다각형의 정점과 변을 기준으로 원과의 교차 여부 판단

(3)연쇄적 그룹화 기능

- Union-Find 자료구조를 활용하여 다수 도형 간 겹침 관계를 효율적으로 관리하고, 겹친 도형들을 그룹으로 시각적으로 구분 및 표현

(4)시각화 및 통계 데이터 제공

- HTML5 Canvas 기반의 직관적인 시각화 기능 제공
- 도형 유형별, 겹침 그룹별 통계 데이터 제공 및 표현

(5)JSON 기반 REST API 인터페이스

- 간단한 REST API 형태로 백엔드 로직과 프론트엔드 시각화를 명확하게 분리하여 유연한 확장성 및 유지보수성 확보

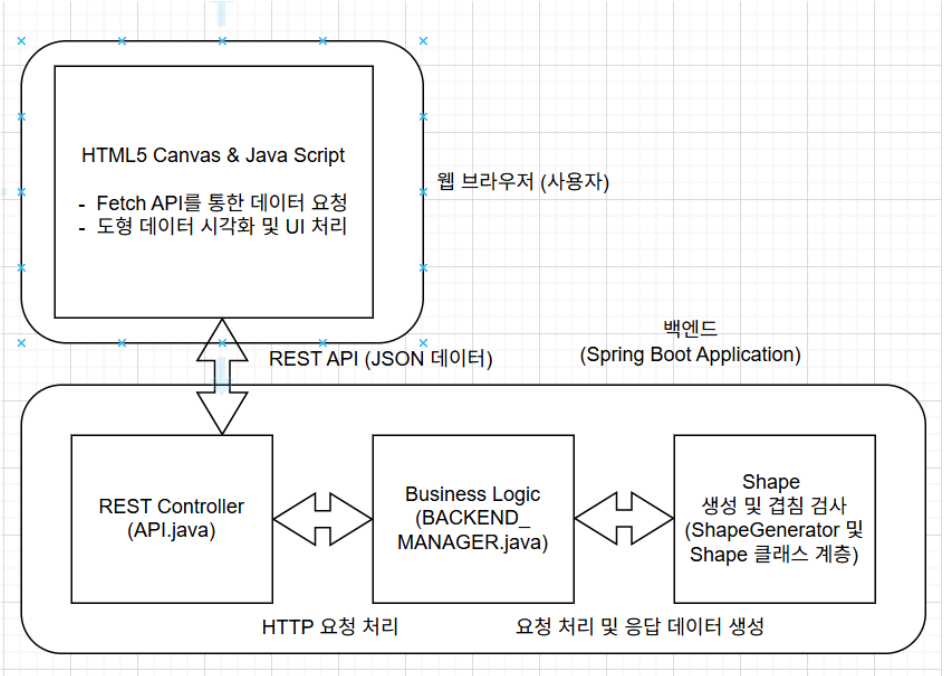
1.2 기술 스택

본 프로젝트에서 사용된 기술 스택은 다음과 같다.

분류	기술 스택	설명 및 역할
프로그래밍 언어	Java 21	객체지향적 설계 및 알고리즘 구현에 사용
백엔드 프레임워크	Spring Boot 3.2.1	REST API 구축 및 백엔드 서비스 구현
빌드 도구	Gradle (Groovy 기반)	프로젝트 빌드 자동화 및 의존성 관리
JSON 처리	org.json 라이브러리	데이터 교환 및 API 응답 형식화
프론트엔드 기술	HTML5, CSS3, JavaScript	사용자 인터페이스(UI) 및 데이터 시각화 구현
클라이언트-서버 통신	Fetch API (JavaScript 내장)	REST API를 통한 데이터 요청 및 수신 처리
개발 환경	IntelliJ IDEA Community Edition 2024.1	통합 개발 환경(IDE)을 통한 효율적 개발

1.3 시스템 아키텍처

본 프로젝트의 시스템 아키텍처는 프론트엔드와 백엔드가 REST API를 중심으로 상호작용하는 구조로 설계하였다. 사용자는 브라우저 기반의 프론트엔드를 통해 도형 생성을 요청하고, 서버는 이를 받아 처리한 후, 결과를 JSON 형태로 반환하여 시각화한다. 아래는 이를 명확히 나타낸 시스템 아키텍처 다이어그램이다.



구성 요소	역할 및 설명
HTML5 Canvas & JavaScript	사용자와의 상호작용 담당 및 시각화 처리
Fetch API	JavaScript에서 REST API와의 통신을 담당
REST API (JSON 데이터)	클라이언트-서버 간의 데이터 전송을 위한 인터페이스 제공
REST Controller (API.java)	클라이언트로부터 들어온 HTTP 요청을 처리하여, JSON 응답 생성 및 전송
BACKEND_MANAGER	비즈니스 로직 처리 (도형 생성, 연산 수행, 데이터 가공 등)
ShapeGenerator	다양한 도형을 무작위로 생성하고, 연쇄적 그룹화 수행
Shape 클래스 계층	기하학적 알고리즘 구현 (Circle, RegularPolygon, IrregularPolygon)

데이터 처리 흐름은 다음과 같다.

- **사용자 입력 및 요청**: 사용자가 프론트엔드의 입력 폼에서 도형 생성을 요청
- **REST API 요청**: 프론트엔드(JavaScript)는 Fetch API를 통해 요청 파라미터를 포함한 HTTP 요청을 서버의 REST Controller로 전달
- **백엔드 REST Controller**: API 컨트롤러가 요청을 받아 파라미터 분석 후 BACKEND_MANAGER로 요청 전달
- **비즈니스 로직 실행**: BACKEND_MANAGER는 요청에 맞는 비즈니스 로직 실행 (도형 생성, 겹침 검사 및 그룹화 등)
- **JSON 데이터 생성**: 결과를 JSON 형태로 가공하여 REST Controller로 전달
- **JSON 데이터 응답**: REST Controller는 가공된 JSON 데이터를 프론트엔드로 응답
- **시각화 처리**: 프론트엔드에서 JSON 데이터를 수신, Canvas에 시각화하여 사용자에게 보여줌

이러한 명확한 구조적 설계를 통해 본 프로젝트는 각 구성 요소의 역할을 명확히 구분하고, 유지보수성과 확장성을 극대화하였다.

2. 기술 분석

2.1 Spring Boot 구조 및 설계

본 프로젝트의 백엔드는 Spring Boot를 기반으로 구축되어 있으며, 이는 REST API 서버로서의 역할을 수행한다.

본 프로젝트에서 활용된 Spring Boot의 주요 구조는 다음과 같다.

구성 요소	상세 설명 및 역할
REST Controller	HTTP 요청과 응답을 처리하는 컨트롤러로, @RestController와 @RequestMapping 어노테이션을 활용하여 API 엔드포인트를 정의한다.
Component	@Component 어노테이션을 통해 BACKEND_MANAGER와 같은 비즈니스 로직 클래스를 Spring의 컴포넌트로 등록하여, 의존성 주입을 통해 관리한다.
Embedded Tomcat	별도의 서버 배포 없이도 Spring Boot 애플리케이션 자체적으로 내장된 웹 서버로 애플리케이션을 빠르게 실행할 수 있도록 지원한다.

2.2 REST API 설계 원칙

본 프로젝트는 REST(Representational State Transfer) 아키텍처 스타일을 엄격히 준수하여 API를 설계하였다. REST는 클라이언트-서버 간의 명확한 역할 분리, Stateless 방식의 상호작용, HTTP 표준 메서드의 활용을 특징으로 하며, 본 프로젝트에서 구현한 REST API는 이러한 원칙들을 다음과 같이 충실히 구현하였다.

(1)Resource 중심 설계

REST API는 리소스를 중심으로 설계된다. 본 프로젝트에서 주요 리소스는 'Shape 데이터'이다. 클라이언트가 API 엔드포인트(/api)로 데이터를 요청하면, 백엔드는 요청된 작업(Action)에 따라 데이터를 처리하고, 해당 리소스 상태를 JSON 형태로 반환한다.

(2)Stateless 원칙

API 요청은 개별적으로 처리되며, 서버는 클라이언트의 상태 정보를 보관하지 않는다. 요청마다 모든 필요한 정보가 포함되므로 서버의 부하를 최소화하며, 확장성을 확보할 수 있다.

(3)Uniform Interface

모든 클라이언트는 일관된 방식으로 API를 사용한다. 본 프로젝트는 HTTP GET과 POST를 통해 JSON 형식의 데이터를 일관된 구조로 주고받아, 서로 다른 시스템 간에도 높은 호환성을 확보하였다.

(4)REST API 데이터 형식 예시

본 프로젝트에서 사용되는 REST API 응답의 예시는 다음과 같다:

```
{
  "REQ": {
    "Action": "ShapesOverlaps",
    "Width": "800",
    "Height": "600",
    "RadiusMax": "50",
    "HowMany": "50",
    "MaxEdges": "15"
  },
  "RES": {
    "STATUS": 200,
    "STATUS_MSG": "OK",
    "RESULT": {
      "shapes": [...],
      "totalCount": 50,
      "overlapGroups": [...]
    }
  }
}
```

이러한 설계를 통해 클라이언트와 서버 간 명확한 데이터 흐름을 제공하여 시스템의 유지보수성과 확장성을 높이고자 하였다.

2.3 객체지향 설계 분석

본 프로젝트는 객체지향 설계의 세 가지 핵심 원칙인 상속(Inheritance), 다형성(Polymorphism), 캡슐화(Encapsulation)를 명확히 구현하였다. 이는 코드의 재사용성을 높이고 유지보수성을 증대시키는 효과를 가진다.

(1)상속

본 프로젝트에서는 추상 클래스인 **Shape**를 기반으로 세부 클래스(Circle, RegularPolygon, IrregularPolygon)가 이를 상속하여 구현되었다. 공통적으로 쓰이는 속성(center, radius, color, id) 및 메서드들을 추상 클래스에서 정의하여 코드의 중복을 최소화하였다.

```
public abstract class Shape {
    protected Point center;
    protected double radius;
    protected String color;
    protected String id;

    public abstract boolean overlaps(Shape other);
    public abstract JSONObject toJSON();
}
```

(2)다형성

다형성은 동일한 메서드 호출이 객체의 실제 타입에 따라 서로 다른 방식으로 작동하는 특성이다. 본 프로젝트의 overlaps() 메서드는 각 클래스마다 다르게 구현되어 있으며, 런타임 시 객체의 타입에 따라 다형적으로 작동한다.

```
// Shape 클래스 내 정의
public abstract boolean overlaps(Shape other);

// Circle에서의 구현 예시
@Override
public boolean overlaps(Shape other) {
    if (other instanceof Circle) {
        return center.distanceTo(other.getCenter()) < (radius + other.getRadius());
    }
    // 추가적인 처리 로직
}

// RegularPolygon, IrregularPolygon 클래스는 다각형 특성을 반영한 SAT 알고리즘 적용
```

(3)캡슐화

캡슐화는 클래스 내부 데이터를 외부에서 직접 접근하지 못하도록 보호하고, 메서드를 통해서만 접근할 수 있도록 제한하는 원칙이다. 본 프로젝트는 모든 데이터 필드를 **protected**나 **private**로 설정하고, **Getter**와 **Setter** 메서드를 통해 접근을 제한함으로써 데이터의 무결성을 보장하고 있다.

```
// 데이터 접근 제한 예시
protected Point center;
protected String color;

// 데이터 접근을 위한 Getter 예시
public Point getCenter() {
    return center;
}

// 데이터 변경을 위한 Setter 예시
public void setColor(String color) {
    this.color = color;
}
```

이러한 객체지향 설계를 통해 본 프로젝트는 유지보수 및 확장성을 용이하게 하였으며, 명확한 클래스 구조와 역할 구분을 통해 향후 추가적인 기능 구현이 쉽도록 하였다.

3. 알고리즘 상세 설명

3.1 도형 겹침 감지 알고리즘 개요

본 프로젝트에서 구현된 도형 겹침 감지 알고리즘은 크게 세 가지 경우로 나누어 접근하였다. 첫째는 **원-원 겹침**을 검사하는 간단한 거리 비교 알고리즘이며, 둘째는 다각형 간의 충돌을 판단하는 **SAT(Separating Axis Thm)** 기반 알고리즘, 셋째는 원과 다각형 간의 겹침을 판단하는 **하이브리드 방식**이다.

(1)원-원 겹침 검사

두 원이 겹치는지 판단하기 위해, 두 원 중심 사이의 거리를 계산하여 두 원의 반지름 합과 비교하는 방식이다. 수학적으로는 다음 조건으로 표현 가능하다.

$$distance < radius1 + radius2$$

(2)다각형-다각형 겹침 검사 (SAT 알고리즘)

SAT 알고리즘은 두 볼록 다각형의 모든 변에 수직인 축을 생성하여 두 다각형의 투영(projection)을 비교해 충돌 여부를 판별하는 방식이다. 충돌 판별을 위해 모든 축에서의 투영 범위를 비교하여 겹침이 존재하는지 확인한다.

(3)원-다각형 겹침 검사 (혼합 방식)

본 프로젝트에서는 다각형의 모든 꼭짓점이 원의 내부에 있는지, 다각형의 각 변이 원과 교차하는지에 따라 겹침 여부를 판단하였다. 이 방식을 통해 볼록 다각형과 원 사이의 정확한 충돌 판별이 이루어진다.

3.2 SAT(Separating Axis Thm.) 알고리즘 상세

SAT는 두 볼록 다각형의 겹침을 정확히 판단할 수 있는 수학적 알고리즘으로, 두 다각형을 명확히 분리하는 축(axis)이 존재하지 않으면 두 다각형이 겹친다는 원리를 기반으로 한다. 본 프로젝트에서의 알고리즘 작동 순서는 다음과 같다.

(1)후보 축 설정

SAT 알고리즘에서 사용되는 후보 축은 두 다각형의 각 변에 수직인 법선 벡터(normal vector)이다.
각 변 (p_1, p_2)의 법선 벡터는 다음과 같이 구할 수 있다.

$$normal\ vector = (-(y_2 - y_1), (x_2 - x_1))$$

(2)투영(projection)

각 후보 축에 대하여 두 다각형의 모든 꼭짓점을 축 방향으로 투영한다. 투영 값(projection value)은 내적(dot product)을 이용하여 계산한다:

$$projection = vertex \cdot axis$$

이 과정에서 각 다각형의 최소 및 최대 투영 값을 저장한다. 두 투영 구간은 다음과 같이 정의된다.

$$projection_1 = [min_1, max_1], projection_2 = [min_2, max_2]$$

(3)겹침 판별

두 다각형의 투영 구간이 모든 후보 축에서 겹치면 두 다각형은 서로 충돌한 것이다. 반대로, 단 하나의 축이라도 투영이 겹치지 않으면 두 다각형은 겹치지 않는다. 수식적으로는 다음과 같이 표현 가능하다.

$$overlaps = (min_1 \leq max_2) \wedge (min_2 \leq max_1)$$

3.3 컨벡스 헐(Convex Hull) 알고리즘 분석

본 프로젝트에서는 무작위로 생성한 다각형의 꼭짓점들이 서로 교차하지 않는 irregular polygons를 이루도록 하기 위해 컨벡스 헐(Convex Hull) 알고리즘을 사용하였다. 컨벡스 헐은 임의의 점 집합을 둘러싸는 가장 작은 irregular polygon을 찾는 알고리즘이다.

(1)정렬 단계

입력된 점 집합을 x좌표 기준으로 오름차순 정렬한다. x좌표가 같을 경우 y좌표로 정렬하여 좌표를 유일하게 정리한다.

(2)하부 헐(Lower Hull) 구성

정렬된 점 집합을 차례대로 검사하면서 현재 검사하는 점이 기존의 헐을 오른쪽으로 돌게 하는지, 왼쪽으로 돌게 하는지를 판단한다. 방향을 판단하는 공식(외적을 이용한 방향 판별 공식)은 다음과 같다.

$$orientation = (q\_x - p\_x)(r\_y - p\_y) - (q\_y - p\_y)(r\_x - p\_x)$$

만약 이 값이 음수이거나 0이면, 기존 헐의 마지막 점을 제거하고 다시 방향을 확인한다. 모든 점에 대해 이 과정을 반복하면 하부 헐이 구성된다.

(3)상부 헐(Upper Hull) 구성

역순으로 같은 과정을 반복하여 상부 헐을 구성한다. 결과적으로 하부 헐과 상부 헐을 합친 것이 완전한 컨벡스 헐이 된다.

3.4 Union-Find 기반 연쇄적 그룹화

본 프로젝트에서는 여러 도형이 복합적으로 겹친 경우 연쇄적으로 그룹을 형성하는 문제를 **Union-Find 알고리즘**으로 해결하였다. Union-Find는 서로소 집합(disjoint-set)을 관리하고, 빠르게 그룹화를 처리할 수 있는 자료구조이다. 본 프로젝트에서 Union-Find의 구현 방식은 다음과 같다.

(1)초기화 단계

각 도형은 초기에 자신을 그룹의 대표(parent)로 하여 독립된 그룹을 형성한다.

```
for (Shape shape : shapes) {
    parent.put(shape.getId(), shape.getId());
    rank.put(shape.getId(), 0);
}
```

(2)Find 연산

각 도형의 그룹 대표를 찾는 과정으로, 경로 압축(path compression)을 통해 효율성을 높인다.

```
private String find(Map<String, String> parent, String x) {
    if (!parent.get(x).equals(x)) {
        parent.put(x, find(parent, parent.get(x)));
    }
    return parent.get(x);
}
```

(3)Union 연산

두 도형이 겹친 경우 해당 그룹을 병합하는 과정이다. 본 프로젝트는 랭크(rank)를 이용한 최적화를 적용하여 트리의 높이를 최소화하여 처리 속도를 향상시켰다.

```
private void union(Map<String, String> parent, Map<String, Integer> rank, String x, String y) {
    String rootX = find(parent, x);
    String rootY = find(parent, y);

    if (!rootX.equals(rootY)) {
        if (rank.get(rootX) < rank.get(rootY)) {
            parent.put(rootX, rootY);
        } else if (rank.get(rootX) > rank.get(rootY)) {
            parent.put(rootY, rootX);
        } else {
            parent.put(rootY, rootX);
            rank.put(rootX, rank.get(rootX) + 1);
        }
    }
}
```

(4)그룹화 결과 생성

Union-Find의 결과로부터 각 그룹별 도형을 추출하여 시각적으로 표현하고 통계 데이터를 제공한다.

본 프로젝트는 이러한 기하학적 알고리즘들을 적절히 활용하여 정확한 도형 겹침 감지와 명확한 그룹화를 수행하였다.

4. 구현 세부사항

4.1 주요 클래스 및 메서드 설명

본 프로젝트의 코드에서 가장 핵심적인 클래스들과 메서드들은 다음과 같다.

클래스명	설명 및 역할	주요 메서드
Shape (추상 클래스)	모든 도형 클래스의 부모 클래스이며 공통 속성과 메서드를 정의한다.	overlaps(), toJSON(), getVertices()
Circle	Shape 클래스를 상속받아 원의 속성과 원-원 및 원-다각형 겹침 로직 구현	overlaps(), getVertices()
RegularPolygon	Shape를 상속받아 정다각형의 속성과 SAT 알고리즘을 활용한 겹침 검사 구현	overlaps(), generateVertices()
IrregularPolygon	Shape를 상속받아 불규칙한 다각형을 생성하고 컨벡스 헐 알고리즘과 SAT 기반의 겹침 로직 구현	overlaps(), generateIrregularVertices()
ShapeGenerator	도형의 랜덤 생성, 겹침 그룹화(Union-Find 알고리즘)를 수행하여 최종 JSON 데이터 생성	generateShapes(), findConnectedComponents()
BACKEND_MANAGER	클라이언트 요청을 처리하고, 요청에 따른 비즈니스 로직을 수행하여 응답 데이터 생성	EXEC_TASK()
API (컨트롤러)	REST API 요청을 수신하여 BACKEND_MANAGER로 전달하고, 결과를 JSON 응답으로 반환	requestParams()

이러한 클래스들은 객체지향 설계 원칙에 따라 역할이 명확히 구분되어 있다.

4.2 JSON 데이터 구조 및 설계

프론트엔드와 백엔드 간의 데이터 전송을 위해 JSON 형식을 사용하며, JSON 구조는 다음과 같이 설계되었다.

[요청(JSON Request) 예시]

```
{
  "Action": "ShapesOverlaps",
  "Width": "800",
  "Height": "600",
  "RadiusMax": "50",
  "HowMany": "50",
```

```
  "MaxEdges": "15"
}
```

[응답(JSON Response) 예시]

```
{
  "REQ": { ... },
  "RES": {
    "STATUS": 200,
    "STATUS_MSG": "OK",
    "RESULT": {
      "shapes": [
        {
          "type": "circle",
          "id": "shape_1716768373654_246",
          "center": {"x": 150.0, "y": 200.0},
          "radius": 35.0,
          "color": "#FF0000"
        },
        {
          "type": "regularPolygon",
          "id": "shape_1716768373655_462",
          "center": {"x": 300.0, "y": 250.0},
          "radius": 45.0,
          "sides": 6,
          "rotationAngle": 1.5708,
          "color": "#00FF00",
          "vertices": [{...}, {...}, {...}]
        }
      ],
      "totalCount": 50,
      "overlapGroups": [
        {
          "shapeIds": ["shape_1716768373654_246", "shape_1716768373655_462"],
          "color": "#FF0000",
          "size": 2
        }
      ]
    }
  }
```

```
}
```

shapes 배열은 생성된 각 도형의 세부 정보를 제공하며, type, id, 중심점(center), 반지름(radius), 색상(color), 꼭짓점(vertices) 등으로 구성된다. 그리고 **overlapGroups** 배열은 겹치는 도형의 그룹 정보로, 그룹 내의 도형 ID와 그룹 색상, 그룹 크기를 명시적으로 포함한다.

본 JSON 구조는 데이터의 명확한 표현과 간결성을 제공하며, 프론트엔드에서의 시각화를 효율적으로 수행할 수 있도록 설계되었다.

4.3 프론트엔드-백엔드 통신 방식

본 프로젝트는 클라이언트와 서버 간의 데이터 통신을 REST API를 통해 이루어지며, 다음과 같은 구조로 처리된다:

(1)클라이언트 요청 생성 (JavaScript Fetch API)

사용자가 입력 폼을 통해 요청하면 JavaScript Fetch API가 호출되어 서버의 REST API 엔드포인트에 HTTP 요청을 전송한다.

```
fetch(`/api?${params.toString()}`)
  .then(response => response.json())
  .then(data => {
    // 데이터 처리 및 시각화
  })
  .catch(error => {
    console.error('API 호출 오류:', error);
  });
```

(2)서버의 요청 처리

서버는 API 컨트롤러(API.java)에서 요청을 수신하고, 요청된 Action에 따라 BACKEND_MANAGER로 로직 처리를 위임한다.

```
@RequestMapping(value="/api", method = {RequestMethod.GET, RequestMethod.POST})
public String requestParams(HttpServletRequest request, HttpServletResponse response) {
    JSONObject params_JSON = new JSONObject(request.getParameterMap());
    JSONObject JSON_RESPONSE = new JSONObject();
    BACKEND_MANAGER.EXEC_TASK(JSON_RESPONSE);
    return JSON_RESPONSE.toString();
}
```

(3)JSON 응답 전송 및 시각화

서버는 가공된 JSON 데이터를 클라이언트에 응답으로 전달하고, 프론트엔드에서는 이를 Canvas를 통해 시각적으로 표현한다.

4.4 에러 처리 전략

본 프로젝트는 다음과 같은 명확한 에러 처리 전략을 구현하였다.

(1)서버측 에러 처리

- 요청 파라미터 유효성 검사

필수 파라미터가 누락되거나 잘못된 값이 전달된 경우 명시적인 에러 메시지를 제공한다.

```
if (!reqJson.has("Width") || !reqJson.has("Height")) {
    throw new Exception("필수 파라미터가 누락되었습니다.");
}
```

-예외 처리

```
try {
    // 비즈니스 로직 수행
} catch (NumberFormatException e) {
    throw new Exception("숫자 형식의 파라미터가 올바르지 않습니다.");
}
```

(2) 클라이언트 측 에러 처리

프론트엔드에서는 API 호출 실패나 에러 응답 수신 시 사용자에게 명확히 알리고, 개발자를 위해 콘솔 로그를 통해 상세한 에러 내역을 기록한다.

```
fetch(`/api?${params.toString()}`)
  .then(response => response.json())
  .then(data => {
    if (data.RES.STATUS !== 200) {
      alert('오류 발생: ' + data.RES.STATUS_MSG);
      console.error(data);
    }
  })
  .catch(error => {
    alert('서버와 통신 중 오류가 발생했습니다.');
    console.error('API 호출 오류:', error);
  });
```

5. 실행 결과 및 검증

5.1 다양한 테스트 케이스 결과

5.2 기능 시연

5.3 성능 최적화 방안