# - Pytet_v0.1 Comments

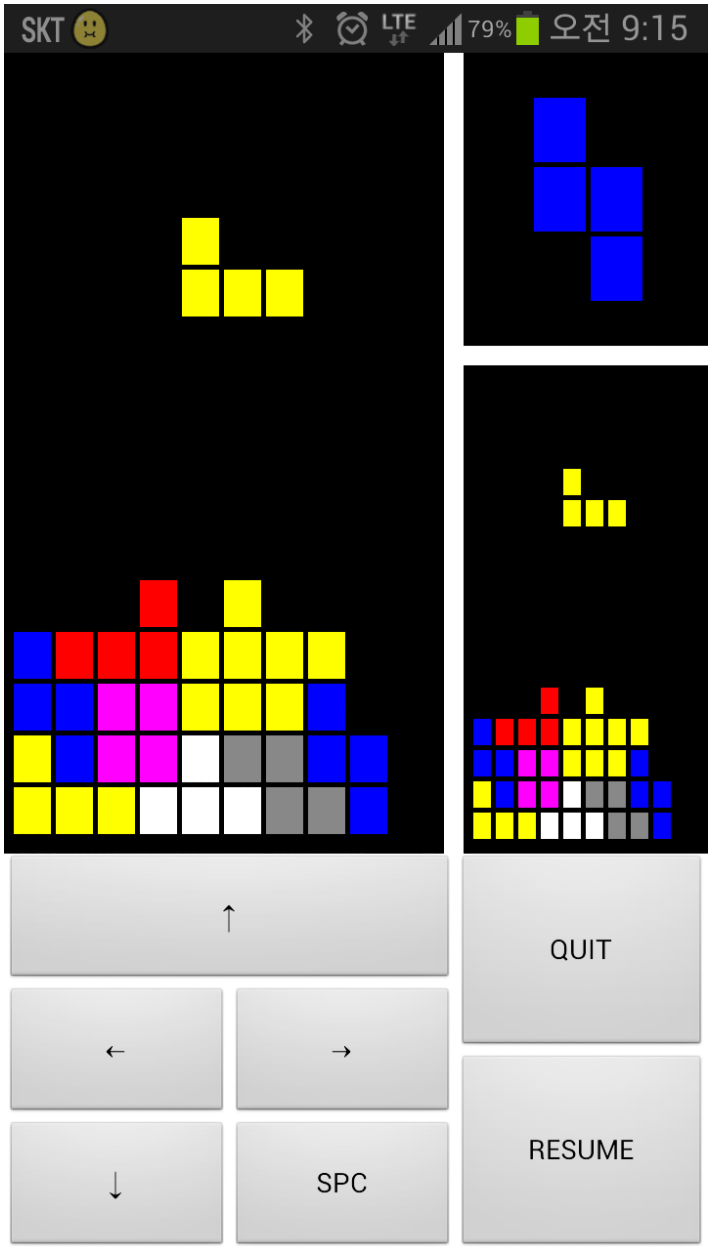
김강희
khkim@ssu.ac.kr

# 소프트웨어 설계

- **대규모 프로젝트 수행시**
  - 요구 분석 → 시스템 설계 → SW 구조 설계 → SW 상세 설계 → UI 설계 → 프로그래밍 …
- **소규모(테트리스) 프로젝트 수행시**
  - **요구 분석** : 그 중요성은 아무리 강조해도 지나치지 않음
    - PC방 주인이 2인용 전투 테트리스 설계를 주문했다고 가정하자^^
  - **프로젝트 로드맵 설계** : 1인용 흑백 콘솔 테트리스 → 2인용 컬러 GUI 테트리스
  - **소스 트리 설계** : deterministic 요소(데이터 모델)과 non-deterministic 요소(외부 인터페이스)로 나누어서 설계
  - **데이터 모델 설계** : 시나리오들의 집합화(결과물: 순서도)와 시나리오 연산화(결과물: 객체간 연산 정의) 위주로 구상
  - **데이터 모델 코딩** : 단순 시나리오 → 시나리오 확장
    - 단순 시나리오 (객체 하나 + 시나리오 하나) → 확장 시나리오 코딩 (객체 다수 + 시나리오 다수)
    - 데이터 모델의 단순성을 추구해야 함

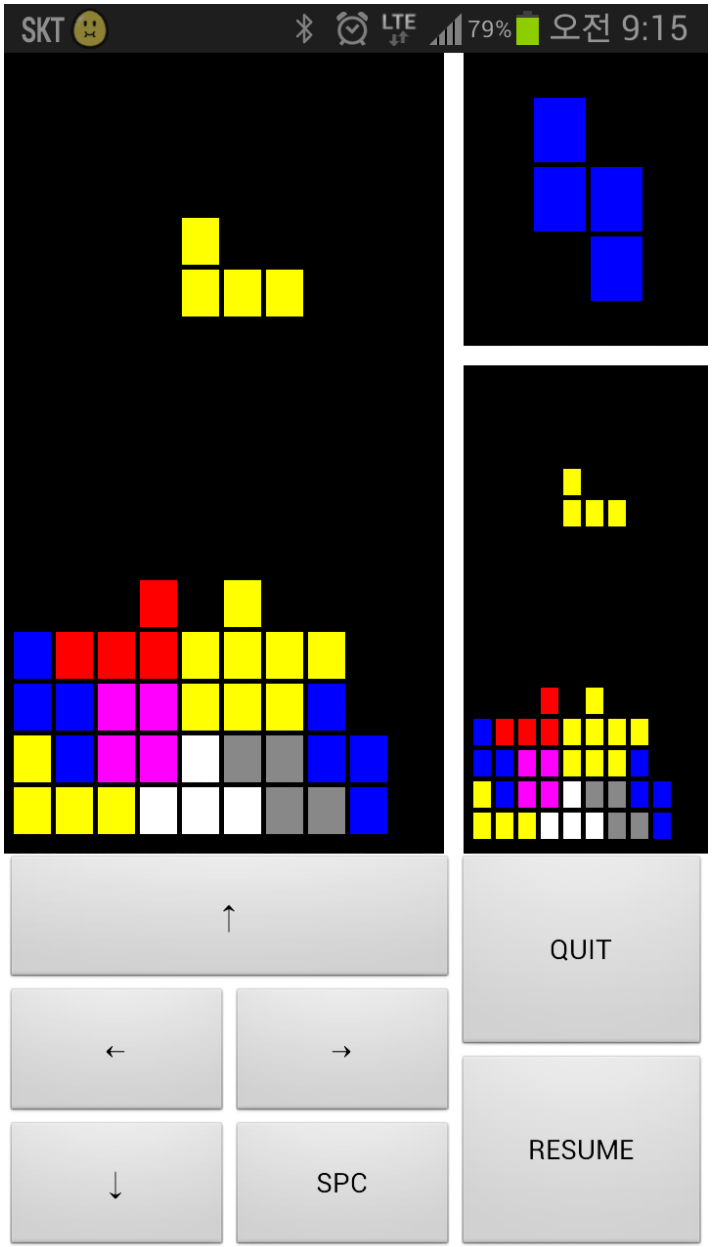
# 테트리스 프로젝트 로드맵 설계

- 콘솔 환경
  - 1인용 흑백 테트리스
  - 1인용 흑백 테트리스 with a duplicated screen
- GUI 환경
  - 1인용 컬러 테트리스 with a duplicated screen
  - 1인용 컬러 테트리스 with Echo server
    - 멀티 쓰레딩 + 소켓
  - 2인용 컬러 테트리스 with Tetris server
    - Tetris Server 프로그래밍 필요



# 테트리스 프로젝트 소스 트리 설계

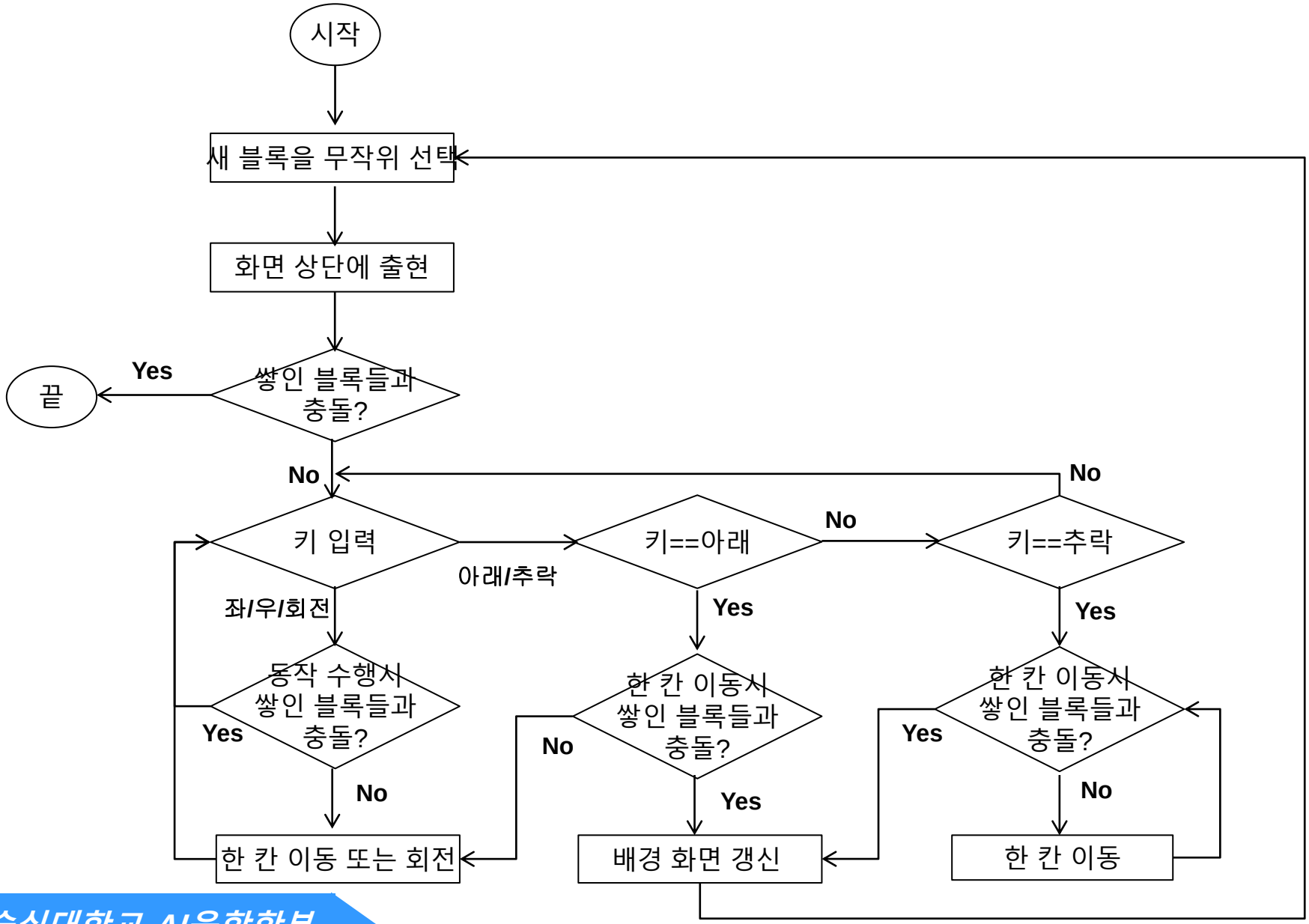
- 시스템 측면 (non-deterministic)
  - 키 입력, 화면 출력, 타이머 구동, 난수 발생 등
- 알고리즘 측면 (deterministic)
  - 블록 출현 후
    - 이동: 좌, 우, 아래, 추락
    - 회전(90도)
  - 블록 충돌
    - 좌/우 충돌
    - 아래/추락 충돌
    - 회전 충돌
  - 행 삭제
  - 배경 화면 갱신 및 신규 블록 출현



# 테트리스 데이터 모델 설계

- **시나리오 연산화** : common scenario 고려 (결과물: 객체 연산의 정의)
  - 주요 객체들 구상 : 배경, 벽, 쌓인 블록들, 내려오는 블록, …
  - 객체 추상화 : 서로 다른 성격의 객체들도 가능한 한 동일한 클래스로 취급할 수 있도록 클래스를 정의 (예: 배경, 벽, 7가지 블록들 → 행렬)
    - 이러한 '공격적인' 추상화가 시나리오 코딩에 미치는 영향을 추후에 신중히 검토해야 함!!
  - 객체 단순화 : 추상화된 클래스가 너무 복잡한 속성들로 정의되지 않도록 최대한 속성을 단순화 (예: 컬러 블록 → 흑백 블록)
- **시나리오들 집합화** : 모든 시나리오 고려 (결과물: 순서도)
  - 가능한 시나리오들을 순서도 형태로 열거한다.
  - 각 시나리오를 선택하여 가급적이면 동일한 타입의 객체의 연산으로 표현한다.

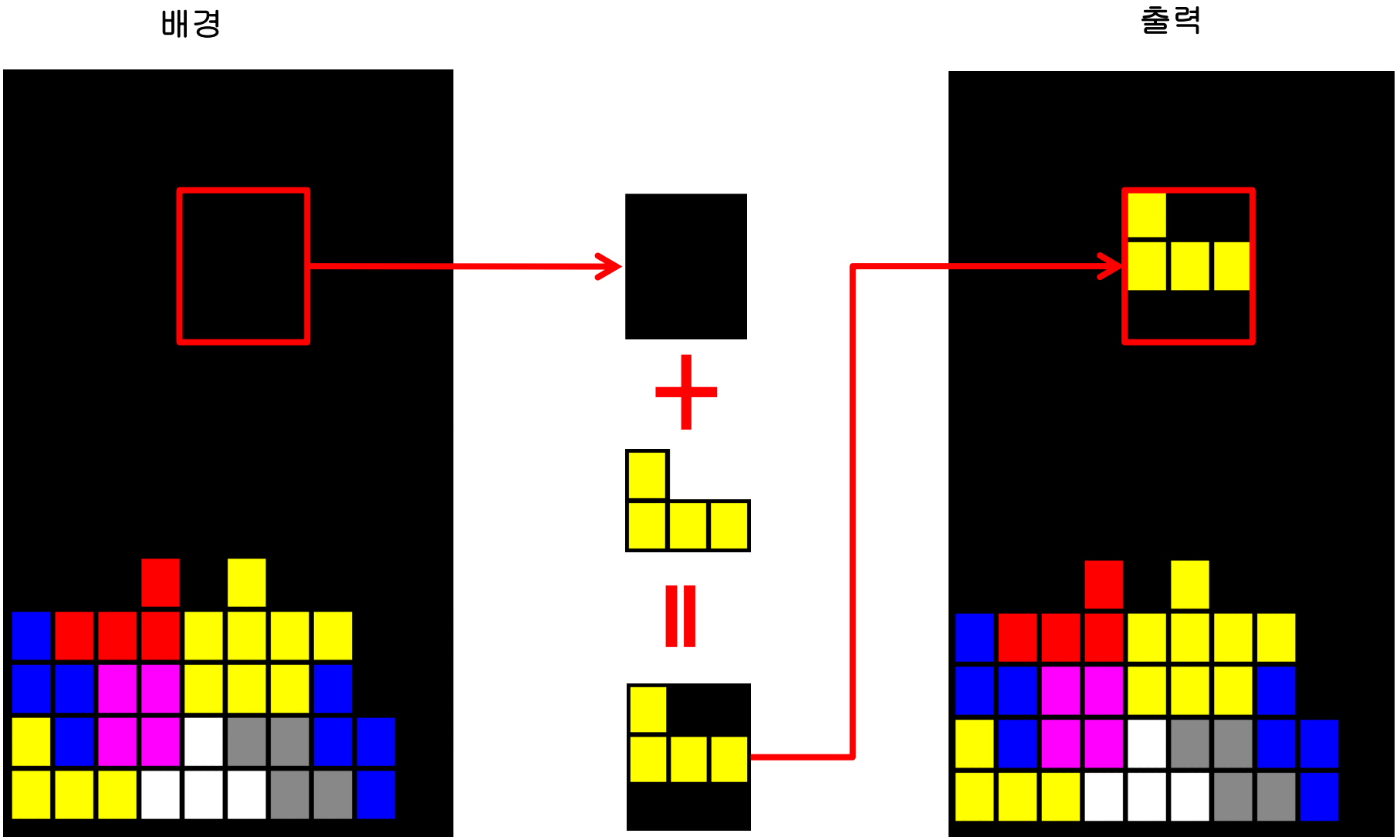
# 테트리스 데이터 모델 설계: 순서도



# 테트리스 데이터 모델 코딩

- **단순 시나리오 코딩** : Matrix class를 이용함
  - 콘솔 입출력(사용자 인터페이스)은 시스템 측면의 코딩이므로 matrix.py 파일과 분리된 pytet.py 파일로 작성함
  - 난수 발생 또는 키 입력과 같은 non-deterministic 요소는 고정된 값을 갖는다고 가정함
  - 데이터 모델 설계의 초기 단계부터 콘솔 프린트(print)를 이용하여 모델을 구상, 보정하는 것은 필수적임
- **확장 시나리오 코딩** : Matrix class의 메쏘드들을 최대한 활용함
  - 추상화된 Matrix 객체들로 모든 시나리오가 해당 객체들의 연산들로 표현 가능한지를 실제로 코딩하면서 검증함 → 그 결과 만들어진 것이 Matrix 클래스임
    - 단순 시나리오 : 객체 하나 + 시나리오 하나
    - 확장 시나리오 : 객체 다수 + 시나리오 다수
  - 데이터 모델의 단순성을 추구해야 함!!
    - Matrix 객체를 이용하여 배경, 벽, 쌓인 블록, 떨어지는 블록을 모두 표현할 수 있고, 그들 사이의 연산들을 몇 가지로 정의할 수 있어야 함
    - 객체를 반복 사용함으로 인해서 버려지는 객체들이 발생할 수 있으며, 이로 인한 메모리 낭비는 데이터 모델 설계 단계에서 코딩의 생산성을 위해서 감수할 수 있음

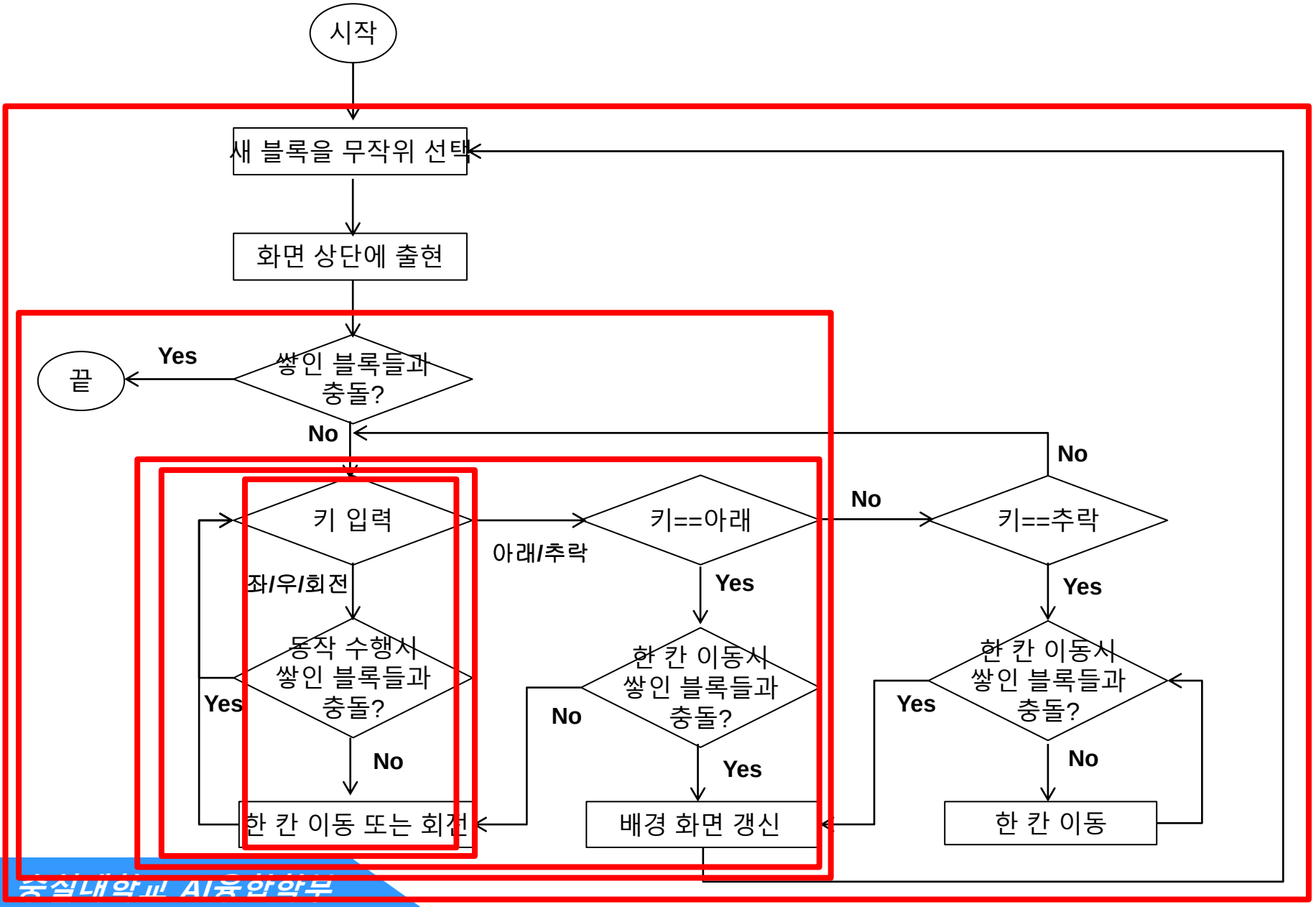
# 대표 시나리오를 수식화



# 테트리스 데이터 모델 코딩: 대표 시나리오

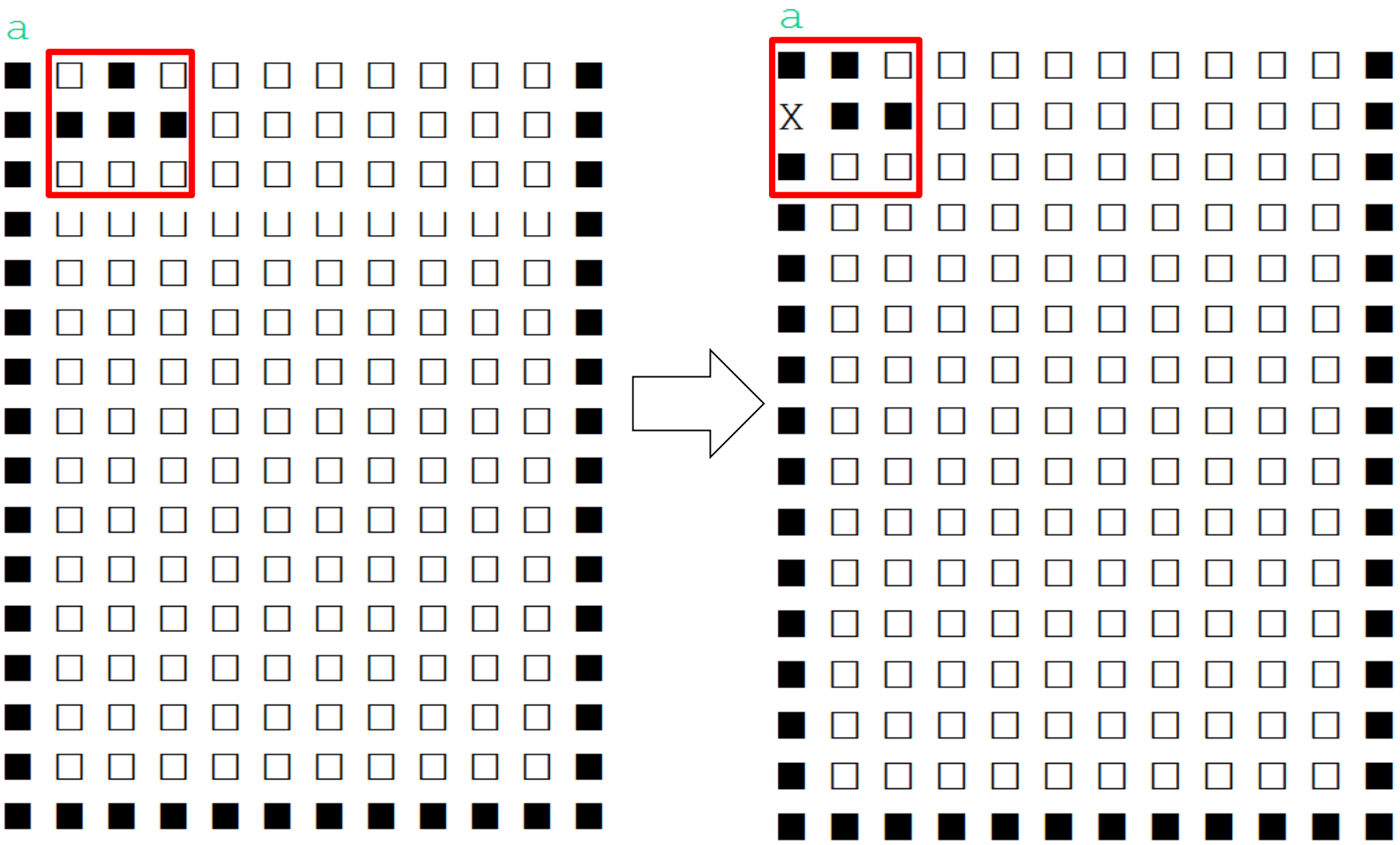
```
top = 0;  // 상태 변수
left = 4;  // 상태 변수
iScreen = Matrix(20,10); // 상태 변수
currBlk = Matrix(arrayBlock); // 상태 변수

tempBlk = iScreen.clip(top, left, top+currBlk.dy, left+currBlk.dx); // 임시변수
tempBlk = tempBlk + currBlk; // tempBlk.add(currBlk);

oScreen = Matrix(iScreen); // 상태 변수
oScreen.paste(tempBlk, top, left);
```

# 테트리스 데이터 모델 코딩: 안쪽에서 바깥쪽으로

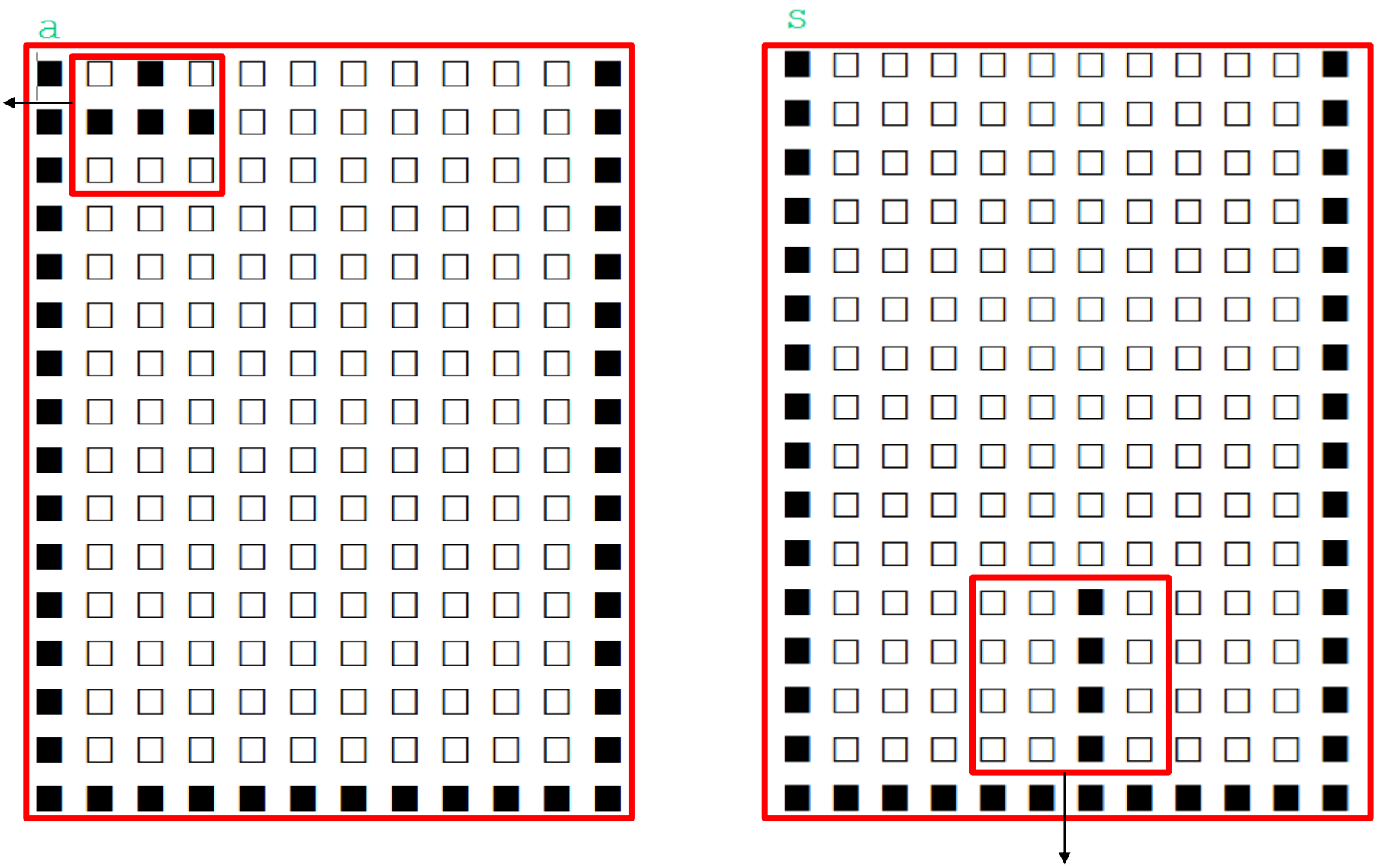


# Pytet (= Tetris in Python)

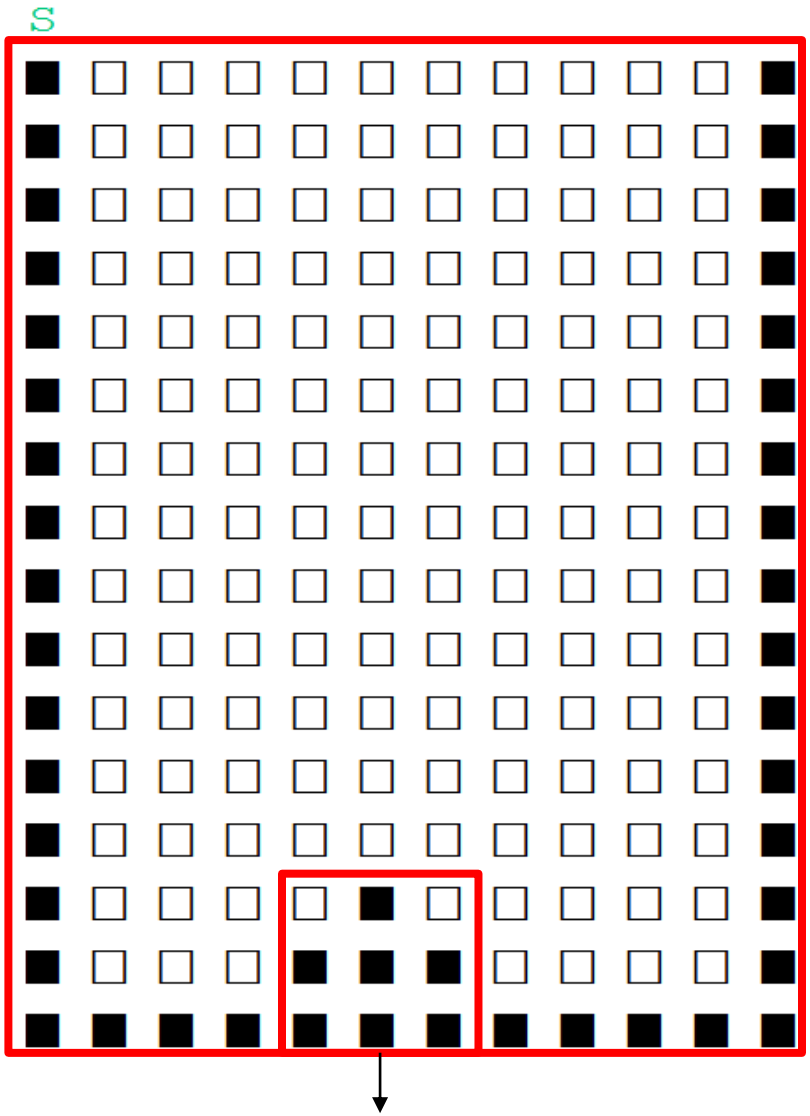
# 벽이 두꺼워야 하는 이유: 두께 1인 벽에 충돌하면?

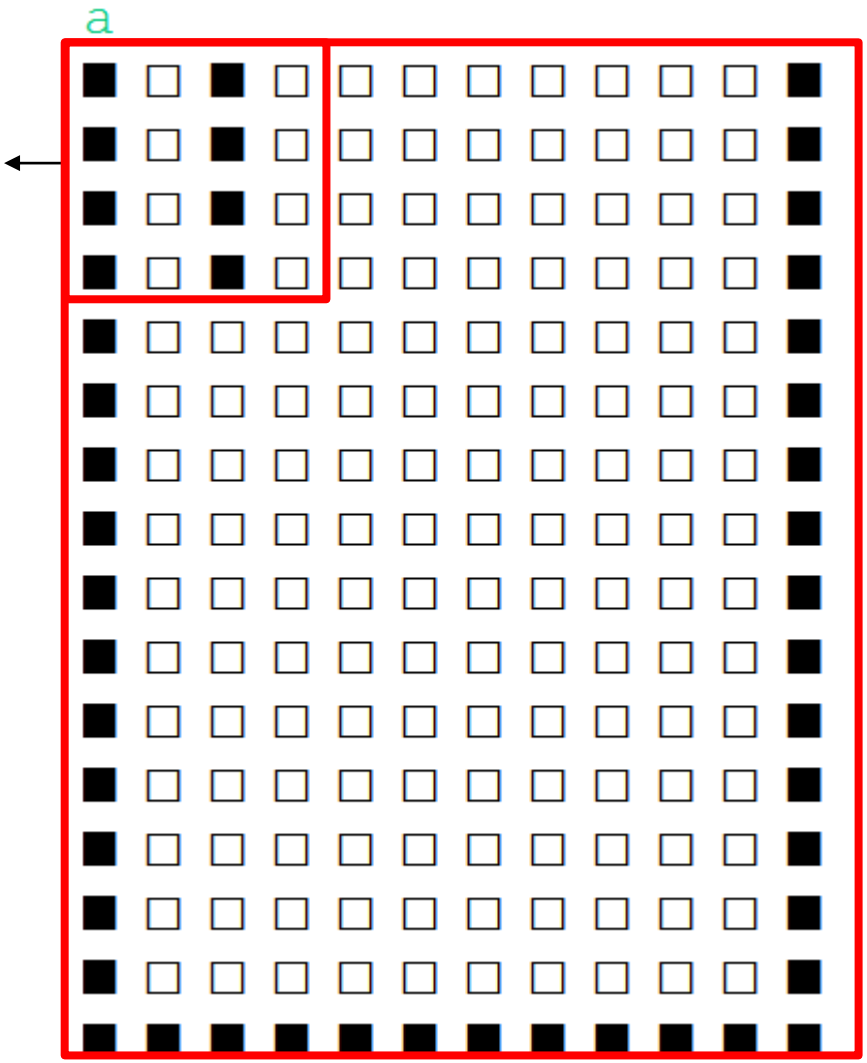


# 벽이 두꺼워야 하는 이유: 블록 경계에서 충돌하면?

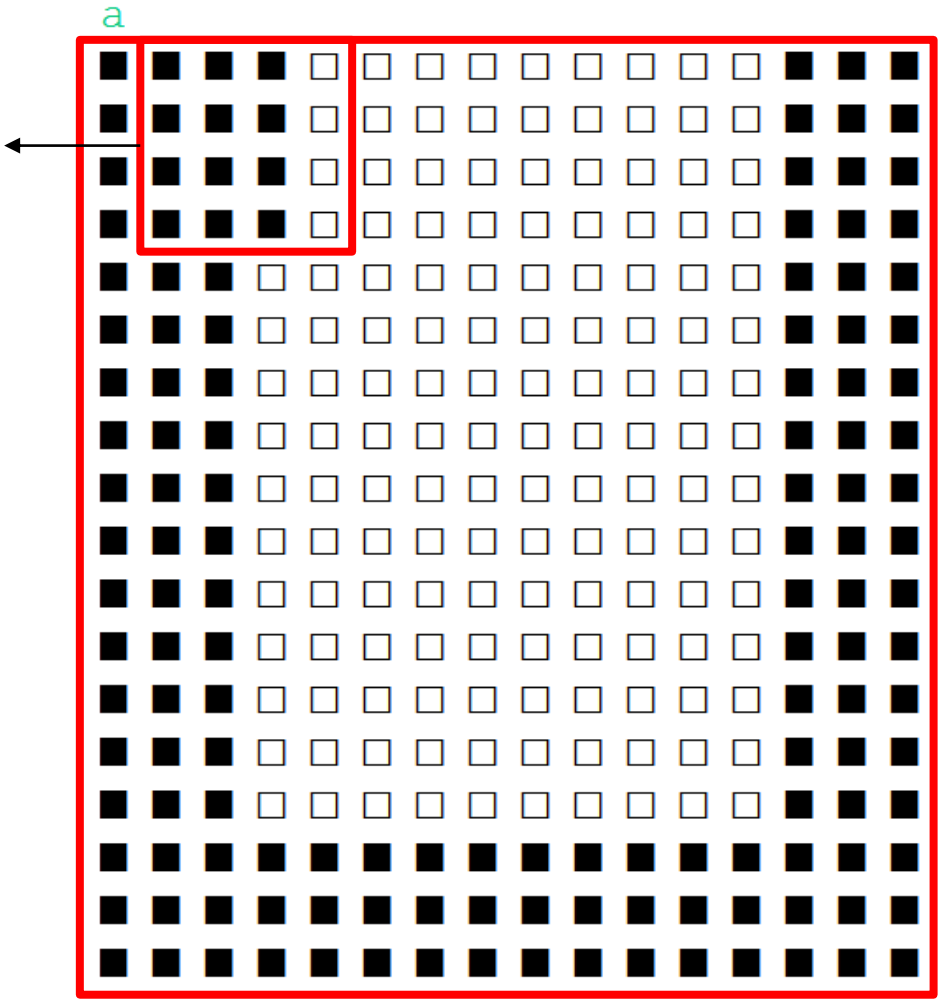


# 벽이 두꺼워야 하는 이유: 블록 내부에서 충돌하면?



# 임의 블록 형태를 수용할 수 있을 만큼 벽이 두꺼워야 함



# 코딩 지침들

- **기능 위주의 코딩을 먼저 해라**.
  - 기능이 완성될 때까지는 성능 최적화를 걱정하지 마라.
  - 컴퓨터 자원(메모리, CPU)을 사용함으로써 코딩이 단순해지면 자원을 사용하라. → 코드 표현력이 커지는 쪽으로 코딩해라.
    - 참고로, 메모리 자원보다는 CPU 자원이 더 소중하다.
- **불필요한 조건문들을 없애라**.
  - 조건문이 많아지면 버그들도 많아진다.
  - 경계 조건을 강제하는 조건문들은 sentinel key 로 없앨 수 있다.
  - 사전 테스트보다 사후 테스트가 유리한 경우도 있다.
  - if – elif – elif – … – else 구조는 가급적 사용하지 않는다.
- **반복되는 코드 조각을 없애라**.
  - 가능하면 하나의 함수로 작성하라.
  - 함수 작성시 "출력=동사(입력1, 입력2, …)" 형태로 정의하라.
  - 함수는 그 로직이 종속되는 파일 안에 코딩하라.
    - 테트리스 로직을 라이브러리 안에 코딩하지 마라.