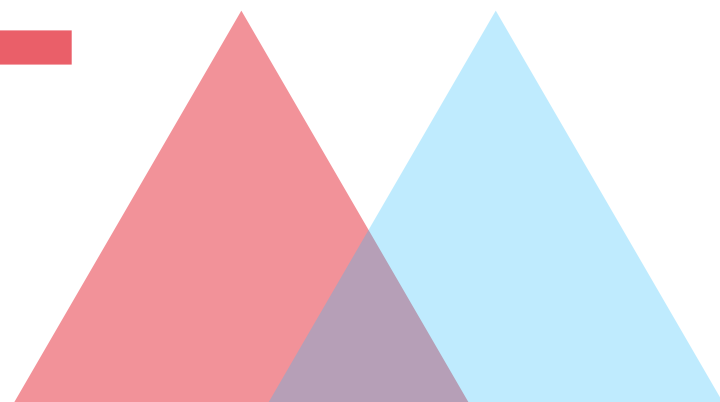


HIDATO PUZZLE

15조



조영선 우승민 최승호 심재욱 박지선 최인정

Contents

1 역할분담 및 진행상황 - 역할분담, 프로젝트 진행상황

2 프로젝트 구성

3 클래스 구조 및 세부설명 - GENERATOR, SOLVER

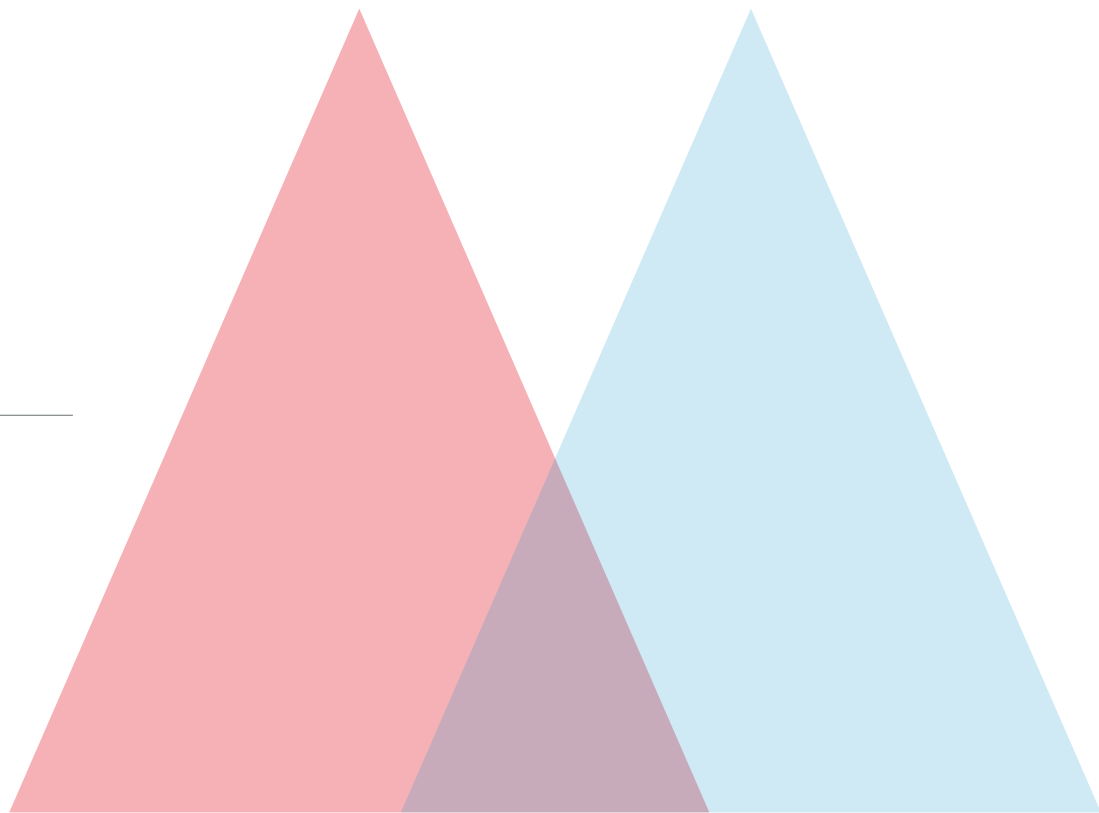
4 개선사항 - 개선 대상, 문제점, 개선 방법

5 실행결과 및 시연

6 Github 주소 및 참고자료

001

역할 분담 및 진행상황



〈 역할 분담 〉

조영선

팀장, 코드 모듈화
설계 및 구현, PM

최승호

외부 코드 분석, 주식 관리,
solver & generator 구현

우승민

solver & generator 세부 구현,
입출력 구축, 코드 형식 통합, ppt

심재욱

solver & generator 세부 구현,
알고리즘 개선, test, 발표

최인정

코드 분석, 프로젝트 계획서 작성,
중간 및 최종 보고서 작성

박지선

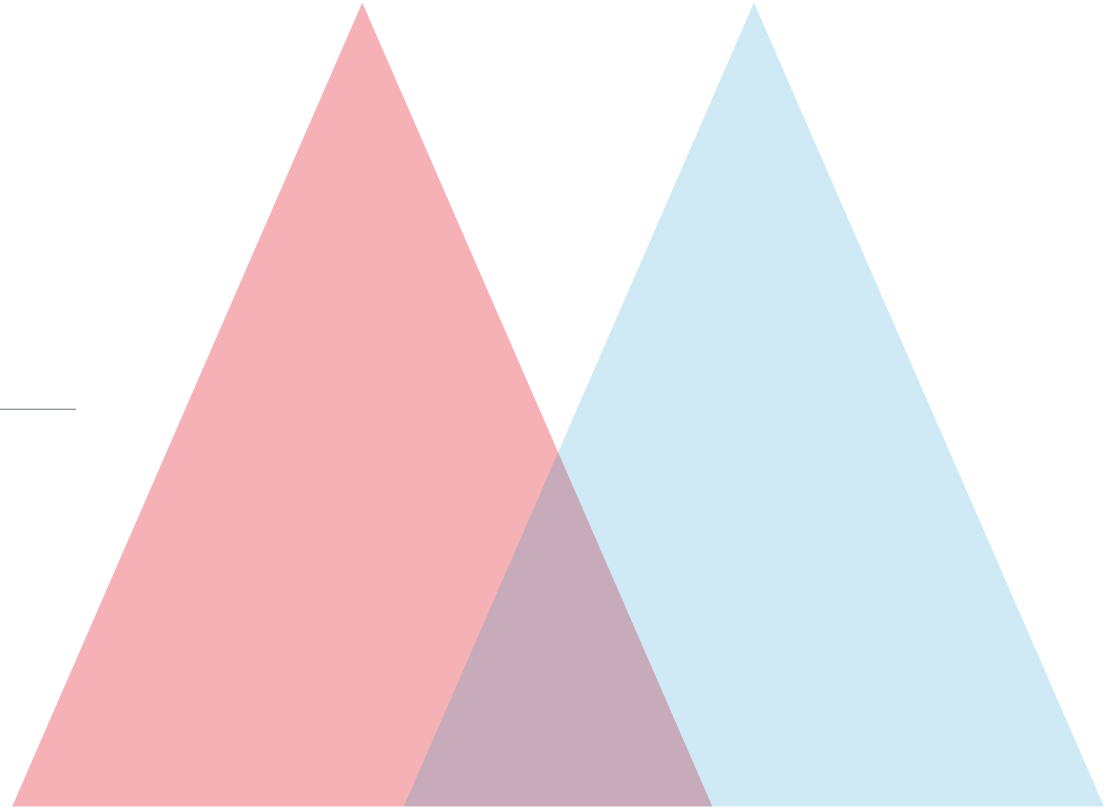
회의록 작성, 발표 ppt 제작,
피드백정리, 참고자료 수집

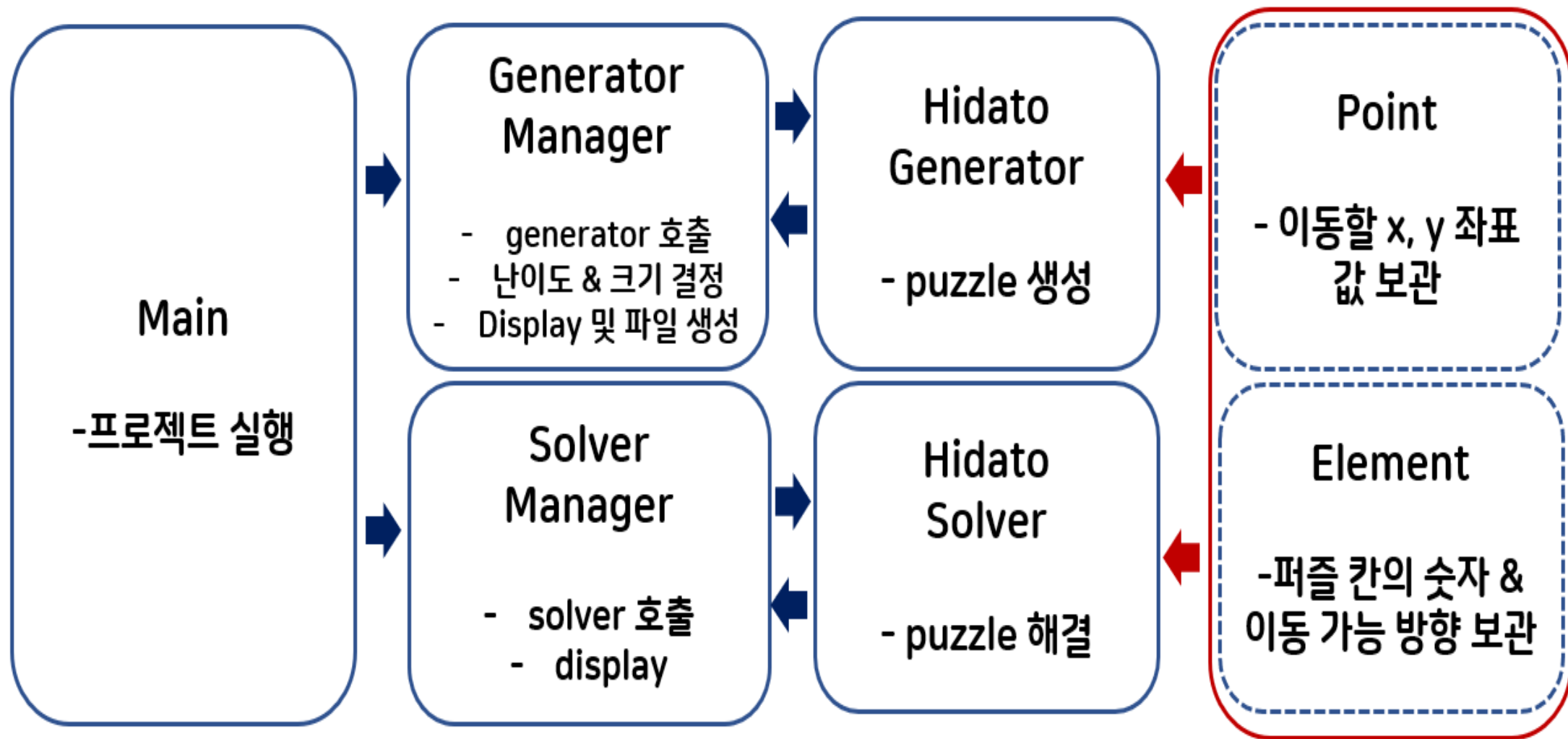
〈 프로젝트 진행상황 〉

1주차 (11.02 ~ 11.06)	SOLVER TASK 분석 및 구조 설계
2주차 (11.07 ~ 11.13)	SOLVER 1차 개발 및 디버깅
3주차 (11.14 ~ 11.20)	SOLVER TEST & 알고리즘 개선 중간 점검 및 보고서 작성
4주차 (11.21 ~ 11.27)	GENERATOR TASK 분석 및 구조 설계
5주차 (11.28 ~ 12.03)	GENERATOR 1차 개발 완료
6주차 (12.04 ~ 12.13)	GENERATOR TEST & 알고리즘 최적화 프로젝트 마무리 및 최종 보고서 작성

002

프로젝트 구성





003

클래스 구조 및 세부설명



1. GeneratorManager

사용자로부터 받은 정보를 HidatoGenerator에게 전달

생성된 히다토 퍼즐을 .txt 파일형태로 저장

생성된 히다토 퍼즐을 화면에 출력

saveInputFile

만들어진 히다토 퍼즐을
input.txt 파일에 저장

printWall

히다토 퍼즐 크기에 맞춰
주변 윤곽선 출력

displayPuzzle

히다토 퍼즐정보를 받아
화면에 출력

2. HidatoGenerator

generatorManager로부터 받은 정보를 기반으로 히다토 퍼즐 생성

setPuzzleSize

주어진 높이&너비 만큼의
빈 배열 할당

**Initialize
Puzzle**

빈 배열을 모두 벽으로
초기화
& 랜덤 시작점 지정

**getRandom
Diff**

난이도에 따라
퍼즐 내 숫자간 간격
범위 지정

makePuzzle

난이도를 고려한
히다토 퍼즐 생성

2-1. getRandomDiff

Easy



Normal



Hard



사용자 설정 난이도에 따라 퍼즐 내 공개된 숫자 사이의 간격을 조정하는 메소드

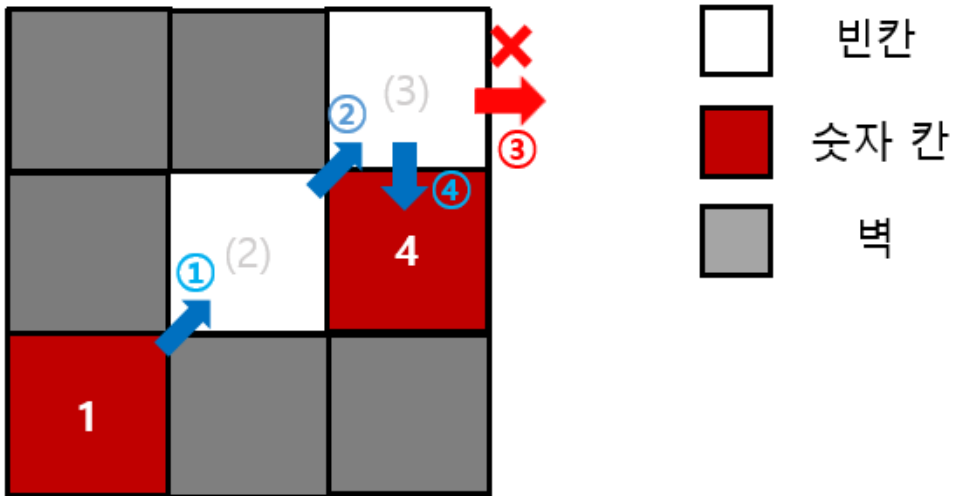
Easy인 경우 : 숫자 간 2~3 칸의 빈칸 생성

Normal인 경우 : 숫자 간 4~7 칸의 빈칸 생성

Hard인 경우 : 숫자 간 8~10 칸의 빈칸 생성

2-2. makePuzzle

Ex) Diff = 3



	초기	①	②	③	④
order	2	2	3	3	4
count	0	0	0	1	1

난이도에 따라 퍼즐을 생성하는 메소드

1. 퍼즐을 전부 벽(-1)으로 초기화(initializePuzzle)
2. 퍼즐 범위를 벗어나거나 이미 숫자가
배정된 칸으로 이동한 경우 count 증가
3. 숫자가 배정되지 않은 빈칸으로 이동할 경우
order 값 할당 후 증가
4. 숫자 간의 차이를 비교하여 빈칸(0)으로 표시

1. solverManager

main 함수에서 넘겨받은 히다토 퍼즐에서 퍼즐 정보 추출

HidatoSolver를 호출

해결된 히다토 퍼즐을 화면에 출력

inputPuzz

.txt 형식의 input



1차원 배열 형태의
히다토 퍼즐/높이/너비 추출

printWall

히다토 퍼즐 크기에 맞춰
주변 윤곽선 출력

displayPuzzle

히다토 퍼즐정보를 받아
화면에 출력

2. HidatoSolver

SolverManager로부터 받은 정보를 기반으로 하는 히다토 퍼즐

Preprocessing

퍼즐에 주어진 숫자 정보 파악
& 퍼즐 구조 변환

findStart

퍼즐이 시작되는
처음 지점의 좌표 반환

solve

퍼즐의 시작점을 기준으로
search 호출

getNeighbors

현재 위치 기준
이동 가능한 방향 정보 반환

search

이동 가능한 방향 정보를 활용하여
recursive 하게 path 탐색

checkSolution

search 함수로 찾아낸
퍼즐 정보를 puzz에 반영

2-1. Preprocessing

int puzz[len]

3	0	0	0	1	9	12	20	0	-1	-1	-1
---	---	---	---	---	---	----	----	---	----	----	----

height

width



preprocessing

Element E_array[len]

--	--	--	--	--	--	--	--	--	--	--	--



Element

value

1	0	0	1	1	0	1	1	/
---	---	---	---	---	---	---	---	---

bool existed[len + 1]

+

X	1	0	1	0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---

퍼즐 길이 만큼의 element Array를 만들어준다

element는 위치의 해당하는 숫자 값과 8방향의 이웃 중 접근 가능한 이웃이 무엇이 있는지의 정보를 담고있다.



existed : 퍼즐에서 주어진 숫자들을 담고있는 어레이이다.

max : 퍼즐을 풀기 위한 퍼즐의 최대값(마지막 값)을 loop를 돌며 저장한다.

2-2. findStart

12	0	0	7	1	0	20	0	-1	-1	-1	-1
----	---	---	---	---	---	----	---	----	----	----	----

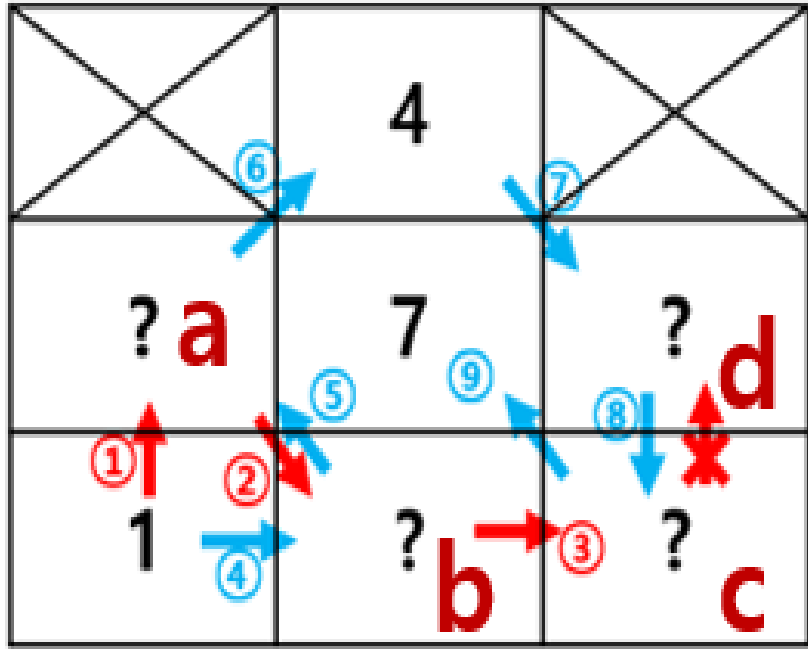
i $len-i$

퍼즐의 처음 시작지점인 1을 퍼즐 내부에서 찾기 위한 메소드

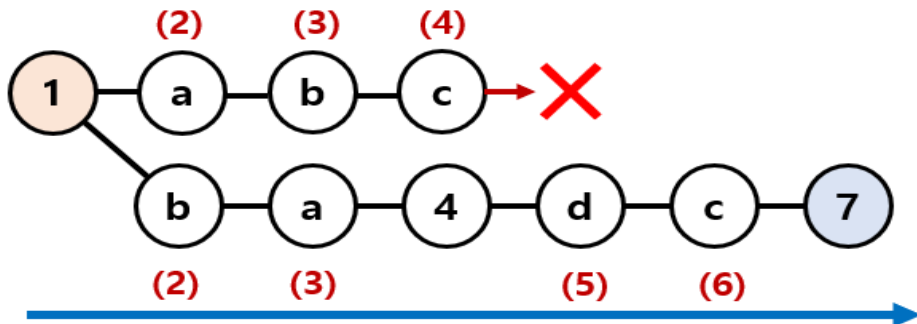
칸의 value가 1인 경우, 해당 칸의 x, y 좌표 할당 후 true 반환
1이 아닌 경우, false 반환

2-3. Search

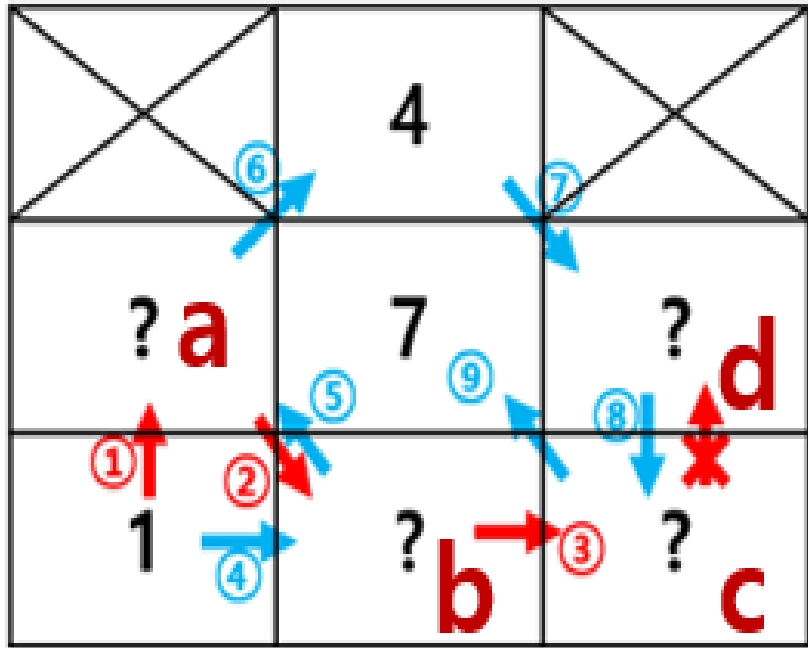


현재 위치하는 퍼즐의 좌표와
찾을 숫자 값을 받아 DFS 방식으로
재귀 호출하여 퍼즐의 해답을 만드는 메소드

getNeighbors()를 호출하여 현재 좌표에서
8개의 방향 중 이동 가능한 방향 정보 생성 후
이웃 칸을 순회하며 진행

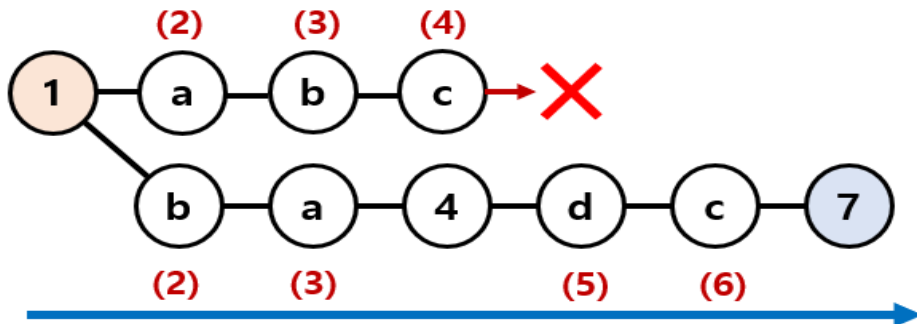


2-3. Search

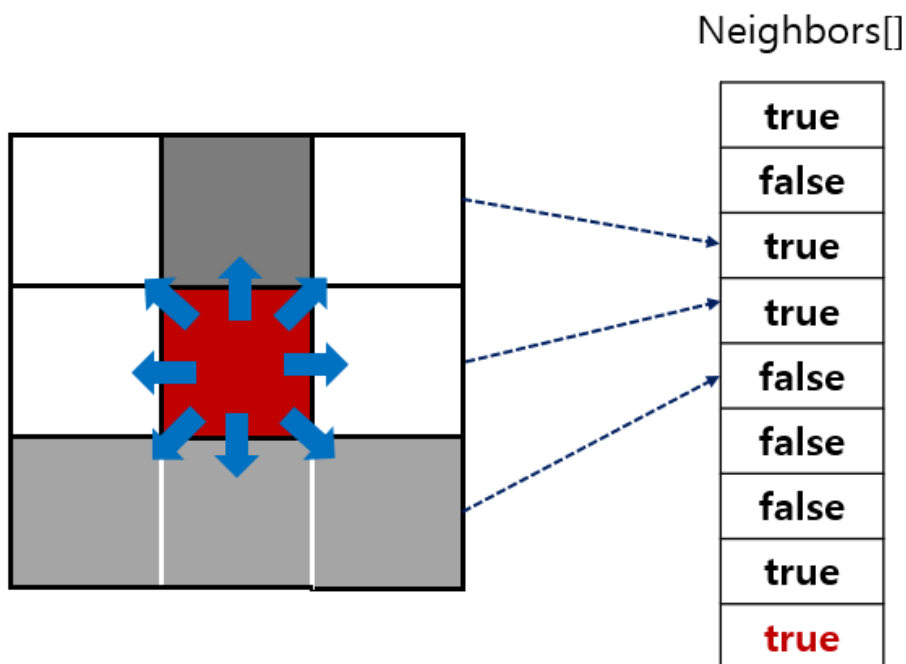


현재 찾을 숫자가 처음에 주어진 퍼즐에 존재할 때
 찾던 숫자를 이웃 칸에서 발견한 경우
 → 찾을 숫자에 +1 하여 다시 search() recursive call
 발견하지 못한 경우 → false 반환

현재 찾을 숫자가 처음에 주어진 퍼즐에 존재하지 않을 때
 → 현재 빈 칸인 이웃의 value를 현재 찾는 order로 가정
 → 찾는 숫자를 +1 하여 search() 호출
 → 호출한 search()가
 true를 반환 : 가정이 맞다고 판단하여 계속 진행
 false를 반환 : 가정이 틀리다 판단하여 가정을 recover



2-4. getNeighbors



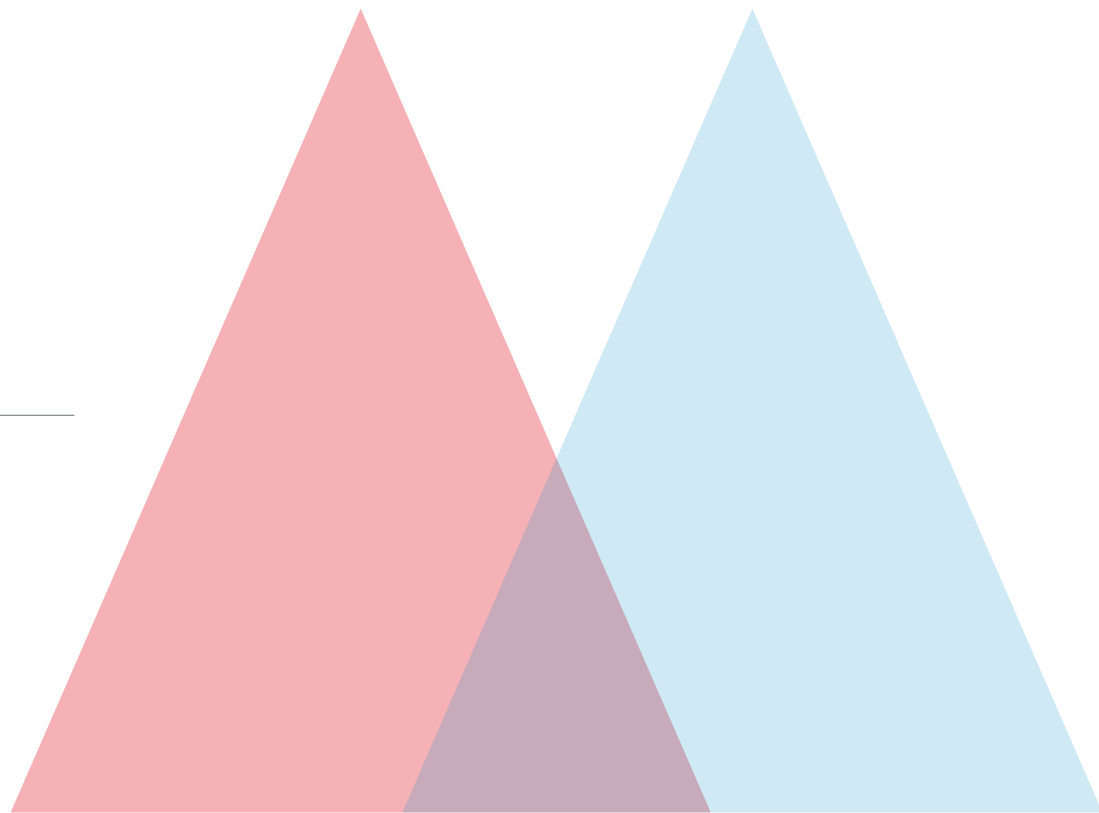
해당 칸에서 이동할 수 있는 칸의
방향정보를 얻는 메소드

현재 칸을 이웃하는 칸으로 이동했음을
가정한 후에 이동 가능 여부 체크

만약 이동한 칸이 빈칸 or 숫자인 경우
해당 element의 neighbors[index]
값을 true로 변경

004

개선사항



1. 개선 대상 〈 GENERATOR 〉

1) 퍼즐생성(makePuzzle)의 size 문제

퍼즐 생성 단계에서 인접한 8칸 중 이동 가능한 칸이 없는 경우,
목표한 너비와 높이를 만족하지 않는 작은 크기의 퍼즐만이 생성됨

2) 퍼즐생성(makePuzzle)의 max값 표기

퍼즐의 마지막 숫자가 난이도 조절을 위한 숫자 간격에 의해 퍼즐에 표기 되지 않아,
퍼즐의 최대 크기를 solver가 파악할 수 없음

1. 개선 방법 〈 GENERATOR 〉

1) 퍼즐생성 size

벽에 부딪히거나 이미 숫자가 들어있어 이동이 불가능한 경우에 대해 발생 횟수를 저장 하여, 해당 숫자가 허용 범위를 초과한 경우 퍼즐생성 종료 퍼즐 생성 종료 후, 완성된 퍼즐의 크기가 목표한 퍼즐 크기의 반보다 작은 경우, 퍼즐 생성을 실패하였다고 가정하고 새로 퍼즐 생성

2) 퍼즐생성 max값 표기

현재 숫자인 order가 max 값이 되었을 때 숫자간 간격에 상관없이 무조건 값을 퍼즐에 표기하는 조건을 추가

1. 개선 대상 〈 SOLVER 〉

3) 이동 가능여부 탐색(search)

현재 숫자인 order가 목표한 max 값이 되었을 때, 직전 값이 있는 퍼즐과 마지막 값이 있는 퍼즐이 연결되었는지 여부를 판단하지 않고 성공으로 가정 후 종료

4) 시작지점 찾기(findStart)

퍼즐의 처음부터 끝까지 순차적으로 loop 하며 퍼즐을 탐색하는 방식으로 이 방법은 효율성이 떨어짐

5) 이동 가능여부 확인(search)

퍼즐을 구성하는 칸에 대한 getNeighbors() 실행 여부를 남기지 않아 경우에 따라 한 칸에 대해 getNeighbors()가 여러 번 실행됨

1. 개선 방법 〈 SOLVER 〉

3) 이동가능여부 탐색

현재 숫자인 order가 목표 값인 max에 도달하였을 때, 무조건 true를 반환하는 이전 방식에서 벗어나 직전 퍼즐 칸과 max 칸이 연결되었는지 판단하는 조건 추가

4) 시작지점 찾기

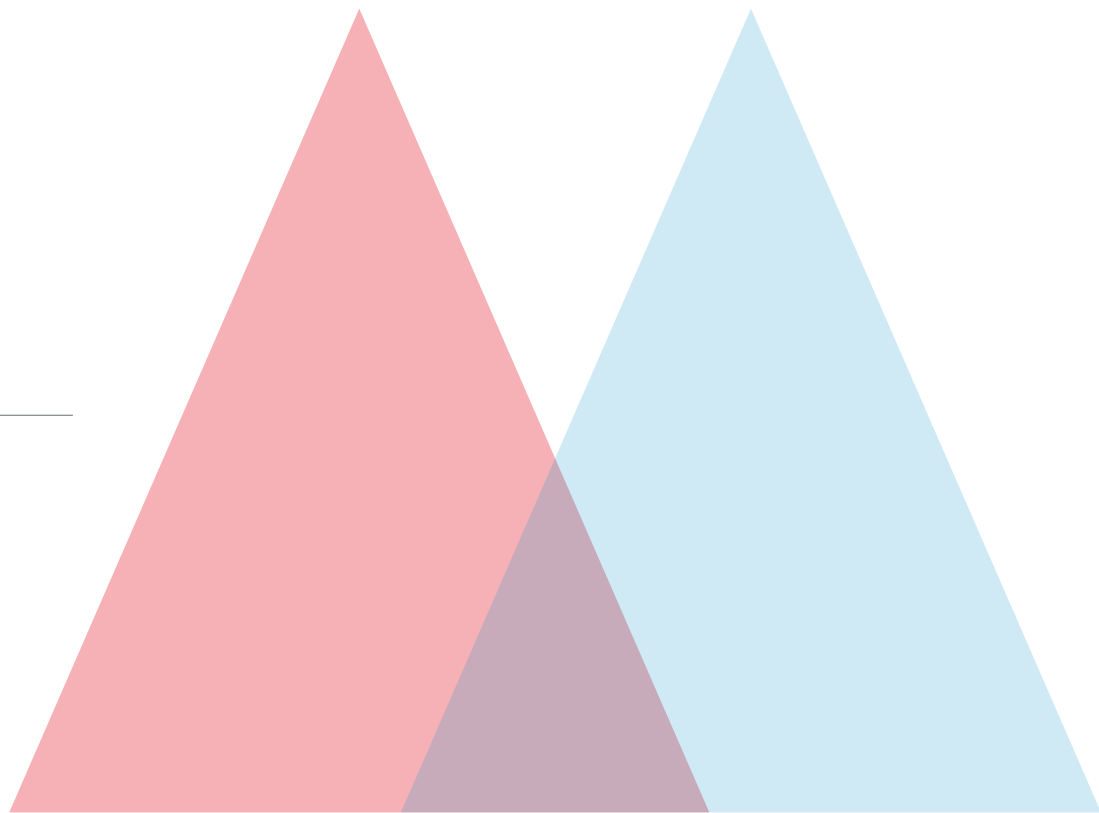
2개의 if 문으로 퍼즐의 처음과 끝을 동시 탐색하는 방식으로 반복문의 실행 횟수 감소시킴

5) 이동 가능여부 확인

방향 정보 외에 메소드 실행 여부를 bool 타입으로 저장하여 메소드 실행이 중복적으로 발생하는 것을 방지

005

실행결과 및 시연



```

=====Hidato Puzzle=====
*****Generator Start*****

          Select Difficulty
        Easy : 1, Normal : 2, Hard : 3

1
-----
          Enter The Number Of Width

5
-----
          Enter The Number Of Height

5
-----
          5 × 5 HidatoPuzzle Created

#####
#####
#####
##### ? ## 5 #####
#####
##### 7 ## ? #####
#####
## 9 ## ? ## 3 #####
#####
## 11 ## ? ##### ? ## 1 ##
#####

*****Solver Start*****

          Check Solution

#####
#####
#####
##### 6 ## 5 #####
#####
##### 7 ## 4 #####
#####
## 9 ## 8 ## 3 #####
#####
## 11 ## 10 ##### 2 ## 1 ##
#####
계속하려면 아무 키나 누르십시오 . . .

```

006

Github 주소 및 참고자료



Github 주소

https://github.com/YoungsunCho/Algorithm_HidatoPuzzle

참고자료 :

https://rosettacode.org/wiki/Solve_a_Hidato_puzzle

<https://github.com/fogleman/Hidato>

https://github.com/YoungsunCho/Algorithm_HidatoPuzzle/blob/master/reference/hidato-master.zip

https://github.com/YoungsunCho/Algorithm_HidatoPuzzle/blob/master/reference/solver.cpp