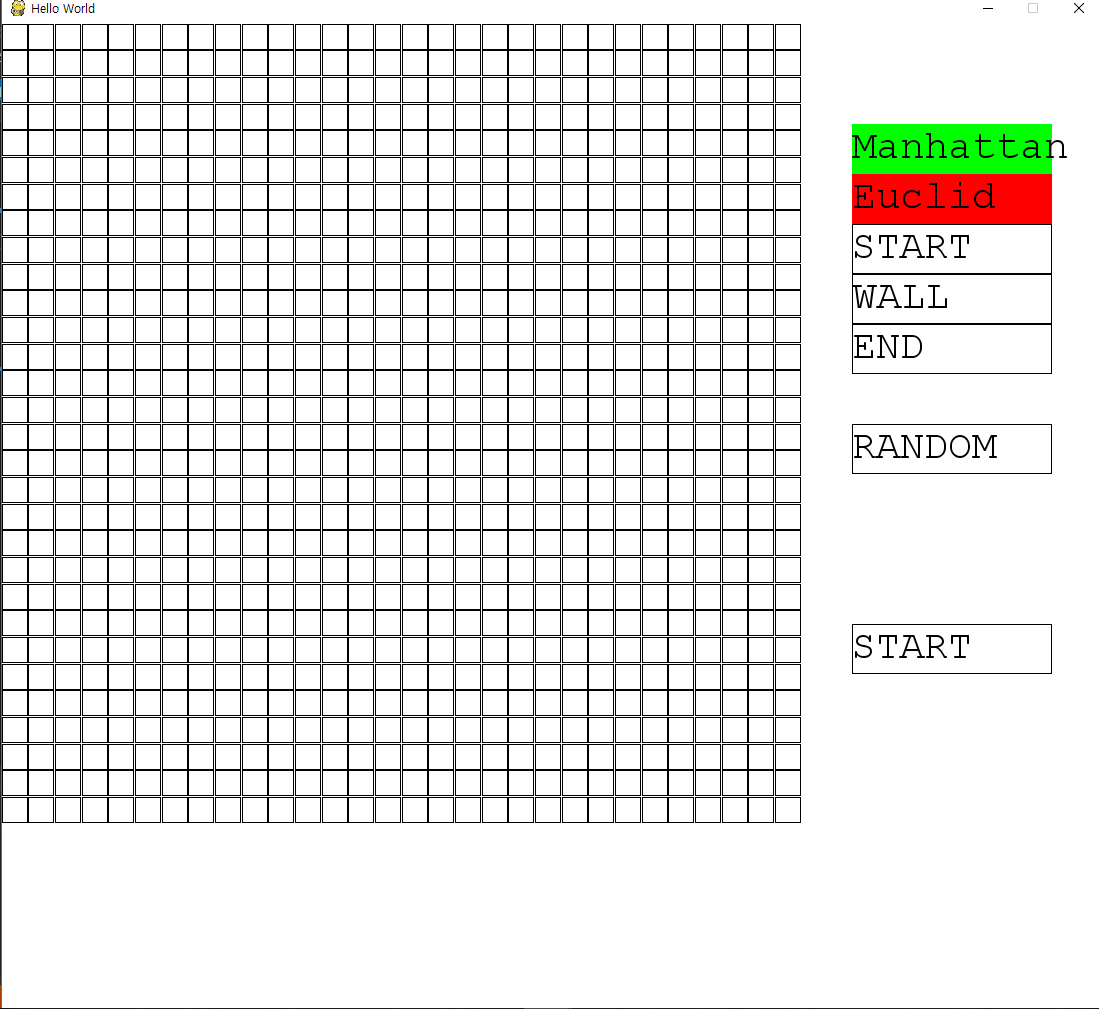
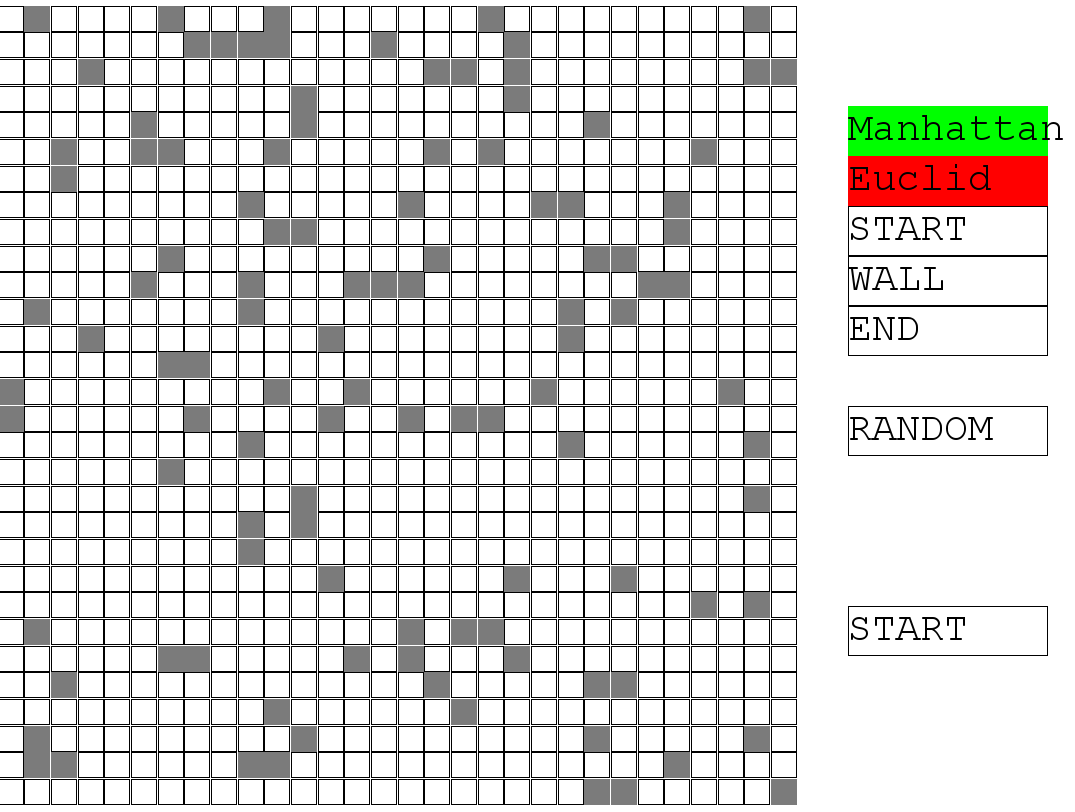
**20154649 강민수 AI\_Assignment 1,2,3 통합 보고서**

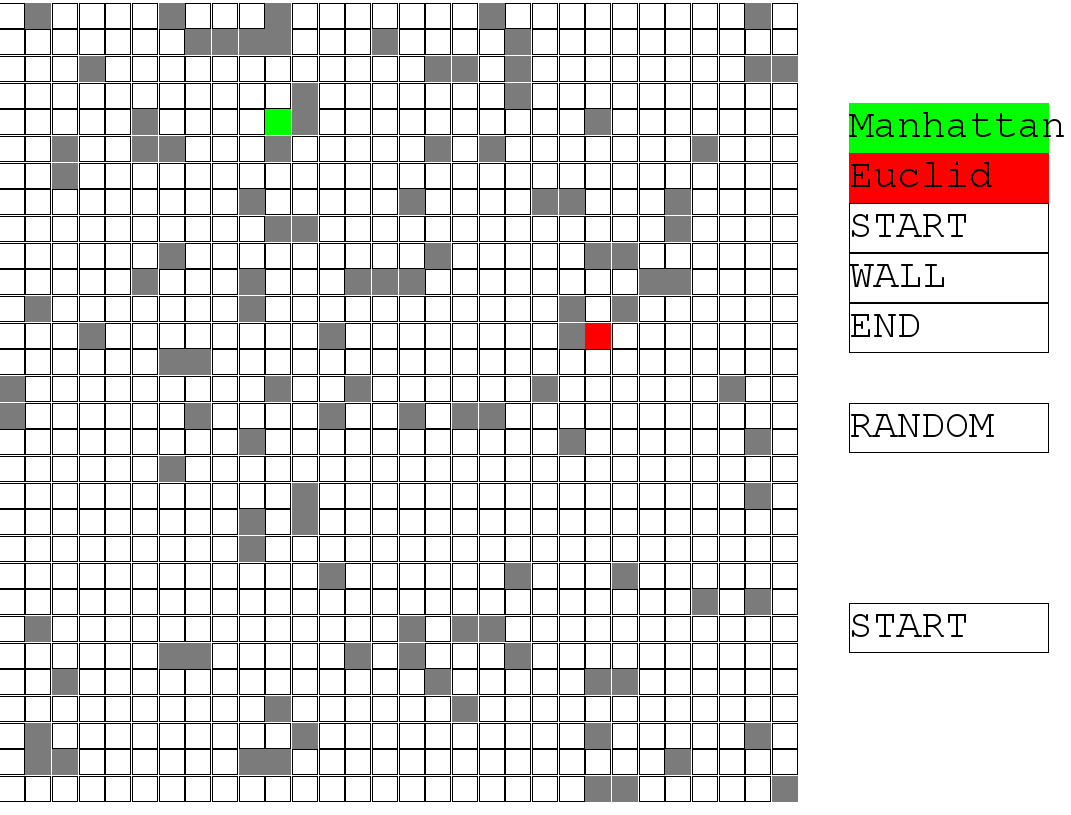
1. **A\*알고리즘을 이용한 길찾기 프로그램**
2. **Grid World크기 – 구현하짐 못하였습니다**
3. **Grid world 화면 출력 –**

****

1. **장애물 랜덤배치. RANDOM키를 누르면 장애물이 저절로 랜덤하게 배치됩니다.(120개 고정)**

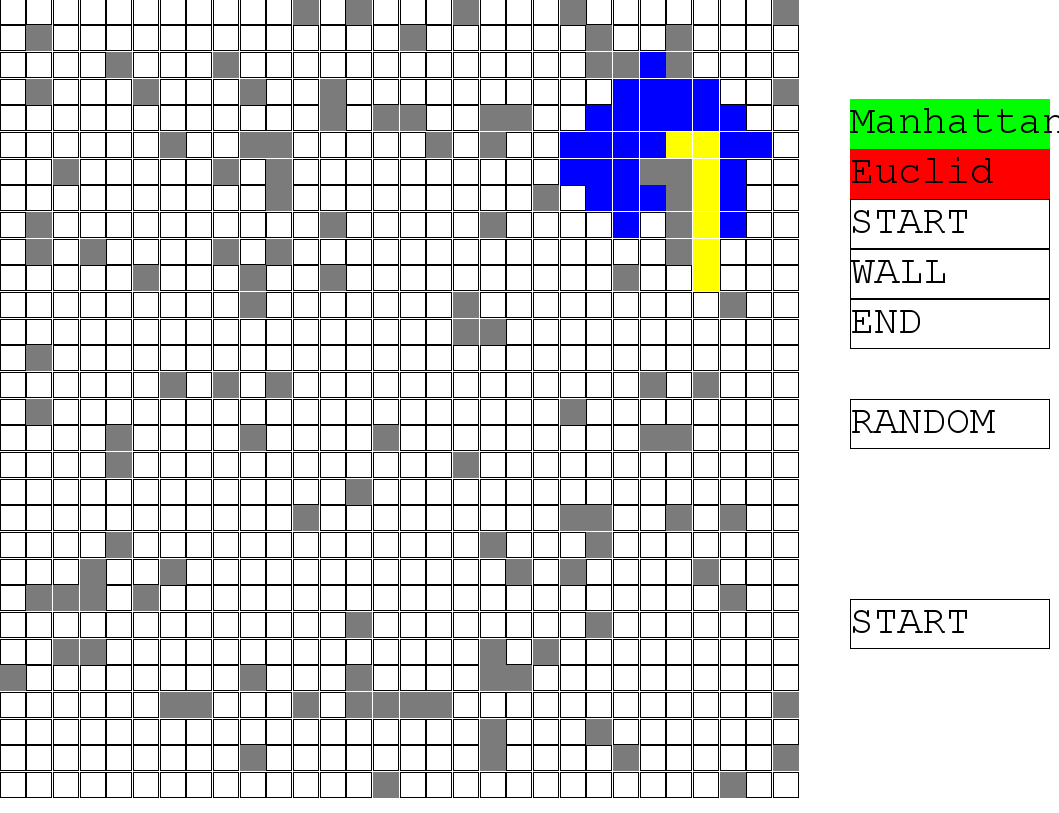
****

1. **시작점과 목표지점. START/END버튼을 누른 후 시작지점과 목표지점으로 설정할 구역에 클릭을 하면 설정됩니다.**

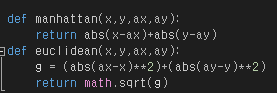
****

1. **A\*수행**

**START버튼을 누르면 콘솔창에 현재 탐색하고있는 부분의 좌표와 그 부모좌표가 나오면서 탐색을 시작합니다. 탐색을 완료하면 탐색했던 구역이 파란색으로 표시되고 탐색한 결과 가장 알맞은 경로가 노란색으로 표시됩니다.**

****

1. **휴리스틱 함수로는 맨해튼과 유클리드 함수가 있고 클릭하면 해당 함수의 버튼이 초록색으로 바뀌면서 함수가 다르게 적용됩니다.**

****

**맨해튼 함수와 유클리디안 함수**

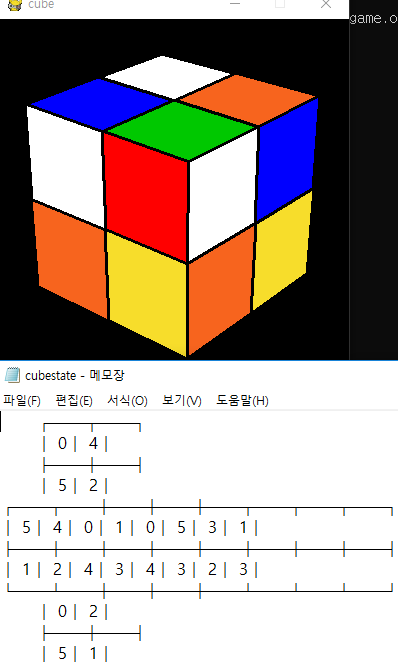
1. **해가 없을때는 콘솔창에 해를 찾지 못하였다고 출력합니다.**

****

**전체 코드는 따로 py파일로 동봉합니다.**

1. **2x2 루빅스 큐브**
2. **Minmove 구현 및 테스트 – 구현하지 못하였습니다**
3. **랜덤 초기화 및 파일로 저장.**

**프로그램 실행 후 키보드 ‘n’키를 누르면 12번을 랜덤으로 이동하고 그에따른 큐브의 상태가 텍스트파일로 저장됩니다.**

****

1. **IDA\*기반 알고리즘 제작**

**큐브를 섞은 후 키보드 오른쪽 숫자키패드에 9를 누르면 알고리즘이 동작합니다.**

**그 후 최적의 해를 콘솔창에 표시해 줍니다**

****

1. **GUI시뮬레이션 구현**

**모든 것을 GUI로 구현했으며**

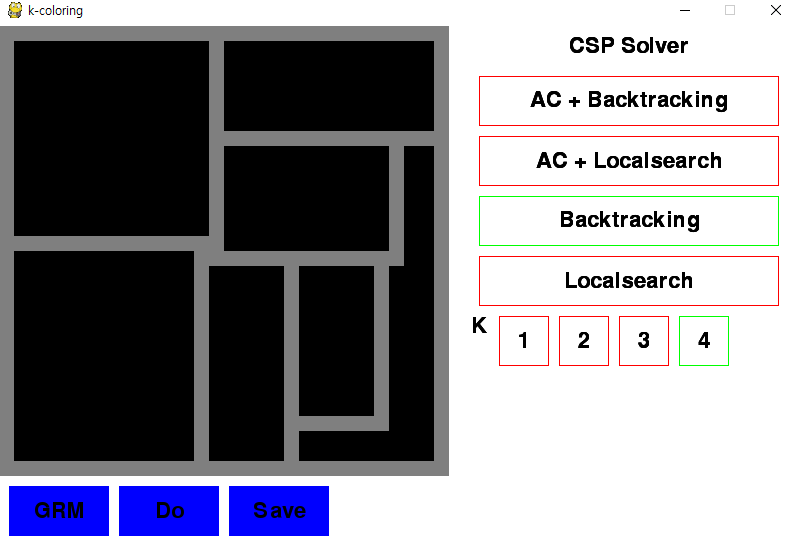
**오른쪽 키패드의 8->U,6->R,4->L,5->F,0->B,2->D 이에 맞게 동작합니다. 오른쪽 키패드의 1을 누르고 있으면 시계 반대방향으로 회전합니다.**

1. **전체 통합**

**오른쪽 키패드의 9를 누르면 최적의 해를 찾고 그에 따라 GUI큐브가 이동합니다. NEXT,STOP기능은 구현하지 못하였습니다.**

**소스코드는 따로 py파일로 동봉합니다.**

1. **K-Coloring**
2. **Random map 생성.**

****

**GRM 버튼을 클릭하면 랜덤맵이 생성됩니다. 여기서 가끔 경계선이 두줄로 생성될 때가 있는데, 이 때 는 두줄짜리 경계로 나뉘어진 지역은 서로 연결되지 않은것으로 취급됩니다.**

1. **인접리스트 그래프로 변환.**

**함수중 gusung()이라는 함수에서 실행됩니다. 함수는 다음과 같습니다. 최종적으로 graph라는 변수를 리턴합니다. 이 변수는 인접행렬로 그래프를 나타냅니다.**

def gusung():

kk=2

xj=0

xxx=0

for i in range(1,29):

for j in range(1,29):

xxx=0

xj=0

if map[i][j] == 0:

if map[i-1][j] != 1 and map[i-1][j] != 0:

xj+=1

xxx=0

if map[i][j-1] != 1 and map[i][j-1] != 0:

xj+=1

xxx=1

if xj == 0:

map[i][j] = kk

kk+=1

elif xj == 1:

if xxx==0:

map[i][j] = map[i-1][j]

else:

map[i][j] = map[i][j-1]

else:

if map[i-1][j] == map[i][j-1]:

map[i][j] = map[i-1][j]

else:

if map[i][j-1] < map[i-1][j]:

map[i][j] = map[i][j-1]

recon(map[i-1],[j],map[i][j-1])

else:

map[i][j] = map[i-1][j]

recon(map[i][j-1],map[i-1][j])

print(kk-2)

for i in range(30):

print(map[i])

mat = []

toc = 0

result = 0

for i in range(30):

for j in range(30):

toc = 0

if map[i][j] == 0 or map[i][j] == 1:

continue

for f in mat:

if map[i][j] == f:

toc = 1

break

if toc == 1:

continue

mat.append(map[i][j])

result += 1

print(result)

print(mat)

rut = []

for i in range(result):

rut.append(50+i)

print(rut)

for i in range(result):

recon(mat[i],rut[i])

for i in range(30):

for j in range(30):

if map[i][j] == 0 or map[i][j] == 1:

continue

map[i][j] -= 48

for i in range(30):

print(map[i])

graph = [[0] \* (result) for i in range(result)]

for i in range(29):

for j in range(29):

if map[i][j] == 0 or map[i][j] == 1:

continue

if i==28 and j == 28:

continue

elif i == 28:

if map[i][j] != map[i][j+2] and map[i][j+2] != 1:

graph[map[i][j]-2][map[i][j+2]-2] = 1

graph[map[i][j+2]-2][map[i][j]-2] = 1

elif j == 28:

if map[i][j] != map[i+2][j]and map[i+2][j] != 1:

graph[map[i][j]-2][map[i+2][j]-2] = 1

graph[map[i+2][j]-2][map[i][j]-2] = 1

else:

if map[i][j] != map[i][j+2] and map[i][j+2] != 1:

graph[map[i][j]-2][map[i][j+2]-2] = 1

graph[map[i][j+2]-2][map[i][j]-2] = 1

if map[i][j] != map[i+2][j]and map[i+2][j] != 1:

graph[map[i][j]-2][map[i+2][j]-2] = 1

graph[map[i+2][j]-2][map[i][j]-2] = 1

for i in range(result):

print(graph[i])

for i in range(result):

a = []

for j in range(result):

if graph[i][j] == 1:

a.append(j)

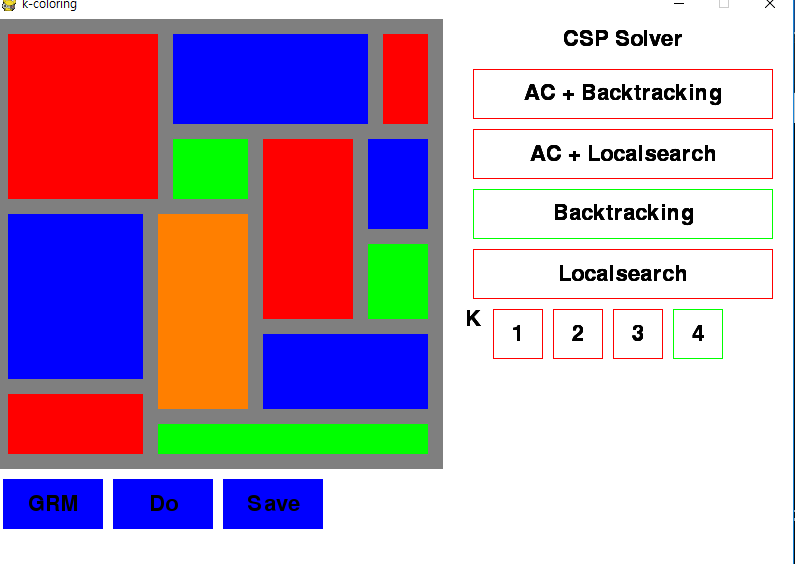
neighbor[i] = a

locals.append(i)

print(neighbor,locals)

return graph

1. **K-coloring 수행. Backtracking 버전만 구현이 가능했습니다. Arc-consistency는 구현했지만 backtraking과 합동해 구현은 실패했습니다.**

****

**소스코드는 py 파일로 같이 첨부합니다.**