|  |
| --- |
| 20172305 이주연  20175162 박건유  2019-12-15 |

|  |
| --- |
| 국민대학교 |
| 소프트웨어 프로젝트 Ⅱ |
| AD Project - Othello 게임 |

목차

**Ⅰ. 서론**3

**Ⅱ. 사항 명세서(SRS)**4

1. 기능적 요구사항4

2. 사용자 인터페이스 요구사항4

3. 비 기능적 요구사항4

**Ⅲ. 소프트웨어 구조 설계서(ADS)** 5

1. 소프트웨어 구조 설계(Architecture Design) 5

2. 클래스 인터페이스 설계5

3. 전체 구조7

**Ⅳ. 소프트웨어 상세 설계서**8

1.Othello.Py9

1) table9

2) clickableButton9

3) ifNearButton9

4) validMoveDirects9

5) keyPressedQ9

6) (R, L) Moved9

7) (U, D) Moved 9

8) reverse10

9) reverse\_btn10

10) gameover10

11) reset 10

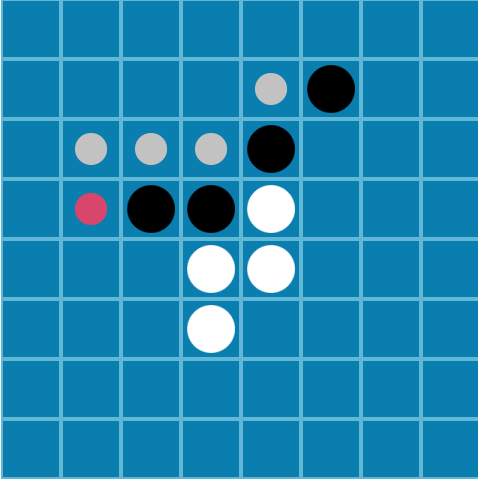
**V. 소스코드**11

**Ⅰ. 서론**

[Othello] 보드 게임의 한 종류이다.  리버시(Reversi)라고도 불린다. 두 명이 8 X 8칸의 Othello 판 위에서 한쪽은 검은색, 다른 한쪽은 흰색인 돌을 번갈아 놓으며 진행된다.

규칙은 다음과 같다.

* 처음에 판 가운데에 사각형으로 엇갈리게 배치된 돌 4개를 놓고 시작한다.
* 돌은 반드시 상대방 돌을 양쪽에서 포위하여 뒤집을 수 있는 곳에 놓아야 한다.
* 돌을 뒤집을 곳이 없는 경우에는 차례가 자동적으로 상대방에게 넘어가게 된다.
* 아래와 같은 조건에 의해 양쪽 모두 더 이상 돌을 놓을 수 없게 되면 게임이 끝나게 된다.
  + 64개의 돌 모두가 판에 가득 찬 경우 (가장 일반적인 경우)
  + 어느 한 쪽이 돌을 모두 뒤집은 경우
  + 한 차례에 양 쪽 모두 서로 차례를 넘겨야 하는 경우 (양쪽 모두 돌을 놓을 자리가 없는 경우)
* 게임이 끝났을 때 돌이 많이 있는 플레이어가 승자가 된다. 만일 돌의 개수가 같을 경우는 무승부가 된다.



[그림 1] Othello 결과 예시

본 문서에서는 [Othello] 게임을 Python과 PyQt5를 이용해 GUI(Graphic User Interface) 기반게임을 구현한 결과와 중간 산출물을 다룬다. 2명의 사용자가 게임을 진행하도록 구현되어 있다.

**Ⅱ. 사항 명세서(SRS)**

**1.기능적 요구사항**

본 게임의 핵심적인 요소는 [Othello] 게임 판에서의 사용자가 입력할 수 있는 위치를 출력하는것과 입력에 맞게 바뀐 결과를 보여주고, 사용자가 입력할 수 있는 위치를 다시 출력한다.

|  |  |
| --- | --- |
| 게임 | * 게임 종료시 결과를 통해 승자를 도출한다 * 자신이나 상대의 차례에 놓을 수 있는 돌의 위치를 출력한다. * 돌이 놓아져 있는 공간이나 돌을 놓을 수 없는 공간의 입력은 받을 수 없어야 한다. * 게임을 새로 하거나 초기화를 할 수 있어야 한다. |

[표 1] 기능적 요구사항

**2. 사용자 인터페이스 요구 사항**

인터페이스의 구성요소는 상단 디스플레이, 게임판, 하단 메뉴바로 구성이 되어 있다.

|  |  |
| --- | --- |
| 상단 디스플레이 | * 상단 디스플레이는 현재 순서가 누구인지, 흑과 백이 딴 돌들을 출력한다. |
| 게임 판 | * 게임판의 경우 현재 놓여 진 돌들과 현재 놓을 수 있는 돌의 위치, 입력으로 놓을 수 있는 위치 중에 선택된 위치를 출력한다. |
| 하단 메뉴 바 | * 새 게임, 도움말 출력 버튼 또는 종료 버튼을 출력한다. |

[표 2 ]사용자 인터페이스 요구사항

**3 비기능적 요구사항**

이 소프트웨어의 구현은 Python을 사용하고, PyQt5 패키지를 사용하여 GUI를 구현한다.

**III. 소프트웨어 구조 설계서 (ADS)**

**1.소프트웨어 구조 설계(Architecture Design)**

게임 구현 시 사용할 모듈은 Othello\_reversi.py이다. 모듈 내 각각 상호작용과 역할들을 정리한 것이 [표 3]이다.

|  |  |  |
| --- | --- | --- |
| 모듈 | 클래스 | 역할 |
| Othello\_reversi.py | Mainwindow | Game Window와 키보드이벤트 처리 |
| Reversi | 사용자 인터페이스와 로직 대부분을 구현 |

[표 3] 모듈 간의 상호작용

**2. 클래스 인터페이스 설계**

각 클래스의 메서드를 간략히 정리하였다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 클래스 | 메서드 | 입력인자 | 출력인자 | 기능 |
| MainWindow | initMW | - | - | 메인 윈도우  출력 |
| setChildrenFocusPolicy | Policy | - | 방향키 이벤트 처리하기위한 포커싱 |
| keyPressedEvent | Event | - | 발생한 키보드 이벤트 중에서 방향키와 Enter키를 처리 |

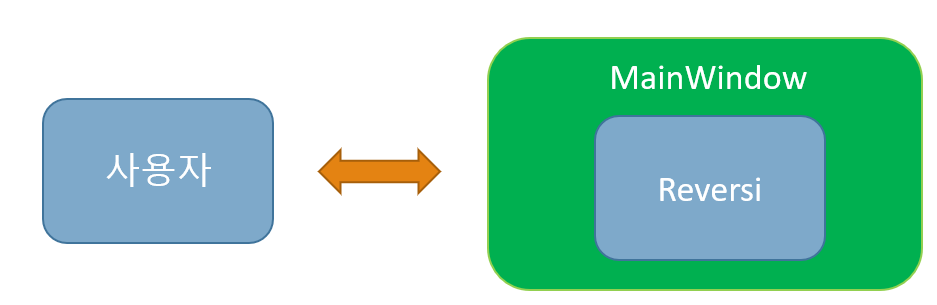
[표 4]클래스 인터페이스

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 클래스 | 메서드 | 입력인자 | 출력인자 | 기능 |
| Reversi | initRV | - | - | 상단 디스플레이, 게임 판, 하단메뉴를 초기화 |
| lists | - | - | 8x8 버튼에 대한 리스트와 돌의 색깔 설정 |
| table | - | - | 8x8버튼 생성 및 배치 |
| ifNearButton | crdnt | TF, nearButtons | 상하좌우 대각선 상에 이미 색깔이 있는 버튼이 있는지 검사 |
| reverse | - | - | 버튼 클릭 시 돌을 뒤집는 출력 |
| clickableButtons | - | - | 현재 차례에 놓을 수 있는 버튼을 표시해 출력 |
| keypressedq | - | - | 놓을 수 있는 버튼 중 현재 가리키고 있는 버튼 출력 |
| clear | - | - | 색이 입혀진 버튼을 초기화하여 출력 |
| validMoveDirectes | senderCoords  , col | truthTable | 돌을 뒤집을 수 있는 돌이 있는지 유효성 검사 |
| score | - | - | 흑과 백의 딴 돌의 개수를 계산 |
| gameOver | - | - | 게임이 끝났을 x때 게임 결과를messageBox출력 |
| intable | xi, yi | TF | 8X8의 범위에 벗어나는지 검사 |
| placeinList | num | - | 1차원상의 table.cords를 인덱싱 값을 x,y좌표로 변환 |
| explainClicked | - | - | 게임에 대한 설명을 알려주는 widget 출력 |
| reset | - | - | 게임을 초기상태로 초기화 |
| Reverse\_Btn | X, Y | - | Enter키를 눌렀을 때 돌을 뒤집어 출력 |
| (L, R, U, D) Moved | - | - | 좌우 방향키 눌렀을 때의 이벤트 처리 |

**3. 전체 구조**

위의 결과를 정리하여 하나의 그림으로 표현한 것이 [그림 2] 이다.

MainWindow가 전체 화면과 사용자의 키보드 이벤트 처리와 안 쪽의 Reversi가 사용자의 인터페이스와 전체적인 화면을 담당한다.



[그림 2]소프트웨어 구조 설계서

**Ⅳ. 소프트웨어 상세 설계서(DDS)**

본 절에서는 모듈 별 구현 방식을 [표 5]에 상세히 기술한다. 구현 방식을 자료 구조와 알고리즘을 중심으로 설명하되, 상단 디스플레이와 하단 메뉴바의 GUI 구현은 일반적인 구현 방식(수업시간 중 다루었던 방식)을 이용하므로 이에 대한 설명은 생략하였다.

|  |  |  |
| --- | --- | --- |
| Attributes | Wbturn | 흑의 차례인지 백의 차례인지를 가지고 있는 정수 |
| Buttons | 실제로 화면에 출력할 8X8  버튼 리스트 |
| colorTable | 각 버튼별의 돌의 색깔들을 저장하는 리스트 |
| zeroColTab | 아직 선택되지 않아 돌의 색깔이 정해지지 않은 버튼들의 리스트 |
| keyBtn | 각 차례마다 놓을 수 있는 버튼의 위치들을 저장한 리스트 |
| selected\_X, selected\_Y | 키보드 이벤트를 위한 현재 가리키고 있는 좌표 (X, Y) |
| clickable | 각 차례마다 놓을 수 있는 위치좌표(X, Y)를 저장한 리스트 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Methods | table | 초기상태의 돌과 nearbutton을 계산 후 화면에 출력 |
| clickableButtons | 차례마다 플레이어가 놓을 수 있는 돌의 위치를 계산해 화면에 작은 회색 원으로 표시 |
| keyPressedq | 놓을 수 있는 회색 위치에서 키보드 이벤트를 위한 선택된 최초위치를 빨간 원으로 출력 |
| reverse, reverse\_Btn | 마우스클릭이다 버튼(Enter) 이벤트 발생 시 입력 받은 버튼의 위치를 기준으로 게임 화면을 갱신 |
| gameOver | 게임 종료 여부를 확인하고 게임이 끝났을 때 점수를 기준으로 승패를 메시지 박스로 출력한다. |
| (R, L, U, D) Moved | 방향키 selected\_X, Y의 위치를 변경한 뒤에 방향키를 기준으로 변경된 위치 출력 |
| reset | new game버튼 클릭 시 attribute와 함수를 새로 호출하여 초기화된 화면 출력 |

[표 5] Othrllo\_proj.py 요약

**1) table**

게임을 시작할 때 초기화면 상태를 출력하는 메소드이다. 8x8의 버튼을 선언하여 각 버튼의 사이즈60으로 설정한 뒤에 화면을 출력한다. 게임 시작 시 중앙 돌에 대한 초기 설정과 버튼 클릭이벤트를 버튼에 연결하고 clickableButtons()를 호출하여 클릭가능한 버튼을 출력한다.

**2) clickableButton**

사용자 차례일 때 버튼 클릭이나 Enter로 선택할 수 있는 버튼을 출력한다. 아무런 돌이 없는zeroClotab를 반복문을 사용하고, ifNearButton와 validMoveDirects를 호출하여 둘 다 True인 경우에만 (32,32)사이즈의 회색 동그라미가 그려진 버튼을 출력한다.

**3) ifNearButton**

선택한 버튼의 8방향(전후좌우, 대각선)의 버튼들을 탐색해 근처 버튼이 존재하는 경우 nearButton, nearButtonsColors에 추가한다. nearButtonColor의 값이 1,2(흑, 백)가 존재하는 경우에는 True를 반환한다.

**4) validMoveDirects**

X, Y의 좌표와 그 좌표의 색깔을 입력하여 좌표 주위의 8방향의 값이 0~7범위에 있는지 검사 후 인접해 있는 값들이 흑백 색깔이 있으면 newl에 추가를 한다. newl의 값이 같지 않다면 그 좌표를 기준으로 또 주위를 탐색한다. 추가한 newl의 값이 1이하인 경우에는 false를 반환하고, newl의 마지막 값이 col과 반대되는 색이라면 false를 반환한다.

**5) keyPressedQ**

키보드 이벤트를 위한 현재 사용자가 클릭가능한 버튼 중에서 어떤 버튼을 가리키고 있는지 계산을 한다. 가리키고 있는 버튼의 좌표는 selected\_X, selected\_Y에 했고, 버튼의 Default 값은 좌 상단을 기준으로 가장 먼 곳으로 하였다. 또한 나중의 키보드 이벤트를 위한 8X8크기의 keyBtn에 클릭 가능한 좌표일 시 1을 넣었고, 그것을 토대로 y축 기준으로 정렬한 2차원 배열 y\_keyBtn을 선언하였다.

**6) (R, L) Moved**

키보드 이벤트 시 좌 우 키를 입력 받았을 때 처리하는 두 메소드이다. y\_keyBtn을 이용하여 처리를 하며 R\_Moved 메소드는 좌 상단을 (0,0)으로 기준을 한 뒤에 왼쪽 방향키를 입력 받았을 때에 같은 y축에 selected\_X보다 작은 값이 존재하지 않는다면 Y축의 크기를 1만큼 줄여 X의 값이 가장 큰 것을 선택한다. 만약 선택가능한 버튼에서 가장 작은 버튼을 선택하여 왼쪽 방향키를 누른다면 y\_ketBtn의 가장 큰 값으로 이동을 한다. R\_moved는 반대로 구현을 하였다.

**7) (U, D) Moved**

키보드 이벤트 시 좌 우 키를 입력 받았을 때 처리하는 두 메소드이다. keyBtn을 이용하여 처리를 하며 U\_Moved 메소드는 좌 상단을 (0,0)으로 기준을 두어 위 방향키를 입력 받았을 때에 같은 x축 선상에 selected\_Y보다 작은 값이 존재한다면 그것을 선택하고 아니라면 X축을 하나 줄여 Y값이 가장 큰 것을 선택한다. 만약 선택 가능한 버튼에서 가장 작은 버튼을 선택하여 위키를 누른다면 KeyBtn의 가장 큰 값으로 이동을 한다. D\_moved는 반대로 구현을 하였다.

**8) reverse**

버튼을 클릭하였을 때 돌을 뒤집는 함수이다. sender를 이용해 클릭을 누른 버튼의 좌표를 입력 받고 ifNearButton, validMoveDirects를 호출해 유효성을 검사하고 유효하다면 zerocoltab의 해당 좌표를 제거 후 wbTurn의 값을 토대로 백돌과 흑돌의 차례일 때 validMoveDirects를 수행하여 값을 받아 h에 저장하여 h의 값이 True 인 경우에만 돌을 뒤집는 행동을 수행한다.

**9) reverse\_btn**

엔터 키를 눌렀을 때 호출이 되는 함수이다 상단의 reverse 함수와 유사하나, sender 대신 selected\_X, selected\_Y에 60값을 곱한 것을 인자로 받아 reverse와 동일한 행동을 수행한다.

**10) gameover**

게임이 끝났는지 확인하는 메서드이다. 게임이 끝났다는 말은 8X8 판을 전부 채워서 둘 곳이 없거나, 게임 규칙적으로 더 이상 둘 곳이 없을 때를 의미한다. clikcables의 크기가 0이면 더 이상 둘 곳이 없다는 뜻이므로 게임을 종료한다.

**11) reset**

게임을 초기화하는 메서드이다. Attributes와 table, clear, list 메서드를 불러와 초기화 한다.

**Ⅴ. 소스코드**

|  |
| --- |
| import sys  from PyQt5 import QtCore  from PyQt5.QtWidgets import QWidget, QMainWindow,QApplication,\  QMessageBox, QVBoxLayout, QLabel, QPushButton, QTextBrowser  from PyQt5.QtGui import QIcon, QColor, QFont,QPixmap  from PyQt5.QtCore import QSize, Qt,QRect  coords = [0, 60, 120, 180, 240, 300, 360, 420]  tableCoords = []  diffs = [(-60, -60), (0, -60), (60, -60), (-60, 0),  (60, 0), (-60, 60), (0, 60), (60, 60)]  directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1),  (0, 1), (1, -1), [1, 0], (1, 1)]  class MainWindow(QMainWindow):  #Reversi 클래스 호출과 initMw호출하는 \_\_init\_\_  def \_\_init\_\_(self):  super().\_\_init\_\_()  self.content = Reversi()  self.setCentralWidget(self.content)  self.setChildrenFocusPolicy(QtCore.Qt.NoFocus)  self.initMW()  # Main Window의 크기와 아이콘, 이름들을 설정  def initMW(self):  menubar = self.menuBar()  menubar.setStyleSheet("background-color: white;"  "color: black")  self.setStyleSheet("background-color: white")  self.move(250, 100)  self.setFixedSize(502, 785)  self.setWindowIcon(QIcon('icon.png'))  self.setWindowTitle('Othello')  self.show()  def setChildrenFocusPolicy (self, policy):  def recursiveSetChildFocusPolicy (parentQWidget):  for childQWidget in parentQWidget.findChildren(QWidget):  childQWidget.setFocusPolicy(policy)  recursiveSetChildFocusPolicy(childQWidget)  recursiveSetChildFocusPolicy(self)  def keyPressEvent(self, event):  # Snake head movement  if event.key() in [Qt.Key\_Return, Qt.Key\_Enter]:  #print(self.content.y\_keyBtn)  self.content.reverse\_Btn(self.content.selected\_X,self.content.selected\_Y)  elif event.key() == Qt.Key\_Right:  #print(self.content.y\_keyBtn)  self.content.R\_Moved()  elif event.key() == Qt.Key\_Left:  #print(self.content.y\_keyBtn)  self.content.L\_Moved()  elif event.key() == Qt.Key\_Up:  #print(self.content.y\_keyBtn)  self.content.U\_Moved()  elif event.key() == Qt.Key\_Down:  #s(self.content.y\_keyBtn)  self.content.D\_Moved()  elif event.key() not in [Qt.Key\_Return, Qt.Key\_Enter, Qt.Key\_Right, Qt.Key\_Left, Qt.Key\_Up,Qt.Key\_Down]:  return  #실질적인 화면과 게임 로직들을 처리하는 reversi 클래스  class Reversi(QWidget):  #initRv()의 호출과 멤버함수들을 선언하는 \_\_init\_\_  def \_\_init\_\_(self):  super().\_\_init\_\_()  #흑백 플레이어의 턴을 정하는 self.wbturn  self.wbTurn = 2  self.col = QColor(10, 126, 175)  self.bg\_col = QColor(4, 36, 63)  #위젯들을 수직으로 나열하는 VBoxLayOut  self.hbox = QVBoxLayout()  #중단 위치의 바둑판과 돌, 놓을수있는 위치 등을 표시하는 labelBoard  self.labelBoard = QLabel()  #하단 메뉴창 위젯  self.labelBottom = QLabel()  self.labelTools = QLabel()  #호출  self.initRV()  #버튼들과 게임판 설계  def initRV(self):  #상단의 플레이어와 차례 표시, 딴 돌을 표시  #hbox에 위젯 3개를 순서대로 추가  self.hbox.addWidget(self.labelTools)  self.hbox.addWidget(self.labelBoard)  self.hbox.addWidget(self.labelBottom)  #labelboard 크기 성정  self.labelBoard.setFixedSize(480.9, 480)  self.labelBoard.setAlignment(Qt.AlignCenter)  self.labelTools.setFixedSize(480.9,195)  self.labelBottom.setFixedSize(480.9,60)  self.setLayout(self.hbox)  #배경 설정  self.labelBoard.setStyleSheet("QWidget { background-color: %s }"  % self.col.name())  self.labelTools.setStyleSheet("QWidget { background-color: %s }"  % self.bg\_col.name())  self.labelBottom.setStyleSheet("QWidget { background-color: %s }"  % self.bg\_col.name())  #흑 플레이어의 딴 돌의 갯수를 보여주는 버튼  self.b = QPushButton(self.labelTools)  self.b.setGeometry(350, 55, 100, 100)  self.b.setStyleSheet("""  border-style: outset;  font: 45px;  color: white  """)  #백 플레이어의 딴 돌의 갯수를 보여주는 버튼  self.w = QPushButton(self.labelTools)  self.w.setGeometry(20, 55, 100, 100)  self.w.setStyleSheet("""  border-style: outset;  font: 45px;  color: white""")  #백 플레이어 이미지 표시  self.w\_img = QLabel(self.labelTools)  self.w\_img.setPixmap(QPixmap("white.png"))  self.w\_img.setGeometry(140, 55, 100, 100)  self.w\_img.show()  #흑 플레이어 이미지 표시  self.b\_img = QLabel(self.labelTools)  self.b\_img.setPixmap(QPixmap("black.png"))  self.b\_img.setGeometry(270, 55, 100, 100)  self.b\_img.show()  #플레이어 이름 표시  self.player1\_text = QLabel(self)  self.player2\_text = QLabel(self)  self.player1\_text.setGeometry(50, 140, 70, 40)  self.player2\_text.setGeometry(380, 140, 70, 40)  self.player1\_text.setText("Player1")  self.player1\_text.setFont(QFont("Arial"))  self.player2\_text.setFont(QFont("Arial"))  self.player1\_text.setStyleSheet(""" background : rgb(4, 36, 63); text-align: center; color:white; Font: 19px""")  self.player2\_text.setText("Player2")  self.player2\_text.setStyleSheet(""" background : rgb(4, 36, 63); text-align: center; color:white; Font: 19px""")  # 누구의 차례인지 표시하는 self.turn  self.turn = QTextBrowser(self)  self.turn.setFont(QFont("Arial",18))  self.turn.setGeometry(170, 20, 160, 50)  self.turn.setStyleSheet(""""border-style: outset;  border-width: 5px;  border-color: white;  """)  self.turn.setText("Player1")  self.turn.setAlignment(Qt.AlignCenter)  #하단 메뉴의 3개의 버튼(new, 도움말, 종료)의 버튼선언과 연동  #self.labelboard, self.tool을 초기화하여 새로 게임을하는 button self.bt\_new  self.bt\_new = QPushButton('새 게임',self.labelBottom)  self.bt\_new.setGeometry(0,0, 160, 60)  self.bt\_new.clicked.connect(self.reset)  self.bt\_new.setFont(QFont("Arial"))  self.bt\_new.setStyleSheet("""  border-style: outset;  border-width: 2px;  border-color:rgb(72,255,151);  font: 28px;  background:rgb(25, 196, 99);  color: rgb(4, 36, 63)  """)  #새 창을 띄워 도움말을 출력하는 bt\_explain  self.bt\_explain = QPushButton('도움말',self.labelBottom)  self.bt\_explain.setGeometry(160, 0, 160, 60)  self.bt\_explain.clicked.connect(self.explainClicked)  self.bt\_explain.setFont(QFont("Arial"))  self.bt\_explain.setStyleSheet("""  border-style: outset;  border-width: 2px;  border-color:rgb(72,255,151);  font: 28px;  background:rgb(25, 196, 99);  color:rgb(4, 36, 63);  """)  #버튼클릭시 프로그램을 종료하는 self\_bt\_quit  self.bt\_quit = QPushButton('종료',self.labelBottom)  self.bt\_quit.setGeometry(320, 0, 160, 60)  self.bt\_quit.setFont(QFont("Arial"))  #버튼클릭시 app.exit를 호출하여 프로그램 종료  self.bt\_quit.clicked.connect(app.exit)  self.bt\_quit.setStyleSheet("""  border-style: outset;  border-width: 2px;  border-color:rgb(72,255,151);  font: 28px;  background:rgb(25, 196, 99);  color: rgb(4, 36, 63)  """)  #점수 8x8버튼 선언 및 출력  self.lists()  self.table()  self.score()  #8x8의 버튼에 대한 리스트와 각각 해당하는 색깔을 선택  def lists(self):  self.colorTable = [[] for i in range(8)]  self.buttons = [[] for i in range(8)]  for i in range(8):  for j in range(8):  self.buttons[i].append("self.b" + str(i + 1) + "\_" + str(j + 1))  #초기상태 설정  if (i, j) == (3, 4) or (i, j) == (4, 3):  self.colorTable[i].append(1)  # 초기상태 설정  elif (i, j) == (3, 3) or (i, j) == (4, 4):  self.colorTable[i].append(2)  else:  self.colorTable[i].append(0)  exec("%s = %d" % (str(self.buttons[i][j]), 1))  self.zeroColTab = []  #8x8 버튼 생성 및 배치  def table(self):  #버튼의 좌표  m = 0  n = 0  #버튼 생성을 위한 2중 for문  for i in coords:  for j in coords:  self.buttons[m][n] = QPushButton(self.labelBoard)  self.buttons[m][n].resize(60, 60)  self.buttons[m][n].move(\*(j, i))  self.buttons[m][n].setStyleSheet("""  border-style: solid;  border-width: 2px;  border-color: rgb(97,184,214)  """)  #게임 시작 시 중앙 돌에 대한 초기화  if (m, n) == (3, 4) or (m, n) == (4, 3):  self.buttons[m][n].setIcon(QIcon("black.png"))  self.buttons[m][n].setIconSize(QSize(48, 48))  #게임 시작 시 중앙 돌에 대한 초기화  elif (m, n) == (3, 3) or (m, n) == (4, 4):  self.buttons[m][n].setIcon(QIcon("white.png"))  self.buttons[m][n].setIconSize(QSize(48, 48))  #일반적인 상황의 경우  else:  self.zeroColTab.append((j, i))  tableCoords.append((j, i))  #각각에 버튼을 클릭하였을 떄 돌을 뒤집는 reverse 함수에 버튼 연결  self.buttons[m][n].clicked.connect(self.reverse)  if n == 7:  m += 1  n = -1  n += 1  #클릭 가능한 버튼들을 화면에 출력하는 함수 호출  self.clickableButtons()  def ifNearButton(self, crdnt):  nearButtons = []  nearButtonColors = []  for i in range(8):  nearcoord = (crdnt[0] + diffs[i][0], crdnt[1] + diffs[i][1])  if nearcoord in tableCoords:  nearButtons.append(nearcoord)  positionx = self.placeinList(tableCoords.index(nearcoord))[0]  positiony = self.placeinList(tableCoords.index(nearcoord))[1]  nearButtonColors.append(self.colorTable[positionx][positiony])  if 1 in nearButtonColors or 2 in nearButtonColors:  return [True, nearButtons]  else:  return [False, nearButtons]  #버튼을 클릭하였을 때 뒤집는 행동을하는 함수  def reverse(self):  sender = self.sender()  senderCoords = (sender.x(), sender.y())  if senderCoords in self.zeroColTab and self.ifNearButton(senderCoords)[0] is True and True in self.validMoveDirects(senderCoords, self.wbTurn)[0]:  self.zeroColTab.remove(senderCoords)  sx = tableCoords.index(senderCoords)  pcx = self.placeinList(sx)[0]  pcy = self.placeinList(sx)[1]  #백돌의 차례일 시  if self.wbTurn == 2:  sender.setIcon(QIcon("white.png"))  sender.setIconSize(QSize(48, 48))  self.colorTable[pcx][pcy] = 2  self.turn.setText("Player2")  self.turn.setAlignment(Qt.AlignCenter)  h = self.validMoveDirects(senderCoords, self.wbTurn)  for i in range(8):  pcx = self.placeinList(sx)[0]  pcy = self.placeinList(sx)[1]  if h[0][i] == True:  j = h[1][i]  addx = directions[i][0]  addy = directions[i][1]  for k in range(j):  pcx = pcx + addx  pcy = pcy + addy  self.buttons[pcx][pcy].setIcon(QIcon("white.png"))  self.buttons[pcx][pcy].setIconSize(QSize(48, 48))  self.colorTable[pcx][pcy] = 2  #흑돌의 차례일 시  elif self.wbTurn == 1:  sender.setIcon(QIcon("black.png"))  sender.setIconSize(QSize(48, 48))  self.colorTable[pcx][pcy] = 1  self.turn.setText("Player1")  self.turn.setAlignment(Qt.AlignCenter)  h = self.validMoveDirects(senderCoords, self.wbTurn)  for i in range(8):  pcx = self.placeinList(sx)[0]  pcy = self.placeinList(sx)[1]  if h[0][i] == True:  j = h[1][i]  addx = directions[i][0]  addy = directions[i][1]  for k in range(j):  pcx = pcx + addx  pcy = pcy + addy  self.buttons[pcx][pcy].setIcon(QIcon("black.png"))  self.buttons[pcx][pcy].setIconSize(QSize(48, 48))  self.colorTable[pcx][pcy] = 1  self.score()  self.clickables.remove(senderCoords)  self.clear()  self.wbTurn = 3 - self.wbTurn  self.clickableButtons()  if len(self.clickables) == 0:  self.wbTurn = 3- self.wbTurn  self.clickableButtons()  if len(self.clickables) == 0:  self.remain = 0  self.gameOver()  #게임종료하는 함수 game over 호출  self.gameOver()  #현재 차례에 놓을 수 있는 버튼들을 표시하여 출력하는 clickableButtons  def clickableButtons(self):  self.clickables =[]  self.keyBtn = [[0 for i in range(8)] for j in range(8)]  self.y\_keyBtn = [[]]  for i in self.zeroColTab:  if self.ifNearButton(i)[0] is True and True in self.validMoveDirects(i, self.wbTurn)[0]:  indx = self.placeinList(tableCoords.index(i))[0]  indy = self.placeinList(tableCoords.index(i))[1]  self.buttons[indx][indy].setIcon(QIcon("avs.png"))  self.buttons[indx][indy].setIconSize(QSize(32, 32))  self.clickables.append(i)  self.keyPressedq()  def keyPressedq(self):  loops = False  self.selected\_X = 0  self.selected\_Y = 0  self.ex\_dot = -1  self.dept =0  for i in range(len(self.clickables)):  yrange = self.clickables[i][0]  xrange = self.clickables[i][1]  yrange = int(yrange/60)  xrange = int(xrange / 60)  self.keyBtn[xrange][yrange] = 1  for i in range(7, -1, -1):  if loops == True:  break  for j in range(7, -1, -1):  if self.keyBtn[i][j] == 1:  self.buttons[i][j].setIcon(QIcon("selected.png"))  self.buttons[i][j].setIconSize(QSize(32, 32))  self.selected\_X = i  self.selected\_Y = j  loops= True  break  for i in range(8):  for j in range(8):  if self.keyBtn[i][j] == 1 and self.ex\_dot == i or self.keyBtn[i][j] == 1 and self.ex\_dot == -1:  self.y\_keyBtn[self.dept].append((i,j))  self.ex\_dot = i  elif self.keyBtn[i][j] == 1 and self.ex\_dot != i and self.ex\_dot !=-1:  self.dept +=1  self.y\_keyBtn.append([])  self.y\_keyBtn[self.dept].append((i, j))  self.ex\_dot = i  elif self.keyBtn[i][j] == 1 and self.ex\_dot == i and self.ex\_dot !=-1:  self.y\_keyBtn[self.dept].append((i, j))  self.ex\_dot = i  self.y\_keyBtn= sorted(self.y\_keyBtn)  self.y\_length = len(self.y\_keyBtn[self.dept])-1  self.x\_length = self.dept  def clear(self):  for i in self.clickables:  indx = self.placeinList(tableCoords.index(i))[0]  indy = self.placeinList(tableCoords.index(i))[1]  self.buttons[indx][indy].setIcon(QIcon())  def validMoveDirects(self, senderCoords, col):  truthTable = [[], []]  newl = []  for direct in directions:  xi = self.placeinList(tableCoords.index(senderCoords))[0] + direct[0]  yi = self.placeinList(tableCoords.index(senderCoords))[1] + direct[1]  if self.inTable(xi, yi) == False:  truthTable[0].append(False)  truthTable[1].append(0)  else:  if self.colorTable[xi][yi] == col:  truthTable[0].append(False)  truthTable[1].append(0)  else:  while self.inTable(xi, yi) == True:  if self.colorTable[xi][yi] != 0:  newl.append(self.colorTable[xi][yi])  if self.colorTable[xi][yi] ==col:  break  else:  break  xi = xi + direct[0]  yi = yi + direct[1]  if len(newl) <= 1:  truthTable[0].append(False)  truthTable[1].append(0)  elif newl[-1]!=col:  truthTable[0].append(False)  truthTable[1].append(0)  else:  truthTable[0].append(True)  truthTable[1].append(len(newl) - 1)  newl = []  return truthTable  #흑과 백 플레이어의 현재 딴돌의개수를 selfcolorTable의 1,2의 개수들을 세서 수정  def score(self):  self.white = 0  self.black = 0  self.remain = 0  for i in self.colorTable:  self.white = self.white + i.count(2)  self.black = self.black + i.count(1)  self.remain= self.remain + i.count(0)  self.w.setText(str(self.white))  self.b.setText(str(self.black))  #게임 종료시 messagebox를 출력하여 승패자와 돌 몇개를 따서 이겼는지 출력  def gameOver(self):  if self.remain == 0:  if self.white > self.black:  QMessageBox.information(self, "GAME OVER", "WHITE WIN \n" + "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n" + "White = "  + str(self.white) + "\nBlack = " + str(self.black))  elif self.black >self.white:  QMessageBox.information(self,"GAME OVER", "BLACK WIN\n"+ "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"+ "Black = " + str(self.black)  + "\nWhite = " + str(self.white))  else:  QMessageBox.information(self, "GAME OVER", "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" + "DRAW" )  def inTable(self, xi, yi):  z = [0, 1, 2, 3, 4, 5, 6, 7]  if xi in z and yi in z:  return True  else:  return False  def placeinList(self, num):  x = int(str(num / 8)[0])  y = num % 8  return [x, y]  #도움말 창 클릭시 이미지와 게임에 대한 설명을 출력함  def explainClicked(self):  self.widget = QWidget()  self.widget.setGeometry(QRect(277, 200, 450, 680))  self.widget.setWindowTitle("도움말")  self.widget.setWindowIcon(QIcon('icon.png'))  self.widget.setStyleSheet("background-color:rgb(244,243,223)")  self.text1 = QTextBrowser(self.widget)  self.text1.setGeometry(10, 10, 430, 400)  self.text1.setStyleSheet("background-color:rgb(244,243,223); color:rgb(130,130,130)")  self.text1.blockSignals(True)  self.text1.setFrameStyle(0)  self.text1.setFont(QFont("Arial", 10))  self.text1.setText("Othello는 가로 세로 8칸의 보드 위에서 한쪽은 검은색,다른 한쪽은 흰색 돌을 번갈아 놓으며 진행하는 전략 게임입니다.\n"  + "게임의 목표는 상대방의 돌 하나나 그 이상을 플레이어의 돌로 애워싸는 것입니다.\n"  + "그러면 돌의 색상이 바뀌면서 상대방의 돌이 플레이어의 돌로 전환됩니다. \n"  + "- 이러한 전술은 가로, 세로, 또는 대각선으로 수행할 수 있습니다.\n"  + "처음에 판 가운데에서 사각형으로 엇갈리게 배치된 돌 4개를 놓고 시작합니다.\n"  + "돌은 반드시 상대방 돌을 양쪽에서 포위하여 뒤집을 수 있는 곳에 놓아야 합니다. 돌을 뒤집은 곳이 없는 경우에는 차례가 자동적으로 상대방에게 넘어가게 됩니다.\n"  + "양쪽 모두 더 이상 돌을 놓을 수 없게 되면 게임이 끝나게됩니다.\n"  + "Ohtello 판에 돌이 많이 있는 플레이어가 승자가 됩니다.\n")  self.imgs = QLabel(self.widget)  self.imgs.setGeometry(150, 450, 150, 150)  pixmap = QPixmap("icon.png")  self.imgs.setPixmap(QPixmap(pixmap))  self.widget.show()  #newgame 버튼클릭시 호출되어 게임을 초기화하는 reset함수  def reset(self):  #hobox의 labelboard,labelbottom을 초기화  self.turn.setFont(QFont("Arial",18))  self.turn.setText("Player1")  self.turn.setAlignment(Qt.AlignCenter)  self.hbox.removeWidget(self.labelBoard)  self.hbox.removeWidget(self.labelBottom)  #차례초기화  self.wbTurn=2  self.labelBottom.clear()  self.keyPressedq()  #label board 새로 선언  self.labelBoard = QLabel()  self.labelBoard.setFixedSize(480.9,480)  self.labelBoard.setAlignment(Qt.AlignCenter)  self.labelBoard.setStyleSheet("QWidget { background-color: %s}"  %self.col.name())  self.hbox.addWidget(self.labelBoard)  self.hbox.addWidget(self.labelBottom)  #버튼생성과,점수생성하는 score 함수 호출  self.lists()  self.table()  self.score()  def reverse\_Btn(self,X,Y):  sender = self.sender()  senderCoords = (Y\*60, X\*60)  if senderCoords in self.zeroColTab and self.ifNearButton(senderCoords)[0] is True and True in \  self.validMoveDirects(senderCoords, self.wbTurn)[0]:  self.zeroColTab.remove(senderCoords)  sx = tableCoords.index(senderCoords)  pcx = self.placeinList(sx)[0]  pcy = self.placeinList(sx)[1]  # 백돌의 차례일 시  if self.wbTurn == 2:  self.buttons[X][Y].setIcon(QIcon("white.png"))  self.buttons[X][Y].setIconSize(QSize(48, 48))  self.colorTable[pcx][pcy] = 2  self.turn.setText("Player2")  self.turn.setAlignment(Qt.AlignCenter)  h = self.validMoveDirects(senderCoords, self.wbTurn)  for i in range(8):  pcx = self.placeinList(sx)[0]  pcy = self.placeinList(sx)[1]  if h[0][i] == True:  j = h[1][i]  addx = directions[i][0]  addy = directions[i][1]  for k in range(j):  pcx = pcx + addx  pcy = pcy + addy  self.buttons[pcx][pcy].setIcon(QIcon("white.png"))  self.buttons[pcx][pcy].setIconSize(QSize(48, 48))  self.colorTable[pcx][pcy] = 2  # 흑돌의 차례일 시  elif self.wbTurn == 1:  self.buttons[X][Y].setIcon(QIcon("black.png"))  self.buttons[X][Y].setIconSize(QSize(48, 48))  self.colorTable[pcx][pcy] = 1  self.turn.setText("Player1")  self.turn.setAlignment(Qt.AlignCenter)  h = self.validMoveDirects(senderCoords, self.wbTurn)  for i in range(8):  pcx = self.placeinList(sx)[0]  pcy = self.placeinList(sx)[1]  if h[0][i] == True:  j = h[1][i]  addx = directions[i][0]  addy = directions[i][1]  for k in range(j):  pcx = pcx + addx  pcy = pcy + addy  self.buttons[pcx][pcy].setIcon(QIcon("black.png"))  self.buttons[pcx][pcy].setIconSize(QSize(48, 48))  self.colorTable[pcx][pcy] = 1  self.score()  self.clickables.remove(senderCoords)  self.clear()  self.wbTurn = 3 - self.wbTurn  self.clickableButtons()  if len(self.clickables) == 0:  self.wbTurn = 3 - self.wbTurn  self.clickableButtons()  if len(self.clickables) == 0:  self.remain = 0  self.gameOver()  # 게임종료하는 함수 game over 호출  self.gameOver()  def L\_Moved(self):  self.max\_len = len(self.y\_keyBtn[self.x\_length])-1  if self.max\_len == 0 and self.dept == 0:  return  elif self.dept ==0:  if self.y\_length != 0:  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  self.y\_length -= 1  self.selected\_Y = self.y\_keyBtn[self.x\_length][self.y\_length][1]  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  elif self.y\_length == 0:  self.y\_length = self.max\_len  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  self.selected\_Y = self.y\_keyBtn[self.x\_length][self.max\_len][1]  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, ta32))  elif self.dept != 0:  if self.y\_length == 0:  if self.x\_length ==0:  self.x\_length = self.dept  self.y\_length = len(self.y\_keyBtn[self.x\_length])-1  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  self.selected\_Y = self.y\_keyBtn[self.x\_length][self.y\_length][1]  self.selected\_X = self.y\_keyBtn[self.x\_length][self.y\_length][0]  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  elif self.x\_length !=0:  self.x\_length -= 1  self.y\_length = len(self.y\_keyBtn[self.x\_length])-1  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  self.selected\_Y = self.y\_keyBtn[self.x\_length][self.y\_length][1]  self.selected\_X = self.y\_keyBtn[self.x\_length][self.y\_length][0]  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  elif self.y\_length != 0:  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  self.y\_length -= 1  self.selected\_Y = self.y\_keyBtn[self.x\_length][self.y\_length][1]  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  def R\_Moved(self):  self.max\_len = len(self.y\_keyBtn[self.x\_length]) - 1  if self.max\_len == 0 and self.dept == 0:  return  elif self.dept == 0:  if self.y\_length != len(self.y\_keyBtn[self.x\_length]) - 1:  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  self.y\_length += 1  self.selected\_Y = self.y\_keyBtn[self.x\_length][self.y\_length][1]  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  elif self.y\_length == len(self.y\_keyBtn[self.x\_length]) - 1:  self.y\_length = 0  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  self.selected\_Y = self.y\_keyBtn[self.x\_length][self.y\_length][1]  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  elif self.dept != 0:  if self.y\_length == len(self.y\_keyBtn[self.x\_length]) - 1:  if self.x\_length == self.dept:  self.x\_length = 0  self.y\_length = 0  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  self.selected\_Y = self.y\_keyBtn[self.x\_length][self.y\_length][1]  self.selected\_X = self.y\_keyBtn[self.x\_length][self.y\_length][0]  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  elif self.x\_length != self.dept:  self.x\_length += 1  self.y\_length = 0  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  self.selected\_Y = self.y\_keyBtn[self.x\_length][self.y\_length][1]  self.selected\_X = self.y\_keyBtn[self.x\_length][self.y\_length][0]  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  elif self.y\_length != len(self.y\_keyBtn[self.x\_length]) - 1:  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  self.y\_length += 1  self.selected\_Y = self.y\_keyBtn[self.x\_length][self.y\_length][1]  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  def U\_Moved(self):  self.max\_len = len(self.y\_keyBtn[self.x\_length]) - 1  flag = 0  if self.dept == 0 and self.max\_len ==0:  return  if self.selected\_X == 0 and self.selected\_Y==0:  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  self.x\_length = self.dept  self.y\_length = len(self.y\_keyBtn[self.x\_length])-1  self.selected\_X = self.y\_keyBtn[self.x\_length][self.y\_length][0]  self.selected\_Y = self.y\_keyBtn[self.x\_length][self.y\_length][1]  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  else :  for j in range(self.selected\_X,-1,-1):  if j == -1:  continue  if self.selected\_X > j and flag != 2 and self.keyBtn[j][self.selected\_Y] == 1:  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  for b in range(self.dept+1):  for a in range(len(self.y\_keyBtn[b])):  if self.y\_keyBtn[b][a][0] == j and self.y\_keyBtn[b][a][1] == self.selected\_Y:  self.x\_length = b  self.y\_length = a  self.selected\_X = j  self.buttons[j][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[j][self.selected\_Y].setIconSize(QSize(32, 32))  flag = 2  if flag ==0:  for i in range(self.selected\_Y-1, -1, -1):  if i == -1:  continue  for j in range(7, -1, -1):  if j == -1:  continue  if self.selected\_Y > i and flag != 2 and self.keyBtn[j][i] == 1:  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  for b in range(self.dept+1):  for a in range(len(self.y\_keyBtn[b])):  if self.y\_keyBtn[b][a][0] == j and self.y\_keyBtn[b][a][1] == i:  self.x\_length = b  self.y\_length = a  self.selected\_X = j  self.selected\_Y = i  self.buttons[j][i].setIcon(QIcon("selected.png"))  self.buttons[j][i].setIconSize(QSize(32, 32))  flag = 2  if flag ==0:  for i in range(7,self.selected\_Y-1, -1):  if i == self.selected\_Y-1:  continue  for j in range(7, -1, -1):  if j == -1:  continue  if flag != 2 and self.keyBtn[j][i] == 1:  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  for b in range(self.dept+1):  for a in range(len(self.y\_keyBtn[b])):  if self.y\_keyBtn[b][a][0] == j and self.y\_keyBtn[b][a][1] == i:  self.x\_length = b  self.y\_length = a  self.selected\_X = j  self.selected\_Y = i  self.buttons[j][i].setIcon(QIcon("selected.png"))  self.buttons[j][i].setIconSize(QSize(32, 32))  flag = 2  def D\_Moved(self):  self.max\_len = len(self.y\_keyBtn[self.x\_length]) - 1  flag = 0  if self.dept == 0 and self.max\_len == 0:  return  for j in range(self.selected\_X+1, 8, 1):  if self.selected\_X < j and flag != 2 and self.keyBtn[j][self.selected\_Y] == 1:  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  for b in range(self.dept + 1):  for a in range(len(self.y\_keyBtn[b])):  if self.y\_keyBtn[b][a][0] == j and self.y\_keyBtn[b][a][1] == self.selected\_Y:  self.x\_length = b  self.y\_length = a  self.selected\_X = j  self.buttons[j][self.selected\_Y].setIcon(QIcon("selected.png"))  self.buttons[j][self.selected\_Y].setIconSize(QSize(32, 32))  flag = 2  if flag == 0:  for i in range(self.selected\_Y, 8, 1):  for j in range(0, 8, 1):  if j == -1:  continue  if self.selected\_Y < i and flag != 2 and self.keyBtn[j][i] == 1:  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  for b in range(self.dept + 1):  for a in range(len(self.y\_keyBtn[b])):  if self.y\_keyBtn[b][a][0] == j and self.y\_keyBtn[b][a][1] == i:  self.x\_length = b  self.y\_length = a  self.selected\_X = j  self.selected\_Y = i  self.buttons[j][i].setIcon(QIcon("selected.png"))  self.buttons[j][i].setIconSize(QSize(32, 32))  flag = 2  if flag == 0:  for i in range(0,self.selected\_Y+1 , 1):  for j in range(0, 8, 1):  if j == -1:  continue  if flag != 2 and self.keyBtn[j][i] == 1:  self.buttons[self.selected\_X][self.selected\_Y].setIcon(QIcon("avs.png"))  self.buttons[self.selected\_X][self.selected\_Y].setIconSize(QSize(32, 32))  for b in range(self.dept + 1):  for a in range(len(self.y\_keyBtn[b])):  if self.y\_keyBtn[b][a][0] == j and self.y\_keyBtn[b][a][1] == i:  self.x\_length = b  self.y\_length = a  self.selected\_X = j  self.selected\_Y = i  self.buttons[j][i].setIcon(QIcon("selected.png"))  self.buttons[j][i].setIconSize(QSize(32, 32))  flag = 2  if \_\_name\_\_ == "\_\_main\_\_":  app = QApplication(sys.argv)  gm = MainWindow()  sys.exit(app.exec\_()) |