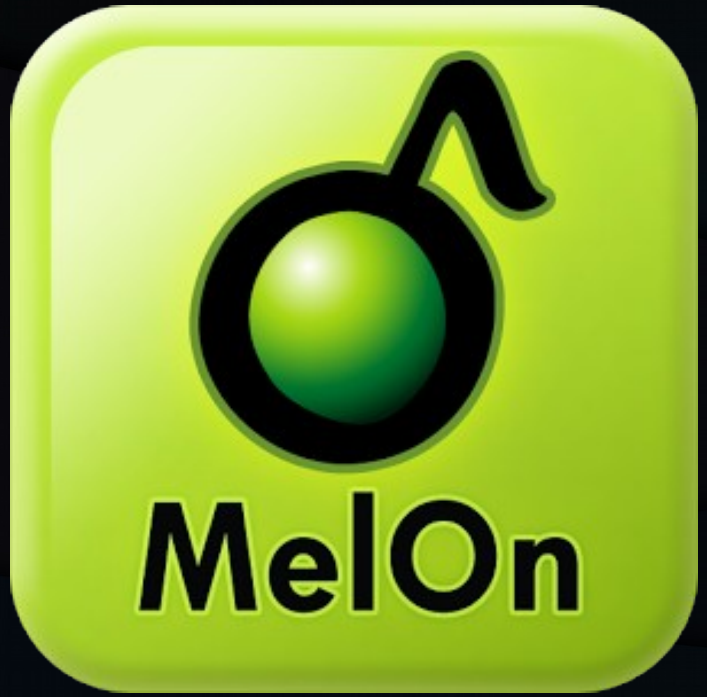

PlayList of Artwork

Data structure design – Personal Project

20164204 컴퓨터공학과 이상민

Goal of Program

- Dynamic Graph 기반의 예술작품 PlayList 구현
  - Dynamic Graph 그래프가 갖는 성질을 활용
    - 노드의 생성과 소멸에 따라 Edge 가 Dynamic 하게
    - 변형되는 그래프
  - 실용성과 효율성에 중점을 둔 프로그램 구현
    - 쉽게 접할 수 있는 PlayList 를 예술작품에 접목
    - 플레이리스트로써 필요한 기능 구현

Dynamic Graph in PlayList

- 재생 목록 = Dynamic Graph
  - 재생목록의 군집
    - (구현부) 동일한 Category (Information) 를 갖는 작품들이 연결된 그래프
    - (추가 , 삭제)User Add // User Delete 에 따라 Dynamic 한 그래프

Node :	Edge :
Artwork	Artwork Information

군집화 ? - 구현부

Edge : 고흐

Edge : 서양

Edge : 동양

Edge : 건축물

ㅇ . 작품 정보중 한 항목이 동일 -> 해당 Edge 로 연결 // 마치 군집

군집화 ? - 추가 , 삭제

- 추가
  - Artwork – Information
  - Information
    1. 작가 2. 연도 3. 장르 4. 동양 / 서 양
  - Information 을 기준으로
  - 기존의 그래프에서 동일한 Information 을 갖는
  - 작품들과 Edge 연결

- 삭제
  - Artwork
  - 해당 Artwork 삭제
  - 해당 Artwork 와 다른 Artwork 의
  - Edge 모두 연결 해제

Programming - python

- Dynamic Graph
  - Dictionary
    - Python data structure
    - Key() : Artwork
    - Values() : Information

- Programming
  - PyQt5 GUI tool
    - 입력 , 출력 , 삭제 선택
    - 입력
    - 출력 : What Category
    - 삭제

Design - Outline

Input or Output.py | Input.py | Delete.py | Output_WhatKind.py | Output.py | Playlist_DynamicGraph.py

```
'''
프로그램 : smart playlist

초기화면 : 입력할지 출력할지 결정 (QCheckBox)
    1) 입력 : 정보 입력 -> node(딕셔너리.키)로 저장, edge(딕셔너리.벨류)로 연결
    2) 출력 :
        a) 어떤 특징을 갖는 작품 list를 출력할 것인지 선택
        b) 해당 특징(벨류)를 갖는 작품(키) 출력 | GUI로 구현
    3) 삭제 : 딕셔너리에서 삭

Dynamic_Graph
입력 : 작품과 정보 -> 기존에 입력된 정보값이 동일한 노드(작품)들과 연결
삭제 : 작품 -> 딕셔너리의 keys()값을 조사하며 있으면 삭제
'''
```

- Consist of 6parts
  - 1. 초기화면 - 추가, 삭제, 재생 선택
  - 2. 추가 module
  - 3. 삭제 module
  - 4. 재생 intro : What Category module
  - 5. 재생 module
  - 6. 재생목록 module
  - & main

핵심 코드 – 1. 초기화면

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QIcon, Qpixmap


#선택 항목
self.radio1 = QRadioButton("Play", self)
self.radio1.move(20, 20)
self.radio1.setChecked(True)
```

핵심 코드 – 2. Artwork 추가

```
#gui - QGridLayout

self.label1 = QLabel("Artwork ")

self.lineEdit1 = QlineEdit()

self.pushButton1= QPushButton("Add on Playlist")

self.pushButton1.clicked.connect(self.inputNode)
self.pushButton2= QPushButton("Not anymore")

layout = QGridLayout()
```

```
#Dynamic Graph 추가 - Node 연결
def inputNode(self):
    Artwork =  self.lineEdit1.text()
    Artist = self.lineEdit2.text()
    preYear = self.lineEdit3.text()
    Year = int(preYear)/100*100 #100년 단위로 분류
  Genre = self.lineEdit4.text()

    n_Artist[Artwork] = Artist
    n_Year[Artwork] = Year
    n_Genre[Artwork] = Genre
```

핵심 코드 – 3. Artwork 삭제

- Not yet

- Dictionary key() 값을
- 조사하며
- 해당 artwork delete.

핵심 코드 - 4. 재생 intro

```
#gui (1) 어떤 Category
self.checkBox1 = QCheckBox("Artist", self)
self.checkBox1.move(10, 20)
self.checkBox1.resize(150, 30)
self.checkBox1.stateChanged.connect(self.checkBoxS
tate_intro)

def checkBoxState_intro(self):
    if self.checkBox1.isChecked() == True:
        msg = "Artist Category : " + tistmsg
    if self.checkBox2.isChecked() == True:
        msg = "Year Category : " + yearmsg
    if self.checkBox3.isChecked() == True:
        msg = "Genre Category : " + enremsg
    self.statusBar.showMessage(msg)
```

- #gui (1) Category의 어떤 요소
-
- Not yet
- Str 자료형으로 Edge (values()) 를 받아옴으로써 선지를 제공
- Checkbox or QLineEdit로 gui 구상 예정

핵심 코드 – 5. 재생

- Not yet

- Dictionary 값을 받아와
- Gui label or QpushButton 을 통해
- 가시화 예

핵심 코드 – 6. 재생목록 (graph)

```
#Dynamic Graph
#by Dictionary
#keys() : Artwork : node
#values() : Information : edge
n_Artist = {}
n_Year = {}
n_Genre = {}
```

- Dictionary 를 통해
- Dynamic graph 구현
- 사용자가 입력 혹은 삭제 하므로
- node 는 기존에 존재하지 않음

Function – Add or not

- 1.
  - Problem : 실행 종료 시 , Dynamic Graph 자동 포멧
  - Function : Dynamic Graph 보관 by 파일 입출력
- 2.
  - Function : 재생에 따른 Artwork 이미지 출력

Q&A

- Next() :

  program 구현 ; # 일단 완성

  Implementation and capture;

  if time :

    additional Function;