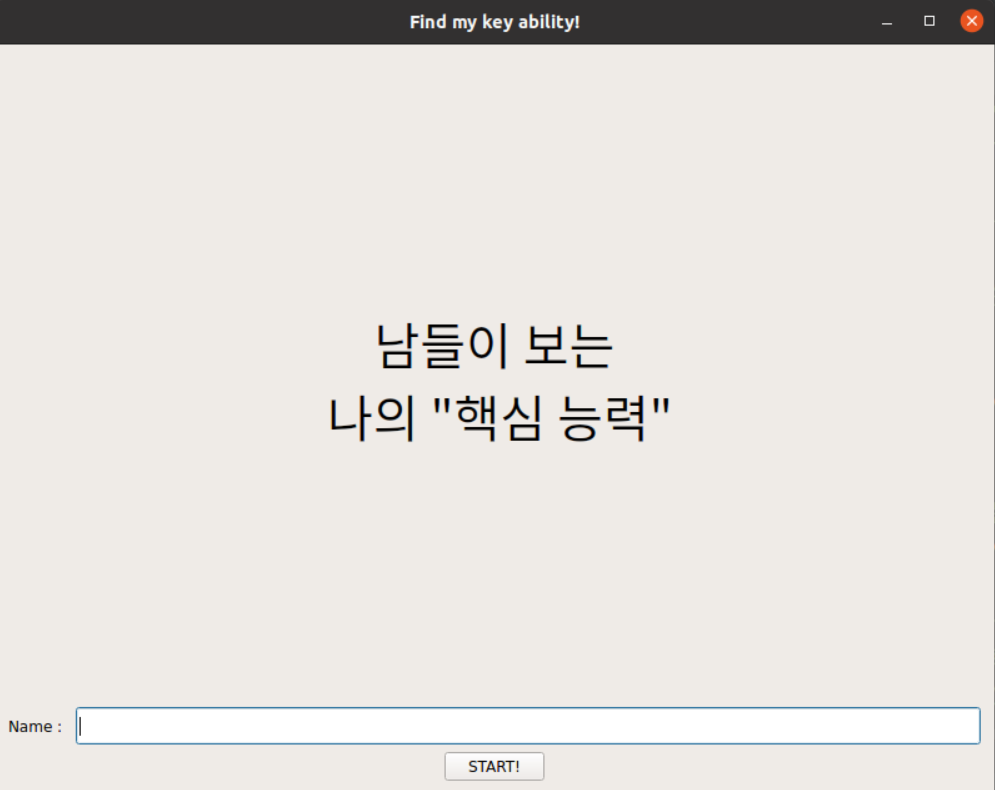
AD 프로젝트 보고서

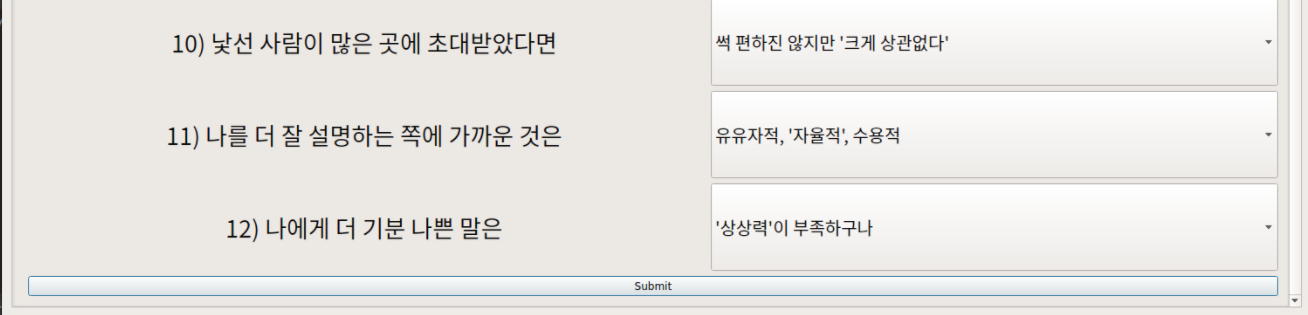
20203074 박정민

1. 요구사항 분석



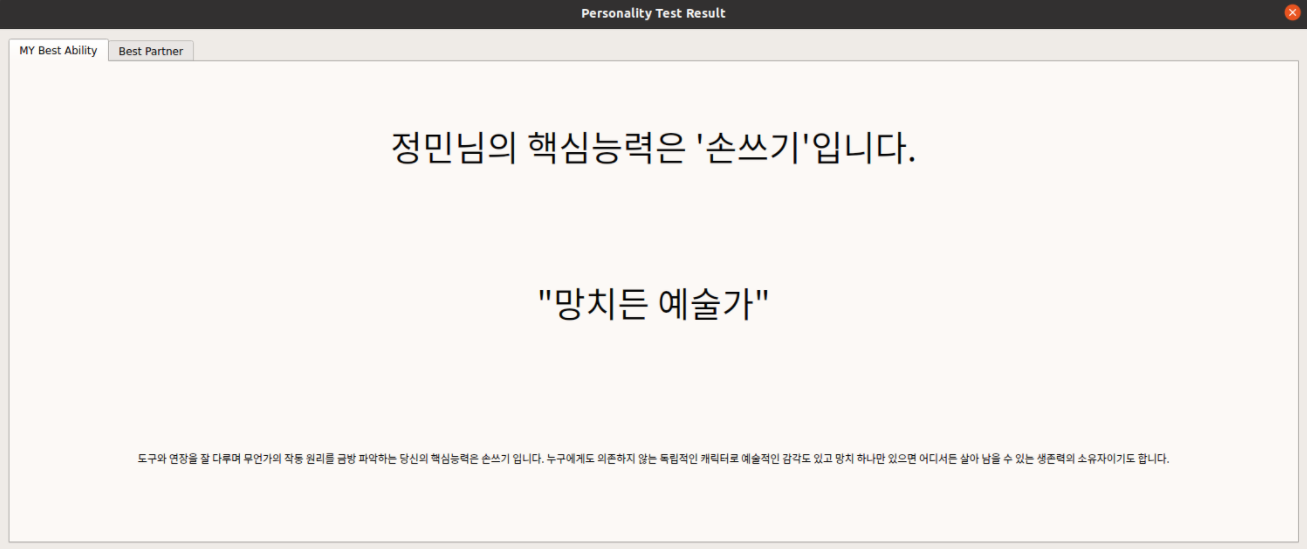
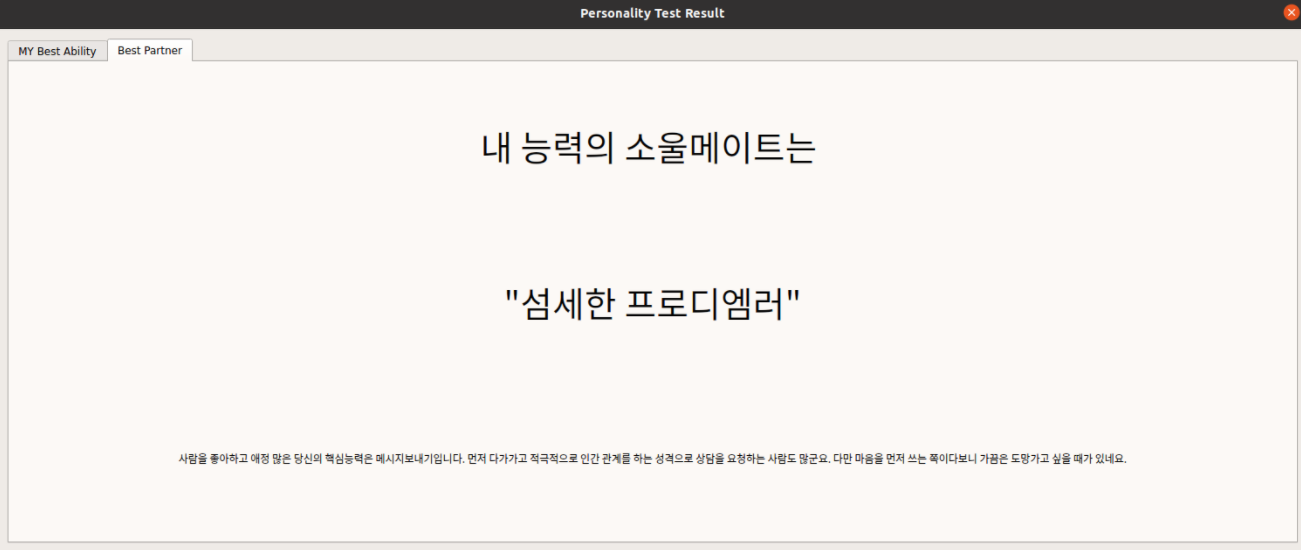
1. StartPage

* 사용자가 원하는 이름을 입력하도록 입력창이 필요(QLineEdit)
* 다음 페이지로 넘어가기 위한 ‘START’버튼 필요
* 사용자가 이름을 입력하지 않아도 실행 가능하도록 함

1. MainPage

* 평가 질문들과 질문들에 대한 선택지 필요(QComboBox)
* 제출을 위한 ‘Submit’버튼 필요
* 한번 제출 후에는 수정이 불가하도록 Submit 버튼을 비활성화

1. ResultPage

* 사용자의 결과를 보여주는 ‘My Best Ability’창 필요
* 사용자와 가장 잘 어울리는 상대방에 대해 알려주는 ‘Best Partner’창 필요

1. 게임 로직의 기능적 요구사항

* 질문에 대한 답들을 전달받아 결과를 산출하여 결과에 맞는 정보들을 제공해주어야 한다.

1. 요구사항 명세서(SRS)
2. 기능적 요구사항
   1. 게임에 이용되는 질문들이 필요하다.
   2. 각 질문들에 대한 선택지가 필요하다.
   3. 각 선택지마다 MBTI유형 매칭이 필요하다.
   4. 결과들을 종합하여 최종 MBTI유형을 산출할 수 있어야 한다.
   5. 최종적인 MBTI유형에 따른 결과를 제공해주어야 한다.
3. 사용자 인터페이스 요구사항

2-1. StartPage

2-1-1. 게임의 제목 표시

2-1-2. 사용자 닉네임 입력

2-1-3. 시작 버튼

2-2. MainPage

2-2-1. 1~12번 문제들과 선택지 두 개씩

2-2-2. 제출 버튼

2-3. ResultPage

2-3-1. 나의 결과창

2-3-1-1. 내가 입력한 닉네임과 산출된 결과 유형을 표시

2-3-1-2. 산출된 결과에 대한 자세한 설명 표시

2-3-2. 나와 가장 잘 어울리는 파트너 결과창

2-3-2-1. 나와 가장 잘 어울리는 파트너 유형을 표시

2-3-2-2. 파트너 유형에 대한 자세한 설명 표시

1. 비기능적 요구사항

3-1. 이 소프트웨어의 구현에는 Python과 PyQt5를 이용한다.

1. 소프트웨어 구조 설계

|  |  |  |  |
| --- | --- | --- | --- |
| 모듈 | 클래스 | 메서드 | 기능 |
| main.py | StartView(QWidget) | initUI | Start창 레이아웃 설정  1. QLabel로 게임 제목 표시  2. QLineEdit으로 사용자 닉네임 입력 받음  3. QPushButton으로 게임 시작  4. 1~3을 각각 QHBoxLayout에 집어넣어 하나의 QVBoxLayout에 추가 |
| startButtonClicked | 1. QLineEdit에 입력된 text를 logic의 setName함수로 전달  2. MainView 생성 |
| MainView(QDialog) | initUI | Main창 레이아웃 표시 |
| submitButtonClicked | 1. combobox에서 선택된 값들을 selected라는 리스트에 담아 logic의 storeTestResult에 전달  2. ResultView 창 생성 |
| ResultView(QDialog) | initUI | Result창 레이아웃 표시  1. QTabWidget을 활용하여 탭창 사용  2.나의 결과를 보여주는 myBestAbility 창과 나와 잘 어울리는 파트너에 대한 결과를 보여주는 bestPartner창 생성  3. 나의 ability에 대해 표시해주는 QHBoxLayout 생성  4.나의 character에 대해 표시해주는 QHBoxLayout 생성  5.detail한 설명에 대해 표시해주는 QHBoxLayout 생성  6. 산출된 MBTI결과를 logic의 getMyMBTI메서드에서 전달받아 personality의 find 메서드를 통해 MBTI유형별 정보를 정리해둔 info 딕셔너리에서 해당 인덱스 번호를 찾음  7. 찾은 인덱스 번호를 사용하여 각각의 정보들을 표시 |
| openDB | character에 대한 detail한 설명을 적어서 따로 저장해둔 txt파일을 load하는 메서드 |
| logic.py | Logic | setName | 사용자가 start창에서 입력한 닉네임을 저장 |
| getName | 저장한 닉네임을 반환 |
| storeTestResult | 1.선택한 선택지들을 리스트로 전달받아 유형을 count함  2. inspectMBTI를 호출 |
| inspectMBTI | testdb 딕셔너리에 저장된 선택된 유형별 횟수를 같은 type끼리 비교하여 최종 MBTI결과 산출 |
| getMyMBTI | MBTI결과 반환 |

1. 구현 상세 설계

<typeCheck.py>

* questionList : 1~12번 문제들을 이 리스트에 저장
* answerList : 각 문제들에 대한 선택지들을 튜플로 묶어 한 개의 리스트로 저장

ex) [("\'따뜻한\' 사람", "\'유능한\' 사람"), ( ), …]

* typeList : answerList의 각각에 대응되는 유형을 질문별로 묶어서 한 개의 리스트로 저장

ex) [(“F”, “T”), (“I”, “E”), …]

* map 메서드 : answerList와 typeList를 key, value로 일대일 매칭하여 mapDic이라는 딕셔너리에 저장

ex) {"\'따뜻한\' 사람" : “F”, "\'유능한\' 사람" : “T”, …}

<personality.py>

* info : 각각의 MBTI유형별 정보들(MBTI, ability, character, filename)을 key,value형태의 딕셔너리로 나누어 하나의 리스트에 저장

ex) [{"MBTI" : "ESTJ", "ability" : "\'일처리하기\'", "character" : "\"자뻑 일잘러\"", "filename" : "jabbeok"}, {"MBTI" : "ENTJ", "ability" : "\'일벌이기\'", "character" : "\"워커홀릭 야근요정\"", "filename" : "workaholic"},…]

* find 메서드 : main의 ResultView에서 mbti유형을 인자로 전달받아 info 딕셔너리에서 해당 mbti의 정보가 저장되어있는 인덱스를 찾아 반환
* getTextfile 메서드 : 해당되는 타입의 인덱스를 전달받아 info 딕셔너리에서 filename에서 찾아 ‘.txt’를 더해 온전한 파일이름을 완성하여 반환
* bestpartner : 서로 잘 맞는 MBTI유형을 key, value 형태로 매칭시켜 딕셔너리로 저장

<logic.py>

* Logic class
* setName 메서드 : Start창에서 입력한 닉네임을 ‘START’버튼을 누르면 startButtonClicked메서드에서 setName 메서드로 넘겨주면 self.playerName 이라는 변수에 저장
* getName 메서드 : Result창에 닉네임을 표시하기 위해 호출되는 함수로 self.playerName 을 반환
* storeTestResult 메서드 : Main창에서 ‘Submit’버튼을 누르면 submitButtonClicked가 실행되는데 이 메서드에서 combobox 의 현재 상태를 selected리스트에 저장해 놓으면 이것이 storeTestResult에 인자로 전달되어 typeCheck.py 에 매칭시켜놓은 mapDic을 통해 각각의 유형을 count 해서 testdb딕셔너리에 업데이트 해 줌. 완료되면 insepctMBTI 메서드를 호출
* inspectMBTI 메서드 : testdb에 저장된 정보들을 바탕으로 counting된 E/I, S/N, F/T, J/P 수를 비교하여 self.result리스트에 추가. 네 카테고리의 검사가 모두 끝나면 self.result에 추가된 것들을 하나의 string으로 합쳐 self.myMBTI변수에 저장.
* getMyMBTI 메서드 : Result창에서 정보를 표시하기 위해 나의 MBTI 결과를 반환해줌

<main.py>

* StartView class
* InitUI :
* self.title 에 QLabel 을 사용하여 테스트 제목을 생성
* self.name에 QLabel을 사용하여 이름 입력창 앞에 표시하는 라벨 생성
* self.nameEdit에 QLineEdit를 사용하여 이름 입력창을 생성
* self.startButton에 QPushButton을 사용하여 시작 버튼 생성. startButtonClicked와 연결
* self.title, self.name과 self.nameEdit, self.startButton을 각각의 QHBoxLayout을 생성하여 하나의 QVBoxLayout에 배치
* startButtonClicked : self.nameEdit에 입력된 text를 logic.py의 setName으로 전달. mainView 실행
* mainView class
* initUI :
* self.submitButton : QPushButton으로 ‘Submit’버튼 생성하고 sbmitButtonClicked 메서드와 연결
* self.number : 문제별로 comobox를 만들기 위해 combobox의 이름을 리스트로 미리 저장
* self.question : QLabel로 typeCheck.py에 있는 questionList를 순서대로 가져와 unit에 배치
* self.number[i] : 문제별로 combbox생성, typeCheck.py에 있는 answerList에서 가져와 combobox 아이템에 추가하고 이를 unit 에 배치
* 각 문제별로 생성된 unit을 formLayout으로 배치하고 이를 groupBox에 추가 (12번 반복)
* 12개의 문제와 선택지 레이아웃 완성 후 submitButton을 추가
* 표시해야 될 내용이 많으므로 scroll이 가능하도록 설정
* submitButtonClicked :
* self.submitButton을 비활성화(한 테스트에서 submit 제출 후 정보 수정을 막기 위함)
* combobox의 currentText들을 selected라는 리스트에 저장
* selected리스트를 logic.py의 storeTestResult에 인자로 넘겨줌
* resultView 실행
* resultView class
* initUI :
* 나의 결과를 보여줄 ‘myBestAbility’ 탭과 나와 잘 맞는 상대를 보여줄 ‘Best Partner’ 탭을 생성
* self.index : logic에서 산출된 MBTI결과를 logic의 getMyMBTI함수를 호출하여 전달받아 personality.py의 find메서드에서 해당 MBTI가 저장된 info 인덱스를 찾아 반환하면 self.index에 저장
* self.result : QLabel로 logic.py의 getName메서드를 호출하여 저장된 닉네임을 반환하고 self.index를 사용하여 해당 ability 를 가져와서 string으로 합쳐 저장
* self.character : QLabel로 self.index를 사용하여 info의 해당 인덱스를 찾아 “character”의 value를 저장
* self.detail : QLable로 openDB메서드에 self.index를 인자로 전달하여 info의 해다 인덱스를 찾아 openDB에서 반환된 값 저장
* self.result, self.character, self.detail을 각각의 QHBoxLayout에 배치하여 하나의 QVBoxLayout에 배치하고 이를 myBestAbiltiy탭에 배치
* self.partnerindex : 나와 잘 맞는 파트너를 personality.py의 bestpartner딕셔너리를 통해 찾고 찾은 bestpartner의 MBTI유형을 find 메서드에 인자로 전달하여 인덱스 번호를 찾아 저장
* self.partner : QLabel로 self.partnerindex를 사용하여 info의 해당 인덱스를 찾아 “character”의 value를 저장
* self.partnerdetail : QLabel로 openDB 메서드에 self.partnerindex를 인자로 전달하여 info의 해당 인덱스를 찾아 openDB에서 반환된 값 저장
* self.explanation, self.partner, self.partnerdetail을 각각의 QHBoxLayout에 배치하여 하나의 QVBoxLayout에 배치하고 이를 bestPartner탭에 배치
* openDB 메서드 :
* defilname : 인자로 받은 인덱스 값을 personality.py의 getTextfile 메서드에 전달하여 완전한 filename을 반환받음
* dbfilepath : dbfilename을 합쳐 완전한 파일 경로를 완성해줌
* 파일을 열어 내용을 읽어와 self.lines에 저장하고 self.lines를 반환