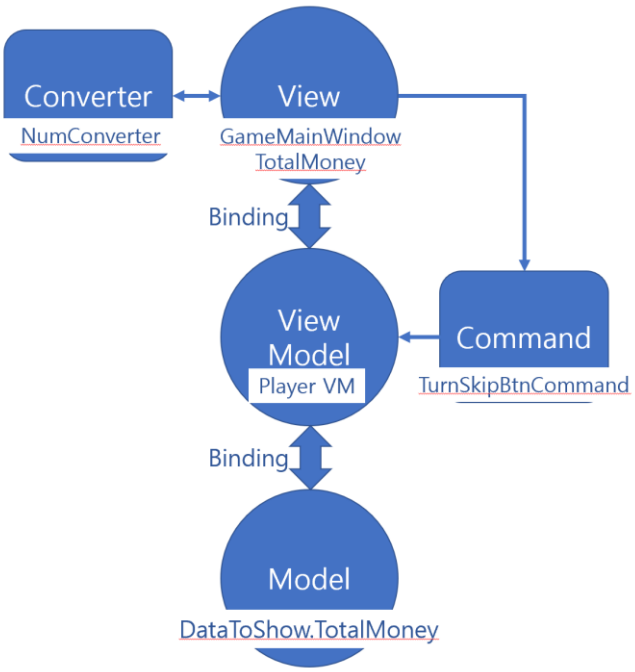
윈도우프로그래밍 최종프로젝트 보고서

201716403 신승원 201716448 정수인

목차

1. MVVM 구조

1-1. 전체 MVVM 구조



전체 자산 표현 과정 예시

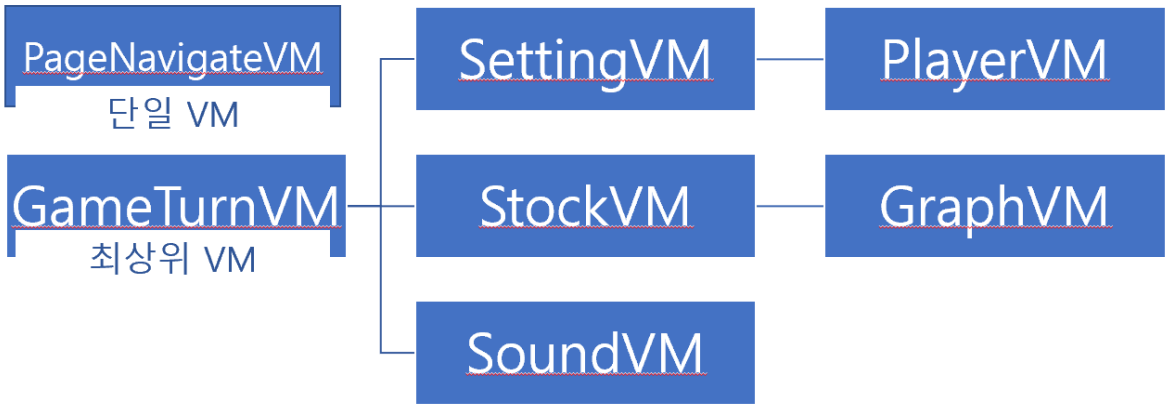
1. TurnSkipBtnCommand 실행

2. PlayerVM에서 DataToShow Model의 TotalMoney 값 수정

3. INotifyPropertyChanged 인터페이스 구현으로 자동으로 바인딩 된 View값 변경

4. NumConverter를 거쳐 최종 출력 값 표시

1-2. VM 계층구조



1-3. VM별 역할(계층구조)

PageNavigateVM : 도움말 창 페이지 이동 구현을 위해 작성한 단일 VM

GameTurnVM: 총괄 VM, 모든 VM의 값과 상당수의 Command와 바인딩

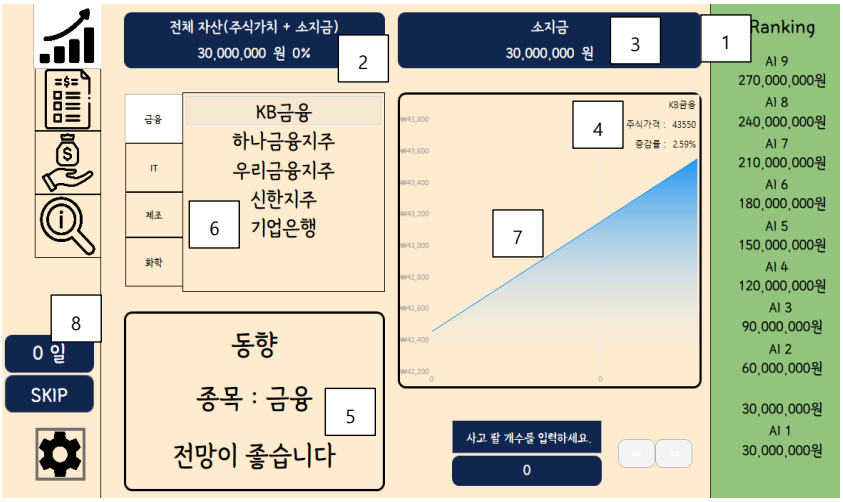
- SettingVM: 게임초기 세팅과 관련한 값들과 바인딩 된 VM
  - PlayerVM: 유저와 AI의 모든 것을 관리, 바인딩된 VM
- StockVM: 주식 API사용과 주식 값, 계산 알고리즘 등 주식과 바인딩된 VM
  - GraphVM: 주식 데이터를 이용하여 그래프와 바인딩된 VM
- SoundVM: 소리설정과 바인딩 된 VM

2. Data Binding

1. MainWindow, SettingWindow

BGM, Effect Btn (<- binding ->) **SoundVM** (<- binding ->) **IsShowBGM(Effect)**

2. StartSettingWindow, GameMainWindow



TurnSkipCommand 실행 시 – 모든 데이터 동기화의 시작 커맨드

1.Ranking ListBox (<- binding ->) **PlayerVM** (<- binding ->) **PlayersDataToShow**

2.TotalMoney Label(<- binding ->) **PlayerVM** (<- binding ->) **TotalMoney**

3.CurMoney Label(<- binding ->) **PlayerVM** (<- binding ->) **CurMoney**

4.Stock Cost, Rate(<- binding ->) **StockVM** (<- binding ->) **Item Rate, Clpr(Cost)**

5.TrandsStock, Rate(<- binding ->) **StockVM** (<- binding ->) **TrandsStock Name, Rate**

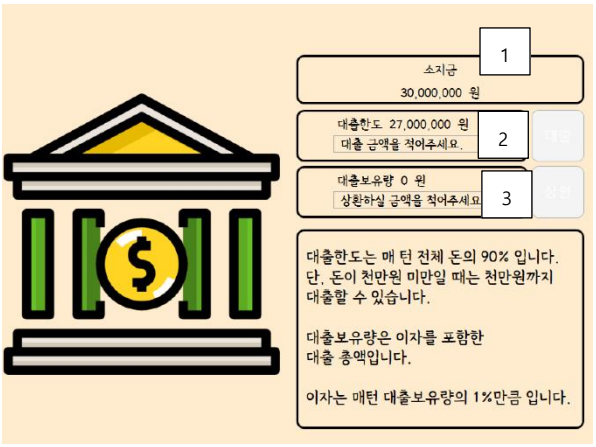
6.SelectedStock(<- binding ->) **StockVM** (<- binding ->) **SelectedStock**

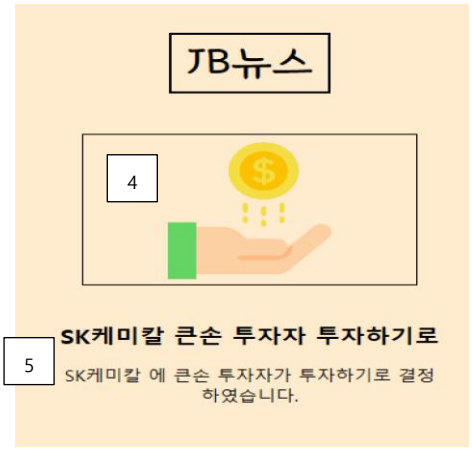
7.StockGraph(<- binding ->) **GraphVM** (<- binding ->) **GraphData**

8.TurnCnt(<- binding ->) **SettingVM** (<- binding ->) **TurnCnt**

9.Level Btn (<- binding ->) **SettingVM** (<- binding ->) **Level**

2. GameMainWindow(Loan), EventPopUpWindow





1.CurMoney(<- binding ->) **PlayerVM** (<- binding ->) **CurMoney**

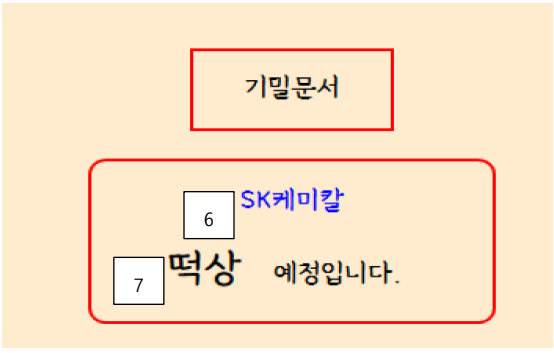
2.CanTakeMaxLoan(<- binding ->) **PlayerVM** (<- binding ->) **CanTakeMaxLoan**

3.LoanMoney(<- binding ->) **PlayerVM** (<- binding ->) **LoanMoney**

4.EventImage(<- binding ->) **SettingVM** (<- binding ->) **_Events.ImgSrc**

5.EventTargetCompnay(<- binding ->) **SettingVM** (<- binding ->) EventCompany

3. ResultWindow, ShowInformationWindow



1.TurnCnt(<- binding ->) **SettingVM** (<- binding ->) **TurnCnt**

2.CurMoney(<- binding ->) **PlayerVM** (<- binding ->) **CurMoney**

3.StockMoney(<- binding ->) **PlayerVM** (<- binding ->) **StockMoney**

4.TotalMoney(<- binding ->) **PlayerVM** (<- binding ->) **TotalMoney**

5.Rank ItemSource (<- binding ->) **PlayerVM** (<- binding ->) **PlayersDataToShow**

6.EventCompany(<- binding ->) **SettingVM** (<- binding ->) **EventCompany**

7. IsGoodEvent (<- binding ->) **SettingVM** (<- binding ->) **GetIsGood**

2-1. Binding을 사용하여 얻은 이점

게임 특성상 각 값들이 유기적으로 연결되어 있는 경우가 많았다. 따라서 하나의 값이 바뀌면 다른 값들도 바뀌게 되고 INotifyPropetyChanged 인터페이스를 통해 자동으로 View의 값들이 모두 바뀌는 장점이 있었다. 또한 중복되는 값을 표현하는 부분이 많았기 때문에 데이터에 Binding하여 재사용이 가능했다.

3. 핵심기능

3-1. 주식 계산

주식변동률의 현실성을 위하여 API를 통하여 랜덤한 기간의 실제 주식데이터를 얻어와서 변동값을 계산한 후에 게임에 쓸 주식에 반영을 한다. 만약 실제 데이터가 존재하지 않는 공휴일 및 주말은 3%~5% 사이의 랜덤한 변동값을 주식에 반영한다.

그 후 각 주식 종목(금융, IT 등)내부에 존재하는 주식들의 계산된 변동 값들의 평균을 계산하여 동향을 구한다.

마지막으로 계산된 주식 데이터는 GraphVM을 통하여 외부 라이브러리인 LiveChart 로 이동하여 사용자에게 이동이 가능하고 확대, 축소가 가능한 주식차트를 제공한다

3-2. Turn(전체적인 실행 알고리즘)

(GameTurnVM을 기반으로 시작 -> 각종 생성자 실행 -> 초기 값 설정)

TurnSkipBtnCommand -> 주식 변동 값 확인, 대입 -> 팝업 이벤트 조건 확인 -> 주식 값을 사용하여 사용자 돈 갱신 -> AI 돈 갱신 -> 주식 값으로 그래프 출력 -> 이자계산