**윈도우 프로그래밍 기말 프로젝트 최종 보고서**

조 이름 : 윈프밍

프로젝트 : wpf를 이용한 같은 그림 맞추기 게임 구현

“맞춰 맞춰 달콤카드”

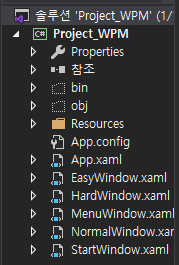
조원

컴퓨터공학부 201617199 김은솔

컴퓨터공학부 201615541 주윤서

**목 차**

1. 프로젝트 개요 및 구성 소개
2. 조원 및 part 소개
3. 개발 전체 일정 및 part별 일정
4. 요구사항 명세
5. part별 프로젝트 구현 및 사용 알고리즘 (part별 코드 소개)
6. 제작 시 문제점 및 해결 방안
7. 향후 보완사항
8. 참고 문헌 및 사이트
9. **프로젝트 개요 및 구성 소개**

* 프로젝트 주제 : 같은 그림 찾기 게임 ( 게임 이름 : 맞춰 맞춰 달콤카드 )
* 주제 선정 이유 :
* 코로나19가 터지며 사회적 거리두기가 시행되고 이러한 여파로 인해 사람들이 집에 머무는 시간이 늘어남에 따라 집에서 누구나 쉽고 재미있지만 단순한 게임이 무엇이 있을까 고민.
* 간단하지만 wpf에서 gui를 윈도우창을 사용해 구현 가능한 프로그램인 같은 그림 맞추기 게임.
* wpf의 다양한 이벤트와 data binding을 이용해 구현할 수 있는 기능들을 많이 구현해 볼 수 있음.
* 프로젝트 구성 :
* 프로젝트를 화면마다 wpf창으로 구성한다.
* 각 화면 (시작 화면, 메뉴화면, easy, normal, hard의 각 모드의 게임 화면)의 wpf창, 배경사진, 버튼 이미지와 같은 source들을 저장할 Resource파일로 구성 되어있다.
* 게임 이용자가 게임을 시작하면 모드를 선택하고 제한 시간 안에 같은 그림을 맞춰 카드를 없애 나가는 게임.



* 게임의 단계는 easy(4\*4), normal(6\*6), hard(8\*8) 모드로 나뉘고, 게임화면은

1. 버튼들이 배치되고 게임이 이루어지는 board

2. 남은 카드 개수를 보여주는 label

3. 일시정지, 나가기 버튼

4. 남은 시간을 알려주는 timer label과 동시에 남은 체력을 알려주는 progressbar.

으로 구성된다.

1. **조원 및 part 소개**

* 201615541 컴퓨터공학부 주윤서
* 게임 화면 전체 UI 코드작성(XAML) 및 디자인
* 타이머 구현 및 사용
* 201617199 컴퓨터공학부 김은솔
* matching 게임 알고리즘 코드 작성

1. **개발 전체 일정 및 part별 일정**



* 프로젝트 개발 (3주차 ~ 6주차)
* 3주차
* 주윤서 : 대략적인 xmal 작성 및 이미지 저장, 참고자료 검색
* 김은솔 : 참고 자료 검색 및 wpf 작동원리 이해
* 4주차
* 주윤서 : xmal 완성
* 김은솔 : 필요한 멤버 변수 및 메소드 정의
* 5주차
* 주윤서 : xmal.cs 알고리즘 작성 및 코드 수정
* 김은솔 : 세부사항 구현 및 수정
* 6주차
* 주윤서 : 오류 수정 및 코드 추가
* 김은솔 : 오류 수정 및 코드 추가

1. **요구사항 명세**

**1.각 단계의 난이도 차이가 존재하는가?**

게임 플레이 시 Easy mode, Normal mode, Hard mode의 난이도 차이가 체감 되도록 카드의 개수와 제한 시간을 다르게 설정.

**2. 카드를 선택했을 때 게임이 잘 작동하는가?**

1. **같은 카드**

* 매치가 되어 남은 개수의 카드가 2개씩 줄어들고 카드가 사라진다.

**(2) 다른 카드**

* 매치가 되지 않아 다른 카드를 클릭하면 카드 선택이 풀린다.

- 남은 카드의 개수는 변하지 않는다.

**3. 게임 성공 시 올바르게 동작하는가?**

* 각 모드마다 정해진 개수의 카드를 제한시간 내에 모두 매치하였을 경우 성공
* 성공 메시지 박스를 출력, 다시 시작할 것인지 묻는다.

**4. 게임 실패 시 올바르게 동작하는가?**

* 정해진 카드 개수를 제한 시간 내에 매치하지 못했을 경우 실패
* 실패 메시지 박스를 출력, 다시 시작할 것인지 묻는다.

**5. 버튼이 원하는 동작을 올바르게 수행하는가**?

* **화면이동** : 버튼 클릭 시 현재 화면을 지우고 원하는 화면으로 이동한다.
* **일시정지** : 타이머를 중단 시키고, 게임이 진행되지 않도록 설정.
* **나가기 :** 게임을 그만하고 싶은 경우 나가기 버튼 클릭으로 메뉴화면으로 이동한다.

1. **part별 프로젝트 구현 및 사용 알고리즘 (part별 코드 소개)**

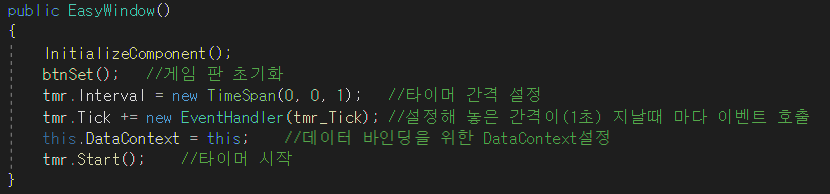
**– 김은솔(Matching Part구현)**

* 제한 시간 내에 같은 그림을 두개 씩 선택해 같은 그림을 전부 맞추는 게임을 구현. 모드별로 다른 난이도를 가지고 랜덤으로 이미지 버튼을 배치하여 사용자가 선택한 두 버튼이 같은 지 비교하고, 그 결과에 따라 게임을 진행하는 코드 진행을 구현하였다.
* 모드마다 EasyWindow, NormalWindow, HardWindow으로 창이 나뉜다. 각 모드(단계)의 차이는 카드 개수인 Matched Property와 시간 제한을 표시하는 Timer의 Count Property의 초기값을 제외하고는 일반적으로 동일한 방식으로 작동한다. 그러므로 코드와 알고리즘 설명은 Easy모드를 기준으로 설명하고 끝에서 Easy Mode와 다른 모드들의 차이점을 설명한다.
* Window창의 멤버변수, 속성Property
* 각 모드의 윈도우 창은 INotifyPropertyChanged 인터페이스를 상속하고 인터페이스의 메소드를 구현한다. 이는 남은 카드 개수를 나타내는 Matched Property와 타이머의 남은 시간을 알려주는 Count Property를 xaml과 데이터 바인딩 하기 위한 것으로, 두 변수들을 Property로 구현한 이유이다.

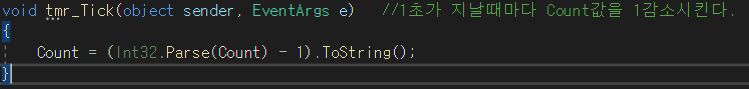




* first, second Button : 사용자는 같다고 생각하는 버튼을 두개 클릭한다. 첫번째 눌린 버튼과 두번째 눌린 버튼을 같은 지 비교하기 위한 버튼을 두개 선언해 저장한다.
* isStop 변수(Boolean type) : 일시정지 버튼 이용 시 현재 상태가 일시정지 상태인지 게임이 진행중인 상태인지를 확인하기 위한 bool type변수. 각 상태에 따라 진행되어야 하는 코드가 다르기 때문에 이를 위해 선언한다.
* Matched Property : 사용자가 앞으로 맞춰야 하는 카드의 개수. xaml의 남은 카드 수를 나타내는 label과 data binding되기 때문에 Property로 구현한다. code behind에서 값이 변경 될 때 마다 binding된 요소에 값을 전달 해주어야 하기 때문에 setter에서 인터페이스를 상속하면서 구현한 OnPropertyChanged를 사용한다.
* Count Property : 타이머에서 남은 시간을 변경하고 저장하기 위한 변수. Easy Mode는 시간 제한이 10초이므로 초기값을 10으로 설정. xaml의 남은 시간과 체력을 나타내는 label과 progressbar와 data binding되기 때문에 Property로 구현되고 Matched 와 동일하게 OnPropertyChanged를 사용한다. Count가 0이 되면 게임이 끝나기 때문에 0이 될 때의 게임 진행 코드를 setter에 넣어준다. (뒤에서 게임 실패 상황 시 다시 설명)
* tmr 타이머: 게임에서 사용할 wpf의 내장 라이브러리 timer 선언.
* 윈도우 창의 생성자 (창이 열리는 순간 호출, 코드 진행)

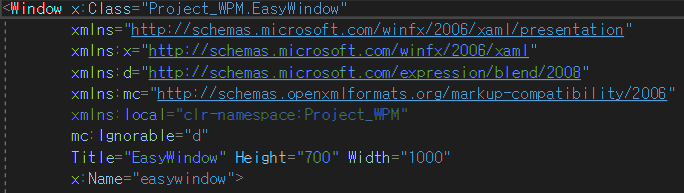


* btnSet() : 버튼들이 배치된 게임 판을 초기화 하는 메소드. 각 모드에 맞는 버튼 개수별로 게임판이 초기화되고, 같은 그림은 같은 tag를 부여해 같은 그림임을 인식하게 한다.
* tmr(타이머) 설정 : 타이머 간격을 1초로 설정하고 1초가 지날때마다 호출할 EventHandler tmr\_Tick()을 등록한다..



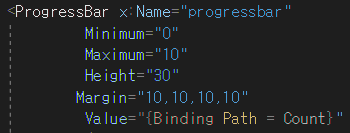
(1초마다 Count를 1씩 감소시키는 Event)

* DataContext설정 : xaml에서 Matched, Count Property를 binding 하면서 어디에 있는 element와 binding해야하는지를 찾을 때 DataContext를 사용한다. 위 코드로 윈도우 창 내의 element를 찾아 binding한다.
* 타이머 시작(게임 시작)

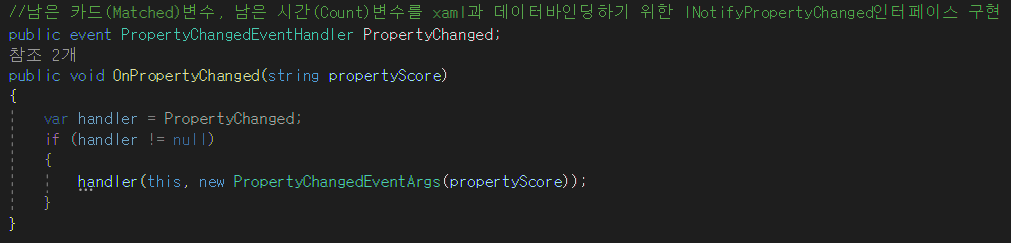




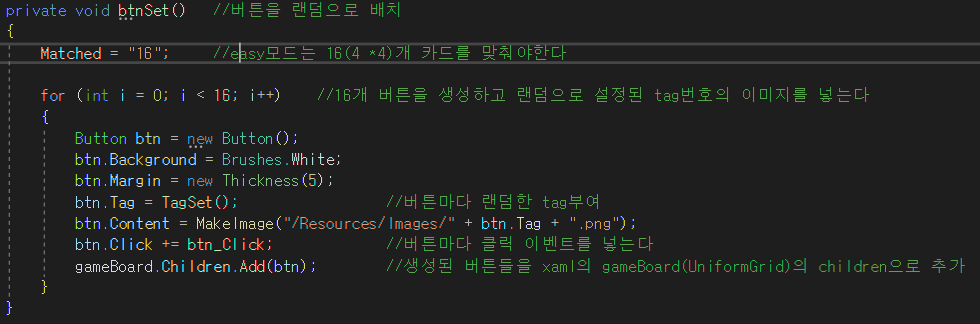
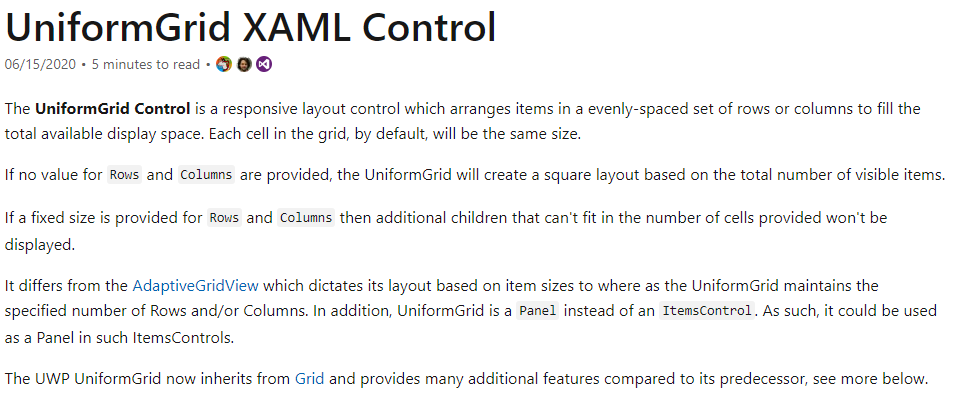
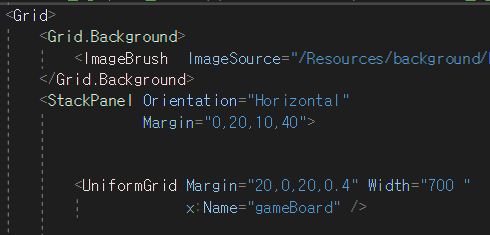


(xaml의 data binding code)

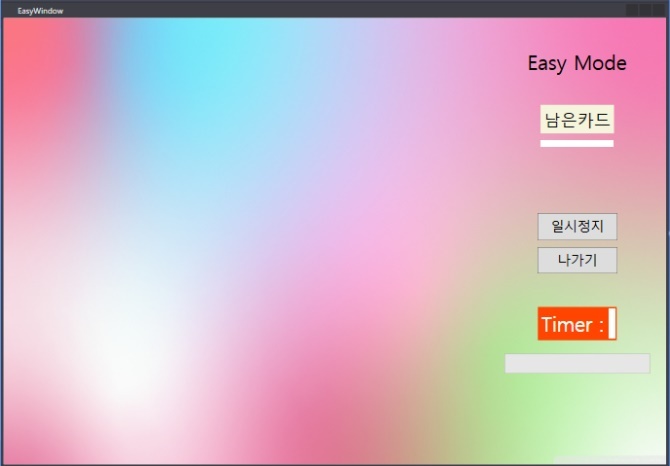
* Data Binding을 위한 INotifyPropertyChanged 인터페이스 메소드 구현



* PropertyChangedEventHandler를 선언하고, OnPropertyChanged메소드를 구현한다. 이는 Matched Property와 Count Property를 xaml과 binding 하기 위한 것으로, 각 변수들의 값이 변할 때마다 setter에서 사용된 OnPropertyChanged로 xaml에 변경된 값을 알려준다.
* btnSet() 메소드 구현

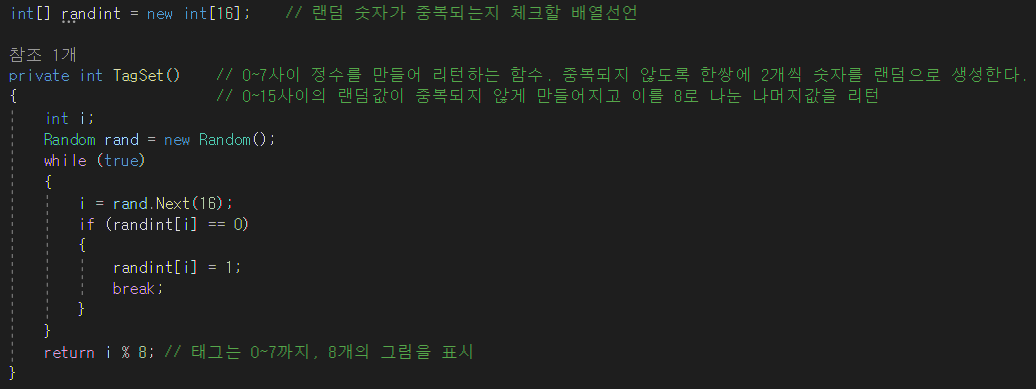
  

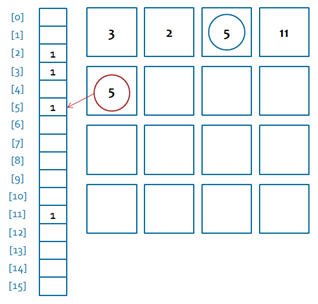
* UniformGrid : 자식객체들의 개수에 따라 균일하게 정렬되는 객체로, 행과 열의 속성을 지정하지 않으면 왼쪽에서 오른쪽으로 위에서 아래의 순서로 자동 정렬되는 xaml의 컨트롤. xaml에서 UniformGrid를 선언해 배치하고 btnSet()을 수행하는 과정에서 생성되는 버튼들이 이 UniformGrid의 자식 객체들로 추가된다.



(자식 객체들이 추가되기 전 UniformGrid만이 배치된 상태)

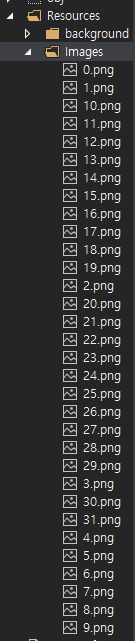
* 생성자에서 호출하는 btnSet()메소드. EasyMode에서는 맞춰야 하는 카드 개수가 4\*4, 16개이므로 Matched를 16으로 설정한다.
* 반복문 안에서 버튼의 개수만큼 버튼을 만들고 속성을 지정. (버튼 background, margin, tag번호, 이미지 contents, 클릭 이벤트)
* 버튼의 contents에 이미지를 넣기 위해 이미지를 불러오는 MakeImage()메소드를 구현, tag번호를 통해 이미지를 불러오기 위해 버튼에 TagSet() 메소드를 통한 tag번호 부여.
* 각 버튼의 속성을 지정한 후 gameboard(UniformGrid)에 자식으로 추가.
* 각 버튼에 tag번호(같은 tag번호를 가지는 버튼은 동일한 버튼으로 취급된다.)를 부여 할 TagSet() 메소드 구현

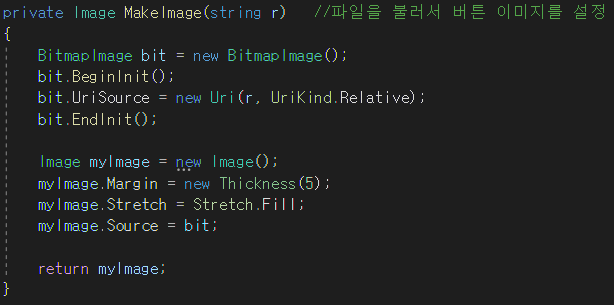


* randint 배열 : 숫자가 중복되지 않아야 하므로 이를 체크하기위한 배열. 그림과 같이 이미 숫자가 나왔는지 체크하고 이미 나온 숫자라면 버리고 다시 만들어 중복을 막는다.
* 랜덤하게 그림에 해당하는 숫자를 만들어 버튼에 저장 : 16/2, 8쌍의 그림을 표시하는 0~7사이의 랜덤한 숫자를 만들어서 리턴. 16개의 버튼에 그림이 한 쌍 씩만 있어야 하므로 랜덤한 숫자는 두개 씩만 만들어져야 한다. 이를 위해 0~ 15의 랜덤 숫자를 만들고 이를 8로 나눈 후 나머지를 리턴한다.

(Tag 속성 : type이 모든 클래스의 조상 클래스인 Object로 어떤 값도 저장이 가능하다. 그러므로 Tag는 필요한 정보를 저장하는데 사용이 가능하다.)

* 버튼에 이미지를 불러오는 MakeImage() 메소드 구현



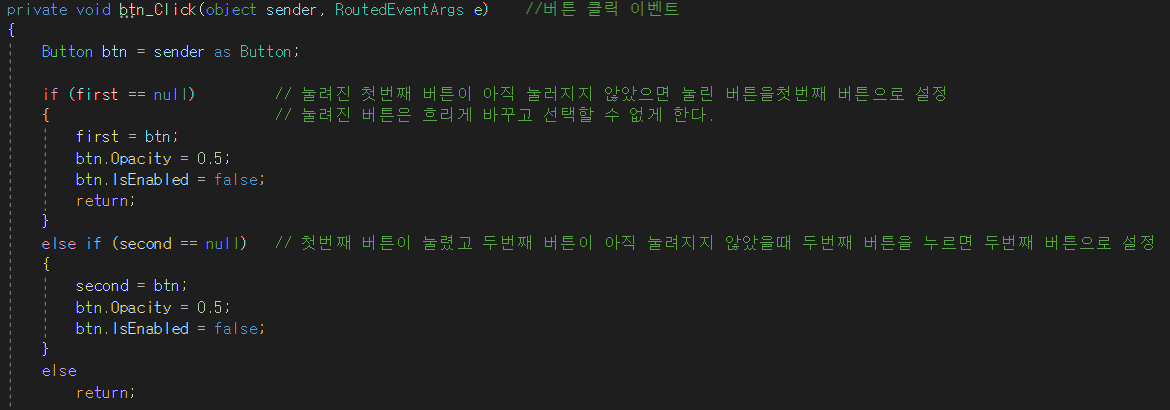


* TagSet() 메소드를 통해 각각의 버튼이 한 쌍 씩 묶여 tag 번호를 가진다. 이때 만들어진 tag번호를 이용하여 각 tag에 해당하는 이름을 가지는 파일 경로를 통해 파일을 불러 비트맵을 만든다.
* 이를 이미지로 변환하고 생성된 이미지를 리턴해 버튼의 contents로 설정한다. 결과적으로 같은 tag를 갖는 버튼의 쌍은 같은 이미지를 contents로 갖는다.

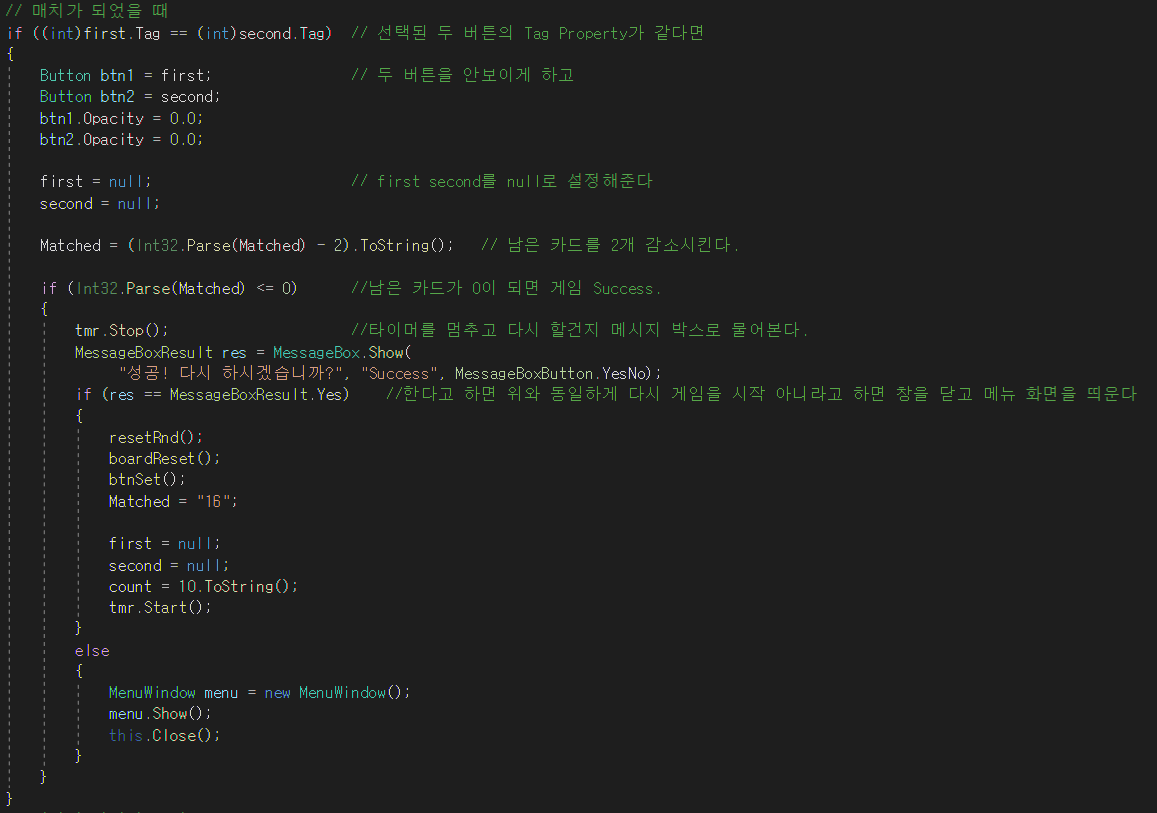
(버튼 이미지들은 Resources 폴더 안에 “숫자.png”형식으로 저장되어 있다. 즉 tag번호로 이름의 숫자를 접근하는 것이 가능하다.)

* 각 버튼에 등록되는 btn\_Click() 메소드 + 게임 성공

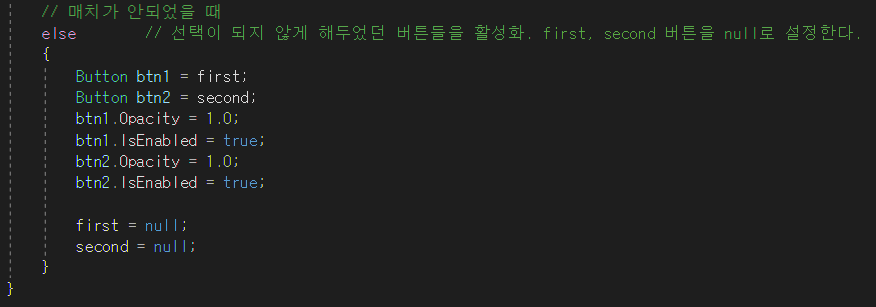




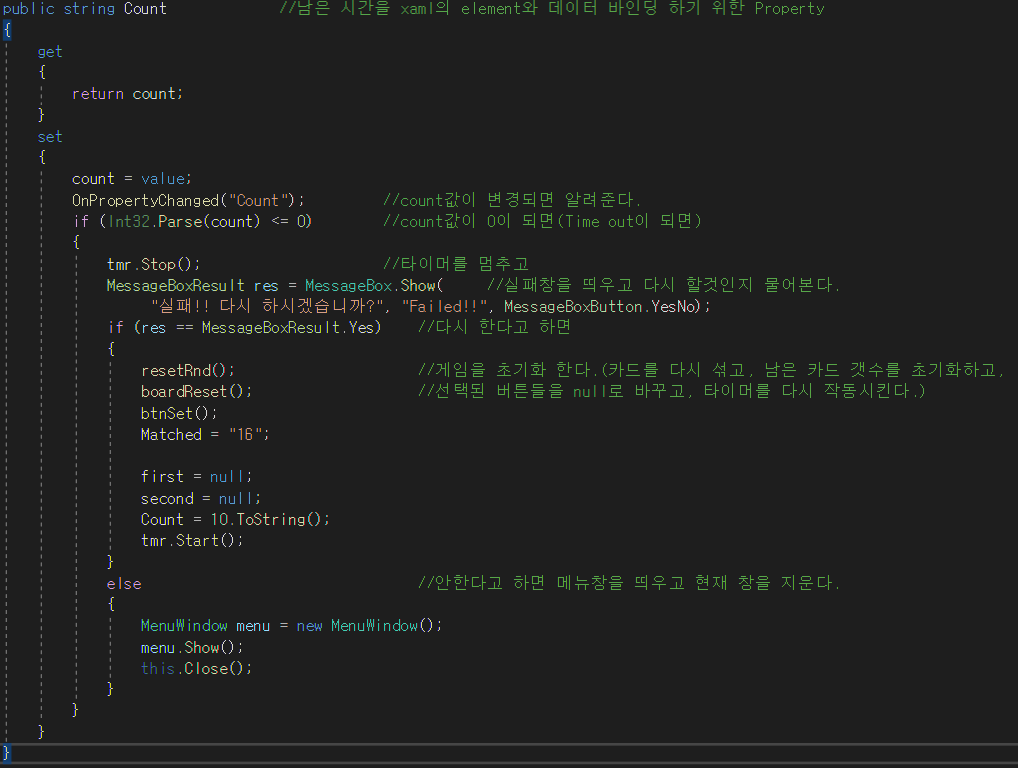
* 먼저 버튼이 눌리면 눌린 버튼을 받아와 첫번째에 눌린 버튼인지(first), 두번째에 눌린 버튼인지(second) 검사한다. first와 second는 null로 초기화 되어있고, first가 null이라면 처음 눌린 버튼이므로 first에 버튼을 저장하고 return한다. (아무것도 하지 않고 click 이벤트를 종료한다.)
* 눌린 버튼들은 Opacity속성을 변경시켜 흐리게 만들고, isEnabled속성으로 비활성화 시켜 두 버튼을 비교하기 전에는 재 선택할 수 없게 한다.

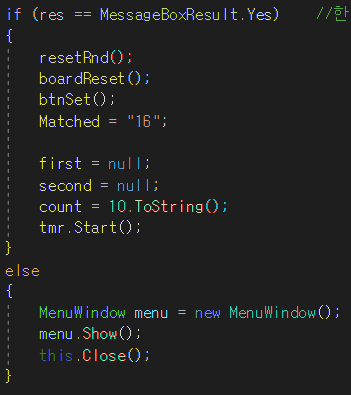


* 두번째 버튼이 눌리면 return하지 않고 계속 진행한다. 첫번째 눌린 버튼과 두번째 눌린 버튼이 저장된 first와 second의 tag 속성을 비교하여 두 버튼이 같은 이미지인지 체크한다.
* 같은 이미지인 경우, 두 버튼을 화면에서 안보이게 만들고 first와 second를 null로 초기화 시킨다. Matched를 2 감소시키고 Data Binding을 통해 xaml의 남은 카드 label이 동시에 변경된다.
* Matched 값이 감소될 때 마다 0이 되었는지 검사한다. 타이머가 0이 되어 게임이 종료되기 전에 Matched가 0이 되었다면 게임이 성공한 것으로. 타이머를 멈추고 성공 메시지박스를 출력한다.
* 메시지박스에서 사용자에게 다시 게임을 할 것인지의 여부를 묻는다. 그렇다고 선택하면 게임을 초기화 시키고 타이머를 재시작해 게임을 재개하고, 아니라면 화면을 끄고 메뉴 화면을 Show()한다.



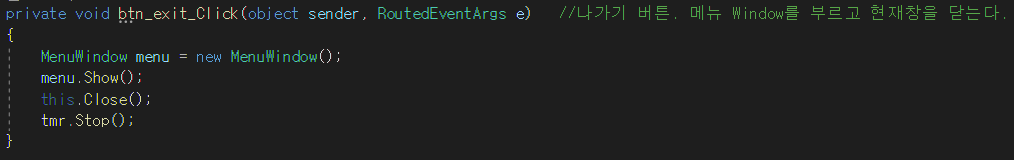
* 눌린 두 버튼의 tag 번호가 다를 시 다른 그림을 선택한 것이므로 비활성화 되어있던 버튼들을 원래대로 되돌리고 first와 second를 다시 null로 초기화한다.
* 게임 실패
* Matched가 0이 되기 전에(카드를 전부 맞추기 전에) 시간이 지나 Count가 0이 되면 게임이 실패한 것으로, Count Property의 setter에서 Count 값이 변경될 때마다 0보다 작은 지 확인한다.



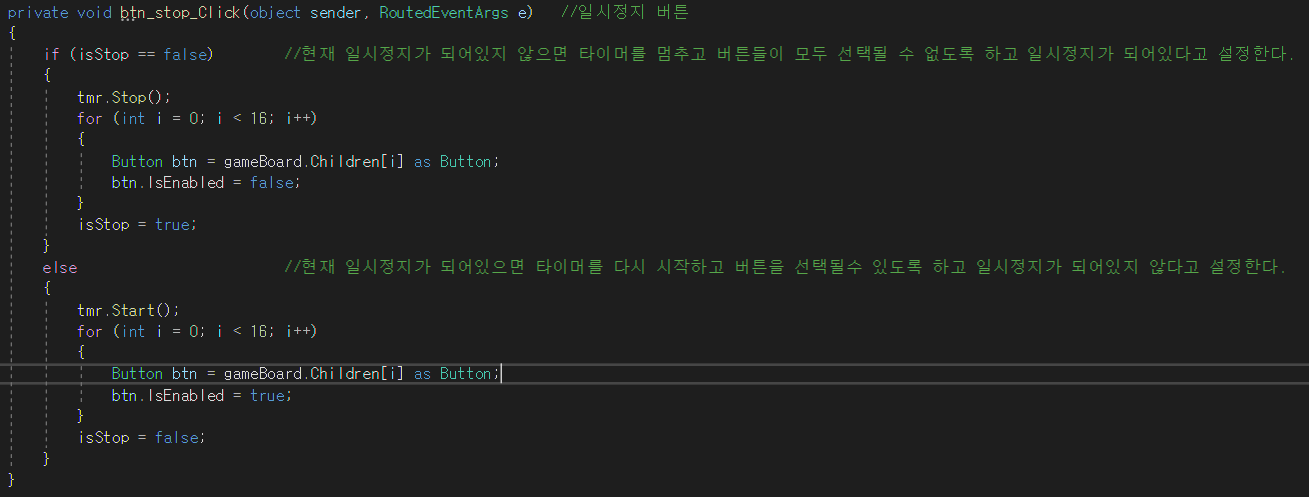
* 실패하면 남은 시간이 음수가 되는 것을 방지하기 위해 타이머를 멈추고 실패했다는 메시지박스를 출력한다. 메시지박스에 대한 사용자의 선택에 따른 게임의 진행은 성공시와 동일하다.
* 게임 재시작과 그만두기
* 사용자가 게임을 재시작 하고싶어 할 때 게임을 초기화한다.
* resetRnd() 메소드는 randint[] 배열을 초기화하고, boardReset() 메소드는 uniformgrid의 children으로 있던 버튼들을 지운다. (clear)
* btnSet() 메소드를 호출해 버튼을 새롭게 만들고 재배열한다.



* Matched와 Count을 초기화하고 현재 저장된 first와 second를 null로 초기화한다.
* 모든 게임 초기화를 마치면 타이머를 재시작 시켜 게임을 다시 시작한다.
* 게임종료시 메뉴창을 show()시키고 현재 창을 닫는다.
* 나가기 버튼과 일시정지 버튼



* 나가기 버튼을 눌렀을 때 호출되는 메소드 : 메뉴창을 show()시키고 현재창을 닫는다. 타이머를 종료 시키지 않으면 타이머가 계속 진행되어 게임 실패 메시지박스를 출력하므로 타이머를 멈춘다.
* 일시정지 버튼을 눌렀을 때 호출되는 메소드 : isStop 변수로 현재 상태가 일시정지 상태인지 아닌지 먼저 확인한다. 일시정지 상태가 아닌 상태에서 버튼이 눌렸다면 타이머를 멈추고 모든 버튼들이 눌리지 않도록 한 후 일시정지 상태라고 변경한다. 일시정지 상태에서 버튼이 눌렸다면 버튼들을 다시 활성화 시키고 일시정지 상태임을 해제한다.

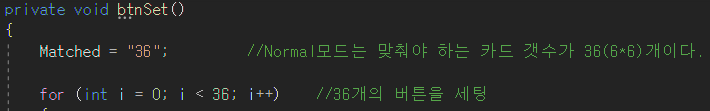


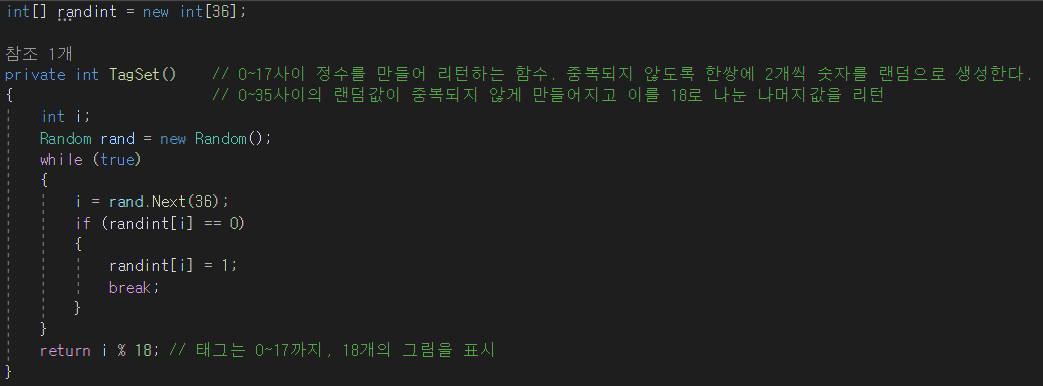
* Normal Mode와 Hard Mode
* Normal 모드와 Hard 모드는 난이도 조절을 위한 맞춰야 하는 카드 개수(Matched)와 정해진 시간 제한(Count)이 달라 이를 위한 코드가 다른 것을 제외하고는 게임이 동일하게 작동한다.

(Normal Mode)





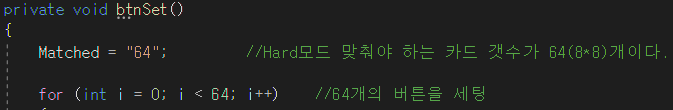


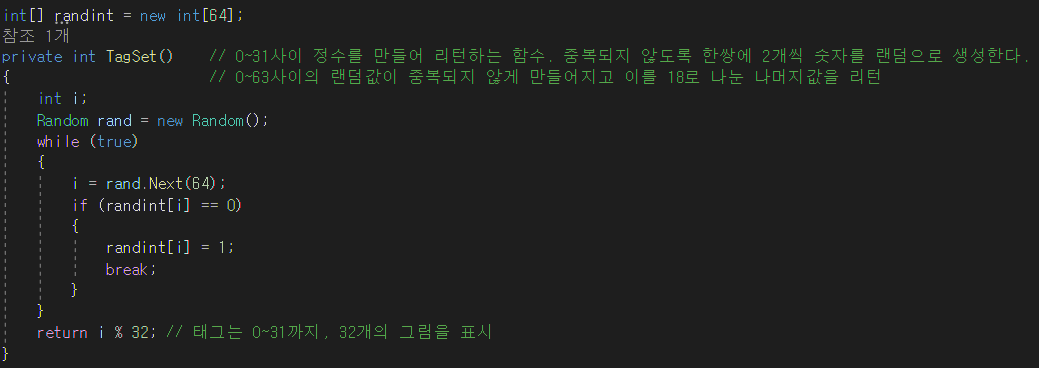


(Hard Mode)









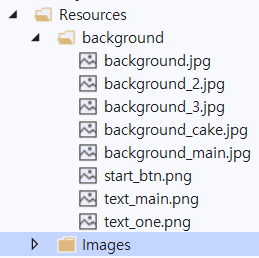
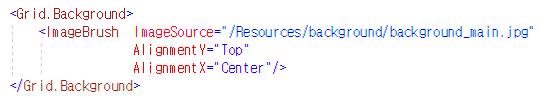
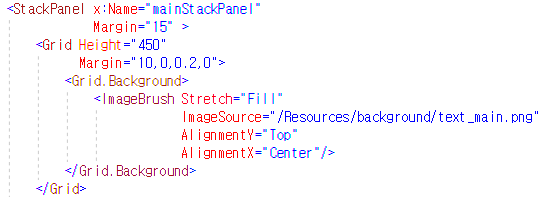
1. **part별 프로젝트 구현 및 사용 알고리즘(part별 코드 소개)**

**– 주윤서(UI 및 타이머 구현)**

**(1) 화면 구성**

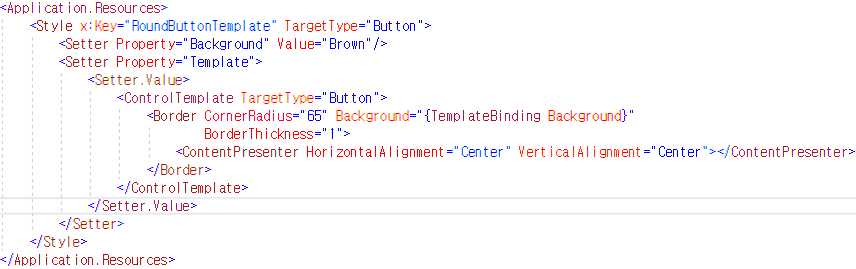
|  |  |  |
| --- | --- | --- |
|  |  |  |
| **초기시작화면(StartWindow)** | **메뉴화면(MenuWindow)** |
|  |  |  |
| **EasyMode화면(EasyWindow)** | **NormalMode화면 (NormalWindow)** | **HardMode화면(HardWindow)** |

**(2) 초기 시작 화면(StartWindow.xaml)**



**1. 전체화면 :** 전체 Panel을 Grid 형태로 설정 하여 Grid.Background의 ImageBrush속성의 ImageSource 경로 설정을 파일 내부의 Resources background 폴더로 설정하였다.

**2. 게임의 제목** : ImageBrush에 들어가는 이미지의 가로 세로 비율을 유지하기 위해 Stretch를 Fill로 설정하여 지정한 화면에 해당하는 비율 만큼으로 설정 할 수 있다.



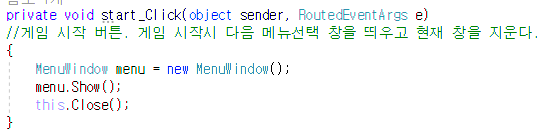
**3-1. 버튼 이미지** : 버튼의 Style을 위해 background 이미지를 설정하였다. 그리고 Button의 Control Template 속성을 새 Control Template으로 설정하여 전체 템플릿을 수정하여 새로운 Button style로 변경하였다.



**3-1(cont.)** : TemplateBinding의 태그 확장기능 사용으로 앞서 만든 RoundButtonTemplate을 binding시켜 이미지에 적용되도록 하였다.

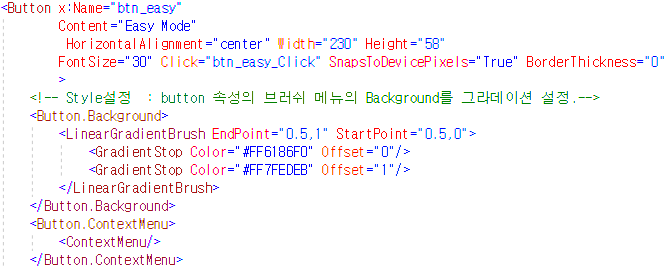
**4. 화면의 배치** : 전체 Grid 패널 내부에 StackPanel을 Vertical 속성과 Margin을 적절히 사용하여 원하는 위치에 버튼과 이미지가 배치되도록 하였다.

**(2-1) 초기시작화면(Startwindow.xaml.cs)**



**1.** 게임 시작버튼 클릭 시 : Start\_Click 이벤트를 발생시켜 Menuwindow창, 즉 다음 창을 보여준다. 그리고 현재 화면을 닫고, 다음 화면만 보일수 있게 Show()와 Close()를 사용하였다.

**(3) 메뉴화면(MenuWindow.xmal)**

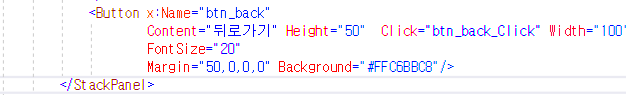


**1. 3가지 모드의 버튼 :** 각각 Easy모드, Normal모드, Hard모드 3가지의 버튼을 생성한다. 생성한 버튼은 Stackpanel의 Verical 속성으로 차례로 배치한다.

**(1. 3가지 모드의 버튼 cont.)** : 왼쪽그림과 같이 btn\_easy 버튼의 속성에서 버튼의 background를 LinearGradientBrush클래스를 사용하였다. 처음과 끝부분만 Gradient를 설정하여 offset을 각각 0과 1로 설정하였다. (btn\_normal버튼과 btn\_hard버튼 역시 마찬가지로 진행하였다)

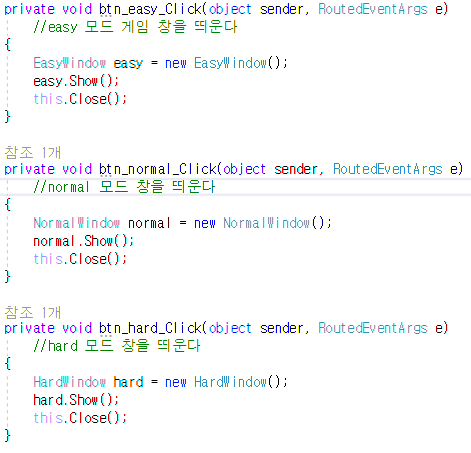
**2. 게임의 제목과 배경** : Resources 파일에서 각각 해당하는 이미지에 맞추어 경로를 설정하였다.

**3. 뒤로가기 버튼**



- 뒤로가기 버튼을 Click할 때 이벤트를 설정하고, StackPanel로 Hardmode버튼과 함께 Horizontal하게 배치하였다.

**(3-1) 메뉴화면(MenuWindow.xmal.cs)**

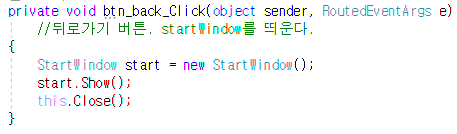


**1. Easymode button 클릭 시** : easymode 창을 띄우기 위해 EasyWindow 클래스의 easy 객체를 선언 후 내장 함수 Show()를 사용하여 easy window를 띄운다. 현재 메뉴 Window를 닫기 위해 this.Close()를 실행한다.

**2. Normalmode button 클릭 시** : normalmode 창을 띄우기 위해 normalWindow 클래스의 normal 객체를 선언 후 Show()를 사용하여 normal window를 띄운다. 현재 메뉴 Window를 닫기 위해 this.Close()를 실행한다.

**3. Hardmode button 클릭 시** : hardmode 창을 띄우기 위해 HardWindow 클래스의 hard 객체를 선언 후 Show()를 사용하여 hard window를 띄운다. 현재 메뉴Window를 닫기 위해 this.Close()를 실행한다.

**4. 뒤로가기 버튼 클릭 시 :**



뒤로가기 버튼 클릭 시 뒤의 창인 StartWindow 즉 게임을 시작하는 초기화면으로 돌아가기 위해 StartWindow class의 start 객체를 선언후 Show()를 하여 초기화면 창을 열고 현재 창을 닫는다.

**(4) 이지 모드 화면 (EasyMode.xmal)**

**1-1. 화면배치-배경** :

검은색 배경을 Grid로 배치하고 앞 window와 동일하게 Grid.Background를 활용하여 ImageSource의 경로를 해당 파일 내부의 이미지로 설정하였다.

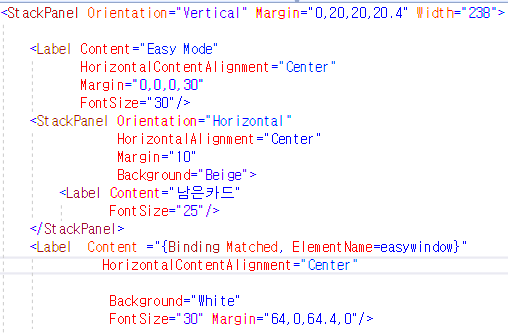




**1-2. 화면배치-게임 화면** : 오른쪽 그림과 같이 4x4 행렬이 필요하다. 따라서 UniformGrid에 자식객체를 하나씩 추가하면 자동으로 정렬 되어 자식객체를 생성한다.

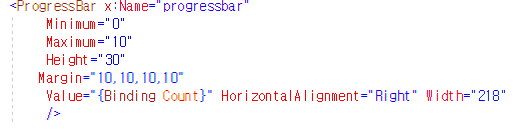


**1-3. 화면배치** –



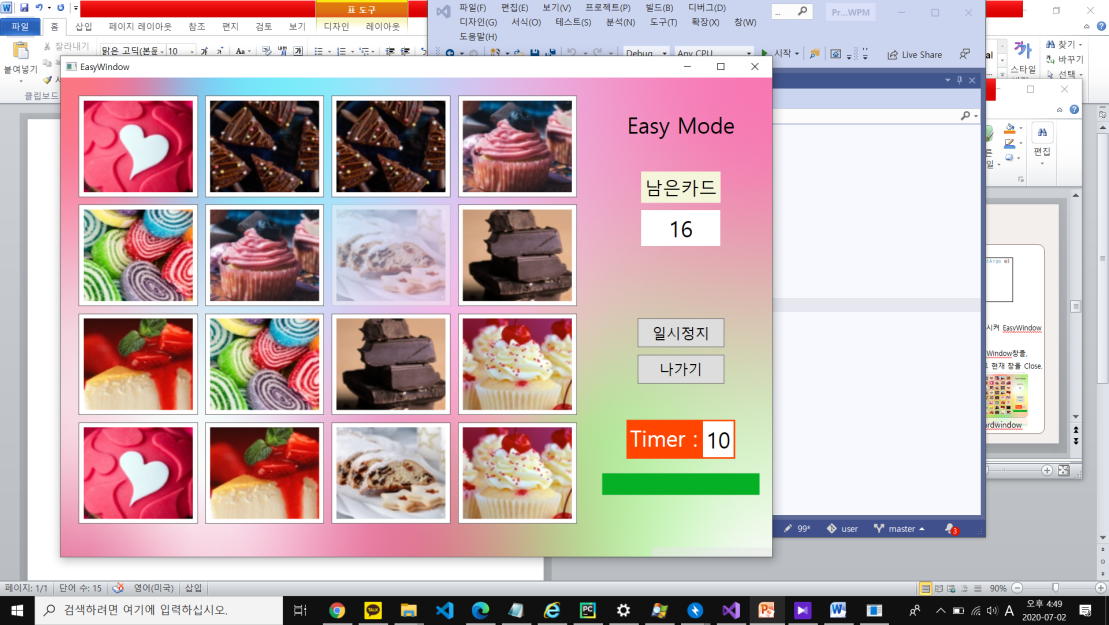
오른쪽의 주황색네모 내부의 modeLabel, 남은개수 label, 숫자 label, 일시정지 버튼, 나가기 버튼, Timer라벨, progressbar를 새로운 StackPanel로 감싸 Orientation을 Vetical설정하여 배치하였다. Timer라벨은 별도의 StackPanel을 또 삽입하여 timer의 이름과 숫자 label이 표시될 수 있게 하였다. 각 라벨과 버튼은 Alignment를 Center로 설정하였다.

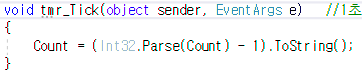
**2. progressbar**



- 게임의 시간 초를 알려주는 progressbar 속성을 사용하였다. 현재 Easymode window이므로 제한시간은 10초로 설정하여 Maxmum 값을 10으로, timeout은 0초이므로 Minimum값을 0으로 설정하였다. Value값은 데이터 바인딩으로 아래 cs파일에서 설명하도록 하겠다.

**(4-1) 이지 모드 화면 (EasyMode.xmal.cs.)**

**1. 게임 시작 시 Timer의 동작**



- 게임을 시작하면 EasyWindow의 생성자에서 게임 판이 초기화됨과 동시에 타이머가 시작된다.

- timer를 사용하기 위해서 Dispatcher Timer클래스를 사용한다.

- tmr\_Tick함수는 count 값을 1초씩 감소시키는 함수이다. Count 변수는 label과 연동되는데, label의 숫자는 String으로 취급되므로 문자열을 숫자로 변환하는 방법이 필요하다. 그러므로 Int32.Parse()를 이용하여 문자열을 숫자로 변환하여 1씩 감소 시킨 후 다시 Tostring을 통해 string으로 변환한다.

- EasyWindow 생성자에서 타이머를 시작하기위해 DispatcherTimer 클래스의 속성들을 사용한다.



- dispatcherTimer\_Tick 이벤트 처리기가 tmr의 Tick이벤트에 추가된다.

- tmr 객체를 선언하여 DispatcherTimer 내장 함수를 불러온다. timer가 1초씩 줄어들도록 하기 위해 tmr.interval을 사용할 수 있다. 기본값은 00:00:00이므로 tmr.interval을 (0,0,1)로 하여 1초로 설정한다.

- 게임이 시작되자마자 타이머가 시작되도록 tmr.Start()함수를 호출한다.

**2. 게임 시작 시 Count와 Binding된 Progressbar와 Timer의 동작**



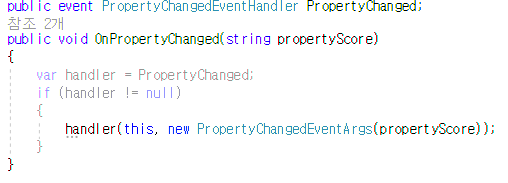
- 다음은 앞 xaml timer의 남은 시간을 데이터 바인딩 하기 위한 Count의 Propety이다.

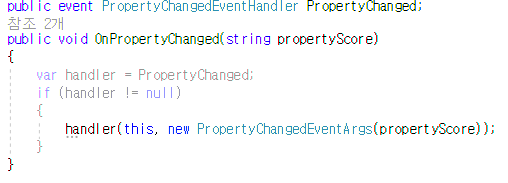
- Count는 남은 시간을 설정하는 변수이다. 이 값을 앞의 xmal에서 progressbar와 남은 시간에 대해 데이터바인딩 할 것이다.



- count값이 바뀌었다는 신호를 보내기 위하여 Easywindow클래스에 INotifyProperty Changed를 상속 받는다.

- INotifyPropertyChanged를 사용하기 위해서는 INotifyPropertyChanged 내에 있는 가상 함수를 오버라이드해야 하고 인터페이스를 구현해야한다.

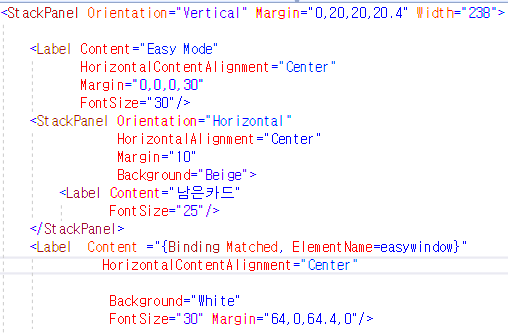


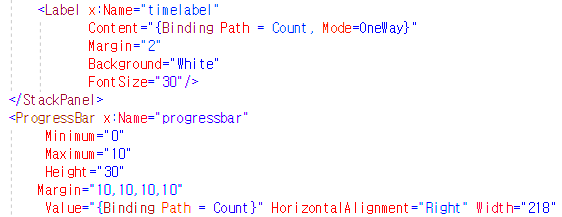


- OnpropertyChange메소드를 통해 propertyscore에 해당하는 데이터가 변화가 생길 때 마다 이벤트를 발생시킨다.



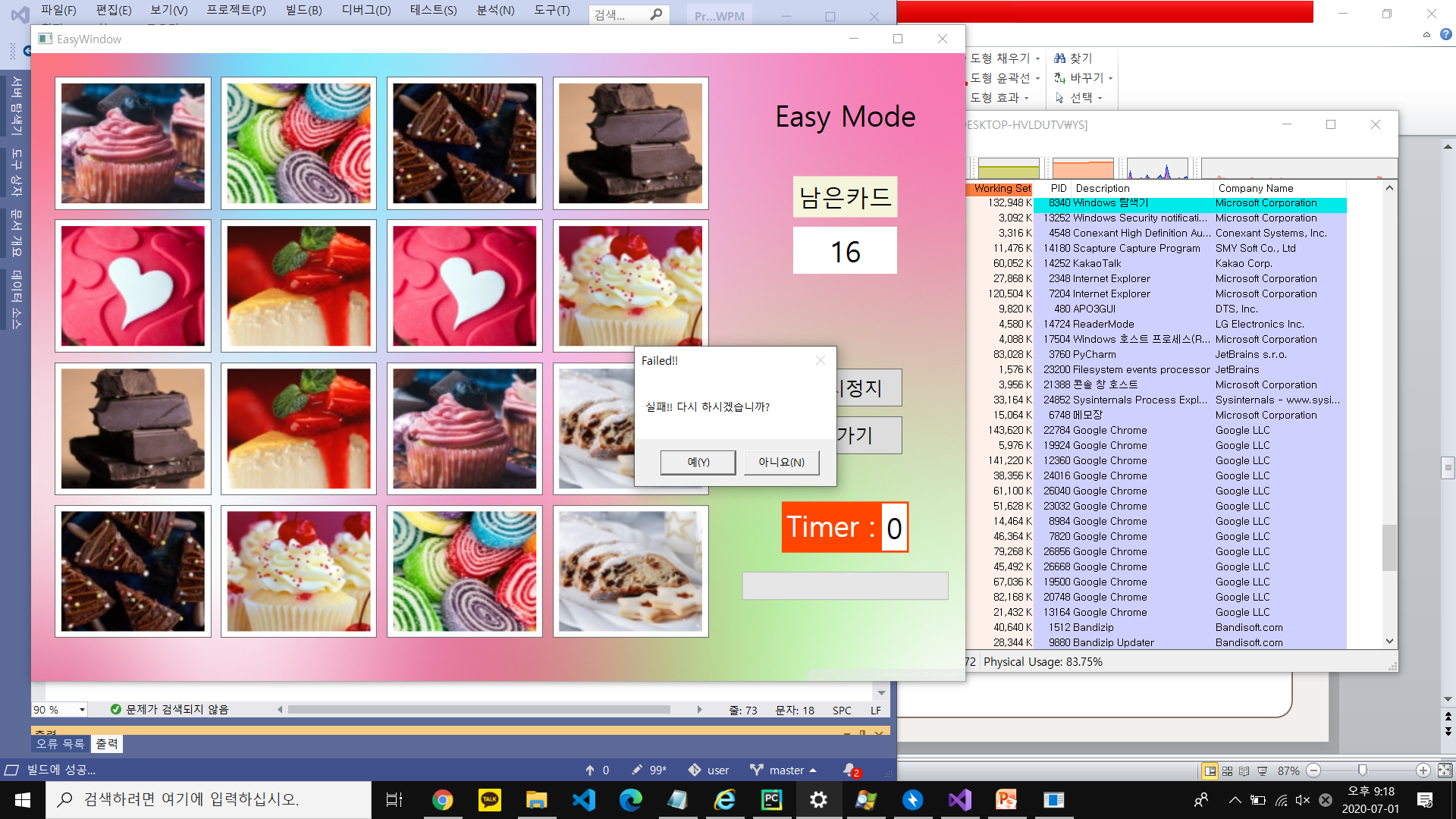
- 즉 setter에서 OnpropertyChanged에 count를 string으로 설정하여 count값이 변경되면 데이터 값이 전달되도록 설정한다.





-이제 앞 xaml에서 남은시간에 해당하는 timelabel과 progressbar에 Value값에 Path= count값으로Data binding시킨다.

- 게임이 시작하면 Timer와 progressbar과 함께 연동되어 줄어드는 모습을 확인할 수 있다.

**3. 게임 실패 시 동작**

- 게임이 실패 될 경우 실패했다는 알림창을 팝업하기 위해 MessageBox 클래스를 사용한다.

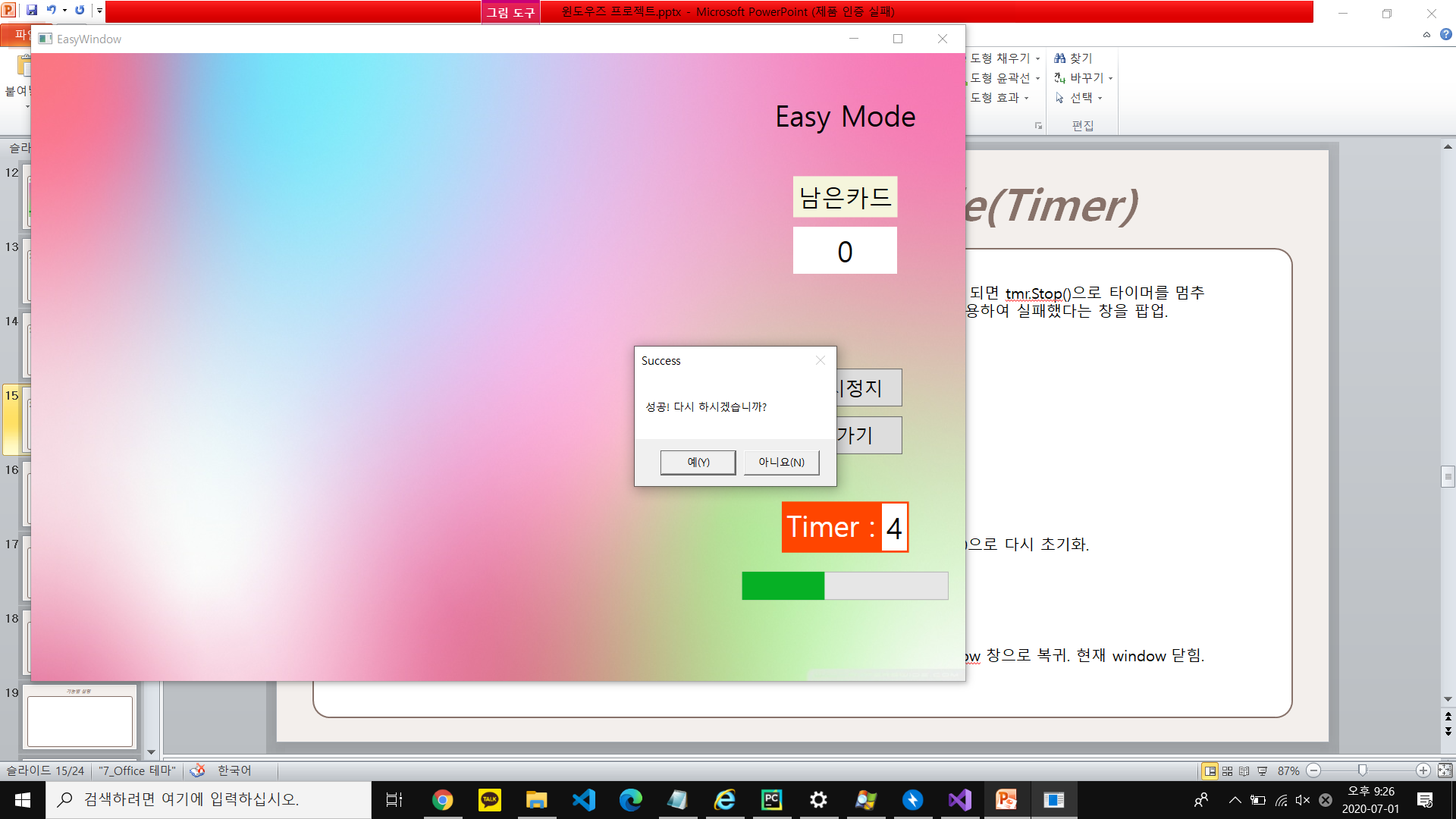
- 게임이 실패 된 경우는 Count값이 0보다 작은 경우 이므로, 이 때 타이머를 Stop()시키고, MessageBox를 출력한다. 실패 창을 띄운 후 재시작할 것인지 묻는다.

****

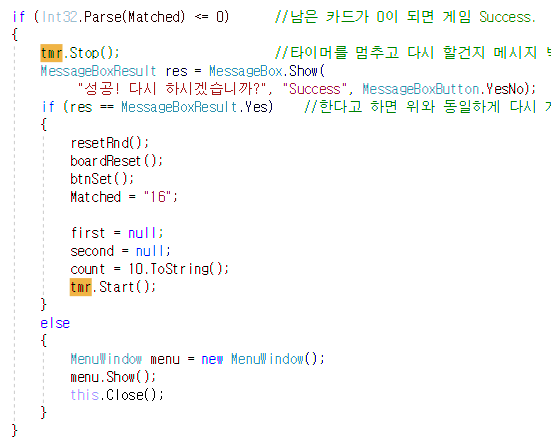
- 게임을 다시 할 경우, 게임을 초기화 시키고, Count변수를 다시 10으로 변경시킨다.

****

- 게임을 그만 할 경우, menu window 창으로 다시 이동하므로 Menuwindow에서 새 객체를 생성한후, 메뉴 창을 Show()하고, 현재 게임 창은 Close()한다.

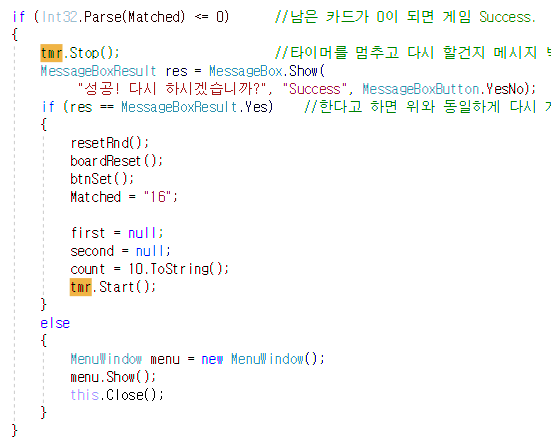


**4. 게임 성공 시 동작**



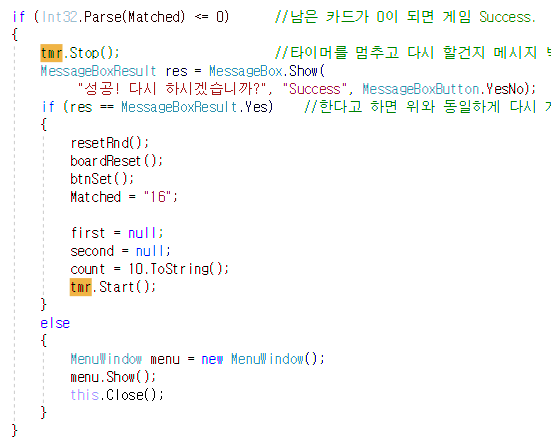
- 게임이 성공 된 경우 남은 카드가 0이 된 경우 이다. Matched는 카드가 매치된 개수를 저장하는 변수이다. 즉 Easymode는 16개의 카드이므로 16개가 0보다 작게 되면 게임이 성공하게 된다.

- 게임이 성공한 경우 역시 타이머를 멈추고, 마찬가지로 재시작 할 것인지에 대해 메시지 박스를 팝업 한다.



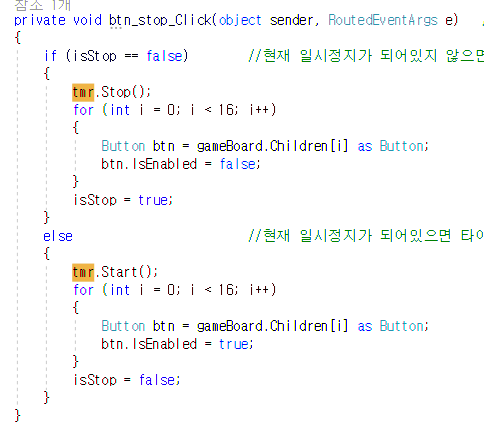
- 만약 재 시작 한다면 게임을 초기화 하고, count변수를 다시 10초로 설정한다.

- 타이머 역시 다시 시작하여 게임을 다시 시작할 경우 같이 동작하도록 한다.



- 재시작 하지 않는다면, 실패 시 ‘아니오’와 동일하게 Menuwindow 창을 Show()하고, 현재 창을 Close()한다.

**5. 일시 정지 시 동작**



5-1) 일시 정지 버튼을 클릭하게 된 경우

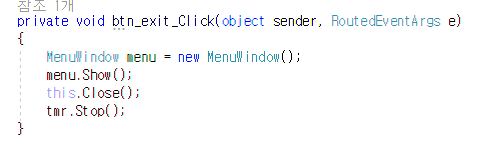
- isStop 변수는 Bool타입으로 만약 일시정지버튼을 누른다면(= 현재 일시정지가 되어있지 않음) false인 상태에서 타이머가 중지된다.

- 모든 버튼이 눌리지 않게 설정되고, isStop변수를 마지막에 true로 설정한다.

5-2) 현재 일시정지 버튼이 눌린상태에서 다시 눌렸을 경우(=일시정지를 풀은경우)

- 일시정지가 풀렸으므로 다시 현재 타이머를 시작시키고, 버튼이 다시 눌리도록 설정된다.

**6. 나가기 버튼 클릭 시**



- 나가기 버튼 클릭 시 btn\_exit\_Click 이벤트가 발생하면서 메뉴 창으로 이동하고, 현재 창을 닫는다.

**7. NormalWindow, HardWindow**

- 앞서 작성한 Easymode와 거의 동일하게 동작한다.

- normal window는 6x6으로 총 36개의 버튼 즉 36개의 카드가 존재하며, hard mode는 8x8 = 64개의 카드가 존재한다

- 게임의 난이도 조정을 위한 count값의 변수 설정과 , progressbar의 maximum값이 서로 다르다.

( Normal mode = 20 sec, Hard mode = 50 sec)

1. **제작 시 문제점 및 해결방안**

|  |  |
| --- | --- |
| **문제1** | 게임을 끝내지 않고 화면 창을 나가면 ‘실패하셨습니다’ messagebox 계속 팝업. |
| **원인** | 창을 숨기는 Hide()를 사용, 나가기 버튼을 눌렀을 때 타이머가 정지되지 않아 Count가 0이 되어 게임이 실패됨. |
| **해결방안** | 창을 Close()으로 종료 시키고, 게임을 나갈 때 타이머를 멈춰 게임 실패 창이 뜨는 것을 막음. |
| **문제2** | Count와 Matched Property가 데이터 바인딩을 구현해도 xaml의 label에 출력되지 않음. |
| **원인** | 화면에 보이는 label은 string type만을 출력 가능. Int type으로 구현 시 출력이 불가능. |
| **해결방안** | Property를 string type으로 구현하고 code behind에서 값 변경 시 int type으로 type casting해 변경 |
| **문제3** | Progressbar와 타이머가 함께 움직이지 못함 |
| **원인** | Xmal은 뒤 cs파일에서 작동하는 데이터 값을 바로 전달 받지 못함. |
| **해결방안** | Count 변수 값을 데이터 바인딩 시켜 함께 초가 감소하도록 함. |
| **문제4** | 두번째 버튼을 클릭하고 매칭이 되지 않았을 때 다시 새롭게 선택한 버튼과 매칭이 되어 검사를 시도함 |
| **원인** | first와 second 가 null이 아니기 때문에 버튼이 눌릴 때마다 매칭 검사를 시도 |
| **해결방안** | 매칭 실패 시 first와 second 를 null로 초기화 해서 새롭게 first 버튼을 저장 |
| **문제5** | 게임을 실패 한 후 다시 시작하면 화면전체가 멈춤 |
| **원인** | 다시 시작할 경우 게임이 다시 작동하는 초기화 단계가 없음. |
| **해결방안** | 초기화 시키는 코드를 삽입, 타이머는 각 모드에 맞는 시간으로 초기화 후 다시 시작하도록 함. |

1. **향후 보완 사항**

1. 데이터 베이스를 활용해 계정을 생성할 수 있도록 보완. 각 계정으로 플레이 할 수 있도록 보완.

2. 데이터 베이스를 활용해 계정 별 성공 시간을 랭킹으로 보여줄 수 있도록 보완.

3. 게임 내부 배경음악 추가.

1. **참고 문헌 및 사이트**

* <https://docs.microsoft.com/ko-kr/visualstudio/designers/getting-started-with-wpf?view=vs-2019>
* 마이크로소프트 wpf doc 설명서
* <https://www.tutorialspoint.com/wpf/>
* wpf 튜토리얼
* <https://mrw0119.tistory.com/category/%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D/C%EC%83%B5>
* c# 티스토리