<구현 보고서>

B993185 유민경

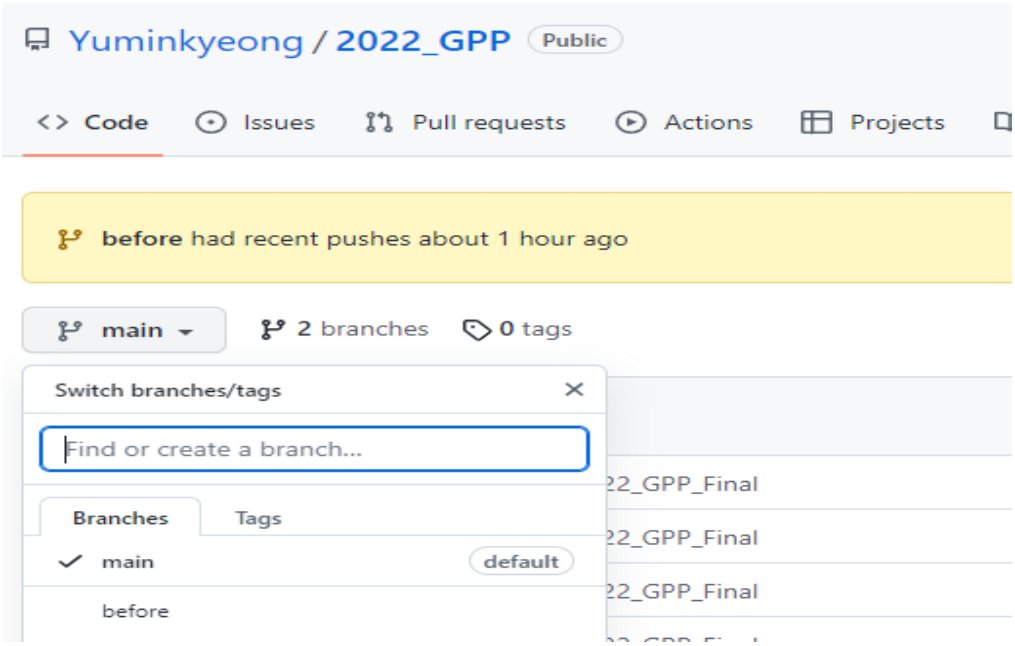
---

*구현 링크: https://github.com/Yuminkyeong/2022_GPP.git

//영상 및 asset 사용으로 인해 너무 느리게 업로드 되어 전체 코드는

github 링크에 올려 공유하게 되었음

branch에 refactoring 전(before branch)/ 후(main) 로 나누어 두어 확인할 수 있음

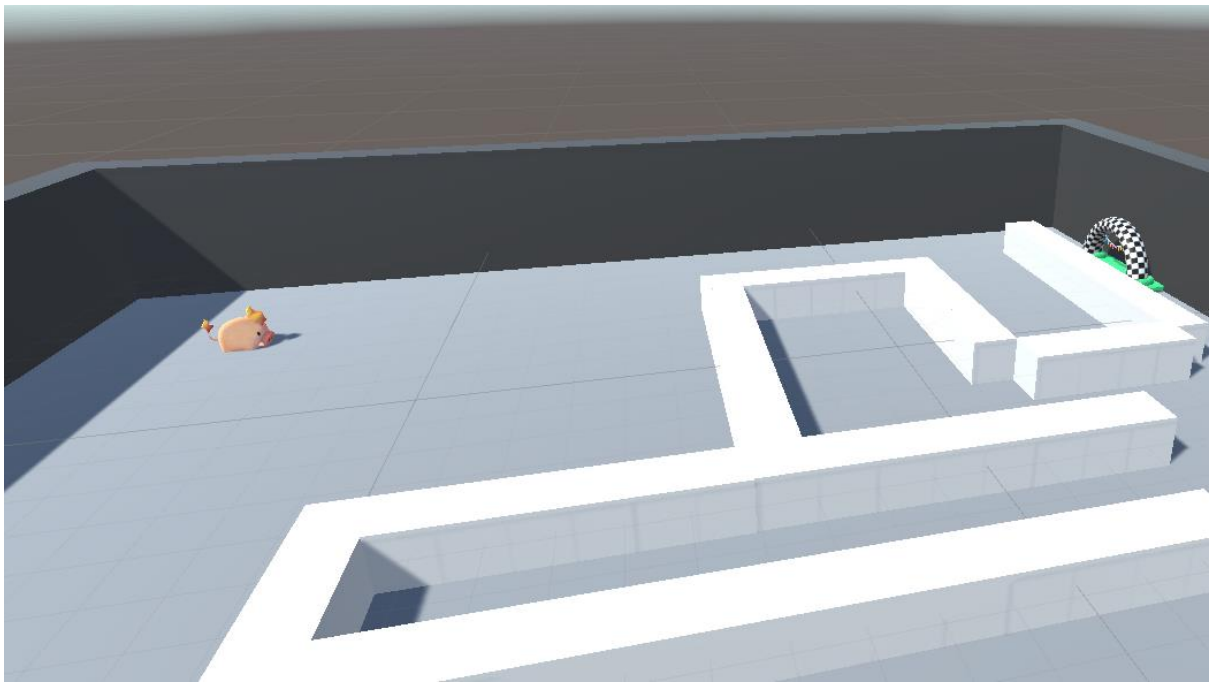


1. 구현 계획

1) 스토리

: '최소로 이쑤시개를 옮겨 강아지 만들기' 퍼즐에서 착안한 게임으로, 벽을 최소한으로

부시고 일반 벽을 최소한으로 생성해서 빠른 시간 안에 게임 목표 지점에 도달하는 게임을 만들고자 함.



2) 길찾기 알고리즘과 Command Pattern

: A* 알고리즘을 사용하고, 벽을 생성, 파괴, 플레이어의 이동 등과 같은 플레이어 중심의 입력은 Command Pattern을 활용해 작성.

GridManager.cs; 카메라에 보이는 영역을 그리고, 해당 공간 안에서 게임 플레이 진행이 가능하도록 함. 길찾기 알고리즘과 이에 따른 player의 움직임을 해당 함수에서 관리

PlayerScript.cs; Undo, Redo, 마우스 왼쪽으로 이동, 마우스 오른쪽으로 일반 벽 생성

CommandManager.cs; Command Pattern을 전체적으로 관리하는 Manager로 Add Command, Execute, Undo 기능들을 수행

CreateWallCommand.cs; 일반 벽을 생성하는 Command로 마우스 오른쪽으로 생성할 수

있고, 생성함에 따라 일반벽을 설치하여 골인 지점까지의 단거리를 설계할 수 있음

Wall2Command.cs; 부술 수 있는 벽으로 흰색->초록색->빨간색->파괴 순으로 변하며, 파괴 시 벽이 있던 지점을 넘어 이동할 수 있음

ReplayCommand.cs; 게임 성공 시 replay가 자동으로 재생되고, 게임이 종료됨

UndoCommand.cs; 게임 뒤로 가기 기능

3) Object Pool, Singleton Pattern

: 자주 생성, 파괴가 이뤄지는 player의 공격을 구현

Bullet.cs; 플레이어의 공격 구현

Object Pool.cs; map의 오브젝트 풀링을 관리하는 cs로 instance를 static으로 선언하고 Queue를 통해 Bullet.script의 총알 생성/삭제를 관리

4) Observer Pattern

: Player의 위치에 따라 동작되는 장애물로, player가 subject가 되어 동작시키는 것으로 Observer Pattern에 맞다고 생각하여 이처럼 구현

FallingBox.cs; 공중에 떨어지는 장애물이 생성 및 동작됨

PlayerScript.cs; 해당 스크립트에서 event Action myEvent를 선언하여 사용

5) State Pattern

; 일정 거리를 배회하는 Monster AI로 player와 가까워졌을 때 Player를 쫓아와서 빠른 play를 방해함

RangeCheck.cs; player와 enemy 사이의 거리를 측정하고 이에 따라 enemy의 상태를 변경시킴

Enemy1.cs, Enemy2.cs; 플레이어 탐지 거리, 이동 속도 등이 다른 몬스터 들로 player가 일정 범위 내에 들어왔을 때 player을 방해하러 움직임

➔ 시간이 될 경우 object pool과 함께 사용하여 player의 목숨 개수를 표현하고 공격력이 다른 enemy1, enemy2를 구현할 예정

6) 기타 스크립트

BtnManager.cs: clear, game over 시 retry or exit 할 수 있도록 기능을 제공

ScoreManager.cs; 게임의 점수를 관리하는 스크립트로 (맵 전체 블록 개수-생성 블록 개수-파괴한 블록 개수)를 계산한 뒤 가중치를 부여하여 점수 표기

TimeManger.cs; 게임 플레이 제한 시간 제어

2. 구현 시 고민이 되었던 부분

1) Observer Pattern으로 떨어지는 장애물을 구현했을 시, Player의 움직임에 따라 달라지는 장애물이므로 이렇게 구현하고자 하였음. 이러한 관점에서 보았을 때, State Pattern 역시 Player의 위치에 따라 Object의 움직임이 달라지는 것인데 굳이 나눠서 Pattern을 적용하는 것이 의미 있을지 고민하게 됨.

2) Object Pool로 맵 형성을 하려고 했으나 알맞은 형태인지 고민하게 됨. 자주 생성/파괴된다는 특성을 가지고 있으나, 개수가 제한되어 있는 리듬 게임의 노트나 위에서 프로그래밍했던 것처럼 장전 개수가 정해져 있는 총 게임의 경우가 더 알맞지 않을까 라는 생각이 들었음.

3) Command Pattern의 근본적인 문제를 고민하게 됨. Command의 개수가 늘어날수록 script가 계속해서 늘어나야 하는데 이것이 좋은 방법인지 잘 모르겠다는 생각이 들었음. 하지만 Undo, Replay에 있어 좋은 방식임에는 틀림없어 이대로 적용하게 됨.

4) Event Queue의 추가 고민. 게임의 시작, 종료를 Event로 처리하고, 중앙 Event에서 scene을 로드 하는 것을 고민했으나, 굳이 Global하게 중앙 event에 불러 둬야 할 필요성을 느끼지 못해 그대로 진행했음

3. 구현 시 문제되었던 부분

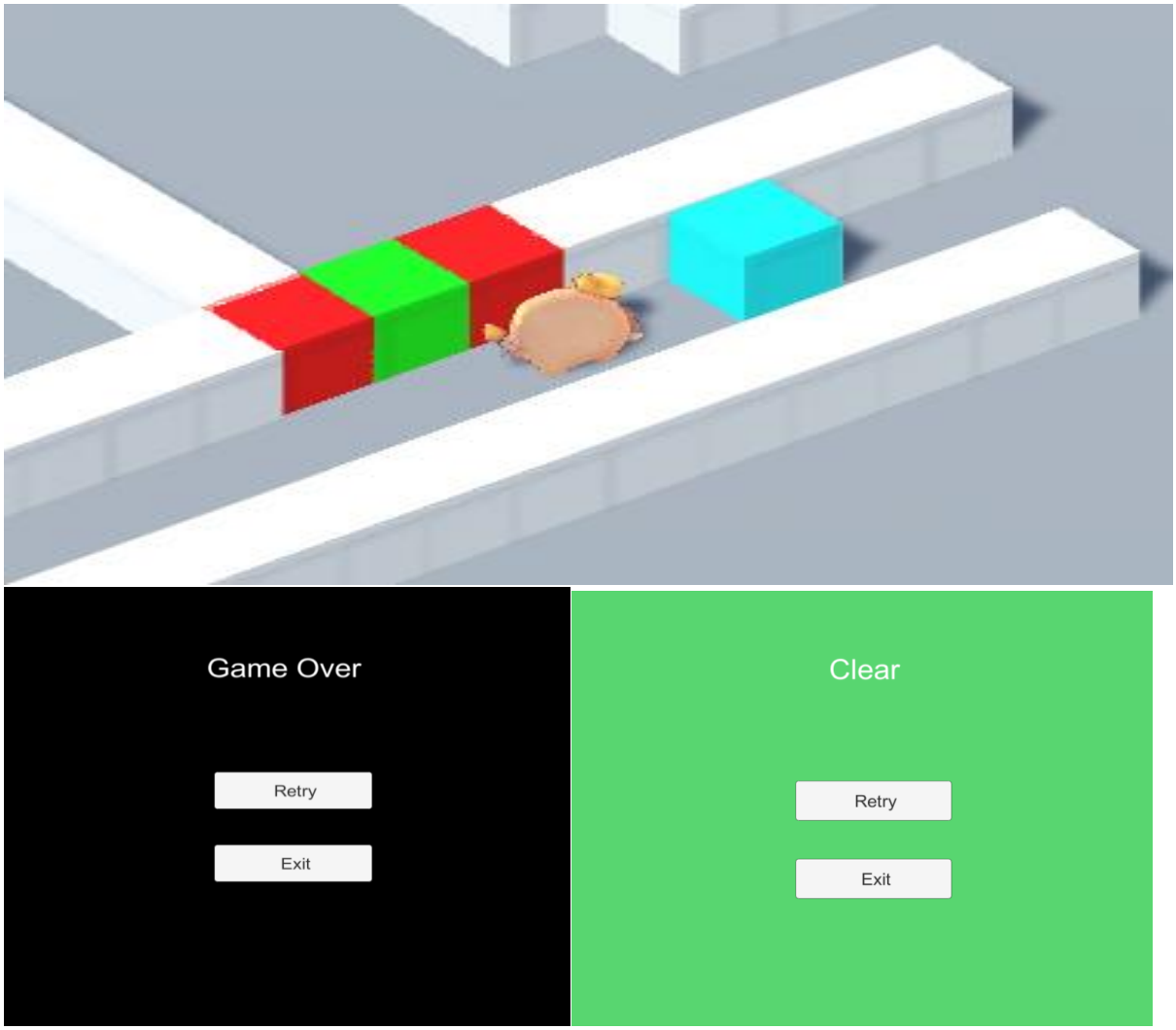
1) Wall2Command.cs에서 gameObject를 받아와 Command 내의 private checkColor 함수를 구현 당시에 초기화를 잘못 해주어 계속 같은 색깔로만 변했었음.

2) Wall2Command.cs에서 코드를 잘못 이해한 부분이 있어 클릭한 hit.point를 계속해서 받아왔으나, 해당 정보가 불필요함을 인지하고 후에 제외시키고 코드를 정리함.

3) TimeTable을 관리하지 않아 시간과 상관없이 Command들이 Replay될 때 일정 시간마다 역재생 되었으나, 후에 Time table 관리하는 코드를 포함해 시간 차이를 두고 동작하였을 때, 이에 맞게 출력되게 변경함.

4) 기존 state pattern에서는 Enemy Manager를 통해 enemy list 를 관리했지만, 저번 과제 때 지적 받았던 것처럼 list 관리가 필요 없으므로 해당 스크립트를 삭제하고 각각의 enemy에서 base.update 되게 하였음. 후에 다른 패턴들을 배우며 이곳저곳에서 update 하는 것이 좋지 않다고 배워 state pattern, 그리고 사용했던 Observer pattern외에 다른 방법이 없을까 고민하게 됨.

4. 구현 결과



1) 일반벽은 처음 의도대로 돌아가려 하고, 파괴할 수 있는 벽은 계속해서 부딪히려고 하기 때문에 꼭 삭제해줘야 함. 때문에 최단 루트를 위해 어떤 벽을 부수고 남겨둬야 할지 판단하면서 게임을 플레이하면 됨

2) Command 추가로 script의 개수는 늘어났지만, Command Pattern을 어느 정도 이해하고 있기 때문에 OverWatch에서 보여주는 것 같은 최고의 플레이 replay 영상과 같은 replay 기능을 만들 수 있었고, 이를 다음에 활용하여 FPS 게임을 만들 때도 활용할 수 있을 것 같음

3) 구현하며 생성했던 필요 없는 script들을 삭제하였고, 어느 정도 프로그래밍 구조를 생각하고 프로그래밍 하였음에도 변동 사항이 발생하는 점을 보았을 때, 설계에 대한 고민을 지속적으로 해야 할 필요성을 느꼈음

5. 패턴을 적용했을 때와 안 했을 때의 차이점

1) Observer Pattern의 장점 덕분에 gameObject나 EnemyObject를 리스트로 관리하지 않아도 되어 코드의 길이가 짧아지고 접근할 때 헷갈림이 덜 해짐.

이번 학기 기능성 게임 프로그래밍을 들을 때 장애물 리스트를 생성하여 관리 했었는데 개수가 늘어나니 Update를 어디서 해줘야 될지, 현재 몇 개의 리스트가 생성되었고 리스트에서 무엇을 삭제 했는 지와 같은 근본적인 것들이 스파게티 코드가 되어 어려움이 있었음.

만약 장애물들이 동작하는 것이 Player가 중심이 되고, 이에 따라 동작하는 Object의 동작 종류가 단순한 pattern 이라면 Observer Pattern, event Action을 이용해서 update를 플레이어 하나에서 관리해주면 헷갈릴 리도 없고 좋았을 것 같음.

➔ Observer Pattern은 Update가 subject에서만 되므로 헷갈릴 일이 적고, 리스트로 관리하지 않아도 된다는 점이 큰 장점인 패턴인 것 같음.

2) Command Pattern의 경우, 실행되므로 Command가 늘어날 때마다 script의 개수도 늘어난다는 단점은 있지만 입력한 Command들이 Command Manager에서 관리된 list에 의해 하나의 Update에서 실행되므로 코드가 꼬이거나 헷갈릴 일이 적었음.

3) State Pattern 과 Observer Pattern을 둘 다 이번 프로젝트에서 사용해보고 느낀 것인데, State Pattern은 동작이 여러 가지이고, Player에 의해 trigger 되는 부분이 있으나 객체 자체로도 기능을 해야 할 때 구현 목적이 더 명확하게 다가왔던 것 같음. Observer Pattern의 경우도 state pattern과 비슷하게 구현할 수 있으나 Command list처럼 list로 관리되지 않기 때문에 오히려 너무 많은 event를 observer pattern으로 받을 경우 좀 더 어렵게 느껴졌던 것 같음. Event를 전역적으로 호출할 수는 없지만 event 자체가 늘어남에 따라 이벤트 생성/해제에 더욱 신경 써야 하기 때문에 사용에 있어 조심스러워졌음. 때문에 섞어 사용하기 보다는 장애물은 Observer, Enemy는 State Pattern 이런 식으로 패턴을 아예 나눠서 코드를 짜게 되었음.

6. 향후 계획

1) 중앙 Event list를 통해 튜토리얼 구현하기

: Event queue의 경우 처음부터 설계를 잘하고 코드를 짜야 문제가 없다고 알려져 있으므로, 현재 동작하고 있는 것들을 튜토리얼 게임으로 분리하여 일정 event를 달성할 때마다 튜토리얼 말풍선이 등장하게끔 진행해볼 예정임. 그 후 기존의 player script에서 event를 관리했을 때와 어떠한 차이점이 있을지 살펴볼 것임

2) State Pattern, animation과 함께 적용. Unity의 animator의 동작을 살펴보면 state pattern, BT 등 여러 가지와 잘 맞을 것 같은 패턴들이 보였음. 우선 적용해봐서 친숙한 State Pattern에 player의 움직임에 따라 공격, 움직이는 애니메이션이 달라지게 만들어 볼 예정

3) Object Pool은 전역적으로 접근할 수 있다는 장점이 있는데, Player의 score를 관리할 때 DontDestroy 함수를 사용하여 score를 다른 scene에서 이용한 적이 있었음. 하지만

위처럼 사용하면 score를 언제 초기화 시켜줄지, score를 가지고 있는 object는 누구인지에 따라 여러 가지 생각해야 할 것이 많은데, 이러한 경우는 object pool로 관리하면 쉽게 할 수 있을 것 같다는 생각이 들었음

4) Event Queue로 겹치면 안 되는 소리들 위주로 관리. 파형이 겹쳐 시끄러워지지 않도록 겹치면 안 될 것 같은 소리들을 따로 관리해 볼 예정임. 기존에는 Audio Soruce를 여러 개 gameObject의 Component로 붙여 사용했음. 위와 같은 방식과 어떠한 차이점이 있고, 어느 방법을 사용하는 것이 좋을지 구현하며 살펴볼 것임. 그 외에도 튜토리얼이나 퀘스트 같은 경우 Event Queue로 구현했을 때 어떨지 고민해 봐야할 것 같음.

-> Event Queue의 장점은 크게 와 닿지 않아 유니티로 적용해보며 기능만 우선적으로 익혀볼 생각임