슈퍼마리오 브라더스

김남석

이 게임의 유니티 프로젝트 파일은 아래의 구글 드라이브에서 확인하실 수 있습니다.

https://drive.google.com/drive/folders/1SayFiac8jnKAX2t0YzdRayp_9N5sBUCP?usp=drive_link

목차

1. 역기획을 통해 설정한 게임 정보

1.1 게임 방식

마리오를 조작해 이 맵의 목적지에 도착하면 클리어입니다. 목적지까지 도착하는 과정에서 몬스터에게 피해를 입거나 낙사로 마리오가 사망할 수 있습니다.

1.2 마리오

마리오의 목숨은 3개이며, 마리오 사망 시 마리오의 목숨이 하나 차감되고 스테이지를 처음부터 다시 시작하게 됩니다. 목숨이 0이 되면 게임오버되어 시작화면으로 돌아갑니다. 마리오는 조작을 통해 좌우 이동과 점프가 가능합니다.

마리오의 이동 : 약 0.3초동안 가속을 받다가 그 후에 최고속력이 됩니다. 최고속력일때 1초에 6칸 정도를 이동합니다. 이후 키를 떼면 0.3초 정도 동안 속력을 줄이면서 전진합니다.

마리오의 점프 : 처음에는 모든 점프가 다 같은 줄 알았지만, 자세히 보니 꾹 누르고 있으면 추가로 더 높이 점프한다. 최저 높이는 타일 세 개의 높이를 가진 지형에 간신히 올라오는 정도이고, 최고 높이는 타일 네 개의 높이를 가진 지형에 간신히 올라오는 정도였습니다.

위와 같은 마리오의 이동과 점프를 구현하기 위해 다음과 같은 수도코드를 먼저 작성해보았습니다. (이 수도코드들을 구현한 코드는 [부록 5.8 Mario 클래스]에서 보실 수 있습니다.)

1.2.1 마리오의 이동 수도코드

Mario::FixedUpdate()

INPUT : *velocity*
 	
IF GetKey(LeftArrow) THEN
  IF *velocity* . x > -1 THEN
  Accelerator ← - 1/ 0.3
  END IF
ELSE IF GetKey(RightArrow) THEN
  IF *velocity* . x < 1 THEN
  Accelerator ← 1/ 0.3
  END IF
ELSE
  Accelerator ← - (Velocity.x / Abs(Velocity.x)) * (1 / 0.3)
END IF
velocity . x ← *velocity* . x + Accelerator * fixedDeltaTime
position . x ← *velocity* . x * 6 * fixedDeltaTime

velocity 는 3차원 벡터 타입의 멤버변수로, 마리오의 속도벡터를 나타냅니다.

1.2.2 마리오의 점프 수도코드

Mario::FixedUpdate()

INPUT : *JumpTime, IsJumpKeyActive*

```
IF   GetKey( SpaceBar ) THEN
      IF   JumpTime < 0.5   &&   IsJumpKeyActive == TRUE   THEN
           Jumpscale ← 90 * (1 - 2 * JumpTime) ^ 4
           AddForce(Vector(0, 1, 0) * JumpScale)
           JumpTime ← JumpTime + fixedDeltaTime
      END IF
END IF
IF   GetKeyUp( SpaceBar )   THEN
      IsJumpKeyActive ← FALSE
END IF
```

JumpTime 은 실수형 멤버변수로, 점프를 시작하고나서 지난 시간을 나타냅니다. 마리오가 지형에 발이 닿게 된다면 0으로 초기화됩니다.

IsJumpKeyActive 은 논리형 멤버변수로, 점프키가 활성화가 되어있는지를 나타냅니다. 점프키를 누르다가 떼는 순간 FALSE가 되며, 마리오가 지형에 발이 닿게 된다면 다시 TRUE로 바뀝니다.

1.3 게임 내 타일 및 아이템 종류

: 이 블록에 부딪치면 특별한 상호작용은 없고 그냥 가로막히기만 한다.

: 이 블록에 머리로 부딪치면 코인이나 아이템을 생성하고 첫번째 블록으로 변한다.

: 이 블록에 머리로 N회 부딪치면 파괴되거나 첫번째 블록으로 변한다. 코인이나 아이템을 제공할 수도 있다.

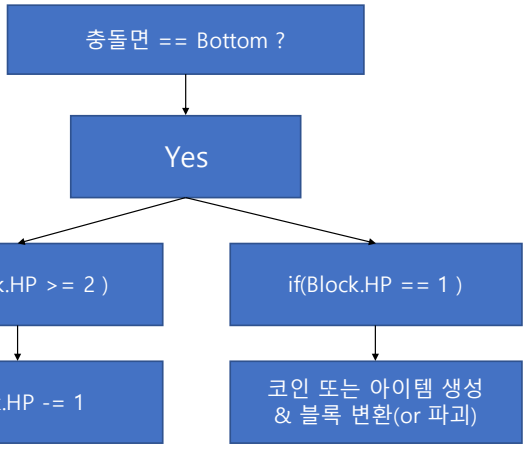
버섯 : 이 아이템을 먹으면 작은 마리오에서 큰 마리오로 성장할 수 있다.

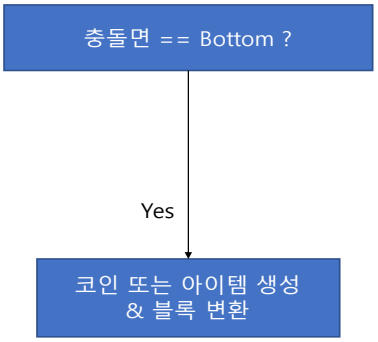
꽃 : 이 아이템을 먹으면 불마법사 마리오로 성장할 수 있다.

별 : 이 아이템을 먹으면 낙사를 제외한 모든 피해에 대해 무적이 되며, 몬스터에 부딪치면 몬스터가 날려지면서 처치된다.

코인 : 이 아이템을 먹으면 점수가 증가한다.

블록의 경우 마리오와 충돌 시 다음과 같은 과정을 거치게 됩니다. 이를 구현한 코드는 각각 [부록5.2 Block1 클래스], [부록5.3 Block2 클래스]에서 보실 수 있습니다.



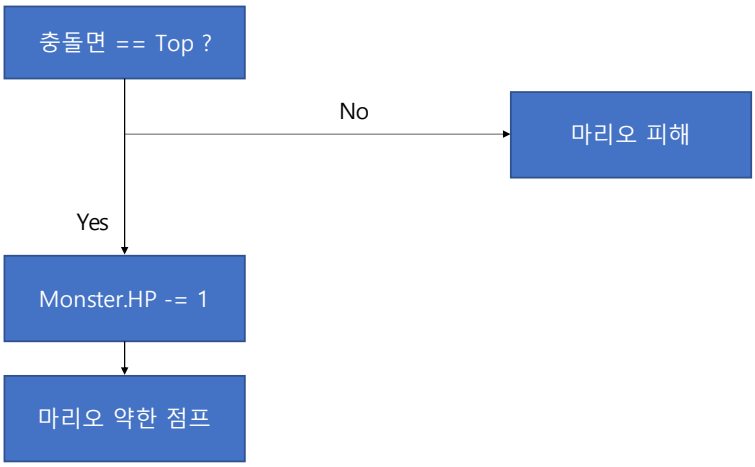


1.4 몬스터 종류

 굼바 : 마리오가 머리로 밟았을 때, 바로 처치되어 사라지는 몬스터입니다.

 거북이 : 마리오가 머리로 밟았을 때, 바로 처치되어 등껍질을 남기고 사라지는 몬스터입니다.

몬스터의 경우 마리오와 충돌 시 다음과 같은 과정을 통해 누가 피해를 입는지 결정이 됩니다. 이를 구현한 코드는 [부록 5.5 Turtle 클래스], [부록 5.6 Gumba 클래스]에서 보실 수 있습니다.



1.5 충돌면 판정 과정 구상

어떤 오브젝트가 다른 오브젝트와 어느 면으로 부딪히는지에 따라 그 결과가 다릅니다. 예를 들어, 마리오가 몬스터의 윗쪽면으로 충돌 시에 몬스터가 처치되고 이외의 면으로 충돌 시에는 마리오가 피해를 받습니다. 그래서 먼저 충돌면을 어떻게 체크할지 아래와 같은 알고리즘을 설계해보았습니다.

- 마리오가 다른 충돌체의 어느 면을 충돌하였는지에 대한 판정

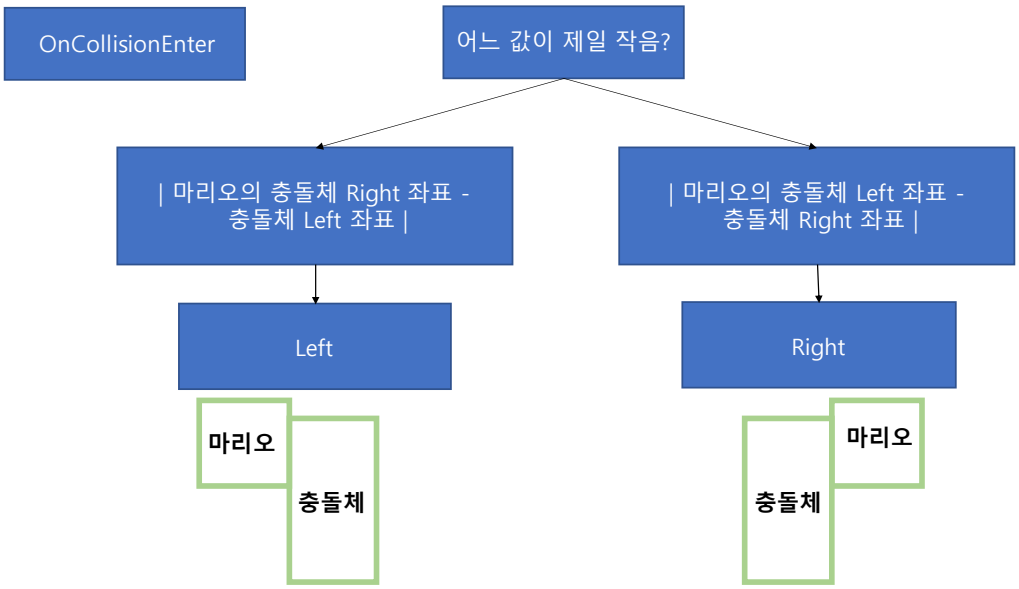
1. 다음과 같이 열거형 데이터를 정의한다.

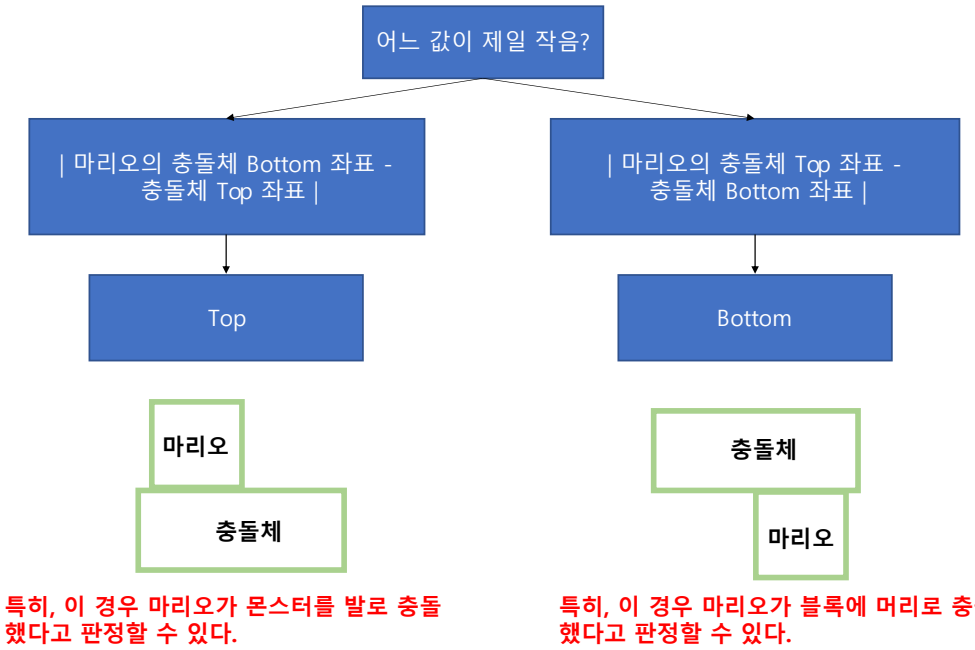
   Left,
   Right,
   Top,
   Bottom

2. 다음 네 가지의 값을 구한다.

마리오의 충돌체 Right 좌표 - 충돌체 Left 좌표
마리오의 충돌체 Left 좌표 - 충돌체 Right 좌표 |
마리오의 충돌체 Bottom 좌표 - 충돌체 Top 좌표 |
마리오의 충돌체 Top 좌표 - 충돌체 Bottom 좌표 |

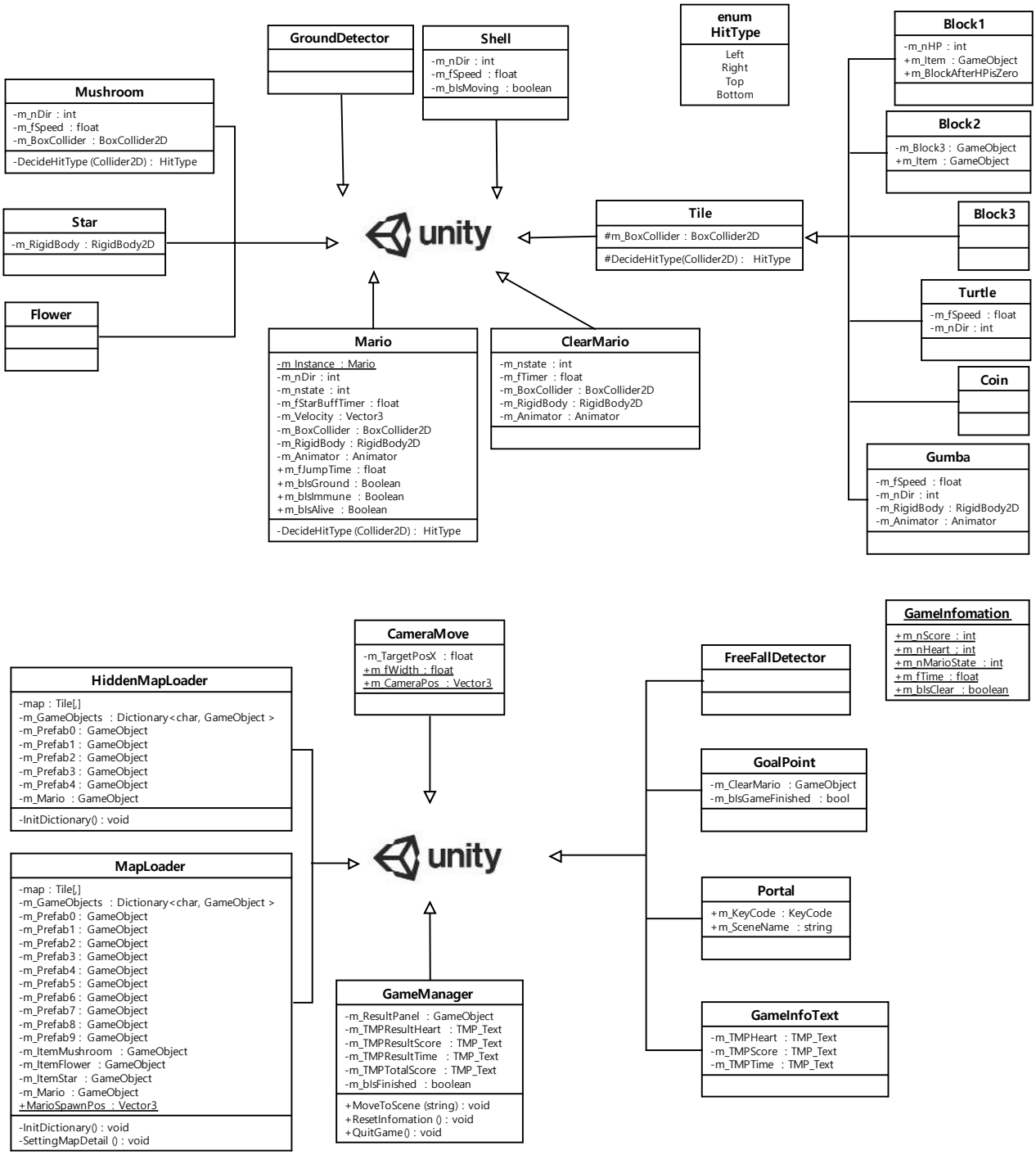
3. 네가지의 값 중 제일 작은 값이 무엇인지에 따라 판정한다.





이를 구현한 내용은 [3.3 충돌 시 어느 면에 충돌하였는지에 대해 판단하는 작업]에서 자세하게 기술하였습니다.

2. UML 다이어그램


GroundDetector
Shell
-m_nDir : int
-m_fSpeed : float
-m_bIsMoving : boolean
enum
HitType
Left
Right
Top
Bottom
Block1
-m_nHP : int
+m_Item : GameObject
+m_BlockAfterHPisZero
Block2
-m_Block3 : GameObject
+m_Item : GameObject
Block3
Mushroom
-m_nDir : int
-m_fSpeed : float
-m_BoxCollider : BoxCollider2D
-DecideHitType (Collider2D) : HitType
Star
-m_RigidBody : RigidBody2D
Flower
unity
Tile
#m_BoxCollider : BoxCollider2D
#DecideHitType(Collider2D) : HitType
Turtle
-m_fSpeed : float
-m_nDir : int
Coin
Gumba
-m_fSpeed : float
-m_nDir : int
-m_RigidBody : RigidBody2D
-m_Animator : Animator
Mario
-m_Instance : Mario
-m_nDir : int
-m_nstate : int
-m_fStarBuffTimer : float
-m_Velocity : Vector3
-m_BoxCollider : BoxCollider2D
-m_RigidBody : RigidBody2D
-m_Animator : Animator
+m_fJumpTime : float
+m_bIsGround : Boolean
+m_bIsImmune : Boolean
+m_bIsAlive : Boolean
-DecideHitType (Collider2D) : HitType
ClearMario
-m_nstate : int
-m_fTimer : float
-m_BoxCollider : BoxCollider2D
-m_RigidBody : RigidBody2D
-m_Animator : Animator
GameInfomation
+m_nScore : int
+m_nHeart : int
+m_nMarioState : int
+m_fTime : float
+m_bIsClear : boolean
CameraMove
-m_TargetPosX : float
+m_fWidth : float
+m_CameraPos : Vector3
FreeFallDetector
HiddenMapLoader
-map : Tile[,]
-m_GameObjects : Dictionary<char, GameObject >
-m_Prefab0 : GameObject
-m_Prefab1 : GameObject
-m_Prefab2 : GameObject
-m_Prefab3 : GameObject
-m_Prefab4 : GameObject
-m_Mario : GameObject
-InitDictionary() : void
GoalPoint
-m_ClearMario : GameObject
-m_bIsGameFinished : bool
unity
MapLoader
-map : Tile[,]
-m_GameObjects : Dictionary<char, GameObject >
-m_Prefab0 : GameObject
-m_Prefab1 : GameObject
-m_Prefab2 : GameObject
-m_Prefab3 : GameObject
-m_Prefab4 : GameObject
-m_Prefab5 : GameObject
-m_Prefab6 : GameObject
-m_Prefab7 : GameObject
-m_Prefab8 : GameObject
-m_Prefab9 : GameObject
-m_ItemMushroom : GameObject
-m_ItemFlower : GameObject
-m_ItemStar : GameObject
-m_Mario : GameObject
+MarioSpawnPos : Vector3
-InitDictionary() : void
-SettingMapDetail () : void
Portal
+m_KeyCode : KeyCode
+m_SceneName : string
GameManager
-m_ResultPanel : GameObject
-m_TMPResultHeart : TMP_Text
-m_TMPResultScore : TMP_Text
-m_TMPResultTime : TMP_Text
-m_TMPTotalScore : TMP_Text
-m_bIsFinished : boolean
+MoveToScene (string) : void
+ResetInfomation () : void
+QuitGame() : void
GameInfoText
-m_TMPHeart : TMP_Text
-m_TMPScore : TMP_Text
-m_TMPTime : TMP_Text

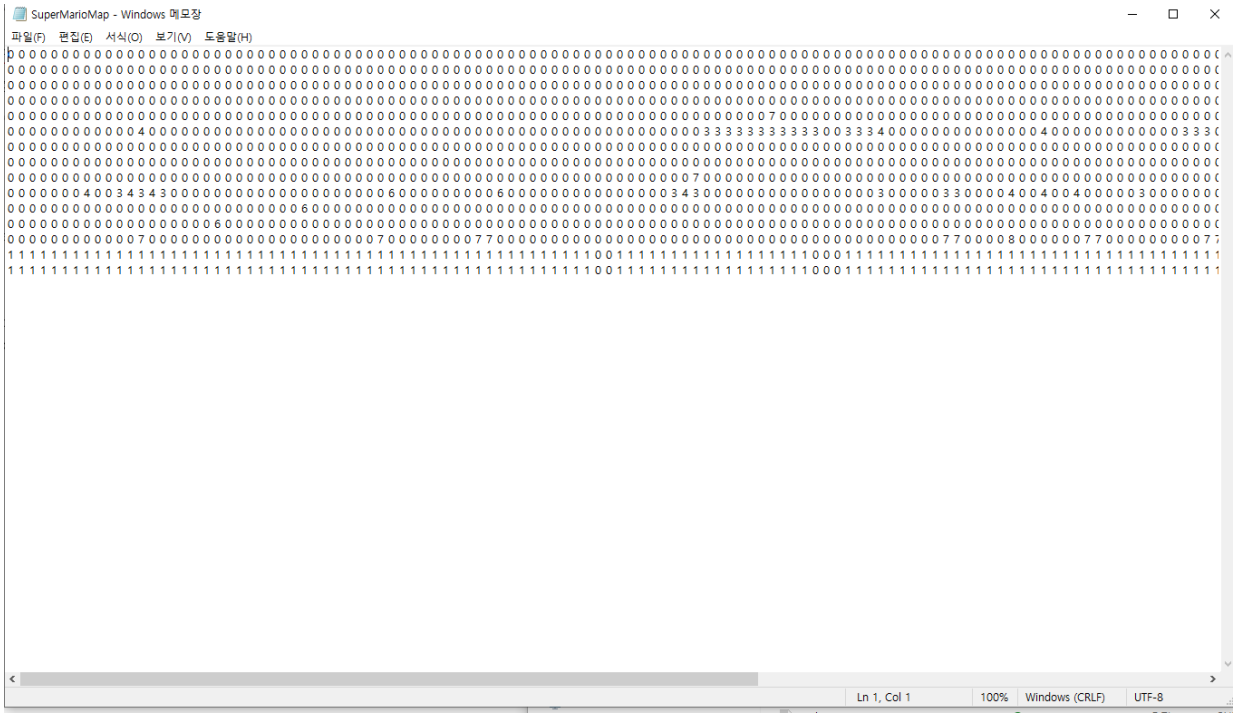
3. 게임 개발을 위한 연구 및 시도

3.1 맵 구성을 위한 텍스트 파일 파싱

3.1.1 개요

이 게임을 처음 만들 당시에 씬에 모든 오브젝트를 배치하고 저장을 하니 그 씬에 대한 데이터만 1MB정도의 크기를 가졌습니다. 그리고 이러한 방식으로 앞으로 스테이지를 늘려가게 된다면 한 맵마다 이 정도의 크기로 저장해야한다는 생각이 들면서, 맵에 대한 정보를 텍스트 파일로 저장할 필요를 느꼈습니다.

3.1.2 구현



먼저 위와 같이 맵에 대한 정보를 텍스트 파일로 만들었습니다. 공백을 통해 칸을 구분하였고, 문자는 실제 맵에서 그 자리에 배치될 오브젝트를 의미합니다.

그리고 씬이 로드될 때 MapLoader 클래스의 Awake함수에서 텍스트 파일을 파싱하여 얻어온 문자의 종류에 따라 알맞은 오브젝트를 배치하도록 코드를 작성하였습니다. (이 수도코드를 구현한 코드는 [부록 5.10 MapLoader 클래스]에서 보실 수 있습니다.)

MapLoader::Awake()
INPUT : *m_GameObjects, map* InitDictionary() sr ← StreamReader(“SuperMarioMap.txt”)

```
index ← 0
line ← sr.ReadLine()

WHILE    line != NULL   THEN
        FOR   i   in   0 : line.Length – 1 THEN
            IF   line[i] == ' ' THEN
                    continue
            ELSE
                    j ← i / 2
                    pos ← Vector3(j, -index, 0)
                    go ← Instantiate(m_GameObjects[line[i]])
                    go.position ← pos
                    map[index, j] ← go.GetTile()
            END IF
        END FOR
         index ← index + 1
         line ← sr.ReadLine()
END WHILE

sr.Close()
```

m_GameObjects 는 MapLoader 클래스의 멤버변수로, C#에서의 Dictionary<char, GameObject> 타입의 변수입니다. 이 변수를 통해 문자를 이용해 게임오브젝트를 검색할 수 있습니다.

map 은 MapLoader 클래스의 멤버변수로, Tile 타입의 2차원 배열입니다.

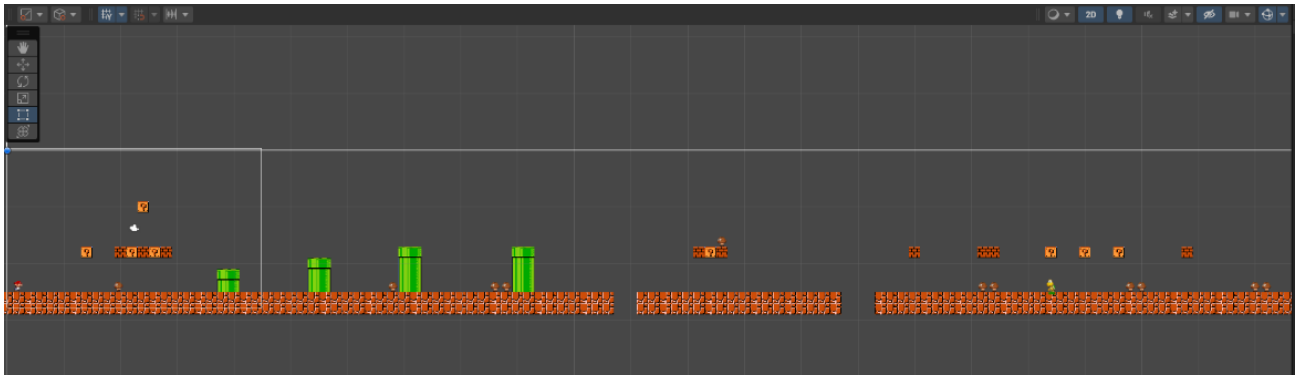
InitDictionary() 함수는 *m_GameObjects* 변수에 (문자, GameObject)쌍들을 추가하는 함수입니다.

StreamReader() 함수는 C#에서의 StreamReader 생성자입니다.

Instantiate()함수는 매개변수인 게임오브젝트 프리팹을 Clone하는 함수입니다.

GetTile() 함수는 게임오브젝트의 Tile 컴포넌트를 가져오는 함수입니다.

3.1.3 결과



위 그림과 같이 알맞게 게임오브젝트들이 배치가 되었고, 이 스테이지에 대한 씬의 데이터 크기가 1MB에서 143KB로 작아져 약 86% 정도 작아졌습니다.

3.2 맵 컬링

3.2.1 개요

맵 크기가 15(세로) * 178(가로)인데, 카메라의 가로 길이가 대략 22.5 입니다. 맵의 가로 크기에 비해 카메라의 가로 길이가 많이 작아 카메라 바깥의 공간에서 불필요하게 렌더링되는 게임오브젝트들이 많아서 카메라 밖의 게임 오브젝트들을 컬링할 필요가 느껴졌습니다.

3.2.2 구현

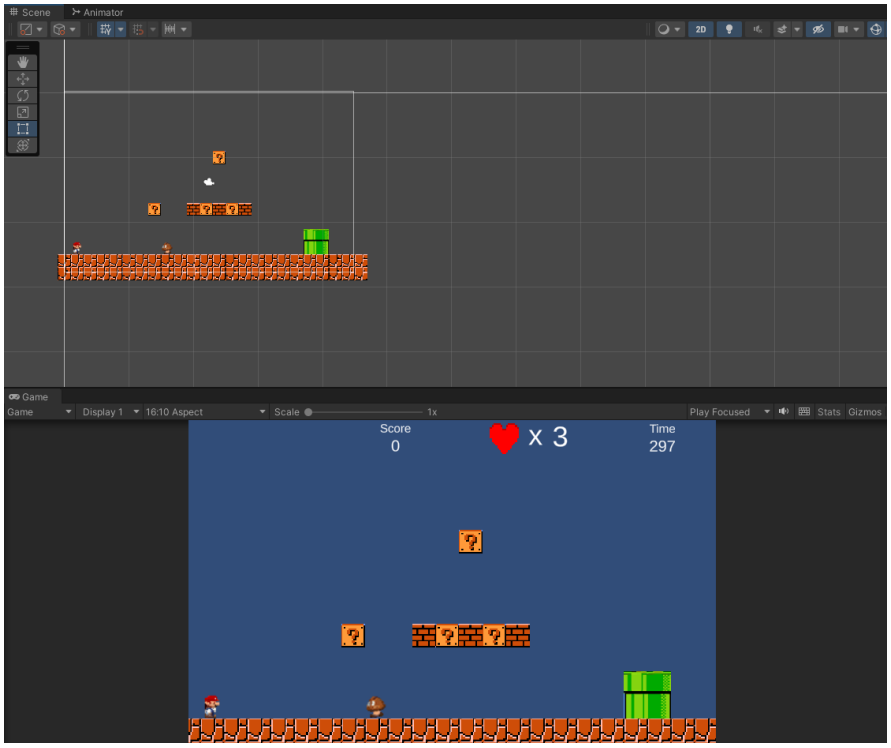
맵에 대한 정보는 MapLoader클래스에 있기 때문에 아래의 수도코드와 같이 MapLoader의 FixedUpdate함수에서 이를 수행하도록 하였습니다. (이 수도코드를 구현한 코드는 [부록 5.10 MapLoader 클래스]에서 보실 수 있습니다.)

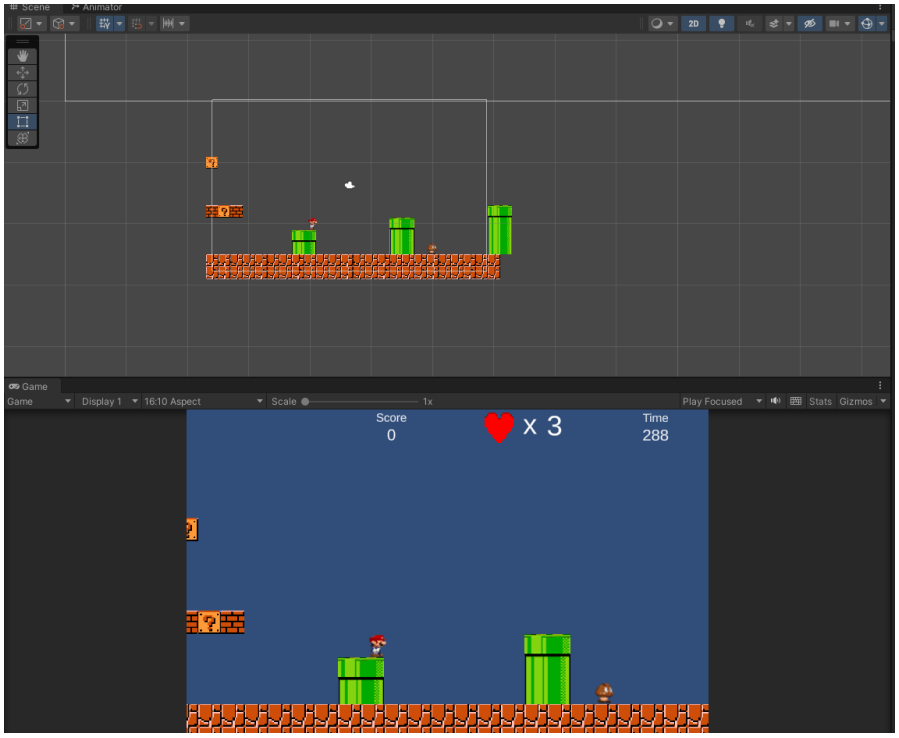
MapLoader::FixedUpdate()

INPUT : *map, CameraPos, CameraWidth*
 	
FOR j in 0 : map.ColumnCount – 1 THEN
  FOR i in 0 : map.RowCount -1 THEN
  X ← Abs(*CameraPos*.x - map[i,j].position.x)
  a ← FALSE
  IF x < (*CameraWidth* / 2 + 1) THEN
  a ← TRUE
  ELSE
  a ← FALSE
  END IF
 	
  map[i, j].SetActive(a)
  END FOR
END FOR

map 은 MapLoader 클래스의 멤버변수로, Tile 타입의 2차원 배열입니다. *CameraPos* 는 3차원 벡터 변수로, 카메라의 위치를 나타냅니다. *CameraWidth* 는 실수형 변수로, 카메라의 가로 너비값를 나타냅니다. Abs() 함수는 매개변수의 절댓값을 반환하는 함수입니다. SetActive() 함수는 해당 오브젝트의 활성화 유무를 결정하는 함수입니다.

3.2.3 결과





두 사진은 카메라의 위치에 따라 오브젝트들이 컬링된 모습을 보여주는 사진입니다. 위쪽 사진에서 카메라가 오른쪽으로 움직임에 따라 아래쪽 사진으로 변화한 것이고, 오른쪽의 오브젝트가 활성화되었고 왼쪽의 오브젝트가 비활성화된 모습을 보여줍니다.

3.3 충돌면을 판단하는 작업

3.3.1 개요

슈퍼마리오 게임은 어떻게 충돌하냐에 따라 충돌 처리가 달라지는 모습을 많이 보여줍니다. 예를 들어, 마리오가 몬스터의 윗부분과 충돌하면 몬스터가 피해를 받고 그 외의 상황은 마리오가 피해를 받습니다. 그리고 마리오가 블록의 아랫면을 머리로 충돌해야 블록과 상호작용이 이루어집니다. 그래서 충돌하기 시작했을 때 '어느 면을 충돌했니?'를 판단하는 것이 꼭 필요하다고 생각해서 이를 구현해야겠다고 판단했습니다.

3.3.2 구현

모든 충돌체를 사각형 충돌체로 정하고 HitType이라는 열겨형 형식을 정의하고(UML 다이어그램 참고), 충돌 후 어느 방향의 충돌면인지를 판정하는 함수를 아래 수도코드와 같이 정의하였습니다. (이 수도코드를 구현한 코드는 [부록 5.1 Tile 클래스]에서 보실 수 있습니다.)

DecideHitType()

INPUT : *col* ,*m_BoxCollider*
OUTPUT : hittype
 	
left ← Abs((*col*.position.x + *col*.Width / 2) - (*m_BoxCollider*.position.x - *m_BoxCollider*.Width / 2))
right ← Abs((*col*.position.x - *col*.Width / 2) - (*m_BoxCollider*.position.x + *m_BoxCollider*.Width / 2))
top ← Abs((*col*.position.y - *col*.Height / 2) - (*m_BoxCollider*.position.y + *m_BoxCollider*.Height / 2))
bottom ← Abs((*col*.position.y + *col*.Height / 2) - (*m_BoxCollider*.position.y - *m_BoxCollider*.Height / 2))
 	
a[] ← { left, right, top, bottom }
 	
k ← 0
min ← 100
 	
FOR i in 0 : 3 THEN
IF a[i] < min THEN
min ← a[i]
k ← i
END IF
END FOR
 	
hittype ← (HitType)k

RETURN hittype
col 은 이 오브젝트와 충돌한 다른 충돌체입니다. *m_BoxCollider* 은 이 오브젝트의 사각형 충돌체입니다. Abs() 함수는 매개변수의 절댓값을 반환하는 함수입니다.

이 수도코드는 다음 네 가지의 값을 구하고 그 중 최솟값을 통해 판정을 하게 되는 코드입니다.

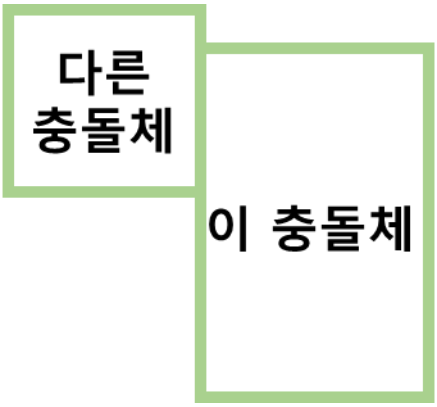
| 다른 충돌체의 Right 좌표 – 이 충돌체 Left 좌표 |

| 다른 충돌체 Left 좌표 – 이 충돌체 Right 좌표 |

| 다른 충돌체 Bottom 좌표 – 이 충돌체 Top 좌표 |

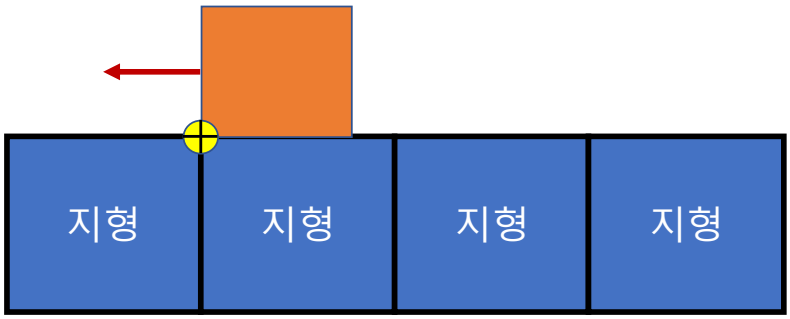
| 다른 충돌체 Top 좌표 – 이 충돌체 Bottom 좌표 |

예를 들어, 위의 값 중 첫번째의 값이 제일 작다면 대략 아래 그림과 같은 상황일 것입니다. 그러므로 '이 충돌체의 왼쪽면을 충돌했다'고 판정합니다.(열거형 형식 HitType 중 Left를 반환)



3.3.3 결과

이 코드를 실행을 하면 대부분의 판정은 잘 이루어졌지만, 한가지의 버그를 발견하게 되었습니다. 이 버그는 (몬스터와 같이) 움직이는 물체가 지형 위를 지나갈 때 발생하게 됩니다. 일단, 몬스터와 같은 오브젝트는 (마리오를 제외한) 다른 충돌체의 왼쪽면이나 오른쪽면을 충돌하게 되면 자신의 좌우방향을 전환하여 이동을 이어나가도록 설계되어있습니다.



그런데, 위 그림과 같이 주황색 오브젝트가 지형 위에서 왼쪽방향으로 움직인다고 했을 때 노란색 점에서의 처리 때문에 저 노란색 점을 넘어서지 못하고 (그림 상에서) 두번째 타일 위에서 좌우로 움직이며 두번째 타일을 벗어나지를 못했습니다. (그림 상에서) 저기 새로 만난 첫번째 지형과 충돌하기 시작함에 따라 어느 면을 충돌했는지 판정을 할 때 |주황색 오브젝트의 Left 좌표 – 첫번째 타일의 Right 좌표| 값과 |주황색 오브젝트의 Bottom 좌표 – 첫번째 타일의 Top 좌표|값이 차이가 크게 나지 않아 주황색 오브젝트가 '자신의 왼쪽면으로 충돌했다'고 판정하여 방향을 바꾼 것입니다. 그래서 이 버그를 수정하기 위해 몬스터와 같이 지형 위를 이동하는 오브젝트에 대해서는 기존 함수에 코드를 추가하였습니다.(이 수도코드를 구현한 코드는 [부록 5.5 Turtle 클래스]나 [부록 5.6 Gumba 클래스]에서 보실 수 있습니다. 추가된 코드는 빨간색으로 표시하였습니다.)

DecideHitType()
INPUT : *col* ,*m_BoxCollider* OUTPUT : hittype left ← Abs((*col*.position.x + *col*.Width / 2) - (*m_BoxCollider*.position.x - *m_BoxCollider*.Width / 2)) right ← Abs((*col*.position.x - *col*.Width / 2) - (*m_BoxCollider*.position.x + *m_BoxCollider*.Width / 2))

```
top ← Abs((col.position.y - col.Height / 2) - (m_BoxCollider.position.y + m_BoxCollider.Height / 2))
bottom ← Abs((col.position.y + col.Height / 2) - (m_BoxCollider.position.y - m_BoxCollider.Height / 2))

a[] ← { left, right, top, bottom }

k ← 0
min ← 100

FOR   i   in   0 : 3   THEN
        IF   a[i] < min   THEN
             min ← a[i]
             k ← i
        END IF
END FOR

l ← Abs(a[3] - a[0])
r ← Abs(a[3] - a[1])

IF   l < 0.05   ||   r < 0.05   THEN
     k ← 3
END IF

hittype ← (HitType)k

RETURN hittype
```

col 은 이 오브젝트와 충돌한 다른 충돌체입니다.

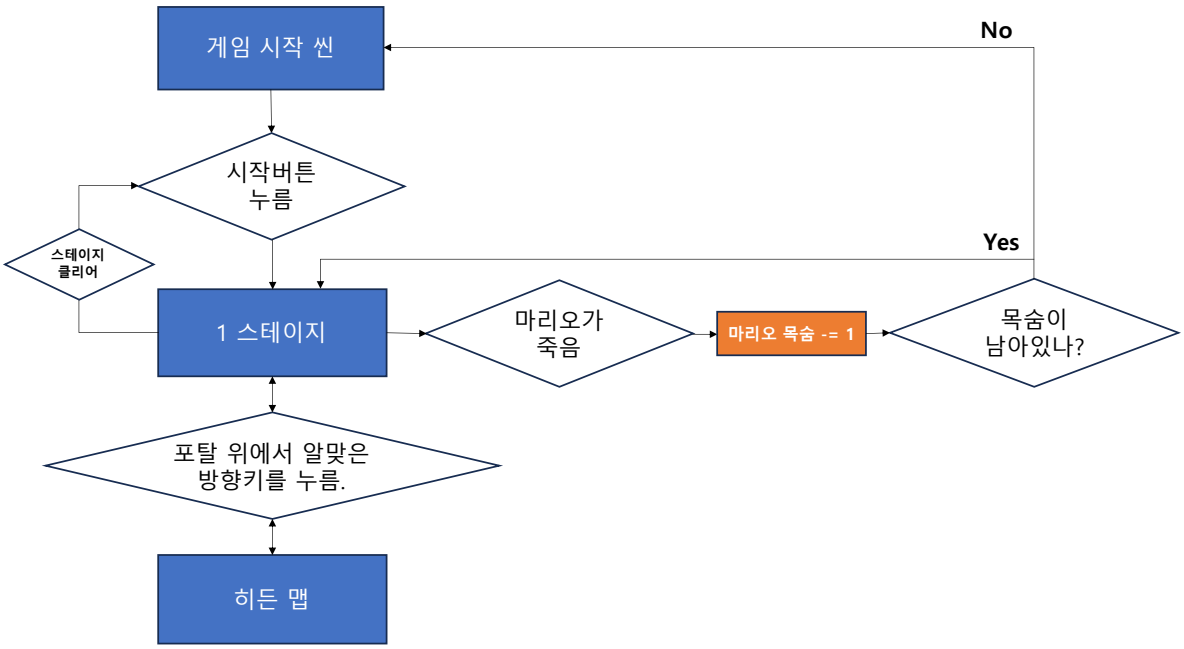
m_BoxCollider 은 이 오브젝트의 사각형 충돌체입니다.

Abs() 함수는 매개변수의 절댓값을 반환하는 함수입니다.

이 코드를 적용 후 버그가 사라져, 몬스터가 잘 이동할 수 있게 되었습니다.

3.4 씬 전환

3.4.1 플로우 차트



게임 진행 시 이루어지는 씬 전환에 대한 플로우 차트는 위 그림과 같습니다. 파란색 사각형 모양의 칸은 씬을 나타내고 마름모 모양의 칸은 게임 진행 중 일어날 수 있는 이벤트를 나타냅니다. 제일 먼저 시작되는 씬은 '게임 시작 씬'입니다. 시작버튼을 누르면 '1 스테이지' 씬으로 전환하고 스테이지를 클리어하거나 마리오의 목숨이 모두 소진될 때까지 1 스테이지를 반복하게 됩니다. 1 스테이지 도중 포탈(특정 배수로)을 타게 된다면 '히든 맵' 씬으로 이동하며, 히든 맵에서도 포탈을 타게 된다면 다시 '1 스테이지'로 돌아오게 됩니다.

3.5 애니메이션

애니메이션은 크게 세 종류가 필요했습니다.

1. Idle : 가만히 서있는 애니메이션
2. Move : 좌우로 움직이는 애니메이션
3. Jump : 점프하는 애니메이션

3.5.1 스프라이트 시트

그리고 작은 마리오, 큰 마리오, 불마법사 마리오 마다 위 세가지의 애니메이션이 필요했습니다. 그래서 아래의 사진들처럼 스프라이트 시트를 찾아 유니티엔진의 Sprite Editor를 이용해 필요한 부분을 가져와 애니메이션에 사용했습니다.

3.5.2 애니메이터

작은 마리오, 큰 마리오, 불마법사 마리오일 때의 상태에 맞게 'SmallMario' , 'BigMario' , 'MageMario' Sub-State Machine으로 들어간 후 다음과 같은 규칙으로 애니메이션이 재생할 수 있도록 설계했습니다.

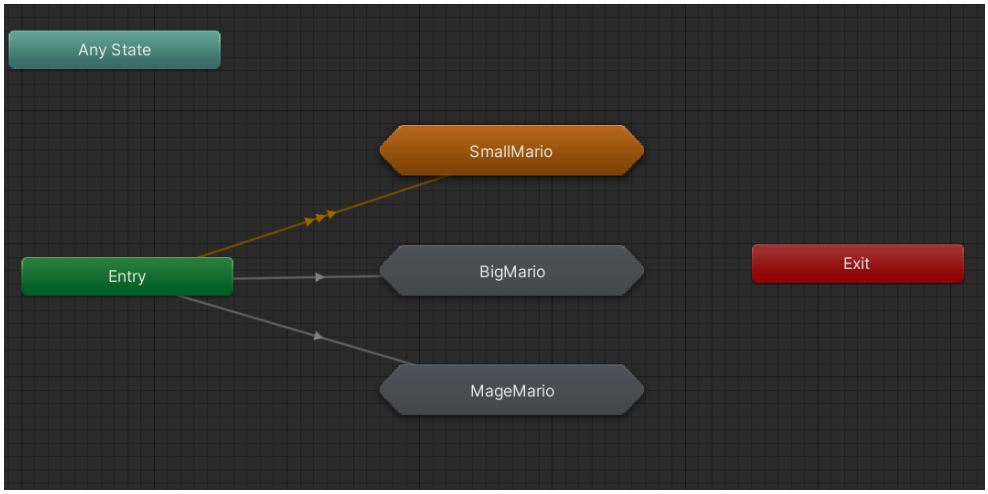
1) 점프를 하고 있다면 Jump 애니메이션을 재생한다. 한 번만 재생하고 마리오가 지형을 밟기 전까지 다른 애니메이션으로 넘어가지 않는다.

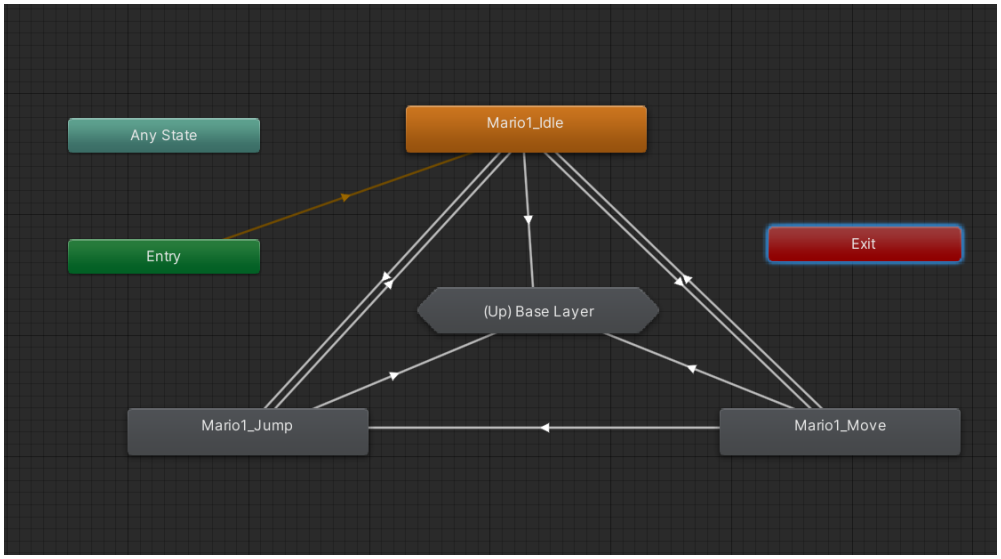
2) 점프를 하고 있지 않지만, 마리오의 속력의 x 성분이 0이 아닐 때 Move 애니메이션을 재생한다.

3) 이 외의 상황은 Idle 애니메이션을 재생한다.

4) 만약 마리오의 상태(작은 마리오, 큰 마리오, 불마법사 마리오)가 달라지면 이 Sub-State Machine을 벗어난다.

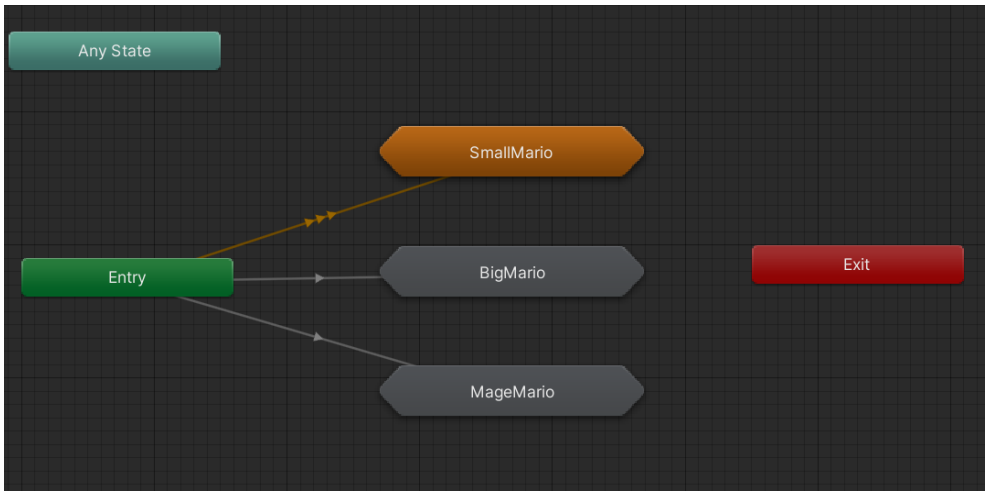
이를 구현하기 위해 처음에는 아래 사진과 같은 구조로 애니메이터를 구성하였습니다.

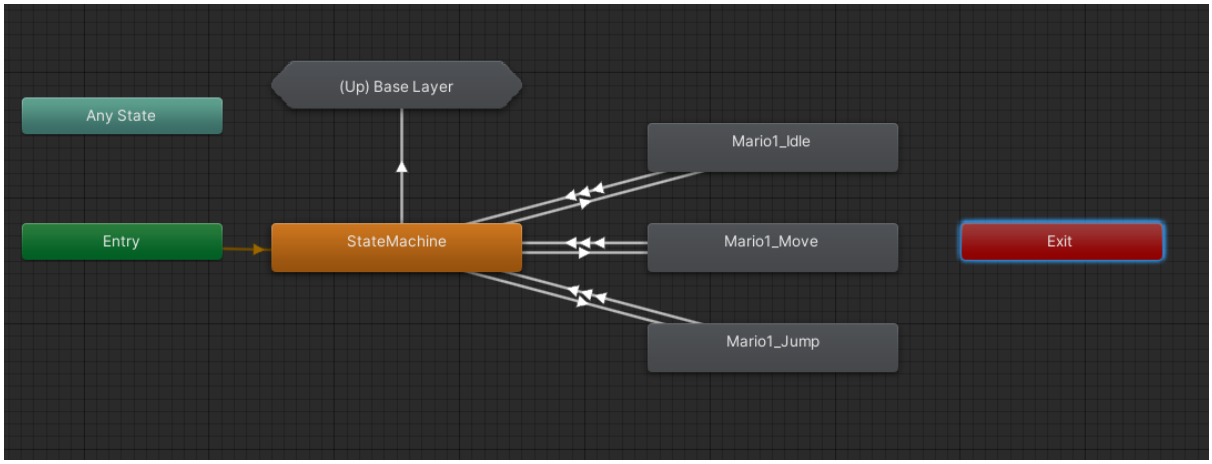




실행에는 문제가 없었지만, 나중에 새로운 행동이 생겨 저기에 새로운 애니메이션을 추가해야할 때 다시 transition을 모두 연결하는 과정과 애니메이터를 수정하는 과정 모두 복잡해질 것 같아 구조 변경이 무조건 필요하다고 생각했습니다.

그래서 아래 사진과 같은 구조로 변경하였습니다.





이런 방식으로 변경하니, 나중에 다른 애니메이션을 추가하거나 transition을 수정해야할 때 복잡하지 않고 손쉽게 수행할 수 있을 것이라고 생각합니다.

4. 결론

슈퍼마리오 브라더스를 역기획하여 이를 구현하는 과정을 통해, 게임을 만들 때 어떠한 방식으로 기획해야하는 지에 대해 감이 잡혔습니다. 전에는 게임 기획을 세밀하게 하지 않아 게임을 구현하는 과정에서 길을 많이 헤맸습니다. 그래서 역기획을 해보기로 했고, 역기획을 하기 위해서 수도 없이 슈퍼마리오 브라더스 1의 1스테이지를 플레이했습니다. 이 과정에서 이동이나 충돌과 같은 부분 등을 디테일하게 관찰했고, 이러한 내용 덕분에 어느 기능이나 함수를 이용해 어떻게 구현해볼지 더 수월하게 구상할 수 있게 되었습니다. 앞으로 게임을 만들 때에 좋은 본보기가 될 것으로 생각합니다.

5. 부록

5.1 Tile 클래스

Tile 클래스

```
using UnityEngine;

//이 사각형의 어느 면이 부딪혔는지 나타내는 열거형 형식
public enum HitType
{
    Left,
    Right,
    Top,
    Bottom
}

public class Tile : MonoBehaviour
{
    protected BoxCollider2D m_BoxCollider;

    //어느 면이 부딪혔는지 판정하는 함수
    protected virtual HitType DecideHitType(Collision2D col)
    {
        //이 충돌체의 left 좌표 - 다른 충돌체의 right 좌표
        float left = Mathf.Abs((col.transform.position.x +
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.x) -
(m_BoxCollider.transform.position.x - m_BoxCollider.bounds.extents.x));
        //이 충돌체의 right 좌표 - 다른 충돌체의 left 좌표
        float right = Mathf.Abs((col.transform.position.x -
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.x) -
(m_BoxCollider.transform.position.x + m_BoxCollider.bounds.extents.x));
        //이 충돌체의 top 좌표 - 다른 충돌체의 bottom 좌표
        float top = Mathf.Abs((col.transform.position.y -
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.y) -
(m_BoxCollider.transform.position.y + m_BoxCollider.bounds.extents.y));
        //이 충돌체의 bottom 좌표 - 다른 충돌체의 top 좌표
        float bottom = Mathf.Abs((col.transform.position.y +
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.y) -
(m_BoxCollider.transform.position.y - m_BoxCollider.bounds.extents.y));

        float[] a = { left, right, top, bottom };

        //네 가지 값 중 최솟값과 인덱스를 구한다.
        int k = 0;
        float min = 100.0f;

        for (int i = 0; i < 4; i++)
        {
            if (a[i] < min)
            {
                min = a[i];
                k = i;
            }
        }
```

```
        return (HitType)k;
    }
}
```

5.2 Block1 클래스

Block1 클래스

```
using UnityEngine;

public class Block1 : Tile
{
    private int m_nHP = 1; //이 블록의 내구도.
    public GameObject m_Item = null; //마리오가 머리로 이 블록을 치면 나오는 아이템. null
이면 생성되는 아이템이 없다.
    public GameObject m_BlockAfterHPisZero = null; //이 블록의 내구도가 0이 될 때 이
게임오브젝트로 변환된다. null 이면 그냥 파괴된다.

    //m_nHP의 get, set 함수
    public int HP
    {
        get { return m_nHP; }
        set
        {
            if (value <= 0)
            {
                //내구도가 0이 될 때 처리하는 코드.
                if (m_BlockAfterHPisZero != null)
                {
                    GameObject go = Instantiate(m_BlockAfterHPisZero);
                    go.transform.position = transform.position;
                    go.transform.parent = null;
                }
                Destroy(gameObject);

            }
            else
                m_nHP = value;
        }
    }
    private void Awake()
    {
        m_BoxCollider = GetComponent<BoxCollider2D>();
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        //마리오가 머리로 이 블록에 부딪혔을 때 처리하는 코드
        if (collision.gameObject.CompareTag("Player") && base.DecideHitType(collision) ==
HitType.Bottom)
        {
            //마리오에게 아래 방향으로 반사시키고, 더 점프를 못하게 강제한다.
            Rigidbody2D rb = collision.gameObject.GetComponent<Rigidbody2D>();
```

```
            Vector2 velocity = rb.velocity;
            velocity.y = -rb.velocity.y;
            collision.gameObject.GetComponent<Rigidbody2D>().velocity = velocity;
            collision.gameObject.GetComponent<Mario>().m_fJumpTime = 1.0f;

            //아이템 생성 코드
            if (m_Item != null)
            {
                GameObject item = Instantiate(m_Item);
                Vector3 pos = transform.position + Vector3.up;
                item.transform.position = pos;
                item.transform.parent = null;

                m_Item = null;
            }

            HP -= 1;
        }
    }
}
```

5.3 Block2 클래스

Block2 클래스

```
using UnityEngine;

public class Block2 : Tile
{
    public GameObject m_Item; //마리오가 머리로 이 블록을 치면 나오는 아이템. null 이면 생성되는 아이템이 없다.
    [SerializeField] private GameObject m_Block3;  //마리오가 머리로 이 블록을 치면 바로 Block3 타입 블록으로 변한다.

    private void Awake()
    {
        m_BoxCollider = GetComponent<BoxCollider2D>();
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        //마리오가 머리로 이 블록에 부딪혔을 때 처리하는 코드
        if (collision.gameObject.CompareTag("Player") && base.DecideHitType(collision) == HitType.Bottom)
        {
            //마리오에게 아래 방향으로 반사시키고, 더 점프를 못하게 강제한다.
            Rigidbody2D rb = collision.gameObject.GetComponent<Rigidbody2D>();
            Vector2 velocity = rb.velocity;
            velocity.y = -rb.velocity.y;
            collision.gameObject.GetComponent<Rigidbody2D>().velocity = velocity;
            collision.gameObject.GetComponent<Mario>().m_fJumpTime = 1.0f;

            //아이템 생성 코드
            if (m_Item != null)
```

```
            {
                GameObject item = Instantiate(m_Item);
                Vector3 pos = transform.position + Vector3.up;
                item.transform.position = pos;
                item.transform.parent = null;
            }

            //Block3 타입 블록으로 변환
            GameObject block3 = Instantiate(m_Block3);
            block3.transform.position = gameObject.transform.position;
            block3.transform.parent = null;

            Destroy(gameObject);
        }
    }
}
```

5.4 Block3 클래스

Block3 클래스

```
using UnityEngine;

public class Block3 : Tile
{
    private void Awake()
    {
        m_BoxCollider = GetComponent<BoxCollider2D>();
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        //마리오가 머리로 이 블록에 부딪혔을 때 처리하는 코드
        if (collision.gameObject.CompareTag("Player") && base.DecideHitType(collision) ==
HitType.Bottom)
        {
            //마리오에게 아래 방향으로 반사시키고, 더 점프를 못하게 강제한다.
            Rigidbody2D rb = collision.gameObject.GetComponent<Rigidbody2D>();
            Vector2 velocity = rb.velocity;
            velocity.y = -rb.velocity.y;
            collision.gameObject.GetComponent<Rigidbody2D>().velocity = velocity;
            collision.gameObject.GetComponent<Mario>().m_fJumpTime = 1.0f;
        }
    }
}
```

5.5 Turtle 클래스

Turtle 클래스

```
using UnityEngine;

public class Turtle : Tile
```

```
{
    private float m_fSpeed = 1.8f;
    private int m_nDir = -1;
    [SerializeField] private GameObject m_Shell;  //이 몬스터의 머리가 밟혔을 때 생성되는
등껍데기 게임오브젝트

    public int m_nDirection
    {
        get { return m_nDir; }
        set
        {
            if (value > 0)
            {
                m_nDir = 1;
                transform.rotation = Quaternion.Euler(0.0f, 180.0f, 0.0f);
            }
            else if (value < 0)
            {
                m_nDir = -1;
                transform.rotation = Quaternion.identity;
            }
        }
    }

    //부모클래스의 DecideType 멤버함수를 재정의
    protected override HitType DecideHitType(Collision2D col)
    {
        float left = Mathf.Abs((col.transform.position.x +
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.x) -
(m_BoxCollider.transform.position.x - m_BoxCollider.bounds.extents.x));
        float right = Mathf.Abs((col.transform.position.x -
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.x) -
(m_BoxCollider.transform.position.x + m_BoxCollider.bounds.extents.x));
        float top = Mathf.Abs((col.transform.position.y -
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.y) -
(m_BoxCollider.transform.position.y + m_BoxCollider.bounds.extents.y));
        float bottom = Mathf.Abs((col.transform.position.y +
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.y) -
(m_BoxCollider.transform.position.y - m_BoxCollider.bounds.extents.y));

        float[] a = { left, right, top, bottom };

        int k = 0;
        float min = 100.0f;

        for (int i = 0; i < 4; i++)
        {
            if (a[i] < min)
            {
                min = a[i];
                k = i;
            }
        }

        // (left  vs  bottom) 또는 (right  vs  bottom) 값차이가 얼마 안나면 아래쪽면과
```

```
부딪혔다고 강제하는 코드.
        if (col.gameObject.CompareTag("Block"))
        {
            float l = Mathf.Abs(a[3] - a[0]);
            float r = Mathf.Abs(a[3] - a[1]);

            if (l < 0.05f || r < 0.05f)
            {
                k = 3;
            }
        }

        return (HitType)k;
    }

    private void Awake()
    {
        m_BoxCollider = GetComponent<BoxCollider2D>();
    }

    private void FixedUpdate()
    {
        //이동하는 코드
        transform.position += Vector3.right * m_nDir * m_fSpeed * Time.fixedDeltaTime;
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        //땅과 아이템과 충돌 시 충돌 처리를 건너뛴다.
        if (collision.gameObject.CompareTag("Ground")||
collision.gameObject.CompareTag("Item"))
            return;

        //마리오가 무적상태일때, 이 몬스터가 피해를 입게 하는 코드.
        if (collision.gameObject.CompareTag("Player"))
        {
            if (collision.gameObject.GetComponent<Mario>().m_blsImmune == true)
            {
                Destroy(gameObject);
                return;
            }
        }

        //충돌면 판정
        HitType hitType = DecideHitType(collision);

        //판정된 충돌면에 따른 처리
        switch (hitType)
        {
            case HitType.Bottom:   //이 몬스터의 아랫면에 부딪혔을 때
                {
                    if (collision.gameObject.tag == "Player")
                    {
                        collision.gameObject.GetComponent<Mario>().m_nState -= 1;
                    }
```

```
                }
                break;
            case HitType.Left:
            case HitType.Right:      //이 몬스터의 왼쪽 또는 오른쪽에 부딪혔을 때
                {
                    //마리오가 부딪혔으면 마리오에게 피해를 입히고, 아니면 이 몬스터의
방향을 바꾼다.
                    if (collision.gameObject.tag != "Player")
                    {
                        m_nDirection *= -1;
                    }
                    else
                    {
                        collision.gameObject.GetComponent<Mario>().m_nState -= 1;
                    }
                }
                break;
            case HitType.Top:        //이 몬스터의 위쪽면에 부딪혔을 때
                {
                    //마리오가 이 몬스터의 머리를 밟았을 때 처리하는 코드.
                    if (collision.gameObject.tag == "Player")
                    {
                        //파괴되고, 마리오가 강제로 약간 점프하게 한다.
                        Destroy(gameObject);
                        collision.gameObject.GetComponent<Rigidbody2D>().AddForce(Vector2.up
* 3.0f, ForceMode2D.Impulse);
                        collision.gameObject.GetComponent<Mario>().m_fJumpTime = 0.5f;

                        //등껍데기를 그 자리에 소환한다.
                        GameObject shell = Instantiate(m_Shell);
                        shell.transform.position = transform.position;
                        shell.transform.parent = null;

                        //점수 증가
                        GameInformation.m_nScore += 100;
                    }
                }
                break;
        }
    }
}
```

5.6 Gumba 클래스

Gumba 클래스

```
using UnityEngine;

public class Gumba : Tile
{
    private float m_fSpeed = 6.0f * 0.3f;
    private int m_nDir = -1;
    private bool m_IsDead = false;
    private Animator m_Animator;
```

```
    private Rigidbody2D m_Rigidbody;

    //m_nDir의 get, set 함수
    public int m_nDirection
    {
        get { return m_nDir; }
        set
        {
            if (value > 0)
            {
                m_nDir = 1;
                transform.rotation = Quaternion.Euler(0.0f, 180.0f, 0.0f);
            }
            else if (value < 0)
            {
                m_nDir = -1;
                transform.rotation = Quaternion.identity;
            }
        }
    }

    //부모클래스의 DecideType 멤버함수를 재정의
    protected override HitType DecideHitType(Collision2D col)
    {
        float left = Mathf.Abs((col.transform.position.x +
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.x) -
(m_BoxCollider.transform.position.x - m_BoxCollider.bounds.extents.x));
        float right = Mathf.Abs((col.transform.position.x -
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.x) -
(m_BoxCollider.transform.position.x + m_BoxCollider.bounds.extents.x));
        float top = Mathf.Abs((col.transform.position.y -
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.y) -
(m_BoxCollider.transform.position.y + m_BoxCollider.bounds.extents.y));
        float bottom = Mathf.Abs((col.transform.position.y +
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.y) -
(m_BoxCollider.transform.position.y - m_BoxCollider.bounds.extents.y));

        float[] a = { left, right, top, bottom };

        int k = 0;
        float min = 100.0f;

        for (int i = 0; i < 4; i++)
        {
            if (a[i] < min)
            {
                min = a[i];
                k = i;
            }
        }

        // (left  vs  bottom) 또는 (right  vs  bottom) 값차이가 얼마 안나면 아래쪽면과
부딪혔다고 강제하는 코드.
        if (col.gameObject.CompareTag("Block"))
        {
```

```
            float l = Mathf.Abs(a[3] - a[0]);
            float r = Mathf.Abs(a[3] - a[1]);

            if (l < 0.05f || r < 0.05f)
            {
                k = 3;
            }
        }

        return (HitType)k;
    }

    private void Awake()
    {
        m_BoxCollider = GetComponent<BoxCollider2D>();
        m_Animator = GetComponent<Animator>();
        m_Rigidbody = GetComponent<Rigidbody2D>();
    }

    private void FixedUpdate()
    {
        //이동하는 코드
        if (m_IsDead == false)
            transform.position += Vector3.right * m_nDir * m_fSpeed * Time.fixedDeltaTime;
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        //땅과 아이템과 충돌 시 충돌 처리를 건너뛴다.
        if (collision.gameObject.CompareTag("Ground") ||
collision.gameObject.CompareTag("Item"))
            return;

        //마리오가 무적상태일때, 이 몬스터가 피해를 입게 하는 코드.
        if (collision.gameObject.CompareTag("Player"))
        {
            if (collision.gameObject.GetComponent<Mario>().m_bIsImmune == true)
            {
                Destroy(gameObject);
                return;
            }
        }

        //충돌면 판정
        HitType hitType = DecideHitType(collision);

        //판정된 충돌면에 따른 처리
        switch(hitType)
        {
            case HitType.Bottom:    //이 몬스터의 아랫면에 부딪혔을 때
                {
                    //마리오가 피해를 입음
                    if (collision.gameObject.CompareTag("Player"))
                    {
                        collision.gameObject.GetComponent<Mario>().m_nState -= 1;
```

```
                    }
                }
                break;
            case HitType.Left:
            case HitType.Right:      //이 몬스터의 왼쪽 또는 오른쪽에 부딪혔을 때
                {
                    //마리오가 부딪혔으면 마리오에게 피해를 입히고, 아니면 이 몬스터의
방향을 바꾼다.
                    if (collision.gameObject.CompareTag("Player") == false)
                    {
                        m_nDirection *= -1;
                    }
                    else
                    {
                        collision.gameObject.GetComponent<Mario>().m_nState -= 1;
                    }
                }
                break;
            case HitType.Top:        //이 몬스터의 위쪽면에 부딪혔을 때
                {
                    //마리오가 이 몬스터의 머리를 밟았을 때 처리하는 코드.
                    if (collision.gameObject.CompareTag("Player"))
                    {
                        //0.2초의 처치 애니메이션과 함께 파괴됨
                        Destroy(gameObject, 0.2f);
                        m_Animator.SetBool("IsDead", true);
                        m_BoxCollider.enabled = false;
                        Destroy(this.m_Rigidbody);
                        m_IsDead = true;

                        //마리오가 강제로 약간 점프하게 한다.
                        collision.gameObject.GetComponent<Rigidbody2D>().AddForce(Vector2.up
* 3.0f, ForceMode2D.Impulse);
                        collision.gameObject.GetComponent<Mario>().m_fJumpTime = 0.5f;

                        //점수 증가
                        GameInformation.m_nScore += 100;
                    }
                }
                break;
        }
    }
}
```

5.7 Shell 클래스

Shell 클래스

```
using UnityEngine;

public class Shell : MonoBehaviour
{
    private float m_fSpeed = 6.0f;
    private bool m_bIsMoving = false;
```

```
    private int m_nDir = 0;
    private BoxCollider2D m_BoxCollider;

    private HitType DecideHitType(Collision2D col)
    {
        float left = Mathf.Abs((col.transform.position.x +
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.x) -
(m_BoxCollider.transform.position.x - m_BoxCollider.bounds.extents.x));
        float right = Mathf.Abs((col.transform.position.x -
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.x) -
(m_BoxCollider.transform.position.x + m_BoxCollider.bounds.extents.x));
        float top = Mathf.Abs((col.transform.position.y -
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.y) -
(m_BoxCollider.transform.position.y + m_BoxCollider.bounds.extents.y));
        float bottom = Mathf.Abs((col.transform.position.y +
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.y) -
(m_BoxCollider.transform.position.y - m_BoxCollider.bounds.extents.y));

        float[] a = { left, right, top, bottom };

        int k = 0;
        float min = 100.0f;

        for (int i = 0; i < 4; i++)
        {
            if (a[i] < min)
            {
                min = a[i];
                k = i;
            }
        }

        if (col.gameObject.CompareTag("Block"))
        {
            float l = Mathf.Abs(a[3] - a[0]);
            float r = Mathf.Abs(a[3] - a[1]);

            if (l < 0.05f || r < 0.05f)
            {
                k = 3;
            }
        }

        return (HitType)k;
    }

    private void Awake()
    {
        m_BoxCollider = GetComponent<BoxCollider2D>();
    }


    private void FixedUpdate()
    {
        //이동하는 코드
```

```
        transform.position += Vector3.right * m_nDir * m_fSpeed * Time.fixedDeltaTime;
    }


    private void OnCollisionEnter2D(Collision2D collision)
    {
        //땅과 아이템과 충돌 시 충돌 처리를 건너뛴다.
        if (collision.gameObject.CompareTag("Ground") ||
collision.gameObject.CompareTag("Item"))
            return;

        //이 오브젝트가 움직이고 있을때에는 마리오와 몬스터 모두에게 피해를 입힌다.
        if (m_blsMoving == true)
        {
            //마리오와 충돌했을 때의 처리
            if (collision.gameObject.CompareTag("Player"))
            {
                //마리오가 무적일 때는 이 오브젝트가 파괴된다.
                if (collision.gameObject.GetComponent<Mario>().m_blsImmune == true)
                {
                    Destroy(gameObject);
                    return;
                }
                //아니면 마리오가 피해를 입는다.
                else
                    collision.gameObject.GetComponent<Mario>().m_nState -= 1;
            }
            //몬스터와 충돌했을 때의 처리
            else if (collision.gameObject.CompareTag("Monster"))
            {
                //점수를 증가시키고, 몬스터에게 피해를 입힌다.
                GameInformation.m_nScore += 100;
                Destroy(collision.gameObject);
            }
            //다른 지형과 충돌했을 때의 처리
            else
            {
                HitType hitType = DecideHitType(collision);

                //다른 지형과 왼쪽 또는 오른쪽으로 부딪혔다면 방향을 바꾼다.
                if(hitType == HitType.Left || hitType == HitType.Right)
                {
                    m_nDir *= -1;
                }
            }
        }

        //마리오가 가만히 멈춰있는 이 등껍데기와 충돌했을 때의 처리
        if (collision.gameObject.CompareTag("Player") && m_blsMoving == false)
        {
            //이 오브젝트의 x좌표에서 마리오의 x좌표를 뺀다.
            float a = transform.position.x - collision.transform.position.x;

            //그 값의 부호에 따라 초기 방향이 결정된다.
            if (a > 0)
```

```
            {
                m_nDir = 1;
            }
            else
            {
                m_nDir = -1;
            }
            //약간의 위치 조정
            transform.position += Vector3.right * a * (1.0f + 0.5f / Mathf.Abs(a));

            m_blsMoving = true;
        }
    }
}
```

5.8 Mario 클래스

Mario 클래스

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class Mario : MonoBehaviour
{
    private static Mario m_Instance;

    private int m_nDir;
    private int m_nstate = 1; //작은 마리오인지 큰 마리오인지 등을 나타내는 멤버변수
    private float m_fStarBuffTimer = 0.0f; //별 버프 지속 시간 체크
    private Vector3 m_Velocity = Vector3.zero;
    private Rigidbody2D m_Rigidbody;
    private BoxCollider2D m_BoxCollider;
    private Animator m_Animator;
    public float m_fJumpTime = 0.5f;
    public bool m_blsJumpKeyActive = true;  //점프 조작이 가능한지 나타내는 멤버변수
    public bool m_blsGrounded = false;   //마리오가 지형을 밟고 있는지 나타내는 멤버변수
    public bool m_blsImmune = false;     //마리오가 무적상태인지 나타내는 멤버변수
    public bool m_blsAlive = true;

    public static Mario Instance
    {
        get { return  m_Instance; }
        private set { m_Instance = value; }
    }

    public int m_nDirection
    {
        get { return m_nDir; }
        set
        {
            if (value > 0)
            {
                m_nDir = 1;
                transform.rotation = Quaternion.identity;
```

```
            }
            else if (value < 0)
            {
                m_nDir = -1;
                transform.rotation = Quaternion.Euler(0.0f, 180.0f, 0.0f);
            }
        }
    }
    //m_nstate 변수의 get, set 함수.
    public int m_nState
    {
        get { return m_nstate; }
        set
        {
            //m_nstate의 값을 정하는 부분. 0이면 즉시 파괴하며,
            //2(큰 마리오) 또는 3(불마법사 마리오)일때 피해를 입으면 무조건 1(작은 마리오)로
강제로 내려버린다.
            if (value == 0)
            {
                m_blsAlive = false;
                Destroy(gameObject);
            }
            else if (m_nstate > value)
                m_nstate = 1;
            else
                m_nstate = value;


            //설정된 m_nstate값에 따라 충돌체 사이즈 조절, groundDetector 위치 조절을 해줌
            Vector2 size;
            size.x = 0.75f;
            size.y = 1.05f;

            switch (m_nstate)
            {
                case 1:
                    {
                        size.y = 1.05f;
                    }
                    break;
                case 2:
                    {
                        size.y = 1.7f;
                    }
                    break;
                case 3:
                    {
                        size.y = 1.7f;
                    }
                    break;
            }
            m_BoxCollider.size = size;

            Vector3 groundDetectorPos;
            groundDetectorPos.x = 0.0f;
```

```
            groundDetectorPos.y = -m_BoxCollider.size.y / 2.0f;
            groundDetectorPos.z = 0.0f;
            transform.GetChild(0).transform.localPosition = groundDetectorPos;
        }
    }
    private void Awake()
    {
        m_Rigidbody = GetComponent<Rigidbody2D>();
        m_BoxCollider = GetComponent<BoxCollider2D>();
        m_Animator = GetComponent<Animator>();

        if (m_Instance == null)
        {
            m_Instance = this;
        }
        else
            Destroy(this.gameObject);
    }
    private void Start()
    {
        //씬이 시작될 때, 저장되어있던 마리오 정보를 가져와서 대입한다.
        m_nState = GameInformation.m_nMarioState;
    }
    private void Update()
    {
        //애니메이터에 정보를 전달해준다.
        bool IsMoving = Mathf.Abs(m_Velocity.x) > 0;
        bool IsJumping = m_fJumpTime > 0;
        int behavior = 0;

        if (IsJumping)
        {
            behavior = 2;
        }
        else if(IsMoving)
        {
            behavior = 1;
        }
        else
        {
            behavior = 0;
        }

        m_Animator.SetBool("IsGround", m_bIsGrounded);
        m_Animator.SetInteger("State", m_nstate);
        m_Animator.SetInteger("Behavior", behavior);

    }

    private void FixedUpdate()
    {
        //좌우 이동 코드
        //어느 방향키를 누르냐에 따라 마리오의 가속도를 조절되도록 설계하였다.
        float acceleration = 0.0f;
        if(Input.GetKey(KeyCode.LeftArrow))
```

```
        {
            if(m_Velocity.x > -1.0f)
            {
                m_nDirection = -1;
                acceleration = - (1.0f/0.3f);
            }
        }
        else if (Input.GetKey(KeyCode.RightArrow))
        {
            if (m_Velocity.x < 1.0f)
            {
                m_nDirection = 1;
                acceleration = 1.0f / 0.3f;
            }
        }
        else
        {
            //아무 키를 안 눌었을 때에는 속력이 점점 줄도록 하였다.
            if(m_Velocity.x != 0.0f)
            {
                acceleration = -(m_Velocity.x / Mathf.Abs(m_Velocity.x)) * (1.0f / 0.3f);
            }
        }
        m_Velocity += Vector3.right * (acceleration * Time.fixedDeltaTime);
        transform.position += m_Velocity * 6.0f * Time.fixedDeltaTime;



        //점프 코드
        if (Input.GetKey(KeyCode.Space))
        {
            if(m_fJumpTime < 0.5f && m_blsJumpKeyActive)
            {
                //오랫동안 누를 수록 마리오에게 위쪽으로 가해지는 힘의 크기가 작아지도록
설계되어있다.
                // f(x) = 80 * (1 - 2x)^4  라는 함수로 정의했다.
                float jumpscale = 90.0f * Mathf.Pow(1.0f - 2.0f * m_fJumpTime, 4.0f) *
Time.fixedDeltaTime;
                Vector2 jumpforce;
                jumpforce.x = 0.0f;
                jumpforce.y = jumpscale;
                m_Rigidbody.AddForce(jumpforce, ForceMode2D.Impulse);
                m_fJumpTime += Time.fixedDeltaTime;
            }
        }
        //점프키를 뗐다면 땅에 닿기 전까지 더 이상 점프할 수 없다.
        if(Input.GetKeyUp(KeyCode.Space))
        {
            m_blsJumpKeyActive = false;
        }

        //무적 버프 코드
        if(m_blsImmune == true)
        {
            if (m_fStarBuffTimer < 10.0f)
```

```
                m_fStarBuffTimer += Time.fixedDeltaTime;
            else
            {
                m_blsImmune = false;
                m_fStarBuffTimer = 0.0f;
            }
        }
        //스테이지 시간 감소 코드.
        GameInformation.m_fTime -= Time.fixedDeltaTime;
    }

    private void OnDestroy()
    {
        //맵 이동을 할때 마리오의 상태 저장
        if (m_blsAlive)
        {
            GameInformation.m_nMarioState = m_nstate;
            return;
        }

        //마리오가 죽었을 때의 코드
        //목숨을 차감하고 어느 씬을 로드할지 결정한다.
        GameInformation.m_nHeart -= 1;
        string scenename;
        //목숨이 남아있다면 1 스테이지로 다시 이동한다.
        if (GameInformation.m_nHeart > 0)
            scenename = "SuperMarioMap";
        //목숨이 모두 소진되었다면 게임 시작 씬으로 이동한다.
        else
        {
            scenename = "StartGameScene";
            GameInformation.m_nHeart = 3;
        }
        //목숨을 제외한 게임 정보를 초기화 한다.
        GameInformation.m_nScore = 0;
        GameInformation.m_fTime = 300.0f;
        GameInformation.m_nMarioState = 1;
        Vector3 NextSpawnPos;
        NextSpawnPos.x = 1.0f;
        NextSpawnPos.y = -12.0f;
        NextSpawnPos.z = 0.0f;
        MapLoader.MarioPos = NextSpawnPos;

        SceneManager.LoadScene(scenename);
    }
}
```

5.9 GroundDetector 클래스

GroundDetector 클래스

```
using UnityEngine;

public class GroundDetector : MonoBehaviour
```

```
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        //지형에 발이 닿아야 점프 시간이 초기화되면서 다시 점프를 할 수 있다.
        gameObject.transform.parent.GetComponent<Mario>().m_blsGrounded = true;
        gameObject.transform.parent.GetComponent<Mario>().m_fJumpTime = 0.0f;
    }
    private void OnTriggerExit2D(Collider2D collision)
    {
        //다른 충돌체가 이 충돌체에서 벗어나면 발에 아무런 지형이 없다고 판정한다.
        gameObject.transform.parent.GetComponent<Mario>().m_blsGrounded = false;
    }
}
```

5.10 MapLoader 클래스

MapLoader 클래스

```
using System.Collections.Generic;
using System.IO;
using UnityEngine;


public class MapLoader : MonoBehaviour
{
    //이 타일맵의 정보를 담는 2차원 배열의 멤버변수
    private Tile[,] map;
    //텍스트 문서에서의 문자를 통해 게임오브젝트 검색을 위한 사전형 멤버변수
    private Dictionary<char, GameObject> m_GameObjects = new Dictionary<char, GameObject>();

    //타일 프리팹
    [SerializeField] private GameObject m_Prefab0;
    [SerializeField] private GameObject m_Prefab1;
    [SerializeField] private GameObject m_Prefab2;
    [SerializeField] private GameObject m_Prefab3;
    [SerializeField] private GameObject m_Prefab4;
    [SerializeField] private GameObject m_Prefab5;
    [SerializeField] private GameObject m_Prefab6;
    [SerializeField] private GameObject m_Prefab7;
    [SerializeField] private GameObject m_Prefab8;
    [SerializeField] private GameObject m_Prefab9;

    //아이템 프리팹
    [SerializeField] private GameObject m_ItemMushroom;
    [SerializeField] private GameObject m_ItemFlower;
    [SerializeField] private GameObject m_ItemStar;

    //마리오 프리팹
    [SerializeField] private GameObject m_Mario;

    //마리오가 스폰될 위치
    public static Vector3 MarioPos = new Vector3(1.0f, -12.0f, 0.0f);

    //m_GameObjects 에 (문자, 게임오브젝트) 쌍을 등록하는 멤버함수
```

```
    private void InitDictionary()
    {
        m_GameObjects.Add('0', m_Prefab0);
        m_GameObjects.Add('1', m_Prefab1);
        m_GameObjects.Add('2', m_Prefab2);
        m_GameObjects.Add('3', m_Prefab3);
        m_GameObjects.Add('4', m_Prefab4);
        m_GameObjects.Add('5', m_Prefab5);
        m_GameObjects.Add('6', m_Prefab6);
        m_GameObjects.Add('7', m_Prefab7);
        m_GameObjects.Add('8', m_Prefab8);
        m_GameObjects.Add('9', m_Prefab9);
    }

    //맵에 디테일한 세팅을 하는 멤버함수 (블록의 아이템 설정, 블록의 내구도 설정 등)
    private void SettingMapDetail()
    {

        map[9, 11].gameObject.GetComponent<Block2>().m_Item = m_ItemMushroom;
        map[9, 35].gameObject.GetComponentInChildren<Portal>().m_SceneName = "HiddenMap";
        map[9, 62].gameObject.GetComponent<Block2>().m_Item = m_ItemFlower;
        map[9, 80].gameObject.GetComponent<Block1>().HP = 5;
        map[9, 80].gameObject.GetComponent<Block1>().m_BlockAfterHPisZero = m_Prefab5;
        map[9, 87].gameObject.GetComponent<Block1>().m_Item = m_ItemStar;
        map[9, 87].gameObject.GetComponent<Block1>().m_BlockAfterHPisZero = m_Prefab5;
        map[5, 95].gameObject.GetComponent<Block2>().m_Item = m_ItemFlower;

    }

    //텍스트 파일의 줄 수를 반환하는 함수
    private int CountNumberOfRow(string fileName)
    {
        int count = 0;
        StreamReader sr = new StreamReader(fileName);

        string line = sr.ReadLine();
        while(line != null)
        {
            count++;
            line = sr.ReadLine();
        }
        sr.Close();

        return count;
    }
    private void Awake()
    {
        InitDictionary();

        StreamReader sr;
        sr = new StreamReader(@".₩Assets₩Resources₩SuperMarioMap.txt");

        //행 갯수를 계산
        int _rowCount = CountNumberOfRow(@".₩Assets₩Resources₩SuperMarioMap.txt");
```

```
        //열 갯수를 계산
        int _colCount = 0;
        string line = sr.ReadLine();
        for (int i = 0; i<line.Length; i++)
        {
            if (line[i]== ' ')
            {
                continue;
            }
            else
            {
                _colCount++;
            }
        }
        map = new Tile[_rowCount, _colCount];


        //텍스트 문서를 읽으면서 문자에 따른 알맞은 게임오브젝트를 배치하고 map 변수에
저장한다.
        int index = 0;
        while (line != null)
        {
            for (int i = 0; i < line.Length; i++)
            {
                if (line[i] == ' ')
                {
                    continue;
                }
                else
                {
                    int j = i / 2;

                    Vector3 pos;
                    pos.x = j;
                    pos.y = -index;
                    pos.z = 0;

                    GameObject go = Instantiate(m_GameObjects[line[i]]);
                    go.transform.position = pos;
                    map[index, j] = go.GetComponent<Tile>();
                }
            }
            index++;
            line = sr.ReadLine();
        }
        sr.Close();

        SettingMapDetail();

        //마리오 생성
        GameObject mario = Instantiate(m_Mario);
        mario.transform.position = MarioPos;
        mario.transform.parent = null;
    }
```

```
    private void FixedUpdate()
    {
        //맵 컬링
        for (int j = 0; j < map.GetLength(1); j++)
        {
            for (int i = 0; i < map.GetLength(0); i++)
            {
                try
                {
                    //오브젝트의 x좌표가 카메라의 영역 밖이라면 그 오브젝트를 비활성화
시킨다.
                    float x = Mathf.Abs(CameraMove.m_CameraPos.x -
map[i,j].gameObject.transform.position.x);
                    bool a;

                    if (x < CameraMove.m_fWidth / 2.0f + 1.0f)
                    {
                        a = true;
                    }
                    else
                    {
                        a = false;
                    }
                    map[i, j].gameObject.SetActive(a);
                }
                catch
                {
                    continue;
                }

            }
        }
    }
}
```

5.11 HiddenMapLoader 클래스

HiddenMapLoader 클래스

```
using System.Collections.Generic;
using System.IO;
using UnityEngine;

public class HiddenMapLoader : MonoBehaviour
{
    //이 타일맵의 정보를 담는 2차원 배열의 멤버변수
    private Tile[,] map;
    //텍스트 문서에서의 문자를 통해 게임오브젝트 검색을 위한 사전형 멤버변수
    private Dictionary<char, GameObject> m_GameObjects = new Dictionary<char, GameObject>();

    //타일 프리팹
    [SerializeField] private GameObject m_Prefab0;
    [SerializeField] private GameObject m_Prefab1;
    [SerializeField] private GameObject m_Prefab2;
```

```
    [SerializeField] private GameObject m_Prefab3;
    [SerializeField] private GameObject m_Prefab4;

    //마리오 프리팹
    [SerializeField] private GameObject m_Mario;

    //m_GameObjects 에 (문자, 게임오브젝트) 쌍을 등록하는 멤버함수
    private void InitDictionary()
    {
        m_GameObjects.Add('0', m_Prefab0);
        m_GameObjects.Add('1', m_Prefab1);
        m_GameObjects.Add('2', m_Prefab2);
        m_GameObjects.Add('3', m_Prefab3);
        m_GameObjects.Add('4', m_Prefab4);
    }

    //텍스트 파일의 줄 수를 반환하는 함수
    private int CountNumberOfRow(string fileName)
    {
        int count = 0;
        StreamReader sr = new StreamReader(fileName);

        string line = sr.ReadLine();
        while (line != null)
        {
            count++;
            line = sr.ReadLine();
        }
        sr.Close();

        return count;
    }

    private void Awake()
    {
        InitDictionary();

        StreamReader sr;
        sr = new StreamReader(@".₩Assets₩Resources₩HiddenMap.txt");

        //행 갯수를 계산
        int _rowCount = CountNumberOfRow(@".₩Assets₩Resources₩HiddenMap.txt");

        //열 갯수를 계산
        int _colCount = 0;
        string line = sr.ReadLine();
        for (int i = 0; i < line.Length; i++)
        {
            if (line[i] == ' ')
            {
                continue;
            }
            else
            {
                _colCount++;
```

```
            }
        }
        map = new Tile[_rowCount, _colCount];


        //텍스트 문서를 읽으면서 문자에 따른 알맞은 게임오브젝트를 배치하고 map 변수에
저장한다.
        int index = 0;
        while (line != null)
        {
            for (int i = 0; i < line.Length; i++)
            {
                if (line[i] == ' ')
                {
                    continue;
                }
                else
                {
                    int j = i / 2;

                    Vector3 pos;
                    pos.x = j;
                    pos.y = -index;
                    pos.z = 0;

                    GameObject go = Instantiate(m_GameObjects[line[i]]);
                    go.transform.position = pos;
                    map[index, j] = go.GetComponent<Tile>();
                }
            }
            index++;
            line = sr.ReadLine();
        }
        sr.Close();

        //마리오 생성
        GameObject mario = Instantiate(m_Mario);
        Vector3 mariopos;
        mariopos.x = 1.0f;
        mariopos.y = 1.0f;
        mariopos.z = 0.0f;
        mario.transform.position = mariopos;
    }
}
```

5.12 CameraMove 클래스

CameraMove

```
using UnityEngine;

public class CameraMove : MonoBehaviour
{
```

```
    private float m_TargetPosX;
    public static float m_fWidth;  //이 카메라의 너비값을 담는 멤버변수이다.
    public static Vector3 m_CameraPos;  //이 카메라의 위치를 담는 멤버변수이다.

    private void Awake()
    {
        //16:10 비율의 카메라라서 가로길이는 세로 길이의 1.6배이다.
        m_fWidth = this.gameObject.GetComponent<Camera>().orthographicSize * 2.0f * 1.6f;
    }
    private void FixedUpdate()
    {
        if (Mario.Instance == null)
            return;

        //마리오 위치에 따라 카메라의 위치를 결정한다.
        m_TargetPosX = Mario.Instance.gameObject.transform.position.x;
        bool leftBound = m_TargetPosX < (m_fWidth / 2.0f) - 3.0f;
        bool rightBound = m_TargetPosX > 178.0f - ((m_fWidth / 2.0f) + 3.0f);

        float x;
        if (leftBound)
        {
            x = 0.0f + m_fWidth / 2.0f;
        }
        else if (rightBound)
        {
            x = 178.0f - m_fWidth / 2.0f;
        }
        else
        {
            x = m_TargetPosX + 3.0f;
        }

        Vector3 pos = transform.position;
        pos.x = x;
        transform.position = pos;

        m_CameraPos = pos;
    }
}
```

5.13 Mushroom 클래스

Mushroom 클래스

```
using UnityEngine;

public class Mushroom : MonoBehaviour
{
    private int m_nDir = 1;
    private float m_fSpeed = 3.0f;
    private BoxCollider2D m_BoxCollider;
```

```
    //어느 면이 부딪혔는지 판정하는 함수
    private HitType DecideHitType(Collision2D col)
    {
        float left = Mathf.Abs((col.transform.position.x +
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.x) -
(m_BoxCollider.transform.position.x - m_BoxCollider.bounds.extents.x));
        float right = Mathf.Abs((col.transform.position.x -
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.x) -
(m_BoxCollider.transform.position.x + m_BoxCollider.bounds.extents.x ));
        float top = Mathf.Abs((col.transform.position.y -
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.y) -
(m_BoxCollider.transform.position.y + m_BoxCollider.bounds.extents.y));
        float bottom = Mathf.Abs((col.transform.position.y +
col.gameObject.GetComponent<BoxCollider2D>().bounds.extents.y) -
(m_BoxCollider.transform.position.y - m_BoxCollider.bounds.extents.y));

        float[] a = { left, right, top, bottom };

        int k = 0;
        float min = 100.0f;

        for (int i = 0; i < 4; i++)
        {
            if (a[i] < min)
            {
                min = a[i];
                k = i;
            }
        }

        // (left  vs  bottom) 또는 (right  vs  bottom) 값차이가 얼마 안나면 아래쪽면과
부딪혔다고 강제하는 코드.
        if (col.gameObject.CompareTag("Block"))
        {
            float l = Mathf.Abs(a[3] - a[0]);
            float r = Mathf.Abs(a[3] - a[1]);

            if (l < 0.05f || r < 0.05f)
            {
                k = 3;
            }
        }

        return (HitType)k;
    }
    private void Awake()
    {
        m_BoxCollider = GetComponent<BoxCollider2D>();
    }
    private void FixedUpdate()
    {
        //이동하는 코드
        transform.position += Vector3.right * m_nDir * m_fSpeed * Time.fixedDeltaTime;
    }
    private void OnCollisionEnter2D(Collision2D collision)
```

```
    {
        //땅이나 몬스터와 충돌 시 충돌 처리를 건너뛴다.
        if (collision.gameObject.CompareTag("Ground") ||
collision.gameObject.CompareTag("Monster"))
            return;

        //마리오와 충돌했을 때의 처리
        if(collision.gameObject.CompareTag("Player"))
        {
            //작은 마리오일때만 큰 마리오로 성장 시킨다.
            if (collision.gameObject.GetComponent<Mario>().m_nState == 1)
            {
                collision.gameObject.GetComponent<Mario>().m_nState = 2;
            }
            Destroy(this.gameObject);
        }

        //충돌면 판정
        HitType hittype = DecideHitType(collision);

        //판정된 충돌면에 따른 처리
        switch (hittype)
        {
            //좌우로 부딪혔을 때만 방향을 전환한다.
            case HitType.Left:
            case HitType.Right:
                {
                    m_nDir *= -1;
                }
                break;
            case HitType.Top:
            case HitType.Bottom:
                break;
        }
    }
}
```

5.14 Flower 클래스

Flower 클래스

```
using UnityEngine;

public class Flower : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        //마리오와 충돌했을 때의 처리
        if (collision.gameObject.CompareTag("Player"))
        {
            //불마법사 마리오로 성장시킨다.
            collision.gameObject.GetComponent<Mario>().m_nState = 3;
            Destroy(gameObject);
        }
```

```
    }
}
```

5.15 Star 클래스

Star 클래스

```
using UnityEngine;

public class Star : MonoBehaviour
{
    private Rigidbody2D m_Rigidbody;

    private void Awake()
    {
        m_Rigidbody = GetComponent<Rigidbody2D>();
    }
    private void FixedUpdate()
    {
        //이동하는 코드
        transform.position += Vector3.right * 2.0f * Time.fixedDeltaTime;
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        //마리오와 충돌했을 때의 처리
        if(collision.gameObject.CompareTag("Player"))
        {
            //마리오에게 무적 버프를 부여한다.
            collision.gameObject.GetComponent<Mario>().m_blsImmune = true;
            Destroy(gameObject);
        }
        //지형에 닿을 때마다 점프를 한다.
        else if(collision.gameObject.CompareTag("Ground") ||
collision.gameObject.CompareTag("Block"))
        {
            m_Rigidbody.AddForce(Vector2.up * 3.0f, ForceMode2D.Impulse);
        }
    }
}
```

5.16 Portal 클래스

Portal 클래스

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class Portal : MonoBehaviour
{
    public KeyCode m_Keycode;
    public string m_SceneName;
```

```
    private void OnTriggerStay2D(Collider2D collision)
    {
        //이 포탈에 아무 설정이 안 되어있다면 건너뛴다.
        if (m_SceneName == null || m_Keycode == KeyCode.None)
            return;

        //마리오가 포탈 위에서 알맞은 키를 누르면 다음에 돌아왔을때 스폰될 위치를 저장하고
다른 맵으로 이동한다.
        if (collision.gameObject.CompareTag("Player") && Input.GetKey(m_Keycode))
        {
            Vector3 NextSpawnPos;
            NextSpawnPos.x = 132.0f;
            NextSpawnPos.y = -10.0f;
            NextSpawnPos.z = 0.0f;
            MapLoader.MarioPos = NextSpawnPos;
            SceneManager.LoadScene(m_SceneName);
        }
    }
}
```

5.17 Coin 클래스

Coin 클래스

```
using UnityEngine;

public class Coin : Tile
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        //마리오와 충돌했을때 점수 증가
        if (collision.gameObject.CompareTag("Player"))
        {
            Destroy(gameObject);
            GameInformation.m_nScore += 200;
        }
    }
}
```

5.18 GoalPoint 클래스

GoalPoint 클래스

```
using UnityEngine;

public class GoalPoint : MonoBehaviour
{
    [SerializeField] private GameObject m_ClearMario;
    private bool m_bisGameFinished = false;

    private void FixedUpdate()
    {
        //마리오가 골인 지점에 도달했을때 깃발 오브젝트가 서서히 내려가는 코드
```

```
        if(m_bisGameFinished)
        {
            if(gameObject.transform.parent.GetChild(0).transform.position.y >
gameObject.transform.parent.position.y + 0.5f)
                gameObject.transform.parent.GetChild(0).transform.position += Vector3.down *
Time.fixedDeltaTime * (15.5f / 3.0f);
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        //마리오가 이 오브젝트에 닿으면 마리오를 파괴하고 클리어 애니메이션이 있는 프리팹을
생성한다.
        if (collision.gameObject.CompareTag("Player"))
        {
            GameInformation.m_nMarioState =
collision.gameObject.GetComponent<Mario>().m_nState;

            GameObject go = Instantiate(m_ClearMario);
            Vector3 pos;
            pos.x = transform.position.x;
            pos.y = collision.gameObject.transform.position.y;
            pos.z = 0.0f;
            go.transform.position = pos;
            go.GetComponent<ClearMario>().m_nState = GameInformation.m_nMarioState;

            Destroy(collision.gameObject);

            m_bisGameFinished = true;        }
    }
}
```

5.19 ClearMario 클래스

ClearMario

```
using UnityEngine;

public class ClearMario : MonoBehaviour
{
    private int m_nstate;  //작은 마리오인지 큰 마리오인지 등을 나타내는 멤버변수
    private float m_fTimer = 0.0f;
    private BoxCollider2D m_BoxCollider;
    private Rigidbody2D m_Rigidbody;
    private Animator m_Animator;

    //m_nstate 변수의 get, set 함수.
    public int m_nState
    {
        get { return m_nstate; }
        set
        {
            m_nstate = value;
```

```
            //설정된 m_nstate값에 따라 충돌체 사이즈 조절
            Vector2 size;
            size.x = 0.75f;
            size.y = 1.05f;

            switch (m_nstate)
            {
                case 1:
                    {
                        size.y = 1.05f;
                    }
                    break;
                case 2:
                    {
                        size.y = 1.7f;
                    }
                    break;
                case 3:
                    {
                        size.y = 1.7f;
                    }
                    break;
            }
            m_BoxCollider.size = size;
        }
    }
    /// <summary>
    /// 3초동안 깃발을 내리고 그 이후에는 오른쪽으로 걸어가면서 성 안으로 들어가도록
설계하였다,
    /// </summary>

    private void Awake()
    {
        m_BoxCollider = GetComponent<BoxCollider2D>();
        m_Rigidbody = GetComponent<Rigidbody2D>();
        m_Animator = GetComponent<Animator>();

        //깃발을 내리는 처리는 등속도 운동으로 적용하기 위해 처음에 중력에 영향을 받지
않도록 하였다.
        m_Rigidbody.gravityScale = 0.0f;

        Destroy(gameObject, 7.0f);
    }
    private void Start()
    {
        //애니메이션 설정
        m_Animator.SetInteger("State", m_nState);
    }

    private void FixedUpdate()
    {
        //3초 동안 깃발을 내리고, 이후 약 4초동안 성 안으로 이동한다.
        if (m_fTimer < 3.0f)
        {
            transform.position += Vector3.down * Time.fixedDeltaTime * (15.5f / 3.0f);
```

```
        }
        else if(m_fTimer > 6.9f)
        {
            GameInformation.m_blsClear = true;
        }
        else
        {
            m_Animator.SetBool("IsMoving", true);
            m_Rigidbody.gravityScale = 1.0f;
            transform.position += 3.0f * Time.fixedDeltaTime * Vector3.right;
        }
        m_fTimer += Time.fixedDeltaTime;
    }
}
```

5.20 FreeFallDetector 클래스

FreeFallDetector

```
using UnityEngine;

public class FreeFallDetector : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        //마리오와 충돌하게 된다면 낙사로 판정한다.
        if(collision.gameObject.CompareTag("Player"))
        {
            collision.gameObject.GetComponent<Mario>().m_blsAlive = false;
        }
        Destroy(collision.gameObject);
    }
}
```

5.21 GameInformation 클래스

GameInformation

```
//게임 정보를 저장하는 정적 클래스
public static class GameInformation
{
    public static int m_nScore = 0; //누적된 점수을 나타내는 변수
    public static int m_nHeart = 3; //남은 목숨을 나타내는 변수
    public static float m_fTime = 300.0f; //남은 시간을 나타내는 변수
    public static int m_nMarioState = 1; //마리오의 상태(작은 마리오, 큰마리오 등)을
저장하는 변수

    public static bool m_blsClear = false; //스테이지가 클리어됐는지 나타내는 변수
}
```

5.22 GameManager 클래스

GameManager

```
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    [SerializeField] private GameObject m_ResultPanel; //스테이지 클리어 시 활성화되어
결과를 알려주는 판넬
    [SerializeField] private TMP_Text m_TMPResultHeart; //클리어 시 남은 목숨을 보여주는 TMP
Text
    [SerializeField] private TMP_Text m_TMPResultScore; //클리어 시 쌓은 점수를 보여주는 TMP
Text
    [SerializeField] private TMP_Text m_TMPResultTime; //클리어 시 남은 시간을 보여주는 TMP
Text
    [SerializeField] private TMP_Text m_TMPTotalScore; //클리어 시 총 점수를 보여주는 TMP
Text
    private bool m_blsFinished = false;


    private void Update()
    {
        if (m_blsFinished)
            return;

        //클리어하면 점수를 집계해 결과 판넬을 활성화한다.
        if(GameInformation.m_blsClear)
        {
            m_ResultPanel.SetActive(true);

            m_TMPResultHeart.text = GameInformation.m_nHeart.ToString();
            m_TMPResultScore.text = GameInformation.m_nScore.ToString();
            m_TMPResultTime.text = ((int)GameInformation.m_fTime).ToString();

            int totalscore = GameInformation.m_nScore + GameInformation.m_nHeart * 300 +
((int)GameInformation.m_fTime) * 3;
            m_TMPTotalScore.text = totalscore.ToString();

            m_blsFinished = true;
        }
    }
    /// <summary>
    /// 아래의 세 함수는 버튼이 클릭되었을때 사용될 함수이다.
    /// </summary>

    // 씬 이동을 하는 함수이다.
    public void MoveToScene(string name)
    {
        SceneManager.LoadScene(name);
    }
    // 게임 정보를 초기화하는 함수이다.
    public void ResetInfomation()
    {
        GameInformation.m_nHeart = 3;
```

```
        GameInformation.m_nScore = 0;
        GameInformation.m_fTime = 300.0f;

        GameInformation.m_nMarioState = 1;

        GameInformation.m_blsClear = false;

        MapLoader.MarioPos.x = 1.0f;
        MapLoader.MarioPos.y = -12.0f;
    }
    //게임을 종료하는 함수이다.
    public void QuitGame()
    {
        Application.Quit();
    }
}
```

5.23 GameInfoText 클래스

GameInfoText

```
using TMPro;
using UnityEngine;

/// <summary>
/// 게임이 진행되는 동안 화면에 나타낼 TMP Text를 관리하는 클래스이다.
/// </summary>
public class GameInfoText : MonoBehaviour
{
    [SerializeField] private TMP_Text m_ScoreText; //누적된 점수를 나타내는 TMP Text
    [SerializeField] private TMP_Text m_HeartText; //남은 목숨를 나타내는 TMP Text
    [SerializeField] private TMP_Text m_TimeText;  //남은 시간를 나타내는 TMP Text

    //누적된 점수, 남은 목숨, 남은 시간을 각각 TMP Text에 입력한다.
    private void Start()
    {
        m_HeartText.text = GameInformation.m_nHeart.ToString();
    }
    private void FixedUpdate()
    {
        m_ScoreText.text = GameInformation.m_nScore.ToString();
        int time = (int)GameInformation.m_fTime;
        m_TimeText.text = time.ToString();
    }
}
```