




THE AVOIDER

게임 디자인

게임제작기초

게임학부 21210091 김민재



목차

1. 개요
2. 특징
3. 게임 플레이 방식
4. 공격 타입
5. 주요 스크립트
6. 데이터 관리
7. 대표 로직 설명
8. 기타

개요

이름	The Avoider
장르	피지컬, 3인칭 피하기 게임
플랫폼	PC, Android, IOS
차원	3D
기획의도	<p>‘언더테일’이라는 게임의 물체를 피하는 방식을 가져와 3D로 재해석해 구현했다.</p> <p>현재 언더테일의 샌즈 라운드를 패러디해 구현중이다.</p> <p>공격을 피하는 공간은 정육면체의 입체 공간에서 이루어진다.</p> <p>오직 물체를 피해서 살아남는 것이 목적이다.</p>

특징

● 어려운 난이도

각 스테이지는 어려운 난이도로 제작할 예정이며 플레이어에게 도전 정신을 일으킬 수 있을 것이다. 회복 가능 횟수를 조정함으로써 더욱 난이도를 높일 수 있을 것이며 한 번도 체력을 소모하지 않고 플레이 하는 유저도 있을 것이라고 추정 할 수 있다.

● 독특한 피하기 게임

현재까지 지정된 공간에서 피하기만 진행하는 3D게임은 잘 없었다. 때문에 플레이어는 신선한 느낌을 받을 것이다.

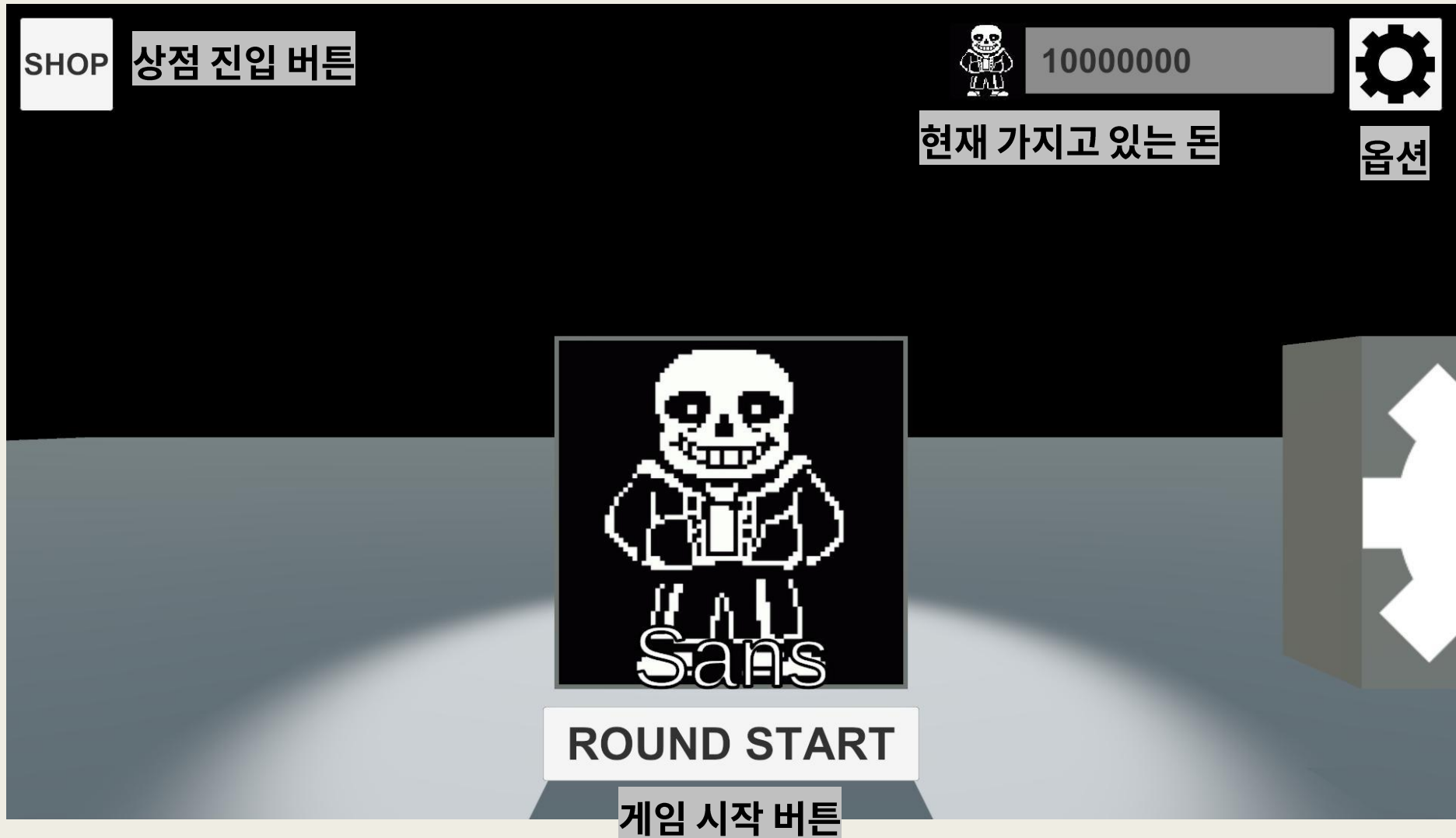
● 단순한 목표

이 게임의 목표는 물체를 피하는 것으로 매우 단순하기 때문에 플레이어는 물체를 피하는 것에 집중 할 수 있다.

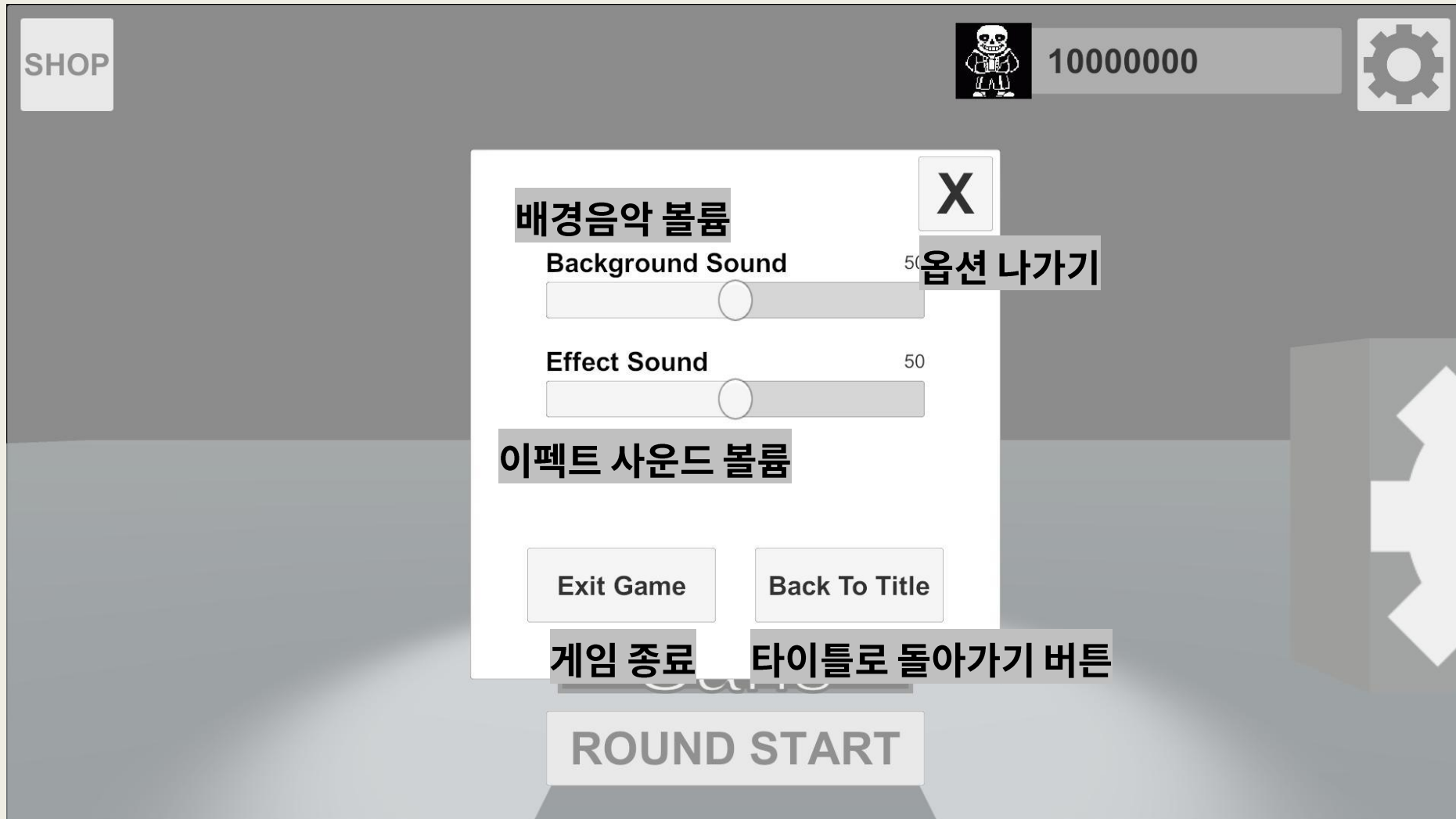
● ‘언더테일’의 샌즈전을 구현

언더테일의 샌즈전을 구현 하는 것을 목표로 제작 중이다. 이것은 언더테일을 알고 있는 사람들에게 친숙한 분위기를 줄 것이다. 하지만 실제로 출시하게 된다면 스킨과 패턴의 변형을 주어서 출시 예정이다.

게임 플레이 방식 (타이틀 UI 설명)



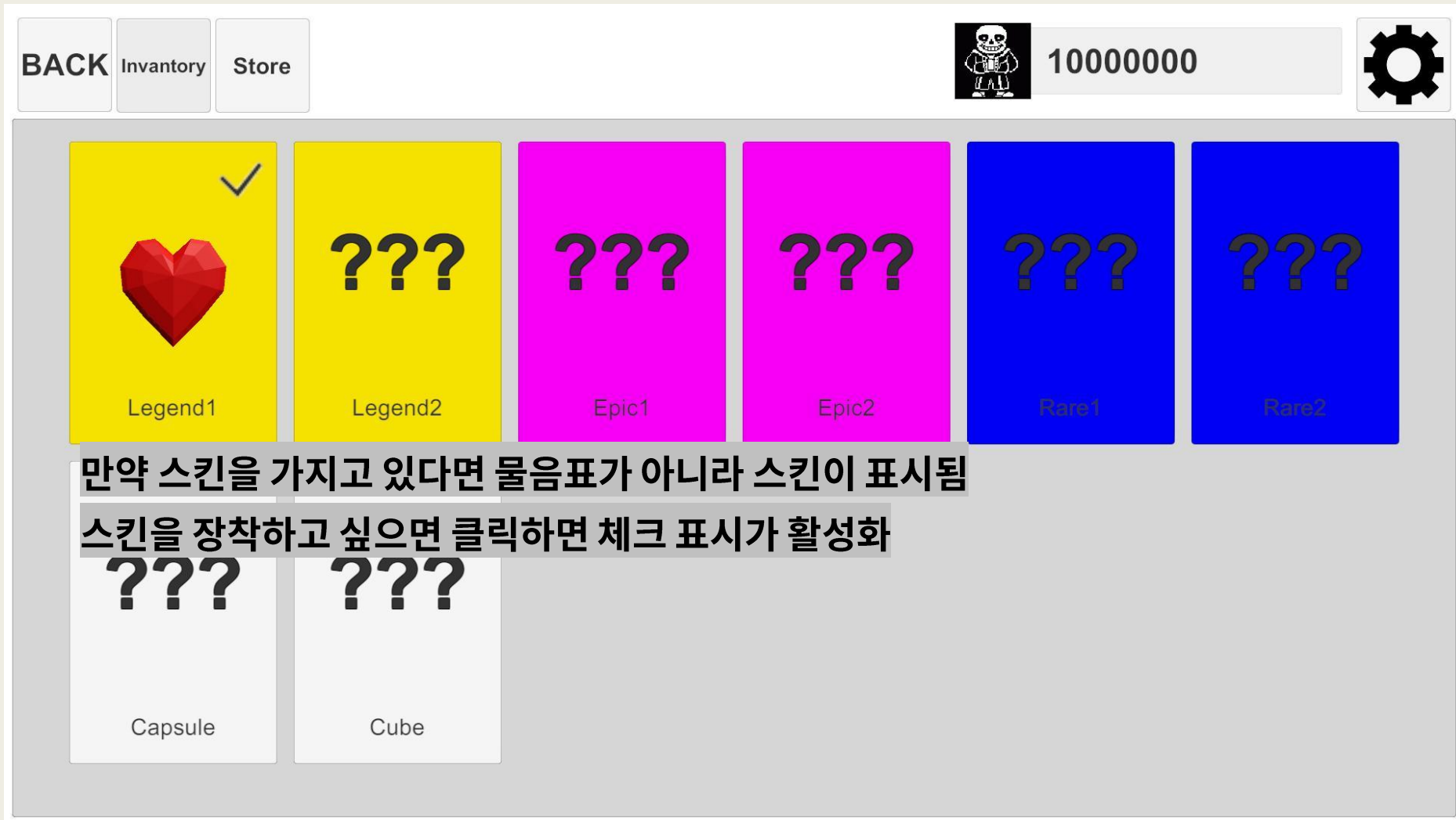
게임 플레이 방식 (옵션 UI 설명)



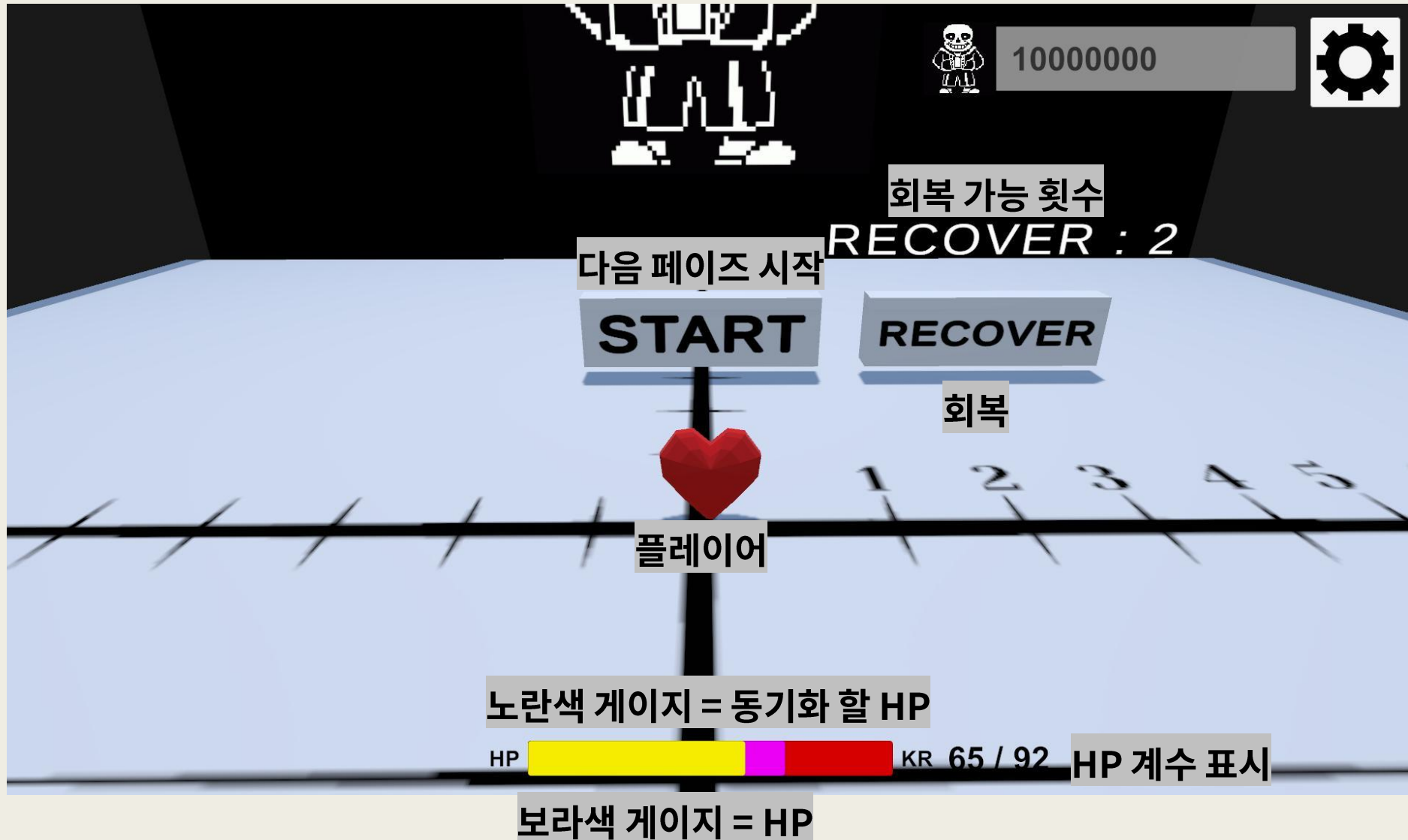
게임 플레이 방식 (상점 UI 설명)



게임 플레이 방식 (인벤토리 UI 설명)



게임 플레이 방식 (메인 게임 UI 설명)



게임 플레이 방식 (조작 방법)

- W =
앞으로 이동

- A =
왼쪽으로 이동

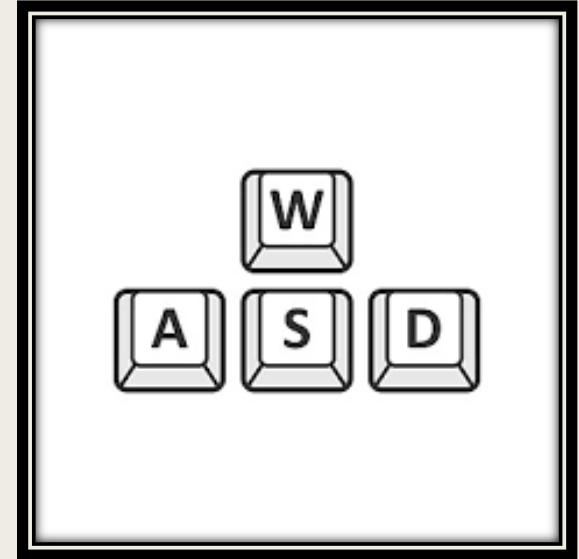
- S =
뒤로 이동

- Escape =
옵션

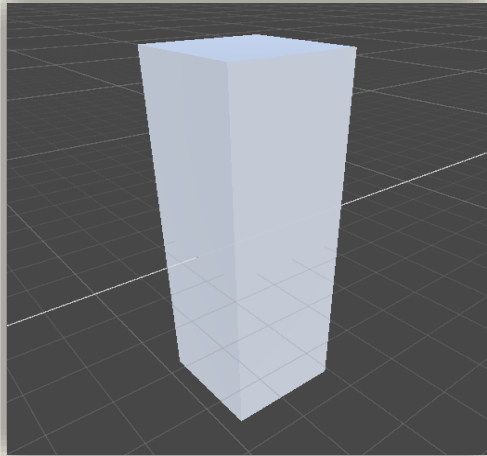
- D =
오른쪽으로 이동

- Space =
점프 유지

- 마우스 왼쪽 클릭 =
버튼 상호작용



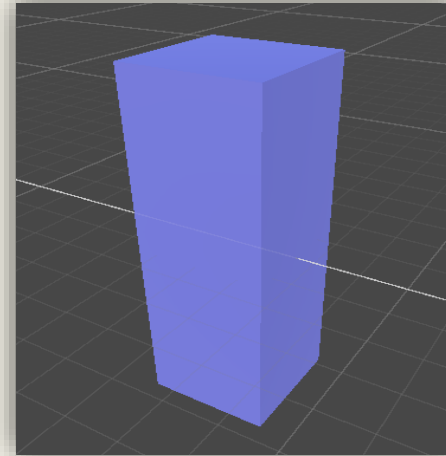
공격 소개



단순 공격

플레이어가 닿으면 체력이 줄어듦

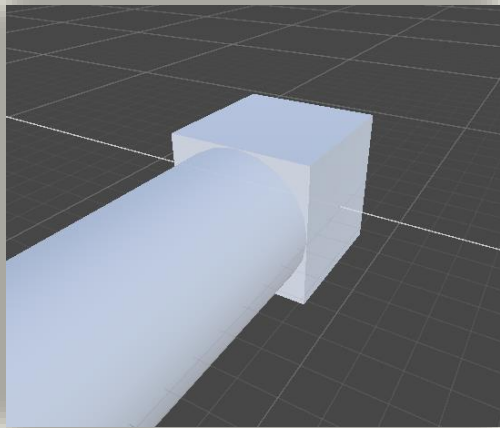
데이터에는 1로 표현



단순 움직임 공격

움직이는 플레이어가 닿으면 체력이 줄어듦

데이터에는 2로 표현

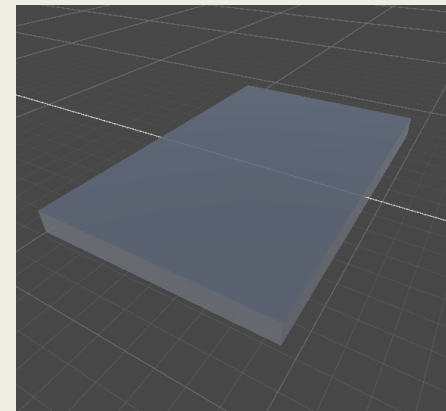


원거리 공격

원거리 공격에서 나오는 원기둥에 닿으면 체력이 줄어듦

원거리 공격 시에는 경고 후 공격

데이터에는 4로 표현



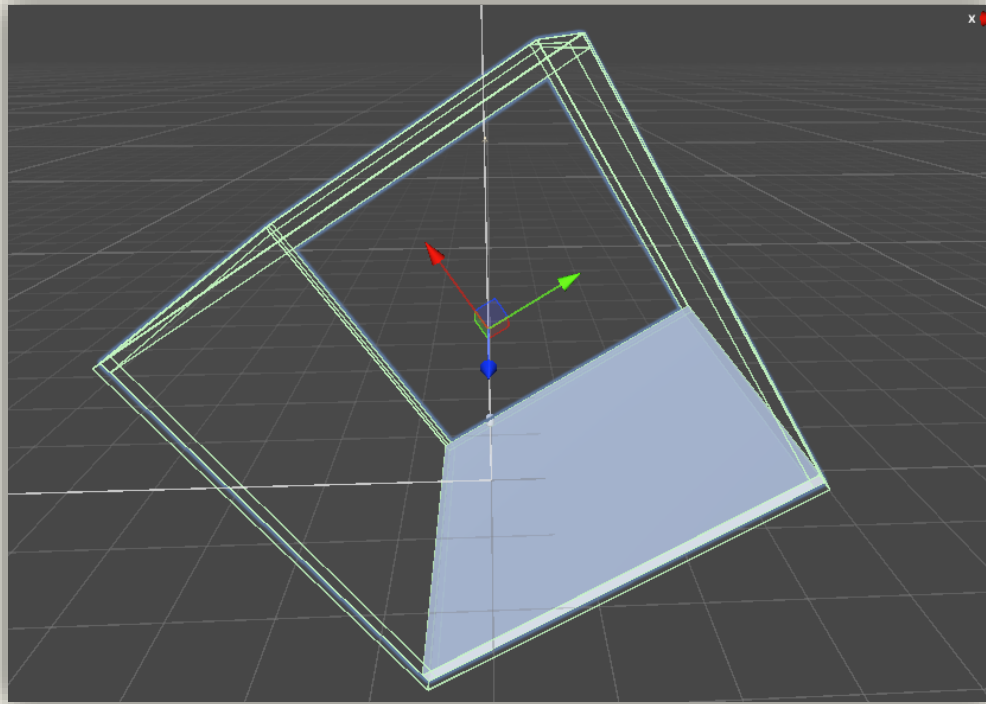
발판 공격

플레이어가 밟을 수 있음

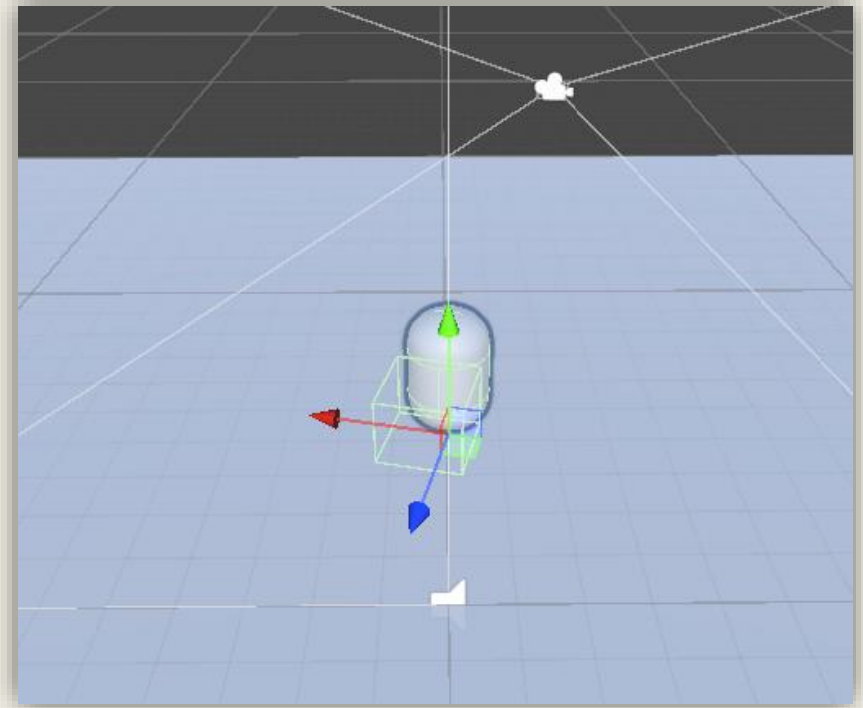
데이터에는 6로 표현

공격 소개

중력 공격



플레이어를 10의 높이까지 들어올림과 동시에 벽을 회전하여 사용할 벽을 바꾼다.

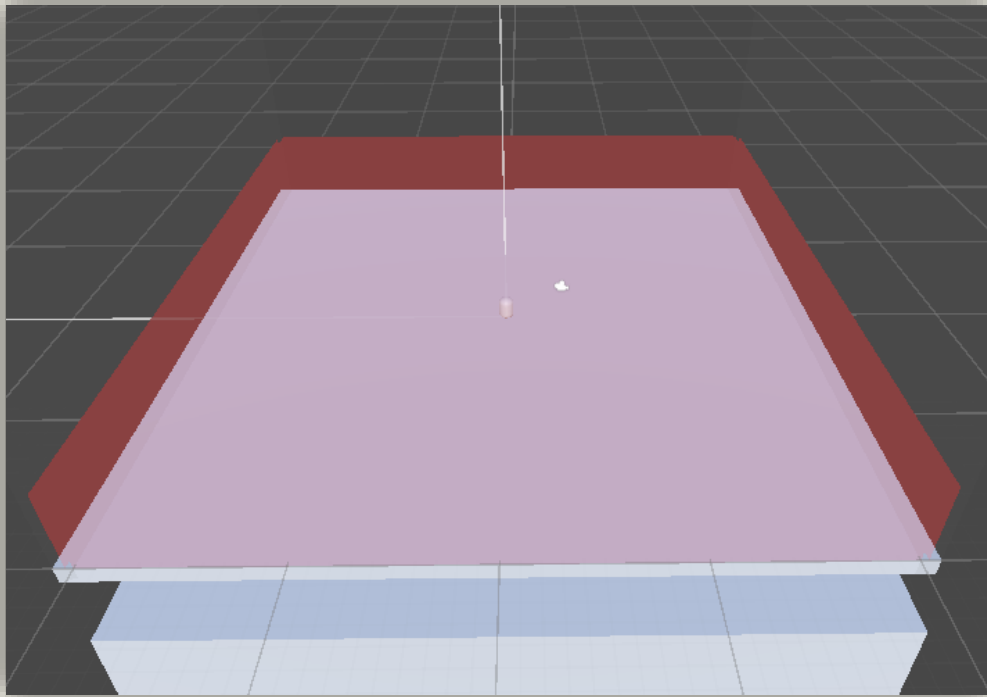


플레이어를 다시 0의 높이까지 내리꽂는다.

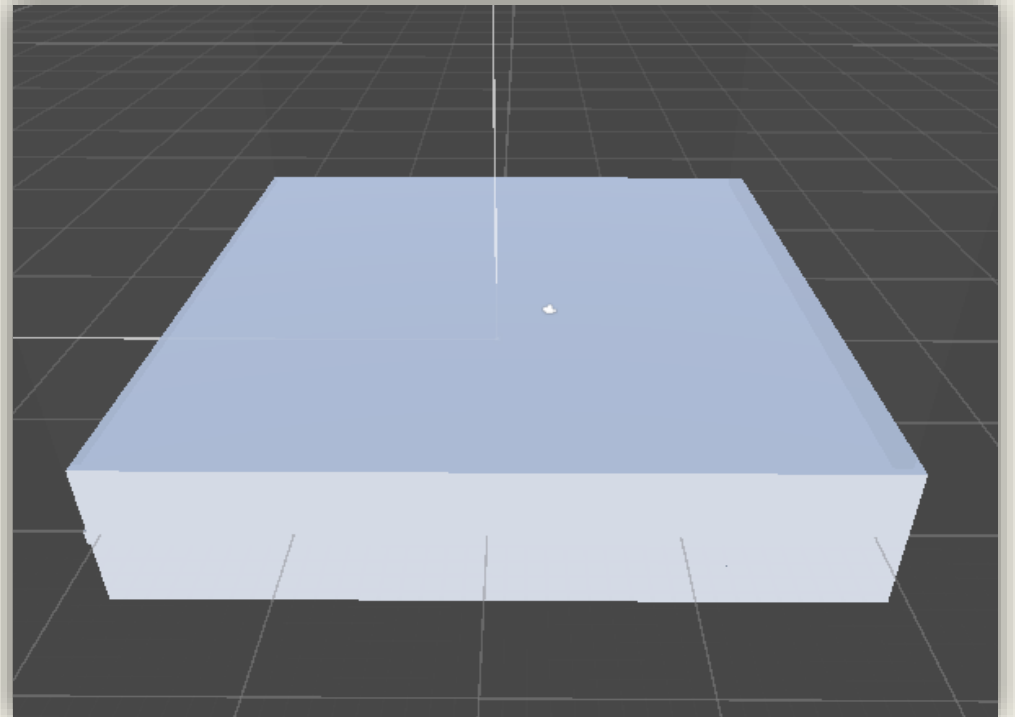
이 공격과 튀어오름 공격을 같이 사용하는 상황이 대부분이며, 해당 공격은 피할 수 없고 데미지가 없다.

공격 소개

튀어오름 공격



지면 전체에 튀어오름 공격을 사용할 것이
라는 경고를 한다.



아래에 있던 단순 공격이 일시에 튀어오르며
사용 중인 벽을 공격한다.

주요 스크립트

GameManager

- 라운드 데이터(ex 샌즈 라운드) 관리 및 플레이어가 저장해야하는 데이터(ex 돈, 클리어한 라운드) 관리

RoundManager, PlayerController

- 해당 라운드 관련 기능 담당 (오브젝트 생성, 타이밍 관리 등)
- 플레이어와 관련된 기능 담당

주요 스크립트

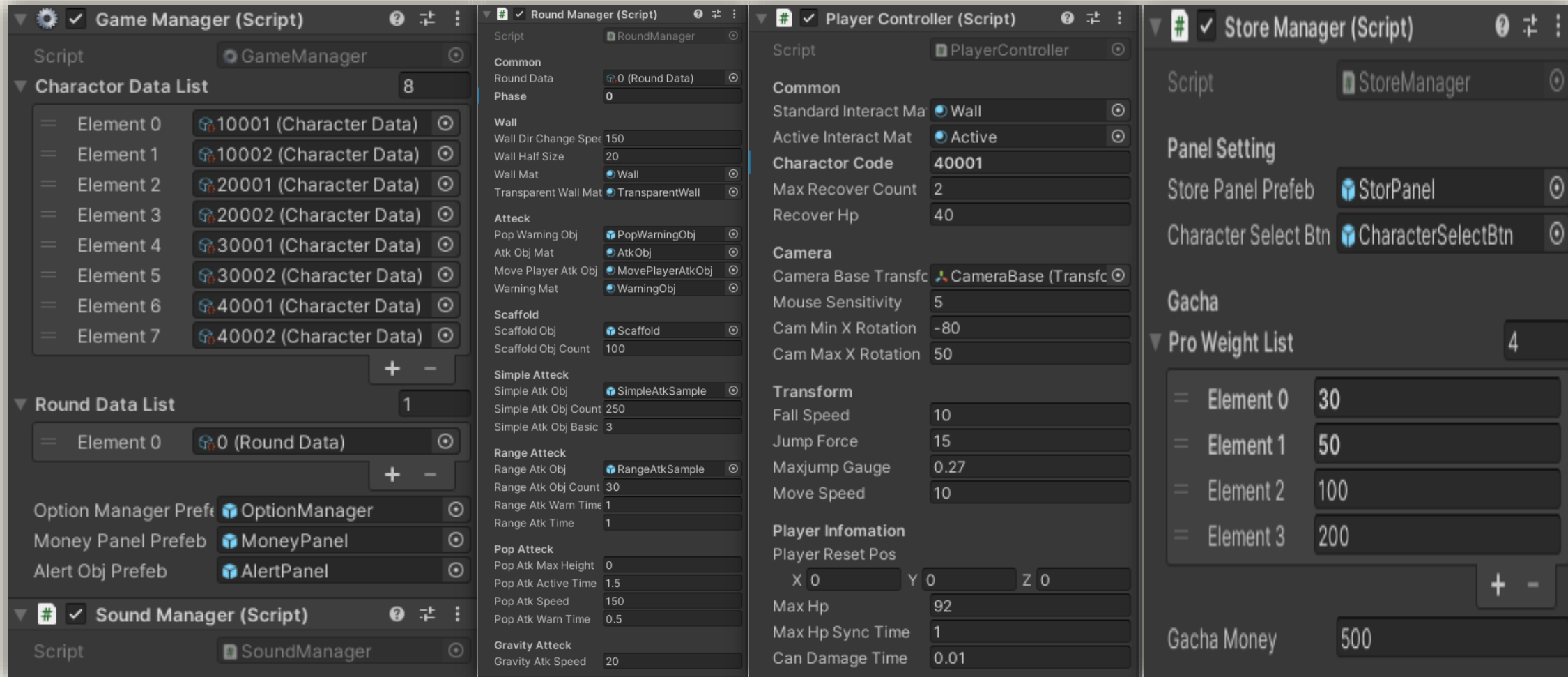
TitleManager

- 타이틀 관리 및 라운드 진입 담당

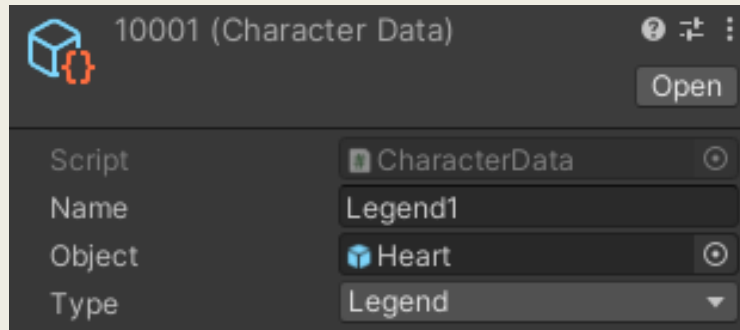
StoreManager

- 상점에서 가챠 기능을 담당

주요 스크립트 이미지



데이터 관리 (ScriptableObj)

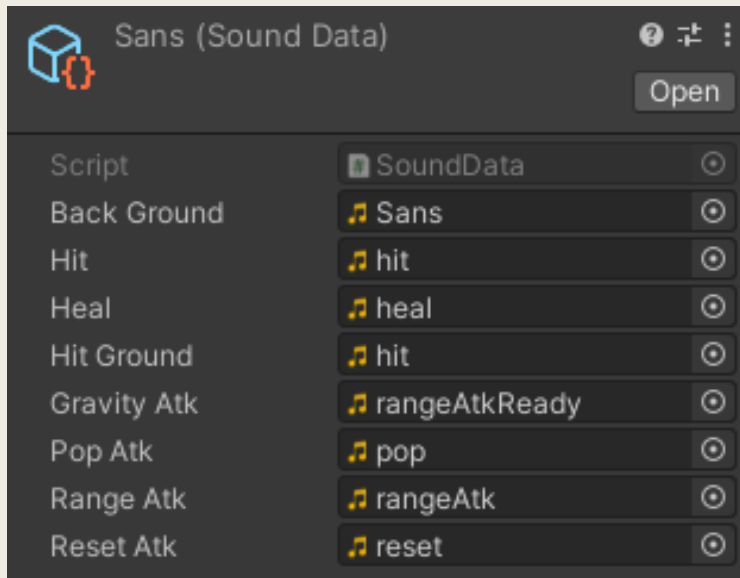


- **CharacterData**

Name = 캐릭터 이름

Object = 스킨 오브젝트

Type = 스킨의 랭크 (희귀도)

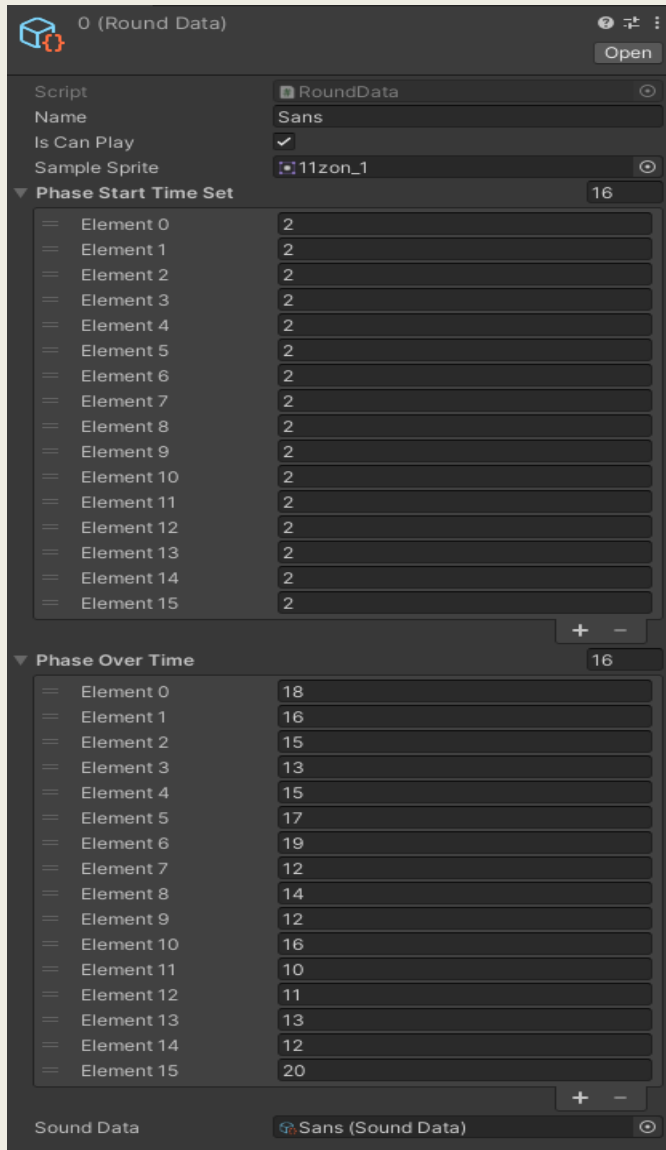


- **SoundData**

각각의 사운드를 설정하여 라운드마다 이펙트 사운드를 설정 가능

이는 게임의 확장성을 위함

데이터 관리 (ScriptableObject)



RoundData

- **Name** = 라운드 이름
- **IsCanPlay** = 현재 플레이 가능한 라운드인지
- **Spmples Sprite** = 라운드의 샘플 이미지
- **PhaseStartTime** = 얼마나 대기하고 페이즈가 시작할 것인지
- **PhaseOverTime** = 이 시간이 경과하면 페이즈를 강제종료
- **SoundData** = 해당 라운드에서 사용할 사운드 데이터

데이터 관리 (CSV 데이터 활용)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	order	type	isMove	speed	genTime	sizeX	sizeY	sizeZ	positionX	positionY	positionZ	rotationX	rotationY	rotationZ
2	-1	6	1	20	1	1	10	17	7.5	5	-30	0	0	0
3	-1	6	1	20	2.5	1	10	17	7.5	5	-30	0	0	0
4	-2	6	1	20	1	1	10	17	-7.5	5	30	0	180	0
5	-2	6	1	20	2.5	1	10	17	-7.5	5	30	0	180	0
6	0	4	0	0	2	15	15	15	-15	0	30	0	180	0
7	1	4	0	0	2	15	15	15	0	0	30	0	180	0
8	2	4	0	0	4	15	15	15	-15	0	30	0	180	0
9	3	4	0	0	4	15	15	15	15	0	30	0	180	0
10	4	4	0	0	6	15	15	15	0	0	30	0	180	0
11	5	4	0	0	6	15	15	15	-15	0	30	0	180	0
12	6	4	0	0	8	15	15	15	15	0	30	0	180	0
13	7	4	0	0	8	15	15	15	-15	0	30	0	180	0

- Order = 기본적으로는 공격 순서를 뜻하지만 음수의 경우에 genTime 제외 같은 수치가 들어간 경우 genTime 기준으로 해당 공격을 반복함
- Type = 공격 타입
- isMove = 해당 공격이 움직이는지
- speed = 움직이는 경우 속도
- genTime = 생성 시간

데이터 관리 (CSV 데이터 활용)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	order	type	isMove	speed	genTime	sizeX	sizeY	sizeZ	positionX	positionY	positionZ	rotationX	rotationY	rotationZ
2	-1	6	1	20	1	1	10	17	7.5	5	-30	0	0	0
3	-1	6	1	20	2.5	1	10	17	7.5	5	-30	0	0	0
4	-2	6	1	20	1	1	10	17	-7.5	5	30	0	180	0
5	-2	6	1	20	2.5	1	10	17	-7.5	5	30	0	180	0
6	0	4	0	0	2	15	15	15	-15	0	30	0	180	0
7	1	4	0	0	2	15	15	15	0	0	30	0	180	0
8	2	4	0	0	4	15	15	15	-15	0	30	0	180	0
9	3	4	0	0	4	15	15	15	15	0	30	0	180	0
10	4	4	0	0	6	15	15	15	0	0	30	0	180	0
11	5	4	0	0	6	15	15	15	-15	0	30	0	180	0
12	6	4	0	0	8	15	15	15	15	0	30	0	180	0
13	7	4	0	0	8	15	15	15	-15	0	30	0	180	0

- Order = 기본적으로는 공격 순서를 뜻하지만 음수의 경우에 genTime 제외 같은 수치가 들어간 경우 genTime 기준으로 해당 공격을 반복함
- Type = 공격 타입
- isMove = 해당 공격이 움직이는지
- speed = 움직이는 경우 속도
- genTime = 생성 시간

데이터 관리 (CSV 데이터 활용)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	order	type	isMove	speed	genTime	sizeX	sizeY	sizeZ	positionX	positionY	positionZ	rotationX	rotationY	rotationZ
2	-1	6	1	20	1	1	10	17	7.5	5	-30	0	0	0
3	-1	6	1	20	2.5	1	10	17	7.5	5	-30	0	0	0
4	-2	6	1	20	1	1	10	17	-7.5	5	30	0	180	0
5	-2	6	1	20	2.5	1	10	17	-7.5	5	30	0	180	0
6	0	4	0	0	2	15	15	15	-15	0	30	0	180	0
7	1	4	0	0	2	15	15	15	0	0	30	0	180	0
8	2	4	0	0	4	15	15	15	-15	0	30	0	180	0
9	3	4	0	0	4	15	15	15	15	0	30	0	180	0
10	4	4	0	0	6	15	15	15	0	0	30	0	180	0
11	5	4	0	0	6	15	15	15	-15	0	30	0	180	0
12	6	4	0	0	8	15	15	15	15	0	30	0	180	0
13	7	4	0	0	8	15	15	15	-15	0	30	0	180	0

- Size = 해당 공격의 크기
- Position = 해당 공격이 생성될 위치
- Rotation = 해당 공격의 방향. 모든 움직이는 오브젝트는 자신 기준 Z 방향으로 이동

대표 로직 설명 (큐, 링크드 리스트)

```
219  /// <summary>
220  /// 플레이어 당으면 단순 공격하는 오브젝트 대기 큐
221  /// </summary>
222  private Queue<SimpleAtk> m_simpleAtkObjWaitQueue = new Queue<SimpleAtk>();
223  /// <summary>
224  /// 원거리 공격 오브젝트 대기 큐
225  /// </summary>
226  private Queue<RangeAtk> m_rangeAtkObjWaitQueue = new Queue<RangeAtk>();
227  /// <summary>
228  /// 원거리 공격 오브젝트 대기 큐
229  /// </summary>
230  private Queue<Scaffold> m_scaffoldObjWaitQueue = new Queue<Scaffold>();
231
232  /// <summary>
233  /// 전장에서 활성화 된 단순 공격 오브젝트
234  /// 사용하고 쓸모 없어지면 대기 큐로 이동
235  /// </summary>
236  private LinkedList<SimpleAtk> m_activeSimpleAtkObjList = new LinkedList<SimpleAtk>();
237  /// <summary>
238  /// 전장에서 활성화 된 원거리 공격 오브젝트
239  /// 사용하고 쓸모 없어지면 대기 큐로 이동
240  /// </summary>
241  private LinkedList<RangeAtk> m_activeRangeAtkObjList = new LinkedList<RangeAtk>();
242  /// <summary>
243  /// 전장에서 활성화 된 발판 오브젝트
244  /// 사용하고 쓸모 없어지면 대기 큐로 이동
245  /// </summary>
246  private LinkedList<Scaffold> m_activeScaffoldObjList = new LinkedList<Scaffold>();
```

- 공격 오브젝트는 큐와 링크드리스트를 사용하여 최대한 최적화 하려고 노력했음
- 큐는 대기중인 오브젝트를 즉시 뽑아서 쓰는 것에 용이하고 링크드리스트는 중간에 있는 값을 제거하는 것에 용이함
- 사용 대기 중인 오브젝트는 큐에 분류 되어 있다가 사용할 시 링크드리스트에 삽입되어 사용됨을 알림
- 모두 사용된 오브젝트는 다시 큐에 삽입되어 사용 대기 상태로 전환됨

대표 로직 설명 (CSV 파싱)

1. 현재 페이지에 맞는 데이터 가져오기
2. 만약 데이터를 찾을 수 없다면 승리 처리
3. AtkData를 생성한 다음 데이터 파싱 후 반복 공격이 아닌 경우 m_atkDataList에 삽입
4. 만약 반복 공격인 경우 같은 order일 때 첫번째, 두번째, 세번째 까지 받아들이며 첫 번째 받아들이는 데이터는 반복 시간 설정에, 세번째로 받아들이는 데이터는 반복이 끝나는 시간 설정에 사용됨
5. 세번째로 받아들이는 데이터는 없어도 무방함

```
355 /// <summary>
356 /// CSV 데이터 가져와서 리스트에 저장
357 /// </summary>
358 void GetCSVData()
359 {
360     int _tmpRepeatAtkIndex = -1;
361     int _tmpRepeatAtkOrder = 0;
362     m_atkDataList = new List<AtkData>();
363     m_repeatAtkDataList = new List<RepeatAtkData>();
364     //Phase data's name = "Phase + N"
365     List<Dictionary<string, object>> _data = CSVReader.Read("Phase" + m_phase);
366
367     if(_data == null)
368     {
369         //데이터가 존재하지 않으면 승리
370         GameOver(true);
371         return;
372     }
373
374     for (int i = 0; i < _data.Count; i++)
375     {
376         if (_data[i]["order"].ToString() == "")
377         {
378             continue;
379         }
380
381         //데이터 생성
382         AtkData _atkData = new AtkData();
383
384         _atkData.m_order = int.Parse(_data[i]["order"].ToString());
385         _atkData.m_type = StrToAtkType(int.Parse(_data[i]["type"].ToString()));
386         _atkData.m_isMove = StrToBool(_data[i]["isMove"].ToString());
387         _atkData.m_genTime = float.Parse(_data[i]["genTime"].ToString());
388         _atkData.m_speed = float.Parse(_data[i]["speed"].ToString());
389
390         _atkData.m_size.x = float.Parse(_data[i]["sizeX"].ToString());
391         _atkData.m_size.y = float.Parse(_data[i]["sizeY"].ToString());
392         _atkData.m_size.z = float.Parse(_data[i]["sizeZ"].ToString());
393
394         _atkData.m_position.x = float.Parse(_data[i]["positionX"].ToString());
395         _atkData.m_position.y = float.Parse(_data[i]["positionY"].ToString());
396         _atkData.m_position.z = float.Parse(_data[i]["positionZ"].ToString());
397
398         _atkData.m_rotation.x = float.Parse(_data[i]["rotationX"].ToString());
399         _atkData.m_rotation.y = float.Parse(_data[i]["rotationY"].ToString());
400         _atkData.m_rotation.z = float.Parse(_data[i]["rotationZ"].ToString());
401
402         //order가 음수인 경우 RepeatAtkData 생성
403         if (_atkData.m_order < 0)
404         {
405             if(_tmpRepeatAtkOrder != _atkData.m_order)
406             {
407                 _tmpRepeatAtkOrder = _atkData.m_order;
408
409                 RepeatAtkData _repeatAtkData = new RepeatAtkData();
410                 _repeatAtkData.m_atkData = _atkData;
411                 _repeatAtkData.m_repeatStartTime = _atkData.m_genTime;
412                 _repeatAtkData.m_repeatTime = 0.0f;
413                 _repeatAtkData.m_repeatOverTime = 0.0f;
414
415                 m_repeatAtkDataList.Add(_repeatAtkData);
416                 _tmpRepeatAtkIndex++;
417             }
418             else
419             {
420                 if(m_repeatAtkDataList[_tmpRepeatAtkIndex].m_repeatTime == 0.0f)
421                 {
422                     m_repeatAtkDataList[_tmpRepeatAtkIndex].m_repeatTime = _atkData.m_genTime - m_repeatAtkDataList[_tmpRepeatAtkIndex].m_repeatOverTime;
423                 }
424                 else if(m_repeatAtkDataList[_tmpRepeatAtkIndex].m_repeatOverTime == 0.0f)
425                 {
426                     m_repeatAtkDataList[_tmpRepeatAtkIndex].m_repeatOverTime = _atkData.m_genTime;
427                 }
428             }
429         }
430         else
431         {
432             m_atkDataList.Add(_atkData);
433         }
434     }
435 }
```

대표 로직 설명 (타이밍 처리)

1. 현재 공격 데이터 리스트 공백 여부 확인 및 타이밍을 체크 중인지 확인
2. 만약 지정된 종료 시간보다 현재 시간이 많을 경우 페이즈 종료
3. 페이즈 시간 설정
4. 반복 공격 작동 체크 후 공격 데이터로 공격 생성
5. 일반 지정 공격 작동 체크 후 공격 데이터로 공격 생성

```
637 /// <summary>  
638 /// 타이밍 체크 루틴  
639 /// </summary>  
640 ///   
641 void TimingCheck()  
642 {  
643     if(m_atkDataList == null || !m_isTiming)  
644     {  
645         return;  
646     }  
647     if(m_phaseOverTime[m_phase] <= m_phaseTime)  
648     {  
649         PhaseOver();  
650     }  
651     m_phaseStartTime += Time.fixedDeltaTime;  
652     m_phaseTime = Mathf.Floor(m_phaseStartTime) + Mathf.Round((m_phaseStartTime % 1.0  
653  
654     //반복 공격 작동  
655     if (m_repeatAtkDataList.Count > 0)  
656     {  
657         for (int i = m_repeatAtkDataList.Count - 1; i >= 0; i--)  
658         {  
659             var item = m_repeatAtkDataList[i];  
660  
661             // 반복 중지 시  
662             if (item.m_repeatOverTime >= m_phaseTime && item.m_repeatOverTime > 0.0f)  
663             {  
664                 m_repeatAtkDataList.RemoveAt(i);  
665                 continue;  
666             }  
667  
668             // 처음 실행 시  
669             if (item.m_repeatStartTime + m_phaseStartTimeSet[m_phase] <= m_phaseTime  
670             {  
671                 item.m_toRepeatTime = m_phaseTime + item.m_repeatTime;  
672                 GenObjAsAtkData(item.m_atkData);  
673                 continue;  
674             }  
675  
676             // 기본 반복 생성  
677             if (item.m_toRepeatTime <= m_phaseTime && item.m_repeatStartTime <= m_pha  
678             {  
679                 item.m_toRepeatTime = m_phaseTime + item.m_repeatTime;  
680                 GenObjAsAtkData(item.m_atkData);  
681                 continue;  
682             }  
683         }  
684     }  
685  
686     //일반 지정 공격 작동  
687     if (m_atkDataList.Count > 0)  
688     {  
689         if(m_atkDataList.Count <= m_atkIndex)  
690         {  
691             return;  
692         }  
693         if (m_atkDataList[m_atkIndex].m_genTime + m_phaseStartTimeSet[m_phase] <= m_p  
694         {  
695             GenObjAsAtkData(m_atkDataList[m_atkIndex]);  
696             m_atkIndex++;  
697         }  
698     }  
699 }  
700  
701 /// </>
```



```

142  /// <summary>
143  /// 캐릭터 뽑기
144  /// </summary>
145  public void RollCharacter()
146  {
147      if(GameManager.Instance.Money < m_gachaMoney)
148      {
149          GameManager.Instance.Alert("Not Enough Money!");
150          return;
151      }
152
153      int _characterKey = 0;
154      float _rand = Random.Range(0.0f, m_allProWeightCount);
155      List<int> _charSortList = GameManager.Instance.CharacterDicSortList;
156      List<int> _selectedCharList = new List<int>();
157
158      //더하면서 가중치 계산
159      float _weight = 0.0f;
160      for(int i = 0; i < m_proWeightList.Count; i++)
161      {
162          _weight = _weight + m_proWeightList[i];
163          if(_weight >= _rand)
164          {
165              _characterKey = (i + 1) * 10000;
166
167              for(int j = 0; j < _charSortList.Count; j++)
168              {
169                  if (_charSortList[j] / _characterKey == 1)
170                  {
171                      _selectedCharList.Add(_charSortList[j]);
172                  }
173              }
174          }
175      }
176
177      //뽑을 등급 설정 후 등급 안에서 인덱스 랜덤 생성
178      int _randInt = Random.Range(0, _selectedCharList.Count);
179      CharacterInfo _characterInfo = GameManager.Instance.GetCharacterInfo(_selectedCharList[_randInt]);
180
181      //이미 가지고 있을 경우 일부 금액 반환
182      if (_characterInfo.m_isHave == true)
183      {
184          GameManager.Instance.Money += m_gachaMoney / 5;
185          m_gachaInfoText.text = "Refund " + m_gachaMoney / 5;
186      }
187      else
188      {
189          _characterInfo.m_isHave = true;
190          ResetInventoryPanel();
191          m_gachaInfoText.text = "New Character: " + _characterInfo.m_data.m_name;
192      }
193
194      //가차 금액 차감
195      GameManager.Instance.Money -= m_gachaMoney;
196
197      if(m_sampleCharacterObj != null)
198      {
199          Destroy(m_sampleCharacterObj);
200      }
201      if(_characterInfo.m_data.m_object != null)
202      {
203          m_sampleCharacterObj = Instantiate(_characterInfo.m_data.m_object, m_rollReadyImage.transform);
204          m_sampleCharacterObj.transform.localPosition = Vector3.zero;
205          m_sampleCharacterObj.transform.localScale = m_sampleCharacterObj.transform.localScale * 100;
206      }
207
208      m_rollReadyImage.enabled = false;
209  }

```

대표 로직 설명 (캐릭터 스킨 가차)

1. 돈이 만약 가차 할 돈보다 없다면 리턴
2. 모든 가중치를 더한 값을 랜덤으로 선정
3. 각 등급(레전드, 에픽, 레어, 노말)등급의 가중치를 참고하여 점점 더해가면서 더한 값의 랜덤 값보다 등급의 가중치가 더 크면 해당 등급을 선정
4. 등급을 선정 후 등급 안에서 어떤 스킨이 나올지 랜덤으로 결정
5. 만약 이미 가지고 있을 경우 가차한 돈의 5분의 1을 반환하고 없을 경우 데이터에 기록
6. 가차 금액을 차감하고 플레이어가 뽑은 스킨의 샘플을 표시

기타

추후 업데이트 내용

1. 언더테일 라운드 구현 종결
2. 공격 오브젝트들의 라운드에 맞는 스킨 추가
3. 더 다양한 효과음 추가
4. 다양한 공격 오브젝트 및 공격 루틴 추가
5. 게임 확장성 증대
6. 언더테일 이미지 탈피 후 독자적인 게임성 구축