안녕하세요

유니티 클라이언트

최시호입니다.

최시호 I CHOISIHO
tlgh10@naver.com
010-6778-4051

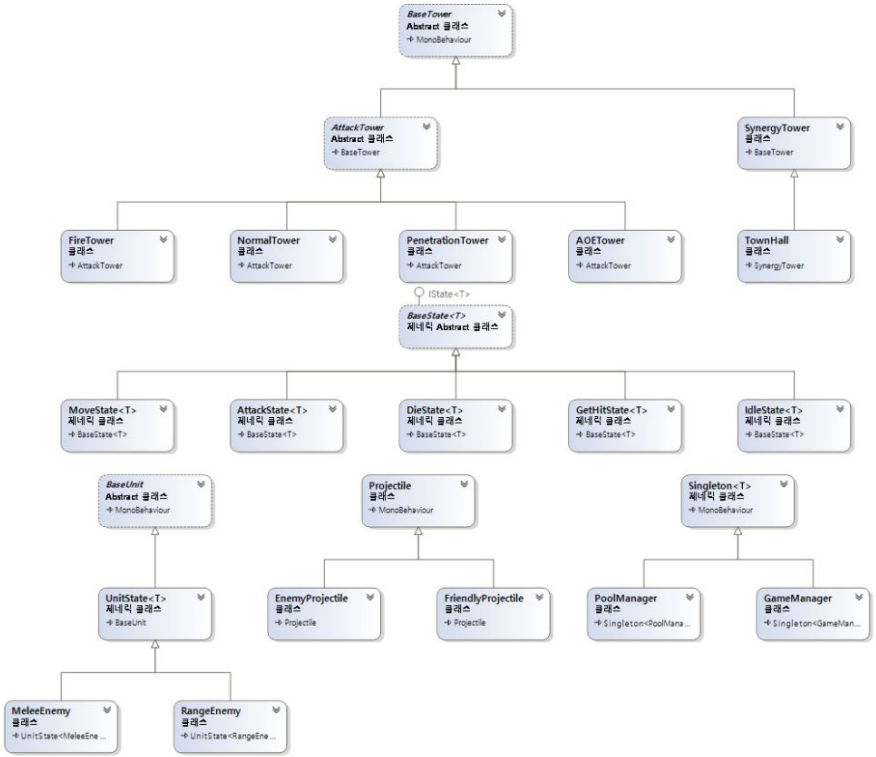
Tower Rouge-Lite

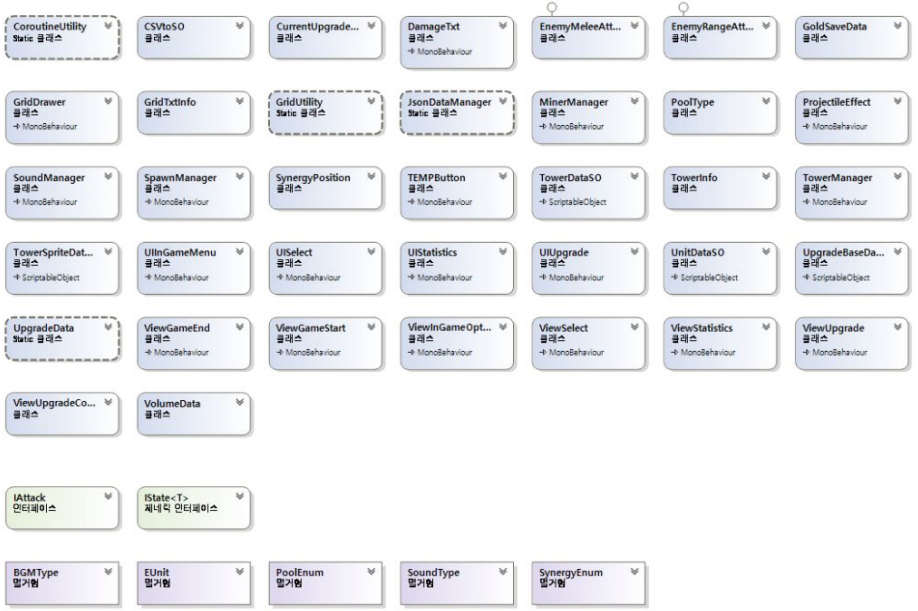
플랫폼	엔진	그래픽/시점	장르	개발 기간
PC	유니티	2D	디펜스	8주

게임 설명	- 사방에서 나오는 적을 처치하며 타워 설치 및 강화
프로그래밍	- 1인
주요 스크립트	- PoolManager => 오브젝트 풀링 - TowerManager => 타워 배치 및 관리 - UISelect => 타워 선택 로직 - GameManger => 게임종료 관리 - UnitState<T> => 적 행동 상태패턴 - SapwnManager => 적 생성 관리 - AttackTower => 타워 전투 로직 - SynergyTower => 시너지 효과 적용 - MinerManager => 골드 채굴 및 관리 - UIUpgrade => 플레이어 성장 - GridDrawer => 타일UI 시각적 표현
깃 허브 주소	https://github.com/SIHO0903/Tower_Rogue_Lite
유튜브 링크	https://youtu.be/ab3zOuBRLXs

영상

다이어그램


BaseTower
Abstract 클래스
MonoBehaviour
AttackTower
Abstract 클래스
BaseTower
SynergyTower
클래스
BaseTower
FireTower
클래스
AttackTower
NormalTower
클래스
AttackTower
PenetrationTower
클래스
AttackTower
AOETower
클래스
AttackTower
TownHall
클래스
SynergyTower
IState<T>
BaseState<T>
제네릭 Abstract 클래스
MoveState<T>
제네릭 클래스
BaseState<T>
AttackState<T>
제네릭 클래스
BaseState<T>
DieState<T>
제네릭 클래스
BaseState<T>
GetHitState<T>
제네릭 클래스
BaseState<T>
IdleState<T>
제네릭 클래스
BaseState<T>
BaseUnit
Abstract 클래스
MonoBehaviour
Projectile
클래스
MonoBehaviour
Singleton<T>
제네릭 클래스
MonoBehaviour
UnitState<T>
제네릭 클래스
BaseUnit
EnemyProjectile
클래스
Projectile
FriendlyProjectile
클래스
Projectile
PoolManager
클래스
Singleton<PoolMana...
GameManager
클래스
Singleton<GameMan...
MeleeEnemy
클래스
UnitState<MeleeEne...
RangeEnemy
클래스
UnitState<RangeEne...


CoroutineUtility
Static 클래스
CSVtoSO
클래스
CurrentUpgrade...
클래스
DamageTxt
클래스
MonoBehaviour
EnemyMeleeAtt...
클래스
EnemyRangeAtt...
클래스
GoldSaveData
클래스
GridDrawer
클래스
MonoBehaviour
GridTxtInfo
클래스
GridUtility
Static 클래스
JsonDataManager
Static 클래스
MinerManager
클래스
MonoBehaviour
PoolType
클래스
ProjectileEffect
클래스
MonoBehaviour
SoundManager
클래스
MonoBehaviour
SpawnManager
클래스
MonoBehaviour
SynergyPosition
클래스
TEMPButton
클래스
MonoBehaviour
TowerDataSO
클래스
ScriptableObject
TowerInfo
클래스
TowerManager
클래스
MonoBehaviour
TowerSpriteDat...
클래스
ScriptableObject
UIInGameMenu
클래스
MonoBehaviour
UISelect
클래스
MonoBehaviour
UIStatistics
클래스
MonoBehaviour
UIUpgrade
클래스
MonoBehaviour
UnitDataSO
클래스
ScriptableObject
UpgradeBaseDa...
클래스
ScriptableObject
UpgradeData
Static 클래스
ViewGameEnd
클래스
MonoBehaviour
ViewGameStart
클래스
MonoBehaviour
ViewInGameOpt...
클래스
MonoBehaviour
ViewSelect
클래스
MonoBehaviour
ViewStatistics
클래스
MonoBehaviour
ViewUpgrade
클래스
MonoBehaviour
ViewUpgradeCo...
클래스
VolumeData
클래스
IAttack
인터페이스
IState<T>
제네릭 인터페이스
BGMType
열거형
EUnit
열거형
PoolEnum
열거형
SoundType
열거형
SynergyEnum
열거형

01. PoolManager

```
31  public override void Awake()
32  {
33      for (int dataIdx = 0; dataIdx < objectDatas.Count; dataIdx++)
34      {
35          PoolType poolData = objectDatas[dataIdx];
36          poolData.pools = new List<GameObject>[poolData.prefabs.Length];
37          poolData.poolNames = new Dictionary<string, int>();
38
39          for (int i = 0; i < poolData.pools.Length; i++)
40          {
41              poolData.pools[i] = new List<GameObject>();
42          }
43
44          for (int i = 0; i < poolData.pools.Length; i++)
45          {
46              poolData.poolNames.Add(poolData.prefabs[i].name, i);
47          }
48      }
49  }
    참조 1개
50  public GameObject Get(PoolEnum prefabType, string name, Vector3 startPos, Quaternion rot, Transform parent = null)
51  {
52      if (parent == null) parent = this.transform;
53
54      PoolType poolData = objectDatas[(int)prefabType];
55      int index = poolData.poolNames[name];
56      List<GameObject> poolList = poolData.pools[index];
57
58      foreach (GameObject obj in poolList)
59      {
60          if (!obj.activeSelf)
61          {
62              obj.transform.SetPositionAndRotation(startPos, rot);
63              obj.SetActive(true);
64              return obj;
65          }
66      }
67
68      GameObject prefab = poolData.prefabs[index];
69      GameObject newObj = Instantiate(prefab, startPos, rot, parent);
70      poolList.Add(newObj);
71      return newObj;
72  }
```

성능 최적화를 위해 오브젝트 풀링을 적용했습니다
이를 통해 대규모 오브젝트 생성 시 발생할 수 있는 렉을 최소화하고 안정적인 프레임을 유지할 수 있습니다
싱글턴 패턴을 적용하여 게임 어디에서든 쉽게 접근하고 관리할 수 있도록 구조화했습니다

02. TowerManager

```
41  void TrySelectTower()
42  {
43      Vector2 mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
44      var hit = Physics2D.Raycast(mousePos, Vector2.zero);
45      if (hit.collider != null && hit.collider.CompareTag("Tower"))
46      {
47          if (hit.collider.TryGetComponent<BaseTower>(out var tower))
48          {
49              selectedTower = tower;
50              ghostObj = CreateGhostObj(mousePos, tower.TowerInfo());
51          }
52      }
53  }
    참조 1개
54  void MoveGhostObject()
55  {
56      if (ghostObj != null)
57      {
58          gridDrawer.gameObject.SetActive(true);
59          Vector2 mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
60          ghostObj.transform.position = mousePos;
61      }
62  }
    참조 1개
63  void PlaceSelectedTower()
64  {
65      if (ghostObj != null && selectedTower != null)
66      {
67          gridDrawer.gameObject.SetActive(false);
68
69          Vector3 targetPos = ClampTilePosition(Camera.main.ScreenToWorldPoint(Input.mousePosition));
70          Vector3 oldPos = GridUtility.SnapToGrid(selectedTower.transform.position);
71
72          if (oldPos == targetPos)
73          {
74              ClearGhost();
75              return;
76          }
77          gridDrawer.UpdateTileGridTxt(oldPos);
78          UpdateTileGrid(oldPos, targetPos, selectedTower.gameObject);
79          ClearGhost();
80      }
81  }
82
```

```
91  public void OnTowerPlaced(GameObject towerObj, Vector2 pos)
92  {
93      if (towerObj.TryGetComponent<SynergyTower>(out SynergyTower synergyTower))
94      {
95          synergyTower.ApplySynergy();
96          UpgradeAllAttackTower();
97      }
98      else if (towerObj.TryGetComponent<AttackTower>(out AttackTower attackTower))
99      {
100         attackTower.LevelChanged(gridDrawer.levelGrid[pos]);
101         gridDrawer.UpdateAttackTowerGridTxt(attackTower.gridTxtInfo);
102     }
103 }
    참조 1개
104 public void UpdateTileGrid(Vector2 oldPos, Vector2 newPos, GameObject towerObj)
105 {
106     if (!tileGrid.ContainsKey(newPos))
107     {
108         Debug.Log("새로운곳에 놓기");
109         tileGrid.Add(newPos, towerObj);
110         tileGrid.Remove(oldPos);
111         towerObj.transform.position = newPos;
112         if(towerObj.TryGetComponent<SynergyTower>(out SynergyTower synergyTower))
113         {
114             synergyTower.RemoveSynergy(oldPos);
115         }
116         OnTowerPlaced(towerObj, newPos);
117     }
118     else
119     {
120         if (newPos == TownHall.Position) return;
121         var otherTower = tileGrid[newPos];
122         otherTower.transform.position = oldPos;
123         towerObj.transform.position = newPos;
124
125         tileGrid[oldPos] = otherTower;
126         tileGrid[newPos] = towerObj;
127
128         OnTowerPlaced(otherTower, oldPos);
129         OnTowerPlaced(towerObj, newPos);
130     }
131
132 }
```

그리드 기반의 타워 디펜스 게임에서 핵심적인 타워 배치 및 이동 시스템을 구현했습니다
사용자 편의성을 위해 '고스트 오브젝트'를 활용한 시각적 피드백과 그리드 스냅 기능을 개발했습니다
딕셔너리 자료구조를 활용하여 타워 위치 정보를 효율적으로 관리했습니다

03. UISelect

```
23          for (int i = 0; i < towerNames.Length; i++)
24          {
25              viewSelect.InitializeButton(i, PoolEnum.AttackTower, towerNames[i],viewSelect.AttackTowerSelects, towerManager.transform);
26          }
27          for (int i = 0; i < viewSelect.SynergyTowerSelects.Length; i++)
28          {
29              viewSelect.InitializeButton(i, PoolEnum.SynergyTower, "Synergy Tower "+i, viewSelect.SynergyTowerSelects, towerManager.transform);
30          }
31          UIInGameMenu.ReStart += ReStartReSelect;
32
33      }
        참조 1개
34      void TowerPlacement()
35      {
36          if (Input.GetMouseButtonDown(0) && viewSelect.TowerObj != null)
37          {
38              Vector2 mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
39              RaycastHit2D hit = Physics2D.Raycast(mousePos, Vector2.zero);
40
41              if (hit.collider != null && hit.collider.CompareTag("TowerTile"))
42              {
43                  Vector2 pos = Constants.SnapToGrid(hit.collider.transform.position);
44                  viewSelect.TowerObj.transform.position = pos;
45                  towerManager.tileGrid[pos] = viewSelect.TowerObj;
46                  viewSelect.TowerObj.SetActive(true);
47
48                  towerManager.OnTowerPlaced(viewSelect.TowerObj, pos);
49                  TowerAmount?.Invoke();
50
51                  towerManager.gridDrawer.gameObject.SetActive(false);
52                  viewSelect.TowerObj = null;
53              }
54          }
55      }
        참조 2개
56      public void GetRandomSelect(int currentLevel)
57      {
58          viewSelect.SetActiveFalseAllSelects();
59          viewSelect.RandomSelect(currentLevel);
60          towerManager.gridDrawer.gameObject.SetActive(true);
61      }
```

플레이어의 마우스 입력을 감지하고, Physics2D.Raycast를 통해 클릭된 그리드 타일을 식별합니다
이 과정에서 GridUtility.SnapToGrid를 사용하여 타워를 그리드 위치에 정확히 스냅합니다
게임 진행 레벨에 따라 공격 타워 또는 시너지 타워를 선택하여 UI에 표시하도록
ViewSelect에 지시하며 그리드UI의 가시성을 제어합니다

04. ViewSelect

```
21  public void InitializeButton(int index, PoolEnum poolEnum, string towerName, Button[] buttons,Transform transform)
22  {
23      GameObject towerPrefab = PoolManager.Instance.Get(poolEnum, towerName, Vector3.zero, Quaternion.identity, transform);
24      towerPrefab.SetActive(false);
25
26      SetButtonDescription(index, towerPrefab, buttons);
27
28      buttons[index].onClick.AddListener(() =>
29      {
30          SoundManager.instance.PlaySound(SoundType.Button);
31          TowerObj = PoolManager.Instance.Get(poolEnum, towerName, Vector3.zero, Quaternion.identity, transform);
32          TowerObj.SetActive(false);
33          SetActiveFalseAllSelects();
34      });
35  }
    참조 1개
36  void SetButtonDescription(int index, GameObject towerPrefab, Button[] buttons)
37  {
38      var texts = buttons[index].GetComponentsInChildren<TMP_Text>();
39      if (towerPrefab.TryGetComponent<BaseTower>(out var baseTower))
40      {
41          texts[0].text = baseTower.GetName();
42          texts[1].text = baseTower.GetDescription();
43      }
44  }
    참조 3개
45  public void SetActiveFalseAllSelects()
46  {
47      foreach (var item in AttackTowerSelects)
48      {
49          if (!item.gameObject.activeSelf)
50              continue;
51          else
52              item.gameObject.SetActive(false);
53      }
54      foreach (var item in SynergyTowerSelects)
55      {
56          if (!item.gameObject.activeSelf)
57              continue;
58          else
59              item.gameObject.SetActive(false);
60      }
61      reRollBtn.gameObject.SetActive(false);
62  }
```

```
63  public void RandomSelect(int currentLevel)
64  {
65      this.currentLevel = currentLevel;
66      ShowReRollBtn();
67      //3의 배수의 레벨마다 시너지 타워선택UI 활성화
68      bool isSynergyLevel = currentLevel % 3 == 0;
69
70      Button[] targetButtons = isSynergyLevel ? SynergyTowerSelects : AttackTowerSelects;
71      int max = targetButtons.Length;
72
73
74      var indices = GetUniqueRandomNumbers(UpgradeData.LevelChoice, 0, max);
75      foreach (var idx in indices)
76      {
77          targetButtons[idx].gameObject.SetActive(true);
78      }
79  }
80
    참조 1개
81  HashSet<int> GetUniqueRandomNumbers(int count, int min, int max)
82  {
83      HashSet<int> unique = new HashSet<int>();
84      while (unique.Count < count)
85          unique.Add(Random.Range(min, max));
86      return unique;
87  }
88
    참조 1개
89  void ShowReRollBtn()
90  {
91      currentReRollCount = UpgradeData.RerollCount;
92      reRollTxt.text = $"리롤 ({currentReRollCount}회 가능)";
93      reRollBtn.gameObject.SetActive(true);
94      reRollBtn.interactable = currentReRollCount == 0 ? false : true;
95  }
```

PoolManager에서 타워 프리팹 인스턴스를 가져와 버튼의 이름과 설명을 동적으로 설정(SetButtonDescription)하고
각 버튼의 onClick 이벤트에 람다(Lambda) 함수를 사용하여 해당 타워 선택 로직을 연결합니다
HashSet<int>를 활용한 중복 없는 난수 생성 로직(GetUniqueRandomNumbers)을 구현하여
타워 선택의 공정성과 게임 플레이의 다양성을 확보했습니다

05. AttackTower

```
104     public void Attack(GameObject target)
105     {
106         animator.SetTrigger("Attack");
107         SoundManager.instance.PlaySound(SoundType.Attack);
108         attackTimer = 0;
109         HeadLookAt(target);
110         GameObject projectile = PoolManager.Instance.Get(PoolEnum.Projectile, projectileName, headSprite.transform.position, Quaternion.identity);
111         if (projectile.TryGetComponent<FriendlyProjectile>(out FriendlyProjectile component))
112         {
113             component.Init(current_Damage, projectileEffectName, current_tier);
114             AttackMethod(component, target);
115         }
116     }
        참조 1개
117     void SearchNearestTarget()
118     {
119         if (currentTarget != null && currentTarget.activeSelf)
120         {
121             float distance = Vector2.Distance(transform.position, currentTarget.transform.position);
122             if (distance <= current_Range)
123             {
124                 Attack(currentTarget);
125                 return;
126             }
127         }
128         Collider2D[] targets = Physics2D.OverlapCircleAll(transform.position, current_Range, LayerMask.GetMask("Enemy"));
129         if (targets.Length > 0)
130         {
131             float minDistance = float.MaxValue;
132             GameObject nearest = null;
133 
134             foreach (var target in targets)
135             {
136                 float dist = Vector2.Distance(transform.position, target.transform.position);
137                 if (dist < minDistance)
138                 {
139                     minDistance = dist;
140                     nearest = target.gameObject;
141                 }
142             }
143             if (nearest != null)
144             {
145                 currentTarget = nearest;
146                 Attack(currentTarget);
```

객체지향 프로그래밍의 핵심 원칙인 상속과 다형성을 적용하여 추상 클래스로 설계했습니다
이를 통해 타워 타입 추가 시 높은 확장성과 코드 재사용성을 확보했습니다
타워 스탯의 레벨 스케일링 로직을 중앙화하여 밸런스 조정 및 데이터 관리가 용이하도록 구현했습니다

06. SynergyTower

```
14      void OnEnable() => towerManager = GetComponentInParent<TowerManager>();
        참조 1개
15      public void RemoveSynergy(Vector2 oldPos)
16      {
17          foreach (var item in SynergyPositions)
18          {
19              Vector2 pos = CalculateSynergyPosition(item, oldPos);
20
21              towerManager.gridDrawer.UpdateSynergyGridTxt(pos, -item.Amount);
22
23          }
24      }
25
        참조 2개
26      Vector2 CalculateSynergyPosition(SynergyPosition item, Vector3? customPos = null)
27      {
28          Vector2 basePos = customPos ?? transform.position;
29          Vector2 offset = GridUtility.EnumToVector3(item.dir) * GridUtility.TileGap * item.Distance;
30          return basePos + offset;
31      }
32
        참조 2개
33      public void ApplySynergy()
34      {
35          foreach (var item in SynergyPositions)
36          {
37              var pos = CalculateSynergyPosition(item);
38              towerManager.gridDrawer.UpdateSynergyGridTxt(pos, item.Amount);
39              towerManager.UpdateTowerGridTxt();
40          }
41      }
```

```
43      public override string GetDescription()
44      {
45          int gridSize = 5;
46          int center = gridSize / 2;
47          string[,] grid = new string[gridSize, gridSize];
48
49          for (int y = 0; y < gridSize; y++)
50              for (int x = 0; x < gridSize; x++)
51                  grid[y, x] = "";
52
53          grid[2, 2] = " T ";
54
55          foreach (var synergy in SynergyPositions)
56          {
57              Vector2 dir = GridUtility.EnumToVector3(synergy.dir);
58
59              int dx = Mathf.RoundToInt(dir.x * synergy.Distance);
60              int dy = -Mathf.RoundToInt(dir.y * synergy.Distance);
61
62              int targetX = center + dx;
63              int targetY = center + dy;
64
65              if (targetX >= 0 && targetX < gridSize && targetY >= 0 && targetY < gridSize)
66              {
67                  string sign = synergy.Amount > 0 ? "+" : "-";
68                  string color = synergy.Amount > 0 ? "#5CC054" : "red";
69                  int absAmount = Mathf.Abs(synergy.Amount);
70
71                  grid[targetY, targetX] = $"<color={color}>{sign}{absAmount} </color>";
72              }
73          }
74
75          StringBuilder sb = new StringBuilder();
76          for (int y = 0; y < gridSize; y++)
77          {
78              for (int x = 0; x < gridSize; x++)
79              {
80                  sb.Append(grid[y, x]);
81              }
82              sb.AppendLine();
83          }
84
85          return sb.ToString();
86      }
```

핵심 메소드는 ApplySynergy()와 RemoveSynergy()로, 타워의 배치/이동/제거 시
CalculateSynergyPosition()을 통해 동적으로 시너지 적용 대상 그리드 좌표를 계산합니다
특히 GetDescription() 메소드는 타워 선택 UI 등에 표시될 시너지 효과의 배치 모양과 수치를 5x5 그리드 형태의 문자열로 동적 생성합니다.
이는 StringBuilder 기능을 활용하여 직관적인 정보 전달을 목표로 합니다

07. UnitState

```
49      public override void Init(int currentLevel)
50      {
51          LevelChange(currentLevel);
52          CircleCollider.enabled = true;
53          gameObject.layer = LayerMask.NameToLayer("Enemy");
54
55      }
        참조 1개
56      void LevelChange(int currentLevel)
57      {
58          Health = UnitData.Health + currentLevel * UnitData.HealthGrowth;
59          Damage = UnitData.Damage + currentLevel * UnitData.AttackGrowth;
60      }
        참조 9개
61      public override void TransitionToState(EUnit estate)
62      {
63          currentState = states[estate];
64          currentState.EnterState();
65      }
66
        참조 2개
67      public Vector2 Dir()
68      {
69          Vector2 dir = Vector2.zero;
70          dir = townHallPos - transform.position;
71          dir.Normalize();
72          spriteRenderer.flipX = dir.x > 0 ? false : true;
73          return dir;
74      }
```

```
75      public float Distance()
76      {
77          return Vector3.Distance(transform.position, townHallPos);
78      }
79
        Unity 메시지 | 참조 0개
80      private void OnTriggerEnter2D(Collider2D collision)
81      {
82          if (collision.CompareTag("TowerProjectile"))
83          {
84              GetHit(GetProjectileDamage(collision), Color.white);
85              TransitionToState(EUnit.GetHit);
86          }
87      }
        참조 3개
88      public override void GetHit(float damage,Color damageFontColor)
89      {
90          if (damage == 0) return;
91          Health -= damage;
92          DamageTxt.Create(transform.position, damage, damageFontColor);
93      }
        참조 1개
94      float GetProjectileDamage(Collider2D projectile)
95      {
96          float damage = 0;
97          if(projectile.TryGetComponent<FriendlyProjectile>(out FriendlyProjectile component))
98          {
99              damage = component.GetDamage();
100         }
101
102         return damage;
103     }
```

IState<T> 인터페이스와 함께 유닛의 각 행동(Idle, Move, Attack, GetHit, Die)을 독립적인 클래스로 분리하여 관리함으로써,
복잡한 행동 흐름을 모듈화하고 코드의 응집도를 높였습니다
유닛의 기본 스탯(체력, 데미지), 물리 컴포넌트(Rigidbody2D), 애니메이터(Animator) 등
모든 유닛이 공유하는 공통적인 속성과 메소드(Dir(), Distance(), GetHit())를 캡슐화하여 코드 재사용성을 극대화했습니다

08. GameManager

```
6   public class GameManager : Singleton<GameManager>
7   {
8       public Action PopUpEnd;
        참조 2개
9       public void DescreaseHealth(float damage)
10      {
11          UpgradeData.TownHallHealth -= damage;
12          SoundManager.instance.PlaySound(SoundType.GetHit);
13          if (UpgradeData.TownHallHealth <= 0)
14          {
15              GameEnd();
16          }
17      }
        참조 1개
18      void GameEnd()
19      {
20          PopUpEnd?.Invoke();
21          GoldSaveData goldSaveData = new GoldSaveData();
22          goldSaveData.gold = MinerManager.Gold;
23          JsonDataManager.JsonSave<GoldSaveData>(goldSaveData, JsonDataManager.JsonFileName.Gold);
24          Debug.Log("게임종료");
25
26      }
27  }
28
```

이벤트 기반(C# Action) 설계를 통해 게임 종료와 같은 중요한 시스템 이벤트 발생 시,
다른 모듈들이 느슨하게 결합되어 반응하도록 구현함으로써 확장성을 높였습니다

09. SpawnManager

```
42  void SpawnTimer()
43  {
44      timer += Time.deltaTime;
45      if (timer > spawnTimer)
46      {
47          SpawnEnemy();
48          timer = 0;
49          DecreaseSpawnTimer();
50      }
51  }
52
    참조 1개
53  void SpawnEnemy()
54  {
55      int tier = GetUnlockedTier(); // 시간 기반 티어 계산
56      int index = UnityEngine.Random.Range(0, tier + 1); // 0 ~ tier 범위 내 랜덤
57
58      Vector3 spawnPos = GetSpawnPosition();
59
60      GameObject enemy = PoolManager.Instance.Get(PoolEnum.Enemy, index, spawnPos, Quaternion.identity, transform);
61      BaseUnit unit = enemy.GetComponent<BaseUnit>();
62      unit.Die = Die;
63      unit.Init(LevelChange(levelTimer));
64      EnemyAmount += 1;
65  }
    참조 1개
66  Vector2 GetSpawnPosition()
67  {
68      Vector2 SpawnPoint = Vector2.zero;
69
70      int rand = spawnPoints[Random.Range(0, 4)];
71      if (rand == -10 || rand == 5)
72      {
73          SpawnPoint.x = rand;
74          int YAxis = Random.Range(-6, 7);
75          SpawnPoint.y = YAxis;
76      }
77      else if (Mathf.Abs(rand) == 6)
78      {
79          SpawnPoint.y = rand;
80          int XAxis = Random.Range(-10, 6);
81          SpawnPoint.x = XAxis;
82      }
83
84      return SpawnPoint;
```

미리 정의된 spawnPoints 배열 중 랜덤하게 위치를 선택하여 적을 스폰하며,
PoolManager를 활용하여 적 오브젝트를 재활용함으로써 스폰 시 발생하는 성능 부하를 최소화했습니다
시간 기반의 동적 난이도 스케일링 시스템을 구현하여 게임의 재플레이 가치와 몰입도를 높였습니다

10. MinerManager

```
40      void Update()
41      {
42          miningTimer += Time.deltaTime;
43          if (miningTimer >= 1f)
44          {
45              MiningPerSecond();
46              miningTimer = 0f;
47              OnGoldUpdate?.Invoke();
48          }
49      }
        참조 1개
50      void MiningPerSecond()
51      {
52          Gold += MinerAmount * miningSpeed;
53          OnGoldUpdate?.Invoke();
54      }
55
        참조 1개
56      public float MiningPerMinute()
57      {
58          Debug.Log("MiningPerMinute : " + Mathf.Round(MinerAmount * miningSpeed * 60 * 10) / 10);
59          return Mathf.Round(MinerAmount * miningSpeed * 60 * 10) / 10;
60      }
        참조 1개
61      public static bool CanPayGold(float amount)
62      {
63          if (Gold < amount)
64              return false;
65          else
66          {
67              Gold -= amount;
68
69              return true;
70          }
71      }
```

```
72      public void AddMiner()
73      {
74          MinerAmount += 1;
75          OnGoldUpdate?.Invoke();
76      }
        참조 1개
77      public void IncreaseMinerSpeed()
78      {
79          miningSpeed += 0.016f;
80          OnGoldUpdate?.Invoke();
81      }
```

채굴꾼 수(MinerAmount)'와 채굴 속도(miningSpeed)에 기반하여 초당 골드 획득량을 계산(MiningPerSecond())하고
이를 Update() 루틴 내에서 주기적으로 적용합니다
유니티의 Time.deltaTime을 활용하여 프레임 독립적인 시간 계산을 통해 이루어집니다

11. ViewUpgradeContext

```
참조 8개
public void Init(UpgradeBaseDataSO upgradeBaseData,int currentLevel)
{
    key = upgradeBaseData.key;
    name = upgradeBaseData.UPName;
    this.currentLevel = currentLevel;
    cost = upgradeBaseData.Cost;
    maxLevel = upgradeBaseData.MaxLvl;
    costMultiplier = upgradeBaseData.CostMultiplier;

    Txt_name.text = name;
    Txt_Progress(currentLevel, maxLevel);
    Txt_cost.text = (cost * (currentLevel+1) * costMultiplier).ToString("N0");
    if (IsMaxLevel()) return;
}

참조 2개
void Txt_Progress(int currentLevel, int maxLevel)
{
    if (maxLevel == -1)
        this.Txt_progress.text = currentLevel.ToString();
    else
        this.Txt_progress.text = currentLevel + " / " + maxLevel;
}

참조 1개
void Update()
{
    if (!MinerManager.CanPayGold(cost))
    {
        Debug.Log("돈이부족합니다.");
        return;
    }
    SoundManager.instance.PlaySound(SoundType.UpgradeButton);
    currentLevel += 1;
    Txt_Progress(currentLevel, maxLevel);
    this.Txt_cost.text = ((currentLevel+1) * cost * costMultiplier).ToString("N0");
    SaveCurrentLevelToJson();
    if (IsMaxLevel()) return;
}
```

```
private void SaveCurrentLevelToJson()
{
    CurrentUpgradeLvl upgradeData = JsonDataManager.JsonLoad<CurrentUpgradeLvl>(JsonDataManager.JsonFileName.Upgrade);

    upgradeData.level[key] = currentLevel;

    JsonDataManager.JsonSave(upgradeData, JsonDataManager.JsonFileName.Upgrade);
}

참조 8개
public void AddBtn(Action action)
{
    button.onClick.AddListener(() =>
    {
        action?.Invoke();
        Update();

    });
}
참조 2개
bool IsMaxLevel()
{
    if (maxLevel == -1)
        return false;

    if (currentLevel >= maxLevel)
    {
        button.interactable = false;
        return true;
    }
    else
        return false;
}
```

플레이어의 업그레이드 시스템과 관련된 UI 요소(업그레이드 이름, 현재 레벨/진행도, 비용 텍스트, 업그레이드 버튼 등)를 제어하고
사용자의 입력을 받아 실제 업그레이드 로직을 실행시키는 UI 컨트롤러(또는 View) 역할을 수행하는 스크립트입니다
View 역할을 수행하며, UpgradeBaseDataSO(데이터), UpgradeData(현재 상태), MinerManager(경제 시스템 등
다양한 데이터 및 로직 모듈들과 유기적으로 상호작용하도록 설계하여 시스템 간의 결합도를 낮추고 유지보수성을 높였습니다

12. GridDrawer

```
41  public void UpdateSynergyGridTxt(Vector2 pos, int lvl)
42  {
43      Vector2 snapPos = GridUtility.SnapToGrid(pos);
44
45      if (levelGrid.TryGetValue(snapPos, out int value))
46      {
47          levelGrid[snapPos] += lvl;
48          UpdateTileGridTxt(snapPos);
49      }
50  }
51
    참조 1개
52  string Sign(int lvl)
53  {
54      string sign = string.Empty;
55      if (lvl > 0) sign = "+";
56      return sign + lvl;
57  }
58
    참조 2개
59  public void UpdateAttackTowerGridTxt(GridTxtInfo gridTxtInfo)
60  {
61      int currentLvl = gridTxtInfo.lvl;
62      int maxLvl = gridTxtInfo.maxLvl;
63      string txt = string.Empty;
64      if (currentLvl > 0)
65          txt = currentLvl + " / " + maxLvl;
66      else
67          txt = "0";
68
69      tiles[gridTxtInfo.pos].text = txt;
70  }
```

```
71  public void WriteGridTxt()
72  {
73      foreach (var item in tiles)
74      {
75          if (levelGrid[item.Key] == 0)
76              tiles[item.Key].text = string.Empty;
77          else
78          {
79              int value = levelGrid[item.Key];
80              tiles[item.Key].text = value > 0 ? "+" + value : value.ToString();
81          }
82      }
83      tiles[townhallPos].text = string.Empty;
84
85  }
    참조 2개
86  public void UpdateTileGridTxt(Vector2 pos)
87  {
88      if (levelGrid[pos] == 0)
89          tiles[pos].text = string.Empty;
90      else
91          tiles[pos].text = Sign(levelGrid[pos]);
92  }
    참조 1개
93  public void ResetLevelGrid()
94  {
95      var keys = new List<Vector2>(levelGrid.Keys);
96      foreach (var key in keys)
97      {
98          levelGrid[key] = 0;
99      }
100
101     WriteGridTxt();
102
103 }
```

levelGrid (Dictionary<Vector2, int>)를 사용하여 각 그리드 좌표에 해당하는 타일의 내부적인 레벨 정보를 저장하며,
이 데이터가 변경될 때마다 tiles 배열에 연결된 UI 텍스트 컴포넌트(tiles[pos].text = Sign(levelGrid[pos]))를 업데이트하여
플레이어에게 실시간으로 시각적인 피드백을 제공합니다

13. MeleeEnemy

```
5   public class MeleeEnemy : UnitState<MeleeEnemy>
6   {
        Unity 메시지 | 참조 3개
7       public override void Awake()
8       {
9           base.Awake();
10          states.Add(EUnit.Idle, new IdleState<MeleeEnemy>(UnitData));
11          states.Add(EUnit.Move, new MoveState<MeleeEnemy>(UnitData));
12          states.Add(EUnit.GetHit, new GetHitState<MeleeEnemy>(UnitData));
13          states.Add(EUnit.Attack, new AttackState<MeleeEnemy>(UnitData,new EnemyMeleeAttack()));
14          states.Add(EUnit.Die, new DieState<MeleeEnemy>(UnitData));
15      }
        Unity 메시지 | 참조 3개
16      public override void OnEnable()
17      {
18          base.OnEnable();
19          TransitionToState(EUnit.Idle);
20      }
        Unity 메시지 | 참조 0개
21      private void Update()
22      {
23          currentState.UpdateState(this);
24      }
25  }
```

불필요한 조건 검사 및 연산을 줄여 Update() 루프의 효율성을 높입니다
이는 CPU 사용량을 최적화하여 게임의 전반적인 성능 향상에 기여합니다
AttackState에 EnemyMeleeAttack과 같은 특정 공격 방식을 설계했습니다
이를 통해 유닛의 핵심 AI 로직을 유지하면서도 다양한 공격 방식을 유연하게 변경하고 확장할 수 있습니다

14. Projectile

```
3   public class Projectile : MonoBehaviour
4   {
5       protected float damage;
        참조 4개
6       public bool HasEffectDamage { get; set; }
7       protected string effectName;
8       protected int current_tier;
9       Rigidbody2D rigid;
10      protected int penetrationCount = 0;
        Unity 메시지 | 참조 0개
11      private void Awake()
12      {
13          rigid = GetComponent<Rigidbody2D>();
14          HasEffectDamage = false;
15      }
        Unity 메시지 | 참조 0개
16      private void OnEnable()
17      {
18          StartCoroutine(CoroutineUtility.SetActiveFalse(gameObject, 3f));
19      }
        참조 5개
20      public void Init(float damage, string effectName, int current_tier)
21      {
22          this.damage = damage;
23          this.effectName = effectName;
24          this.current_tier = current_tier;
25      }
26
        참조 7개
27      public void Shoot(Vector2 direction, float projectileSpeed = 5, int penetrationCount = 0)
28      {
29          rigid.velocity = direction.normalized * projectileSpeed;
30          rigid.transform.up = direction;
31          this.penetrationCount = penetrationCount;
32      }
33
```

모든 투사체의 공통적인 동작을 추상화한 베이스 클래스 역할을 수행합니다
이를 통해 다양한 종류의 투사체(화살, 포탄, 전기 구체 등)를 구현할 때 코드를 재사용하고 유지보수를 용이하게 했습니다

깃 허브

https://github.com/SIHO0903/Tower_Rogue_Lite

유튜브 링크

https://youtu.be/ab3zOuBRLXs

감사합니다