Implementation of a Dungeon-Crawler Game in Unity Engine

Team nr 04

s24387 Illia Tkachenko
s24884 Hubert Hagel
s24238 Yan Novikau

Supervisor: Pavlo Zinevych
Reviewer: Rafał Masłyk

February 2025

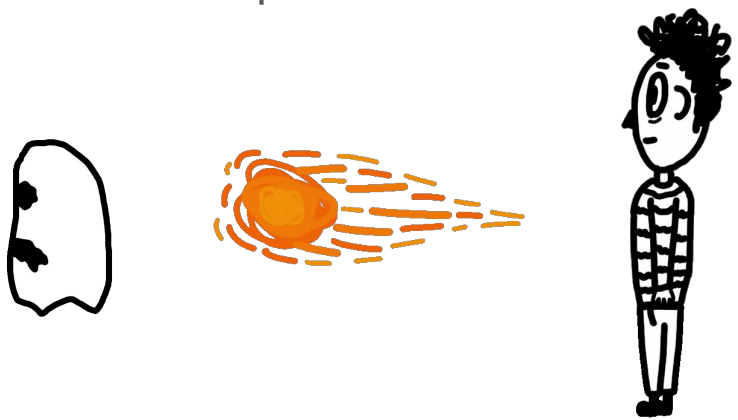
Project Overview

Sketchy in a Video Game

Inspired by the style and gameplay of Don't Starve, Enter the Gungeon, and The Binding of Isaac, this dungeon crawler puts players into a surreal, *sketchy* world full of dangers.

Unique Selling Points

The game combines hand-drawn art style with dungeon crawler experience to create an engaging and visually distinctive experience.



Project Overview

Genre:
Dungeon Crawler

Setting:
Hand-drawn, Surreal

Main mechanics:
Combat, Dungeon Exploration

Target audience:
16-35 years old casual PC players.

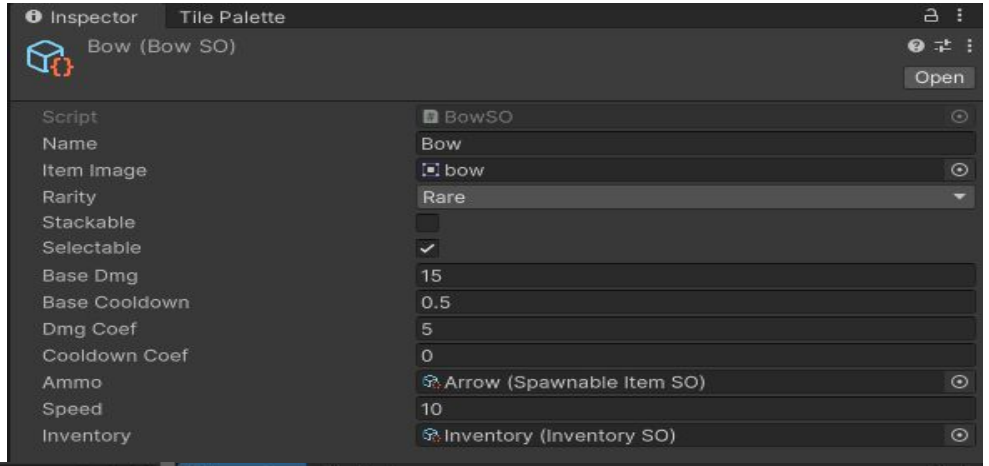
Inspirations

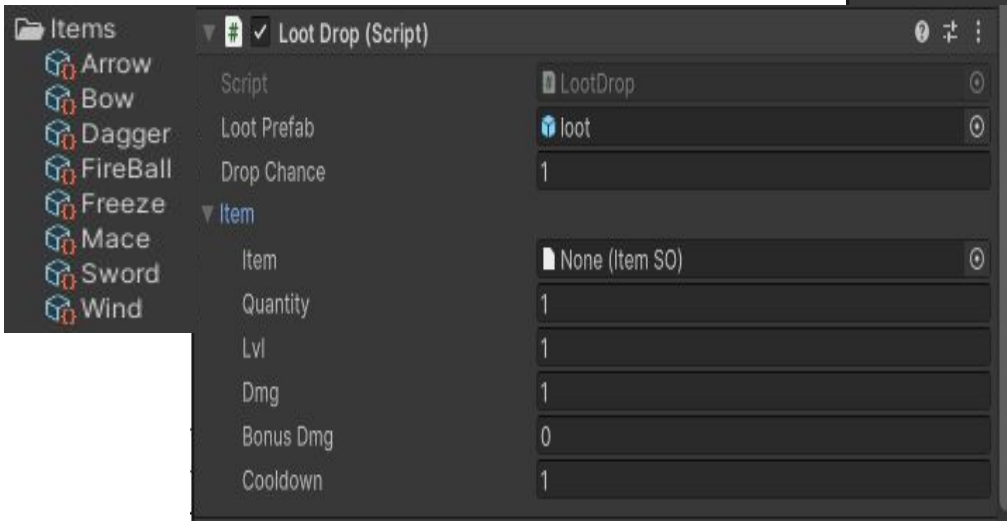
Tools

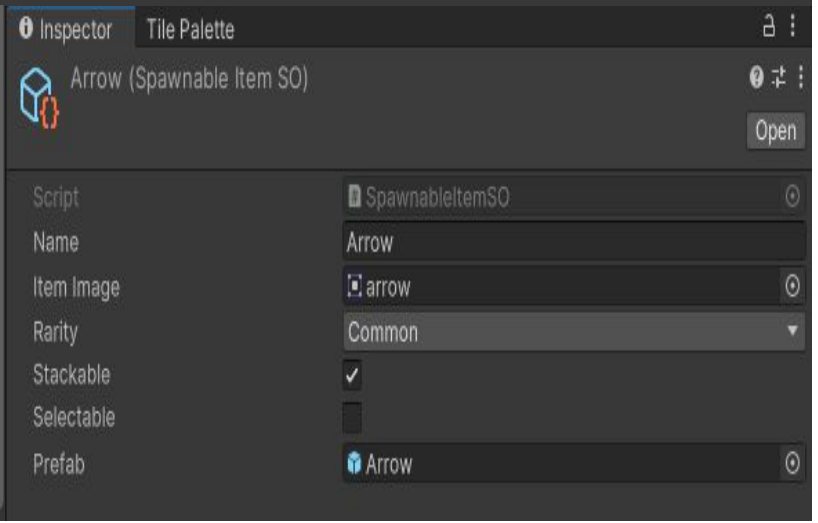
- Unity
- Github
- Adobe Illustrator
- Adobe Animate

Methodology

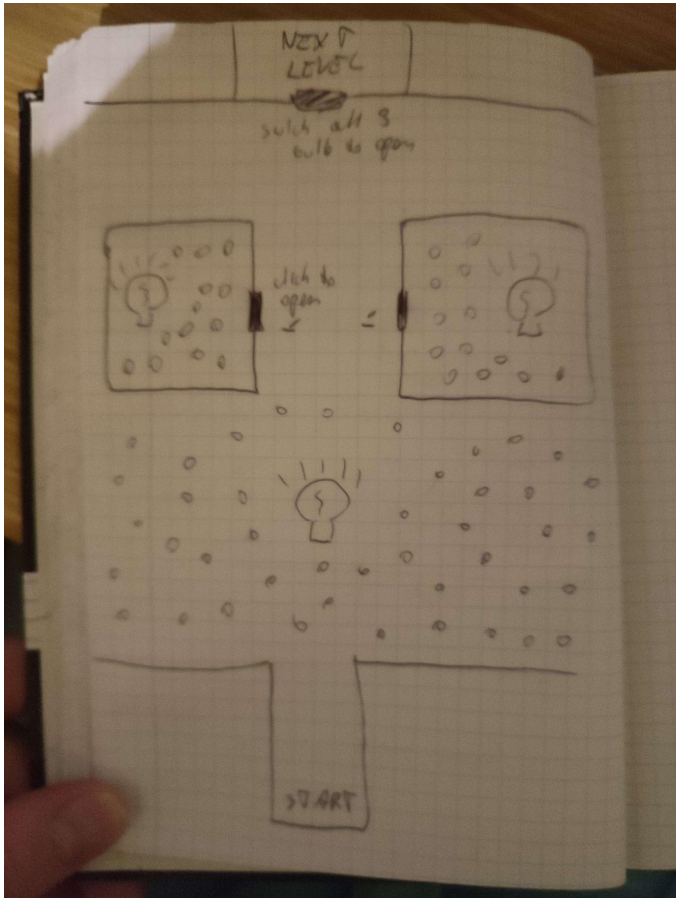
- Component-based design
- Sciptable Object-based architecture
- MVC

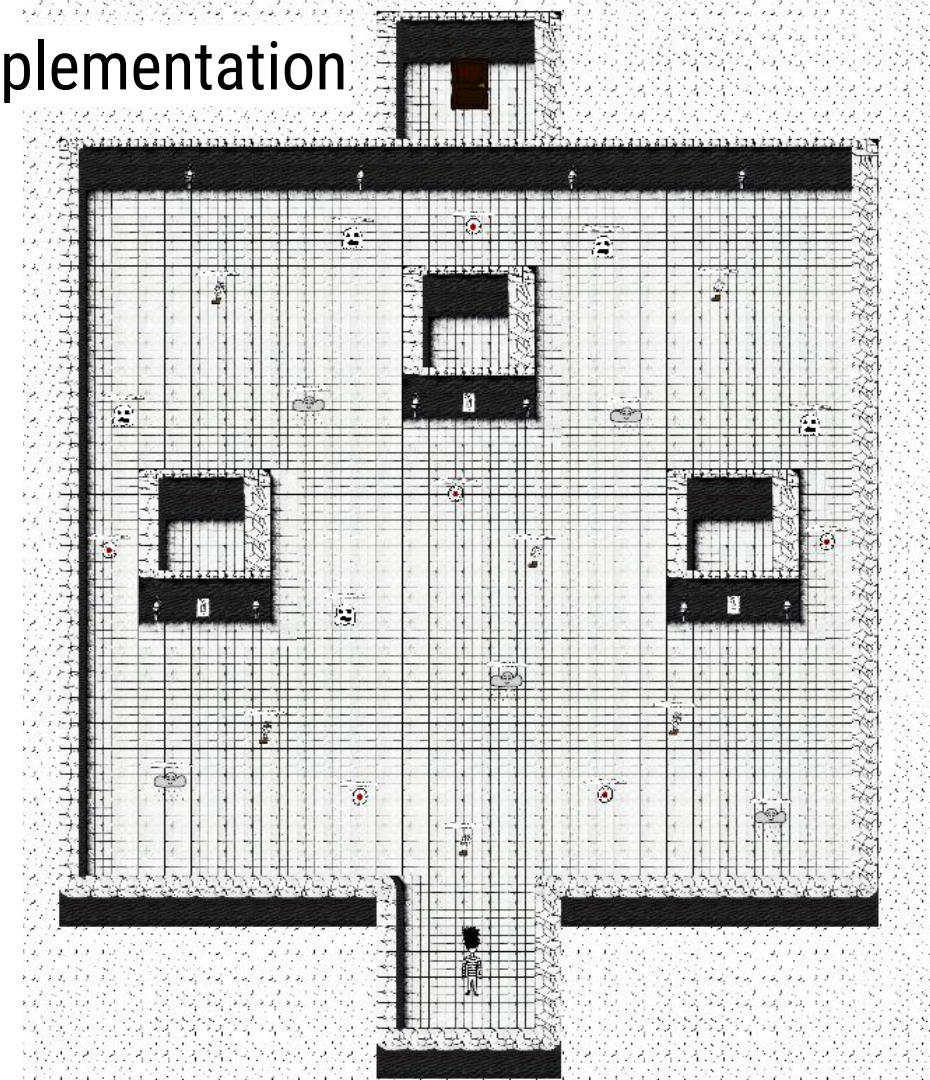




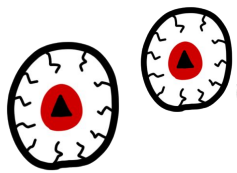


Level Design and Implementation





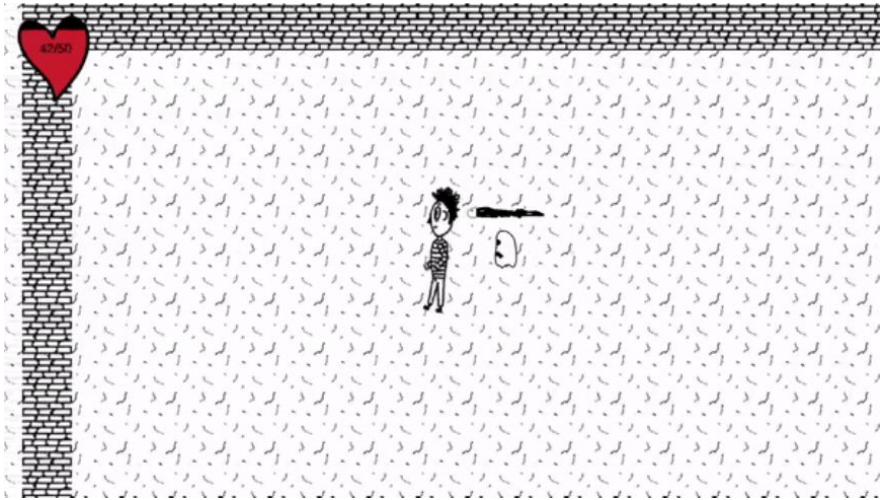
Design and Implementation



xr_c

Challenges and Solutions

- Limitations for 2.5D projects in Unity
- ScriptableObjects limitations
- Abstraction of scripts



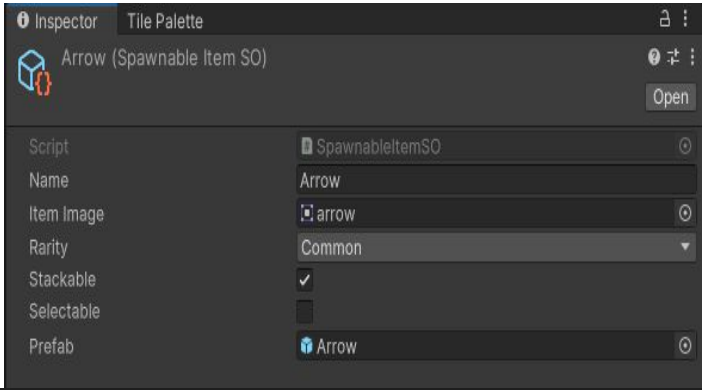
ScriptableObjects limitations

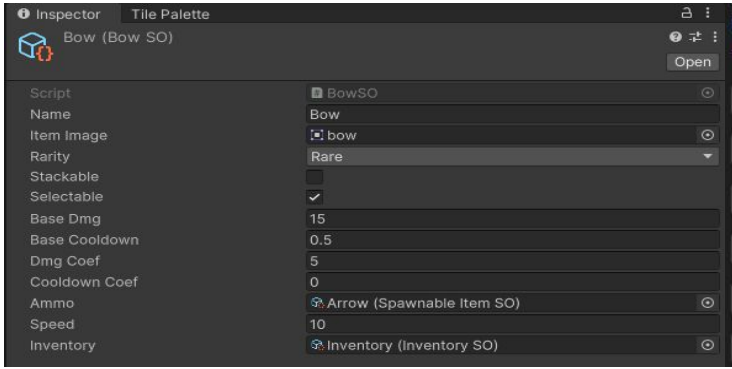
MonoBehaviour

collision handling, require instance

ScriptableObject

don’t have MonoBehaviour, easily configurable






```
public class InventoryItem
{
    30 references
    [field: SerializeField] public ItemSO Item { get; set; }
    10 references
    public int Quantity = 1;
    5 references
    [field: SerializeField] public int Lvl { get; set; } = 1;
    5 references
    [field: SerializeField] public float Dmg { get; set; } = 1f;
    4 references
    [field: SerializeField] public float BonusDmg { get; set; } = 0f; //random dmg bonus
    6 references
    [field: SerializeField] public float Cooldown { get; set; } = 1f;
```

Abstraction of scripts

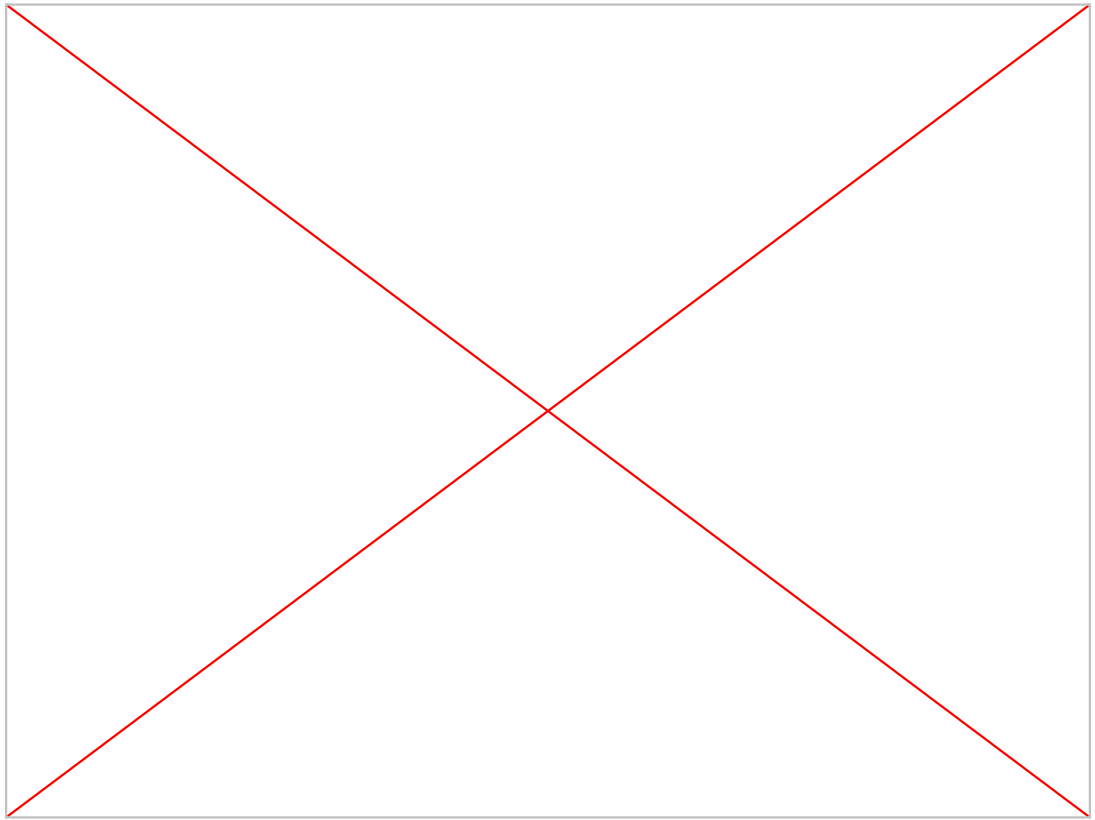
Instead of creating separate scripts for each creature in the game, abstract scripts were created.

- StatsCalculation: manages health, attack, defence and taking damage for all creature;
- Enemy: handles similar enemy behaviors like attack, cooldown, loot drop, etc.
- EnemyMovement: handles similar movement behaviors of enemies like random movement, chasing and taking distance from player;

```
public abstract class Enemy : StatsCalculation
{
    [Serializable]
    public class AttackManager
    {...}

    [SerializeField] protected List<AttackManager> _attacks = new();
    protected LootDrop _lootDrop;
    protected EnemyMovement _movement;
    protected Boss1Movement _movement2;
    [SerializeField] private int xpGiven = 10;


    [SerializeField] protected float _generalCooldown = 3f;
    protected float remainingGeneralCooldown = 0f;
    protected bool _canAttack = true;
    protected GameObject player;
    protected PlayerStats playerStats;
    protected AttackManager lastUsedAttack;
    private Animator _animator;

    protected override void Start()
    {
        base.Start();
        player = GameObject.Find("Body");
        playerStats = player.GetComponent<PlayerStats>();
        _lootDrop = GetComponent<LootDrop>();
        _movement = GetComponent<EnemyMovement>();
```

```
public abstract class EnemyMovement : MonoBehaviour
{
    [SerializeField] protected float _movementSpeed;
    [SerializeField] private float _minInterval;
    [SerializeField] private float _maxInterval;
    [SerializeField] protected float _chaseRadius;
    [SerializeField] private float _keepingDistance;
    protected Animator _animator;

    private GameObject _player;
    protected Vector2 _playerPosition;
    private bool _isChasing = false;
    protected Vector2 _targetDirection;
    private float _changeDirectionInterval;

    public bool CanMove { get; set; } = true;
```

Game Trailer

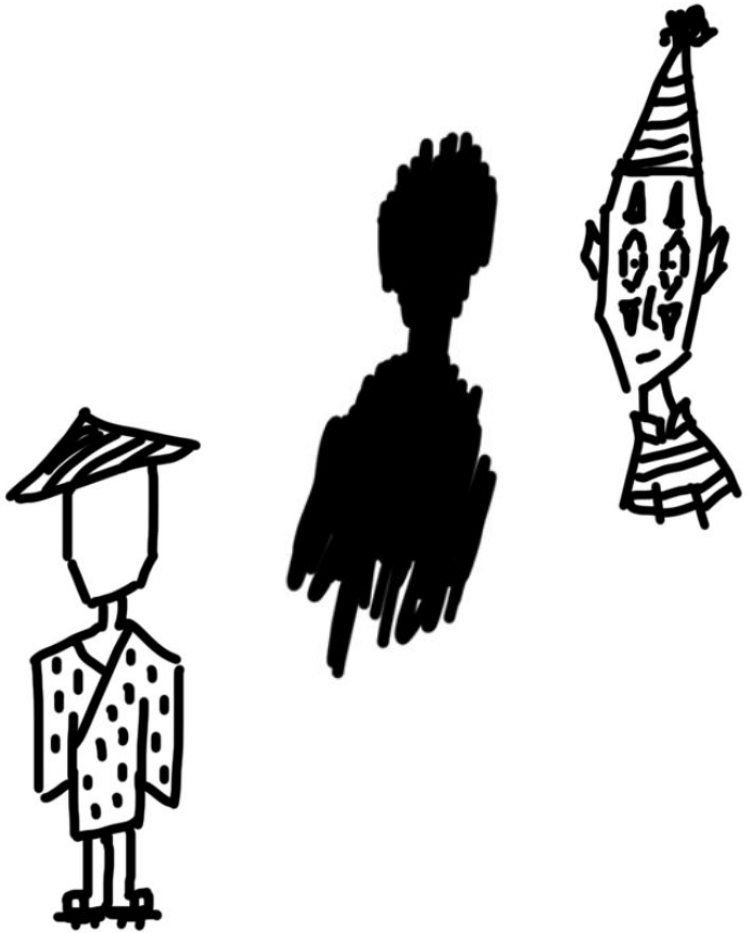


Results and Discussion

- Inventory:
  Changed from “Inventory Screen” to in-game inventory. After testing ammo and “out of ammo” indicators were added.
- Weapons and Skills:
  Initially scripts to independently manage movement, collisions, etc., but changed to Scriptable Objects.
- Enemies:
  The abstract architecture of enemy design greatly decreased code duplication, which made the system simpler to administer, test and adjust balance after.

Future Work

- Expanding Game Content
- Enhancing the Upgrade System
- Polishing Animations and Visual Feedback
- Balancing
- Procedural Dungeon Generation
- Multiplayer Implementation



Conclusion

- Process was hard but interesting, requiring creativity and technical skills
- Success in combining uncommon style and dungeon-crawler gameplay
- Represents a short, but completed project
- Knowledge gained provides a foundation for more ambitious future projects

Q & A

Thank you for your attention