The background is a dense, intricate line drawing of a steampunk-style mechanical system. It features numerous gears of various sizes, pistons, valves, and interconnected pipes. The drawing is rendered in a light, monochromatic style, possibly sepia or light brown, against a slightly darker background. The overall impression is one of complex, vintage industrial machinery.

„Dirty Game Jams Triks in Unity.“



Mateusz Pusty
Unity Developer @ Robot Gentleman

 @mateuszpusty

- Unity Developer @ Robot Gentleman
- Pracuje nad *60 Parsecs!*
- Jeden z organizatorów spotkań Poznań Unity User Group
- Jeden z organizatorów All Play Game Jam
- Założyciel Koła Naukowego „Pyra” na Uniwersytecie im. Adama Mickiewicza w Poznaniu



60 Seconds!

- 60 Seconds! – komediowa „atomic adventure” w której mamy 60 sekund na zebranie potrzebnych do przeżycia surowców.
- Wydana w 2015 na PC i Mac. W 2016 na iOS.
- Obecnie trwają prace nad wersjami na konsole i Androida.



60 Parsecs!

- 60 Parsecs! – kosmiczna kontynuacja pomysłu z 60 Seconds!
- Ogłoszona w tym roku podczas targów Poznań Game Arena.
- Planowane wydanie na PC oraz konsole.

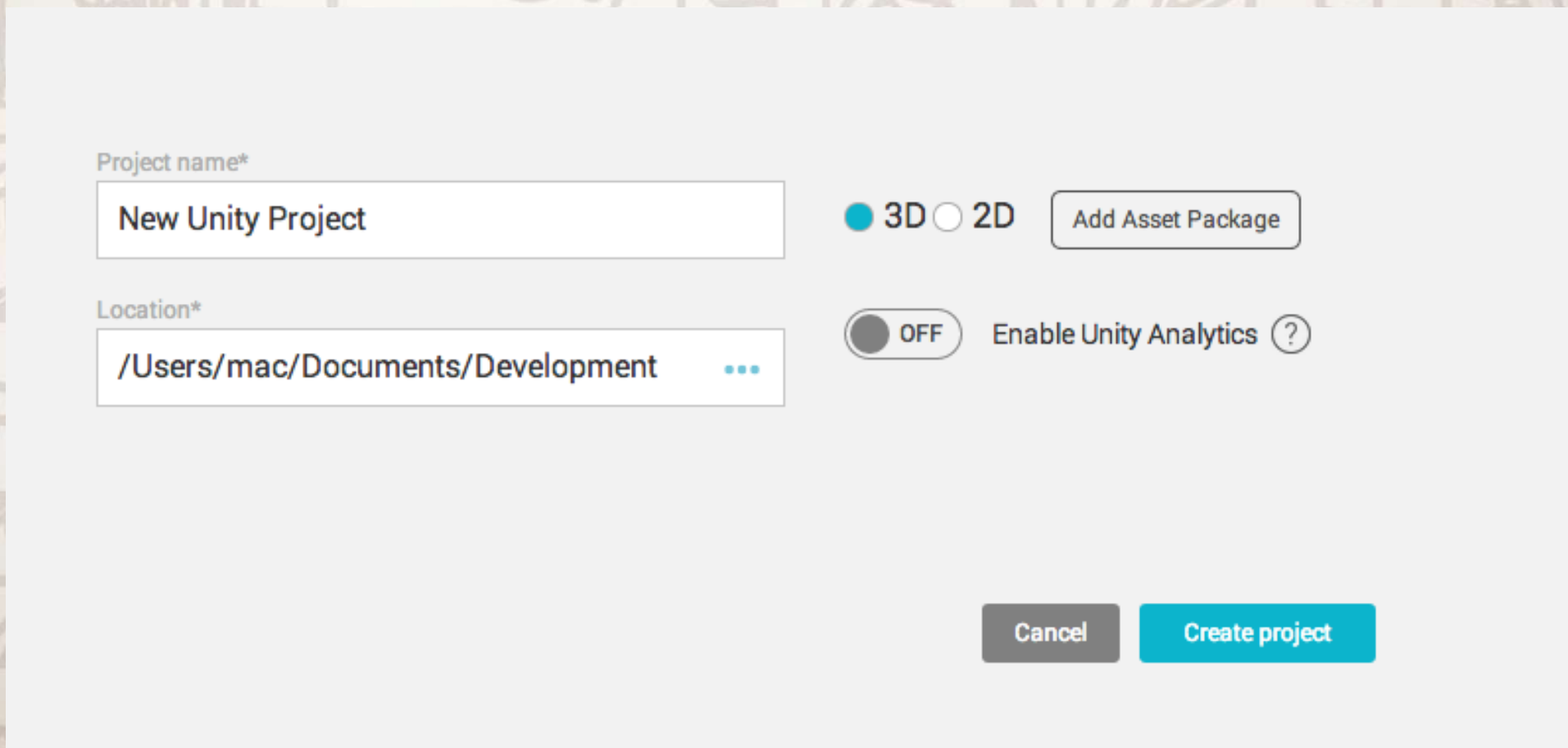


The background is a dense, intricate line drawing of a steampunk-style mechanical system. It features numerous gears of various sizes, pistons, valves, and pipes. There are several circular gauges or pressure meters with needle indicators. The overall aesthetic is that of a complex, vintage industrial machine. The text is centered over this background.

Przed dzemem.

Przed dżemem.

1. Stwórz projekt.



Project name*

New Unity Project

3D 2D

Add Asset Package

Location*

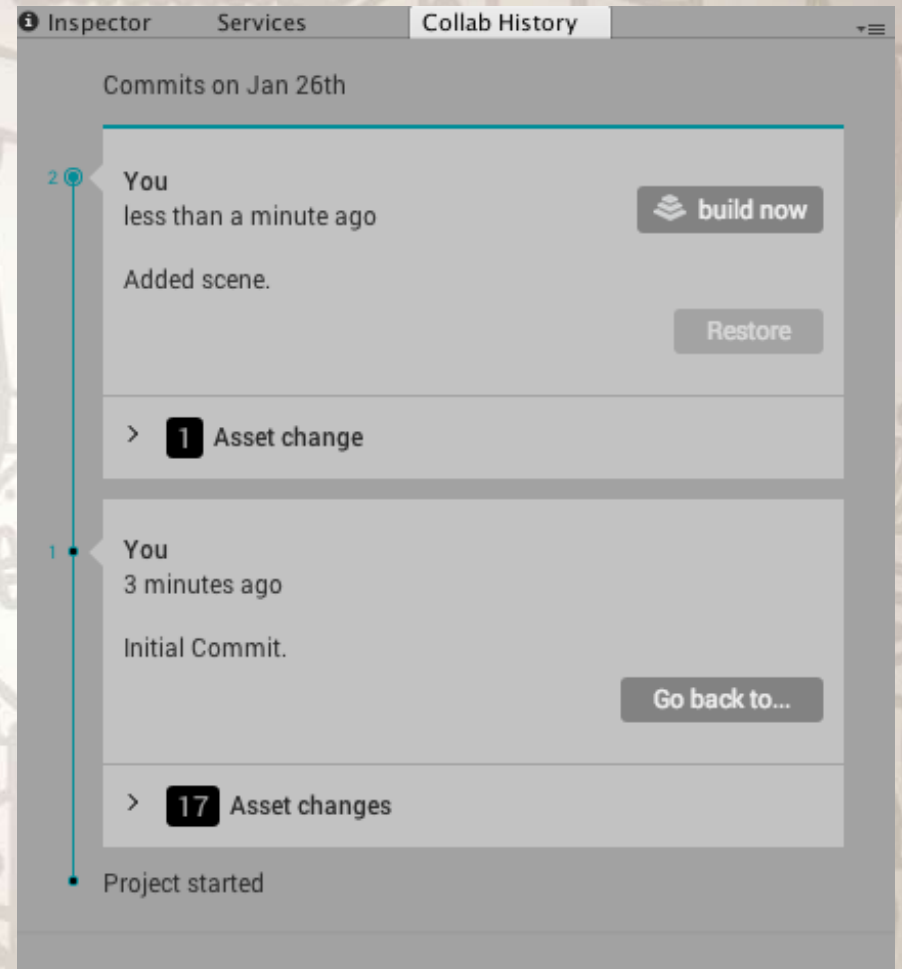
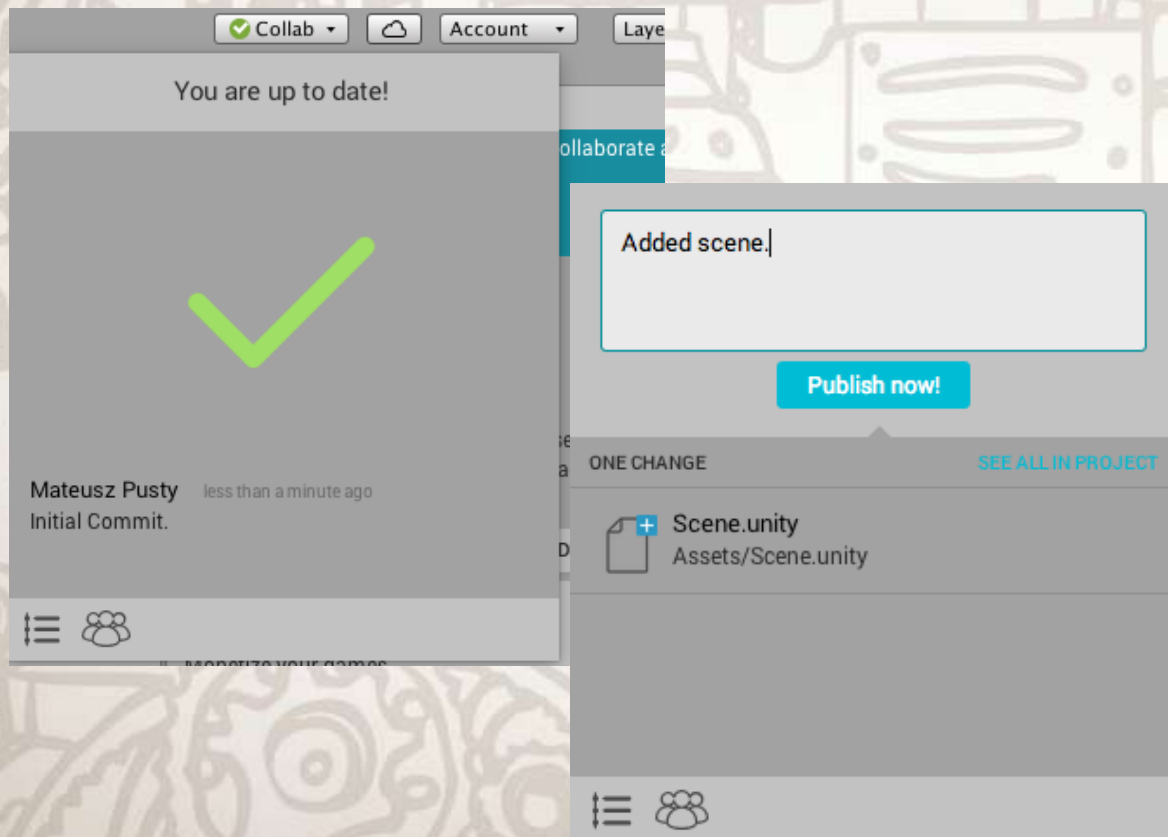
/Users/mac/Documents/Development ...

OFF Enable Unity Analytics ?

Cancel Create project

Przed dżemem.

1. Stwórz projekt.
2. Collaborate - system kontroli wersji w Unity!



Przed dżemem.

1. Stwórz projekt.
2. Collaborate - system kontroli wersji w Unity!
3. Dołącz wszystkie zewnętrzne pluginy, z których korzystasz.



Przed dżemem.

1. Stwórz projekt.
2. Collaborate - system kontroli wersji w Unity!
3. Dołącz wszystkie zewnętrzne pluginy, z których korzystasz.
4. Wyśpij się!



The background is a dense, intricate pattern of steampunk-style mechanical components. It features various gears of different sizes, pistons, valves, and mechanical linkages, all rendered in a light beige or tan color against a slightly darker, textured background. The overall aesthetic is that of a complex, vintage industrial machine.

Początek dżemu.

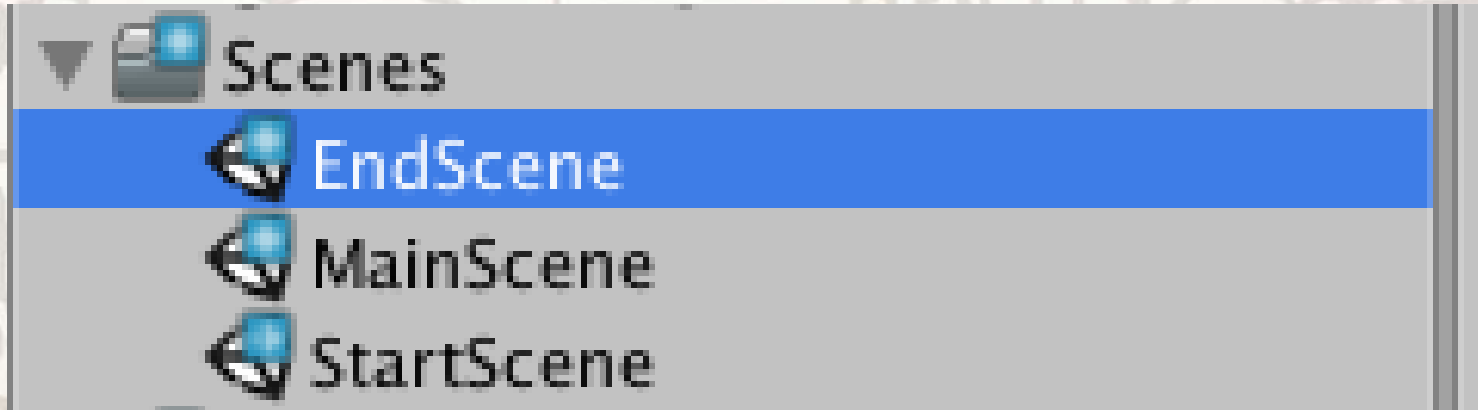
Początek dżemu.

1. Globalny GameManager

```
3  
4 public class GameManager : MonoBehaviour  
5 {  
6     public static GameManager Instance;  
7  
8     private void Awake()  
9     {  
10         if (Instance == null)  
11         {  
12             Instance = this;  
13             DontDestroyOnLoad(gameObject);  
14         }  
15         else if (Instance != this)  
16         {  
17             Destroy(gameObject);  
18         }  
19     }  
20 }
```

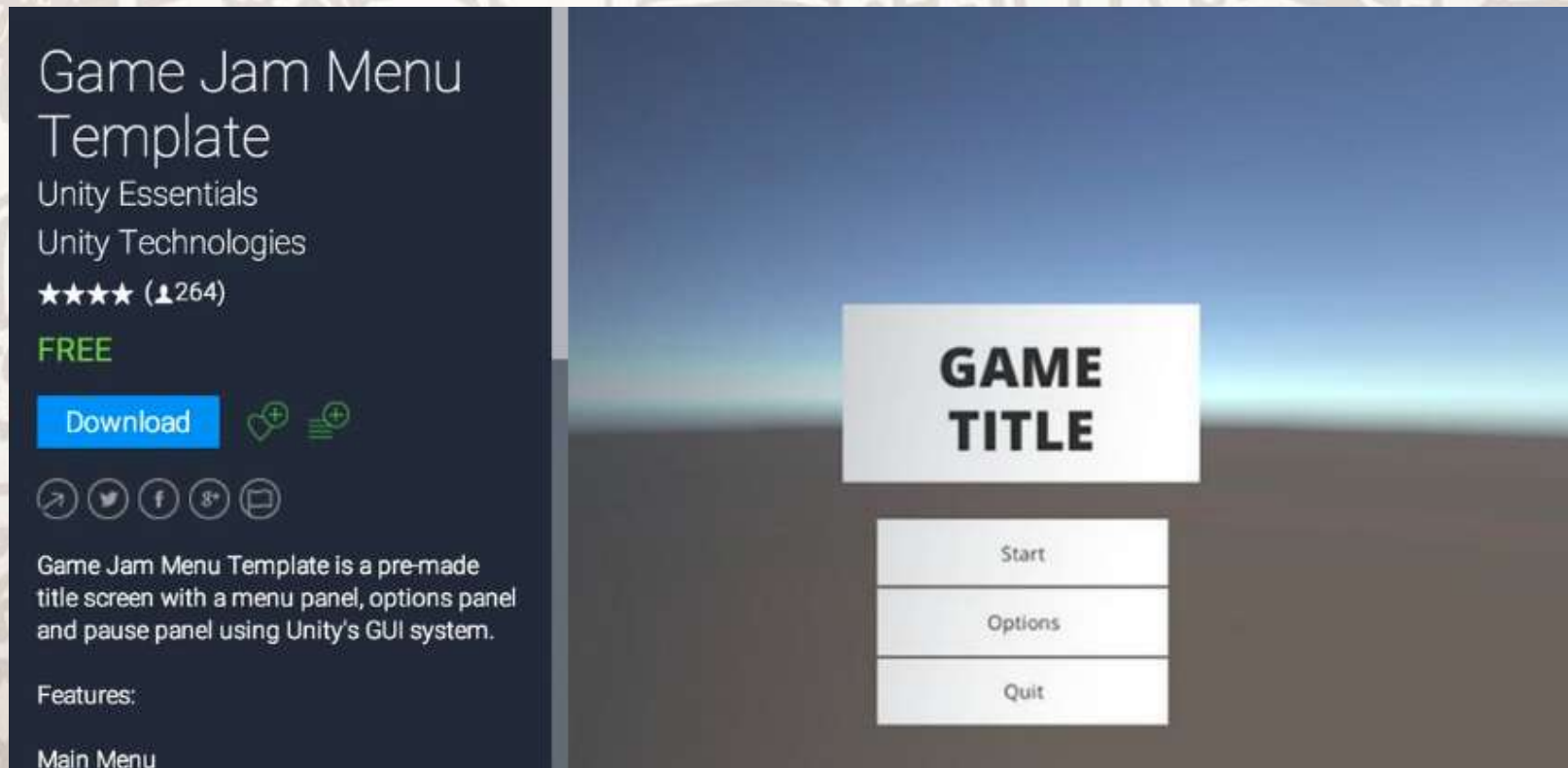
W trakcie dzemu.

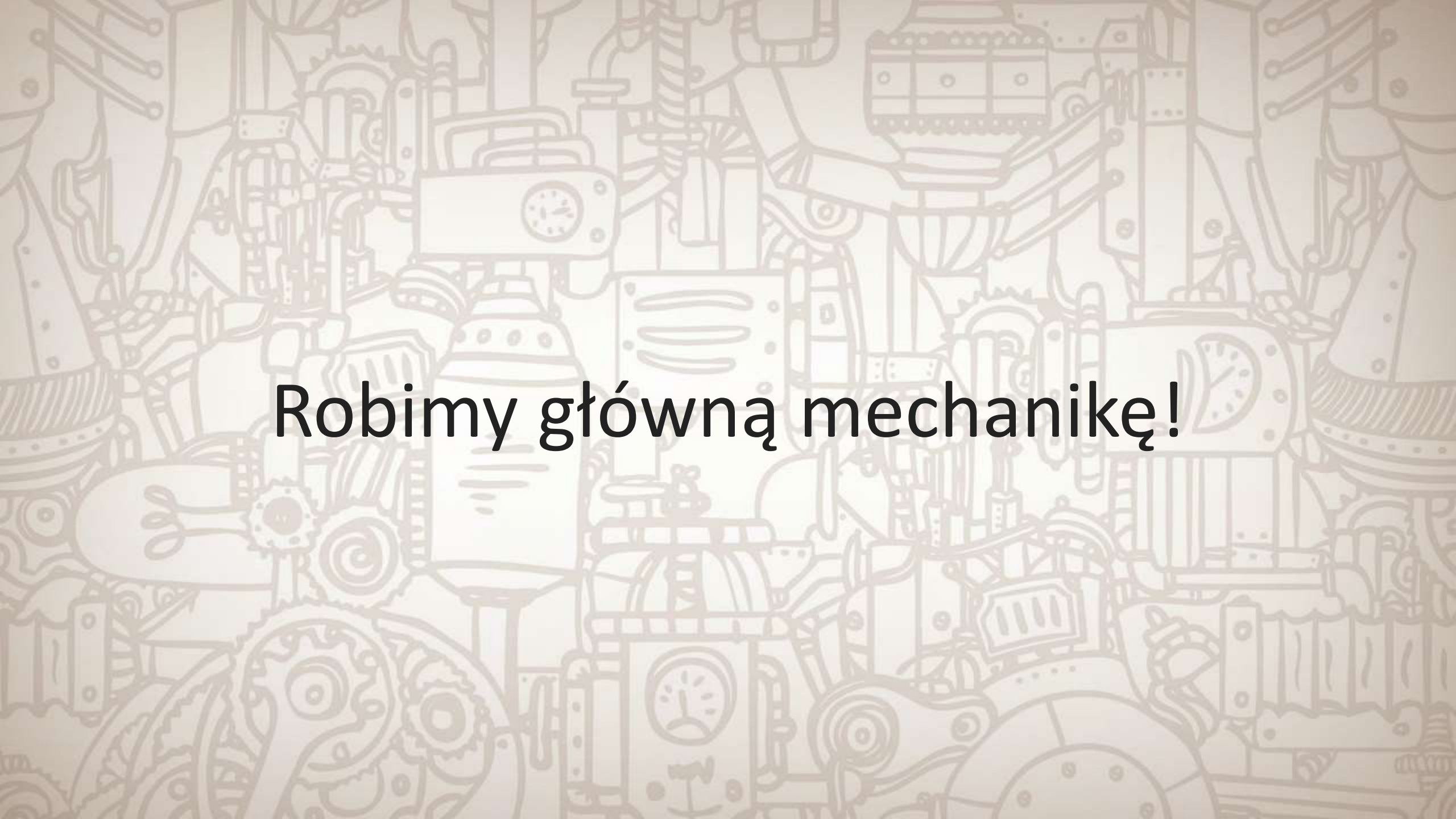
1. Globalny GameManager
2. Utwórz 3 sceny (Startowa, Główna, Końcowa)



W trakcie dzemu.

1. Globalny GameManager
2. Utwórz 3 sceny (Startowa, Główna, Końcowa)
3. Zrób proste menu, przejścia pomiędzy scenami i reset gry.

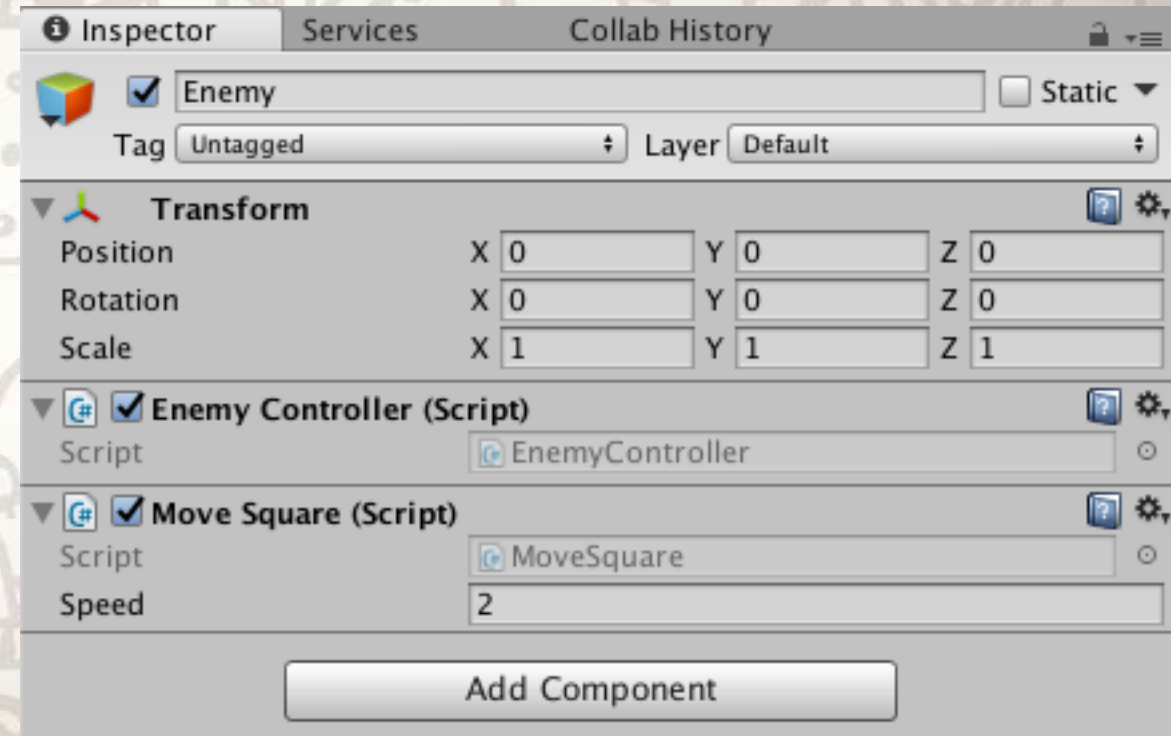
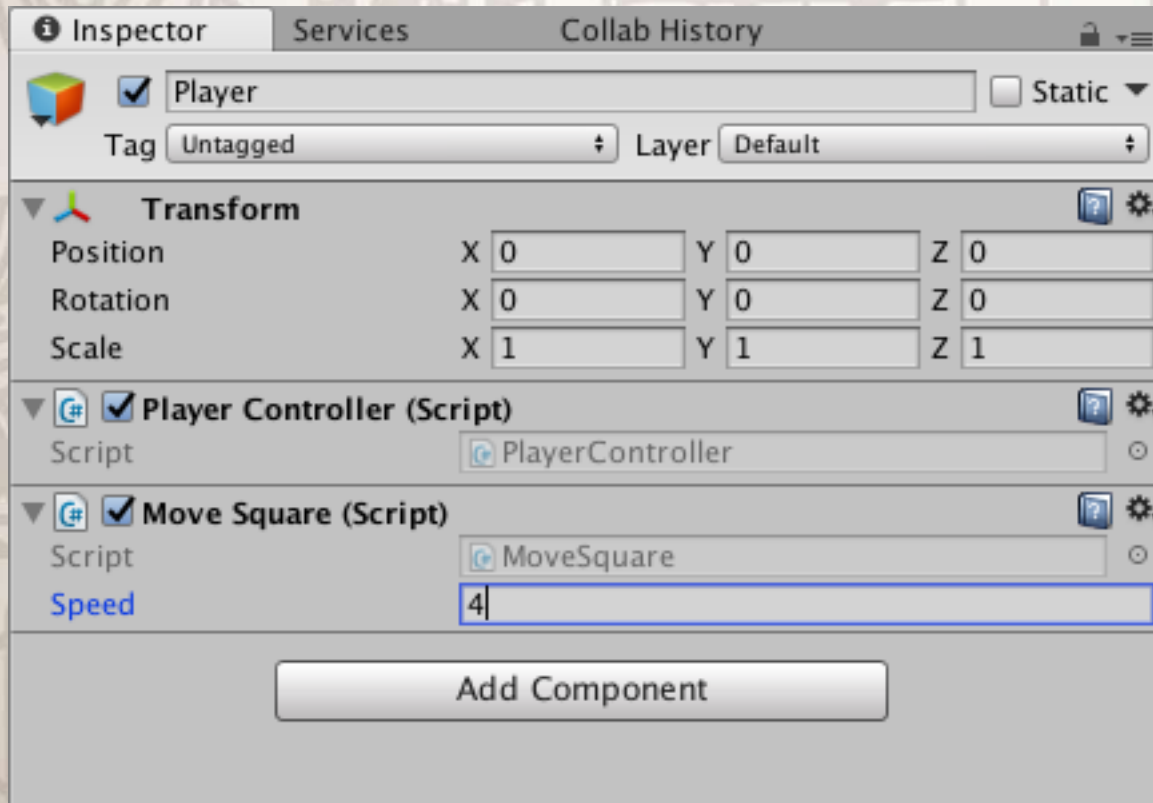


The background is a dense, intricate line drawing of a steampunk-style mechanical system. It features numerous gears of various sizes, pistons, valves, and pipes. There are several circular gauges or pressure meters with needles. The overall aesthetic is industrial and mechanical, rendered in a light, sketchy style. The text is centered over this background.

Robimy główną mechanikę!

Robimy główną mechanikę!

1. Wyrzucić wspólne elementy mechaniki do osobnych kontrolerów.



Robimy główną mechanikę!

1. Wyrzuć wspólne elementy mechaniki do osobnych kontrolerów.
2. Używaj referencji z inspektora gdzie jest to możliwe i sensowne.

```
0 references
public class ChangeSquareColor : MonoBehaviour
{
    [SerializeField]
    private SpriteRenderer squareRenderer;

    /*
     * Some code here.
     */
}
```

▼ Player

- Square
- Circle
- Triangle
- UIManager
- Enemy
- Enemy (1)
- Enemy (2)

```
public class PlayerController : MonoBehaviour
{
    public int Points;

    public string Name;
    public float Health;
    public float ShootingSpeed;

    public EnemyController[] Enemies;
    public UIManager UI;

    0 references
    public void KillEnemy()
    {
        UI.SetPoints(Points);
    }

    /*
     * Some code here.
     */
}
```

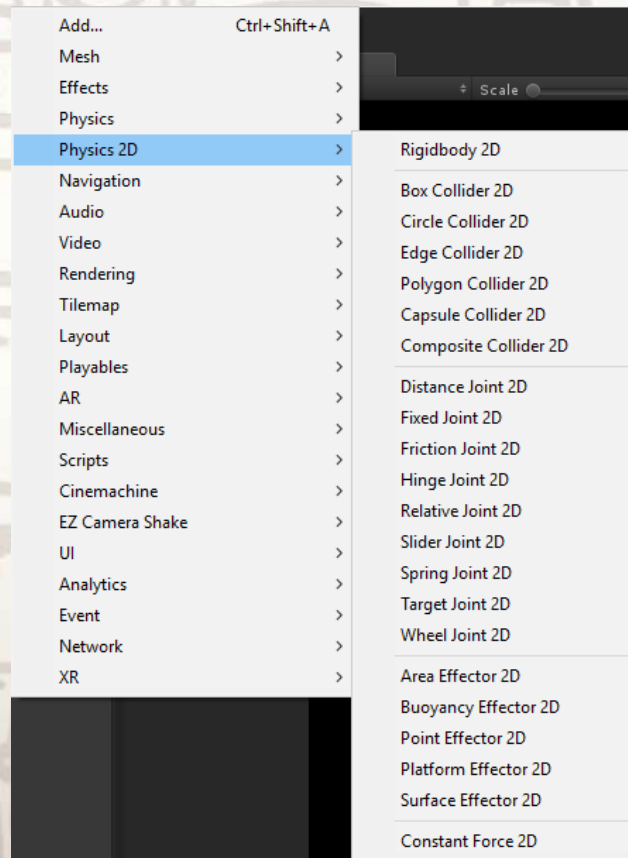
The screenshot shows the Unity Inspector window with the following components and their settings:


- Player Controller (Script)**
 - Script: PlayerController
 - Points: 0
 - Name: Ted
 - Health: 100
 - Shooting Speed: 5
 - Enemies**
 - Size: 3
 - Element 0: Enemy (EnemyController)
 - Element 1: Enemy (1) (EnemyController)
 - Element 2: Enemy (2) (EnemyController)
 - UI: UIManager (UIManager)
- Change Square Color (Script)**
 - Script: ChangeSquareColor
 - Square Renderer: Square (Sprite Renderer)
- Move Square (Script)**
 - Script: MoveSquare
 - Speed: 4

At the bottom, there is an "Add Component" button.

Robimy główną mechanikę!

1. Wyrzuć wspólne elementy mechaniki do osobnych kontrolerów.
2. Używaj referencji z inspektora gdzie jest to możliwe i sensowne.
3. Staraj się korzystać z gotowych komponentów Unity gdzie tylko się da.

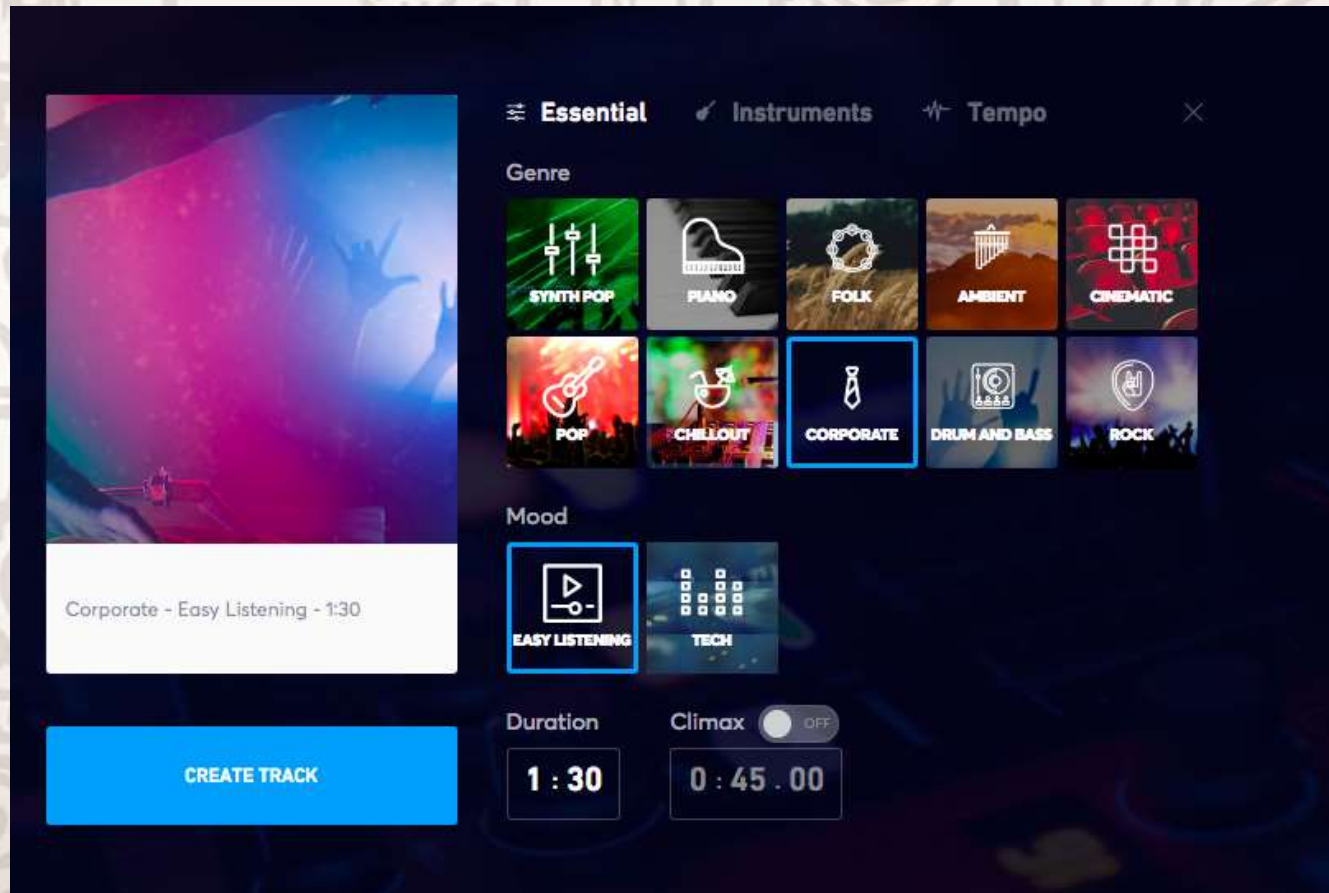


The background is a dense, intricate line drawing of a steampunk-style mechanical system. It features numerous gears of various sizes, pistons, valves, and pipes. Some components have circular gauges or pressure meters with needle indicators. The overall aesthetic is that of a complex, vintage industrial machine. The text is centered over this background.

Dodajemy Audio!

Dodajemy Audio!

1. Jukedeck – muzyka wygenerowana komputerowo! - <https://www.jukedeck.com/>



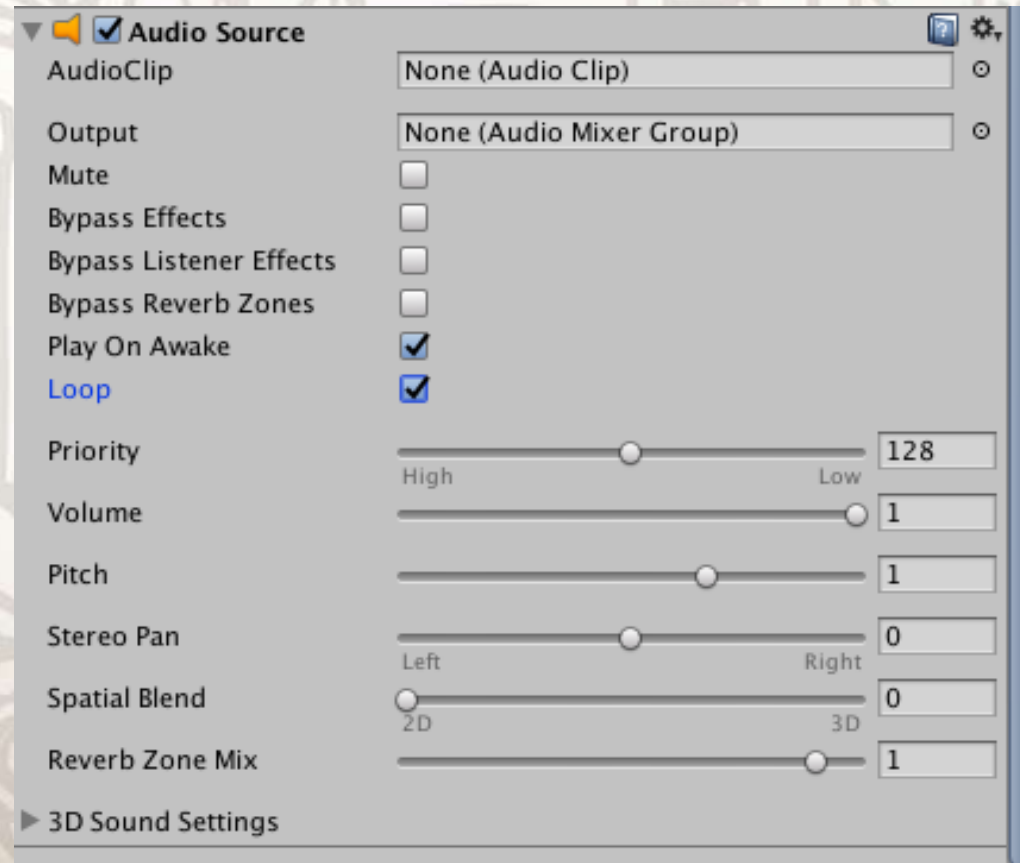
Dodajemy Audio!

1. Jukedek – muzyka wygenerowana komputerowo! - <https://www.jukedek.com/>
2. Bfxr – generujemy efekty dźwiękowe!



Dodajemy Audio!


1. Jukedek – muzyka wygenerowana komputerowo! - <https://www.jukedek.com/>
2. Bfxr – generujemy efekty dźwiękowe!
3. Dodaj AudioSource na GameManager aby mieć muzykę, która będzie żyć pomiędzy scenami!



Dodajemy Audio!

1. Jukedek – muzyka wygenerowana komputerowo! - <https://www.jukedek.com/>
2. Bfxr – generujemy efekty dźwiękowe!
3. Dodaj AudioSource na GameManager aby mieć muzykę, która będzie żyć pomiędzy scenami!
4. AudioSource.PlayClipAtPoint() zagrany na główną kamerę.

```
private void PlayAudio(AudioClip clip)
{
    AudioSource.PlayClipAtPoint(clip, Camera.main.transform.position);
}
```


The background is a dense, intricate line drawing of a steampunk-style mechanical system. It features numerous gears of various sizes, pistons, valves, and pipes. There are several circular gauges or pressure meters with needle indicators. The overall aesthetic is that of a complex, industrial machine from a fictional era. The text is centered over this background.

Nieczyste zagrywki

Nieczyste zagrywki.

1. Public statics! Public statics everywhere!

```
0 references
public class PlayerController : MonoBehaviour
{
    public static string Name;
    public static float Health;
    public static float ShootingSpeed;

    /*
     * Some code herer.
     */
}
```


Nieczyste zagrywki.

1. Public statics! Public statics everywhere!
2. Co się da w inspektorze.

```
1 reference
public class PlayerController : MonoBehaviour
{
    public int Points;

    public string Name;
    public float Health;
    public float ShootingSpeed;

    public EnemyController[] Enemies;
    public UIManager UI;

    0 references
    public void KillEnemy()
    {
        UI.SetPoints(Points);
    }

    /*
     * Some code herer.
     */
}
```

Nieczyste zagrywki.

1. Public statics! Public statics everywhere!
2. Co się da w inspektorze.
3. Stringi i magiczne liczby.

```
0 references
public void KillEnemy(string enemyType)
{
    switch (enemyType)
    {
        case "big":
            UI.SetPoints(5);
            break;
        case "small":
            UI.SetPoints(2);
            break;
        case "giant":
            UI.SetPoints(10);
            break;
    }
}
```

```
0 references
public void KillEnemy(EnemyType enemyType)
{
    const int SMALL_ENEMY_POINTS = 2;
    const int BIG_ENEMY_POINTS = 5;
    const int GIANT_ENEMY_POINTS = 10;

    switch (enemyType)
    {
        case EnemyType.SMALL:
            UI.SetPoints(SMALL_ENEMY_POINTS);
            break;
        case EnemyType.BIG:
            UI.SetPoints(BIG_ENEMY_POINTS);
            break;
        case EnemyType.GIANT:
            UI.SetPoints(GIANT_ENEMY_POINTS);
            break;
        default:
            throw new ArgumentOutOfRangeException("enemyType", enemyType, null);
    }
}
```


Nieczyste zagrywki.

1. Public statics! Public statics everywhere!
2. Co się da w inspektorze.
3. Stringi i magiczne liczby.
4. `GameObject.Find()`, `FindObjectsOfType<T>()` – tu ujdą Ci na sucho!

```
0 references
public void Start()
{
    UIManager UItype = FindObjectOfType<UIManager>();

    UIManager UIfind = GameObject.Find("UIManager").GetComponent<UIManager>();
}
```

```
0 references
public void Start()
{
    EnemyController[] Enemies = FindObjectsOfType<EnemyController>();
}
```

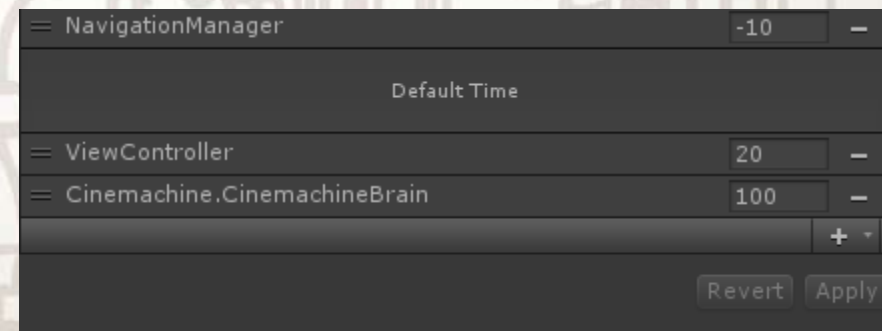
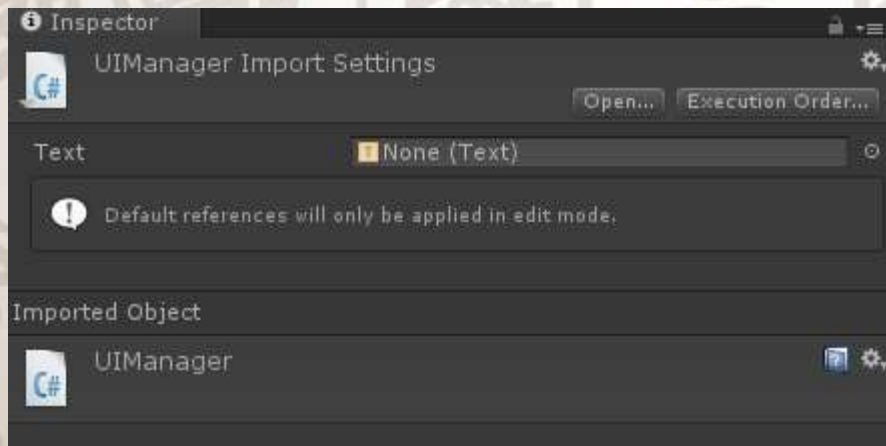
Nieczyste zagrywki.

1. Public statics! Public statics everywhere!
2. Co się da w inspektorze.
3. Stringi i magiczne liczby.
4. `GameObject.Find()`, `FindObjectsOfType<T>()` – tu ujdą Ci na sucho!
5. `Resources.Load()` – jedyny przypadek, w którym można z niego korzystać!

```
0 references  
public void Start()  
{  
    var textFile = (TextAsset) Resources.Load("Text");  
}
```


Nieczyste zagrywki.

1. Public statics! Public statics everywhere!
2. Co się da w inspektorze.
3. Stringi i magiczne liczby.
4. `GameObject.Find()`, `FindObjectsOfType<T>()` – tu ujdą Ci na sucho!
5. `Resources.Load()` – jedyny przypadek, w którym można z niego korzystać!
6. Script Execution Order!



Nieczyste zagrywki.

1. Public statics! Public statics everywhere!
2. Co się da w inspektorze.
3. Stringi i magiczne liczby.
4. `GameObject.Find()`, `FindObjectsOfType<T>()` – tu ujdą Ci na sucho!
5. `Resources.Load()` – jedyny przypadek, w którym można z niego korzystać!
6. Script Execution Order!
7. Bezpośrednie odwołania do klawiszy.

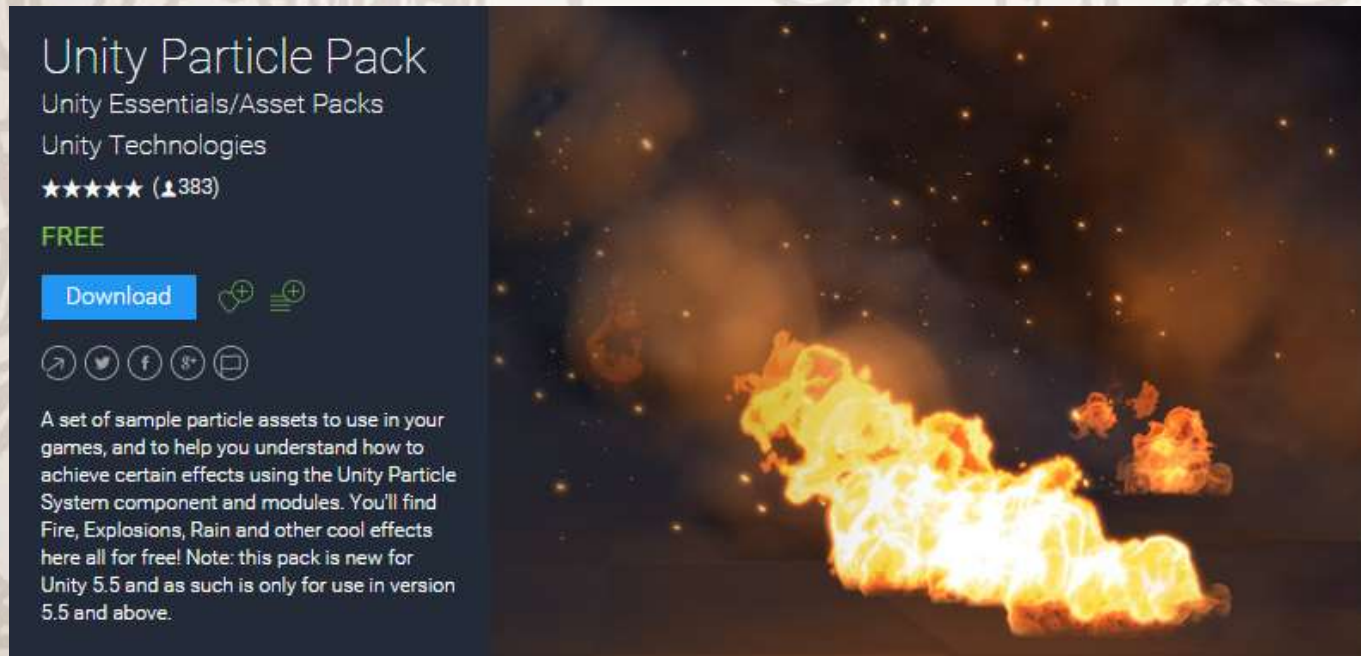
```
0 references  
void Update()  
{  
    if(Input.GetKeyDown(KeyCode.Space))  
        Jump();  
}
```


The background is a dense, intricate line drawing of a steampunk-style mechanical system. It features numerous gears of various sizes, pistons, valves, and pipes. There are several circular gauges or pressure meters with needle indicators. The overall aesthetic is that of a complex, vintage industrial machine. The drawing is done in a light, sketchy style with fine lines.

Poliszing

Polishing

1. Particles!



Particle System	
Duration	5.00
Looping	<input checked="" type="checkbox"/>
Prewarm	<input type="checkbox"/>
Start Delay	0
Start Lifetime	5
Start Speed	5
3D Start Size	<input type="checkbox"/>
Start Size	1
3D Start Rotation	<input type="checkbox"/>
Start Rotation	0
Randomize Rotation	0
Start Color	
Gravity Modifier	0
Simulation Space	Local
Simulation Speed	1
Delta Time	Scaled
Scaling Mode	Local
Play On Awake*	<input checked="" type="checkbox"/>
Emitter Velocity	Rigidbody
Max Particles	1000
Auto Random Seed	<input checked="" type="checkbox"/>
Stop Action	None

Polishing

1. Particles!
2. Post Processing Stack

Post Processing Stack

Unity Essentials
Unity Technologies

★★★★★ (1046)

FREE

Download

Note: version 2.0 of the post-processing stack is currently [in development on Github](#).

The new Unity post-processing stack is an über effect that combines a complete set of image effects into a single post-process pipeline. This has a few advantages :



☒ Bloom

☒ Color Grading

Tonemapping

Tonemapper

Neutral

Neutral Tonemapper



Black In

0.0384

White In

10

Black Out

0

White Out

10

White Level

5.3

White Clip

10

Basic

Post Exposure (EV)

0.18

Temperature

0

Tint

0

Hue Shift

0

Saturation

1

Contrast

1

Channel Mixer

Channel

Red

Green

Blue

Red

1

Green

0

Blue


0

Trackballs

Linear

Log

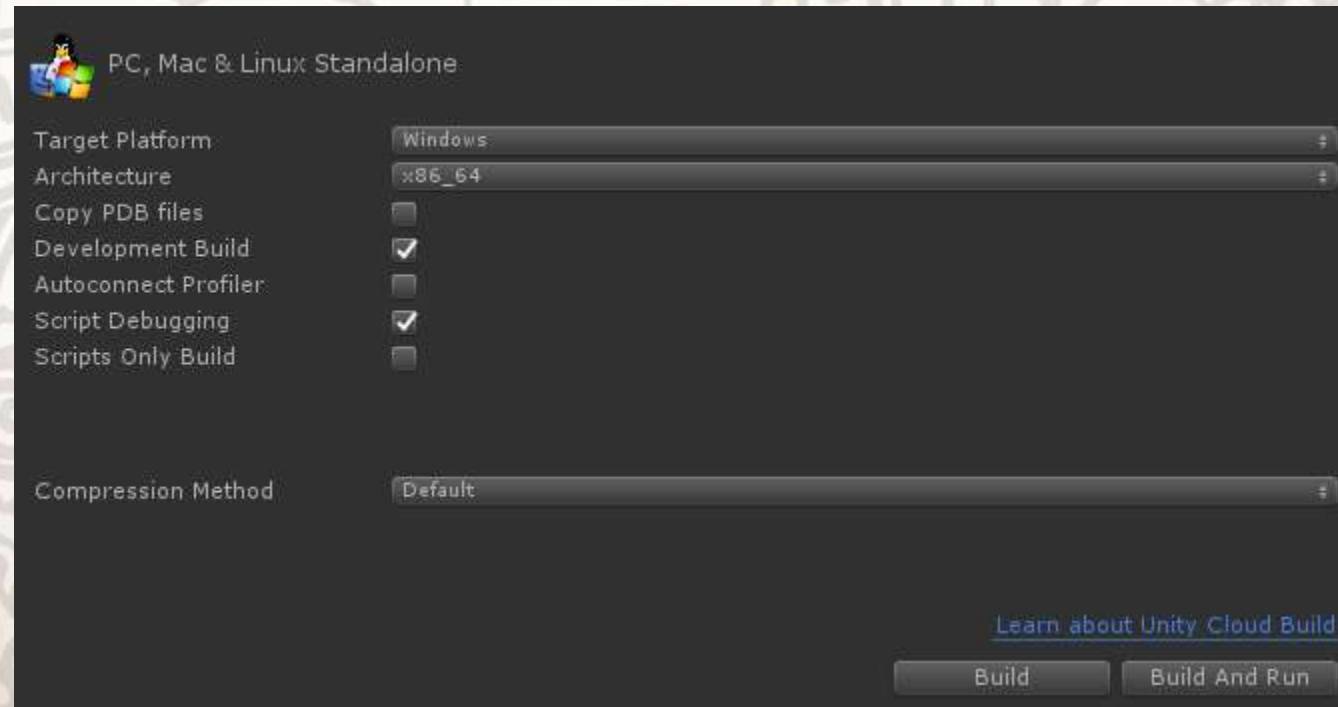
Log Controls



1.00 1.00 1.00 1.00 1.00 1.00 0.00 0.00 0.00

Polishing

1. Particles!
2. Post Processing Stack
3. Build



PYTANIA?





mateusz@robotgentleman.com
[@mateuszpusty](https://github.com/sygan)
<https://github.com/sygan>

