TECHNICAL DOCUMENTATION

TABLE OF CONTENTS

1. INTRO

Reel It In is a local couch-party **1v3** game featuring a battle between an all-powerful *cannon* and three *runners* armed with nothing but a rope. With no chance of outrunning the cannon's attacks, the runners must *work together*—reeling each other to safety. However, teamwork isn't the only option—*sabotage* is always on the table.

1.1. TEAM MEMBERS

NAME	EMAIL	MAJOR
Balder Huybreghs	balder.huybreghs@student.howest.be	GD
Jens Fierens	jens.fierens@student.howest.be	GGP
Agnese Cais	agnese.cais@student.howest.be	GGP
Arthur van den Barselaar	arthur.van.den.barselaar@student.howest.be	GD
Ioana Raileanu	ioana.raileanu@student.howest.be	GD
Ziqqy Ziqlam	ziqqy.ziqlam@student.howest.be	GGP

2. DEVELOPMENT STACK

2.1. DOCUMENTATION

We chose **Markdown** for documentation due to its *open standard* and *text-based format*. This makes it easy for our **source control tool** to merge files and resolve conflicts seamlessly, allowing multiple contributors to work on the same file simultaneously. In contrast, the alternative format (**.docx**) is a proprietary binary format, requiring us to use Perforce's file-locking feature—an approach that doesn't align with our workflow.

2.2. COMMUNICATION & MANAGEMENT

All links can be found in **Useful Links**

2.2.1. HacknPlan

HacknPlan is used for task management, providing a structured weekly overview of what needs to be done, who is responsible for each task, and which tasks are already in progress.

2.2.2. Miro

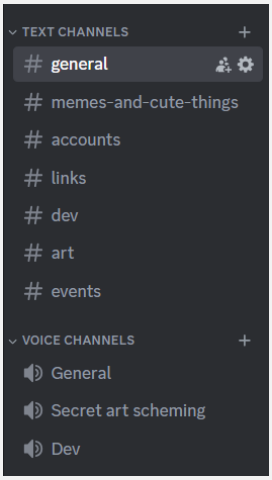
Miro serves as a collaborative space for brainstorming, sketching, and visualizing ideas. It helps in explaining concepts and maintaining an organized overview of all tasks.

2.2.3. Itch.io

Itch.io is our platform for game distribution and developer blogs, where we share progress updates and announcements.

2.2.4. Discord

We use Discord as our primary communication platform. A dedicated server, linked in the Useful Links section, helps us stay organized with various channels for different topics.



2.3. GAME ENGINE

- **Unity 6000.0.38f1 LTS**

We're choosing **Unity 6** for our game because it better aligns with our goals: *fast development*, *simple stylized 2.5D visuals*, and *minimal reliance on realistic physics*.

After prototyping a simple version of our core mechanic (reeling objects/actors) in both **Unreal** and **Unity**, we found that while Unreal provided realistic physics, it was restrictive due to its precise physics system. In contrast, Unity allowed us to implement the mechanic quickly and customize physics more freely to fit our theme.

Key reasons for our choice:

- **Great for stylized games** – Unity makes it easier to achieve a cartoonish look.
- **Simpler workflow** – Learning and working in Unity is more straightforward.
- **Less unnecessary complexity** – Unreal comes with many advanced features we don't need.
- **Customizable systems** – Unity allows us to write our own physics and gameplay logic more easily.
- **Strong documentation & tutorials** – Unity's resources make problem-solving faster.

Overall, Unity's *flexibility* and *ease of use* make it the best choice for our project.

2.3.1. Render pipeline

- **Universal Render Pipeline (URP)** is used for its performance benefits and ease of use.

2.4. SOURCE CONTROL

- **Perforce (Helix Core, P4V)** is used for version control.
- **Perforce server**: `p4.howest.be:1666`
- **Art Artifacts Export**:
    - **3D models** should be structured properly before committing to Perforce.
- Changelist descriptions should be clear and concise, explaining the changes made in the files.
- For the folder structure refer to the **Folder Structure** section.

2.5. UNITS & METRICS

Metric system: **1 unit = 1 meter**

2.6. ART ARTIFACTS EXPORT

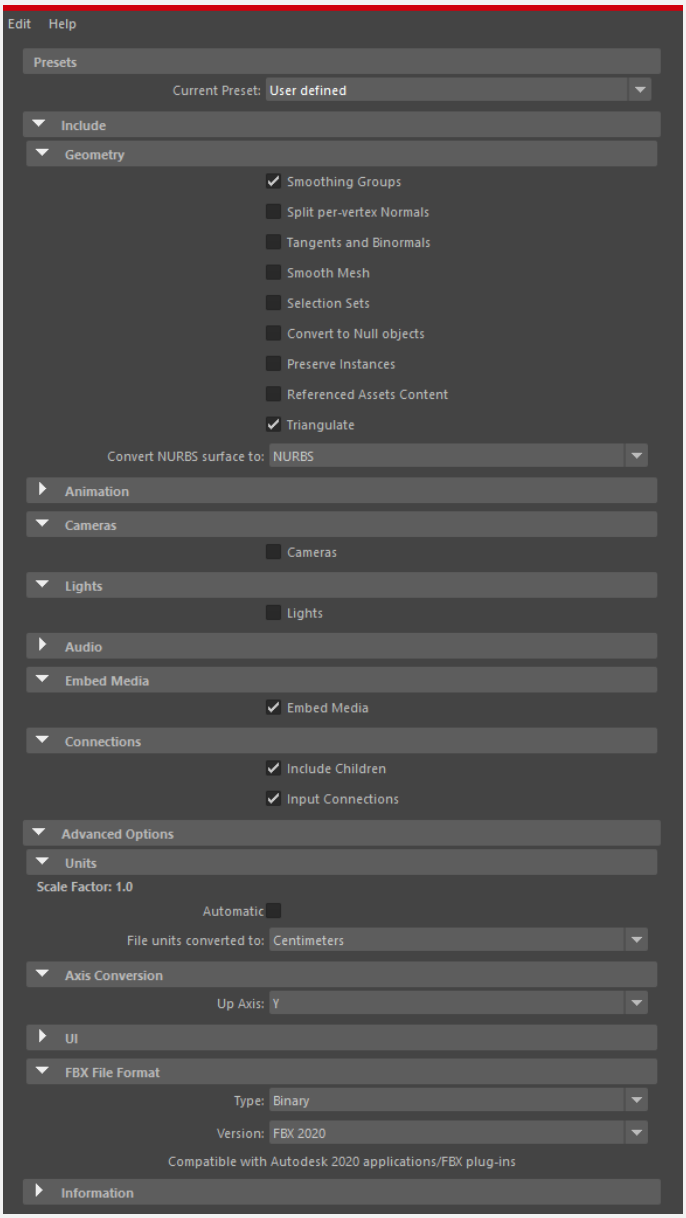
2.6.1. 3D Models

- Exclusively **.fbx binary** format following the right naming conventions.

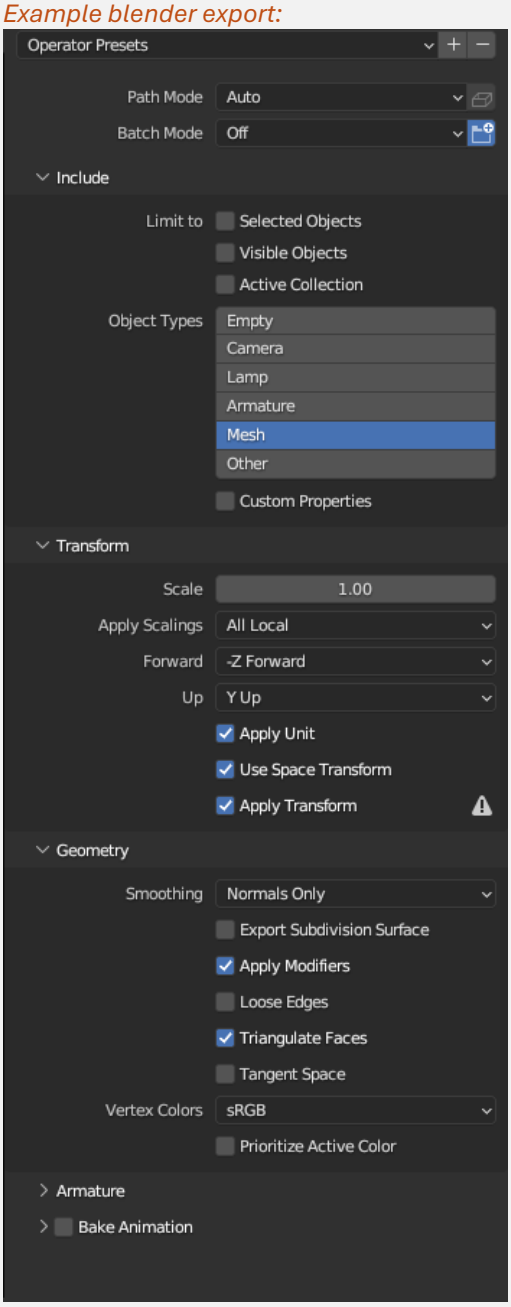
2.6.2. Static Meshes

- This is an example export template for static meshes (not rigged/animated)

Example Maya export:

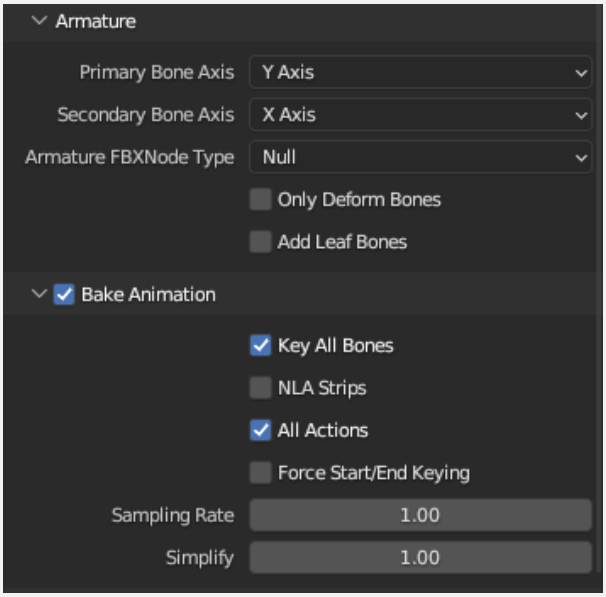


Example blender export:



2.6.3. Skinned Meshes

- We will use **blender** for rigging and animation. The extra export settings will be the following:



2.6.4. Textures

Photoshop export:

- We will export in 3 different sizes (these guidelines might change once we get to prototype some materials) :

`2048x2048` maps for **detailed meshes** -> characters, predominant environment elements

`1024x1024` maps for **mid detailed meshes** -> side background elements, additional props in game

`512x512` for **low detailed meshes** and **VFX** -> least relevant background elements, particle materials

- Textures will be saved in Photoshop .png on **HIGH** (2048x2048) resolution always.
- **Transparency UNCHECKED unless the texture is meant for a translucent material.**
- **The color space should always be sRGB**

The resize process should be done in the engine, so it can be easily manipulated if changes are implemented further down the production.

The compression settings in Unity can vary depending on the texture content. These are the options provided and their advantages:

Properties

Mitchell	Default high quality resize algorithm.
Bilinear	Might provide better result than Mitchell for some noise textures preserving more sharp details.

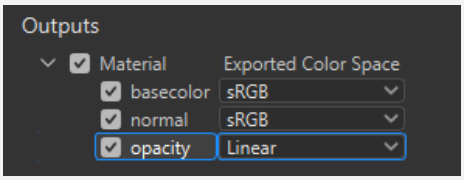
Substance painter export:

- We decided **against** using standard PBR maps as it would be unnecessary for the art direction we're going for.

The maps that we're going to be using are these 3:

- One RGBA map for **base color** (with baked AO) and eventual Opacity.
    - If the texture doesn't include opacity it can be saved as a **full black map**.
    - This is for easier handling with master materials.
- One RGB **normal** map.
    - The normal mapping used in Unity is OpenGL.
- One GRAYSCALE **emissive** map.
    - This will be exported **ONLY** if the texture needs an emissive map.

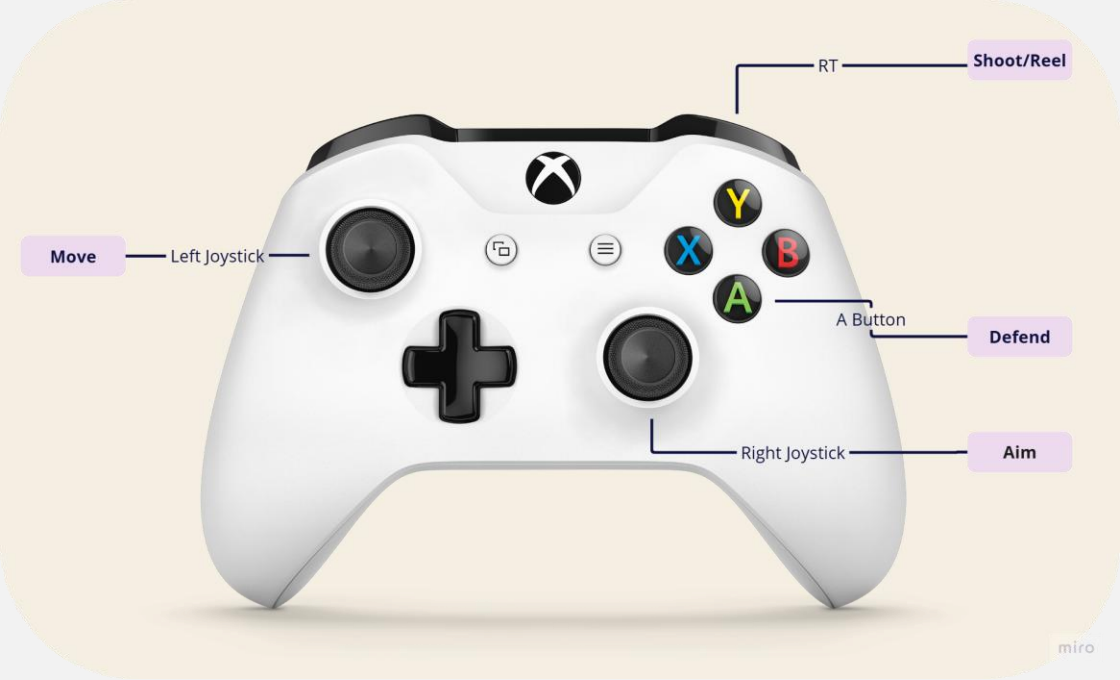
- The same conventions apply for **Substance Designer**. The texture packing can be handled in **photoshop**.



2.7. SOFTWARE & PROGRAMMING LANGUAGE

CATEGORY	TOOLS & SOFTWARE
Game Engine	Unity 6000.0.38f1 LTS
Programming Language	C#
Development Environment	VisualStudio 2022, Jetbrains Rider 2024
Modeling	Maya, Blender
Rigging & Animation	Blender
Materials & Textures	Substance Designer, Substance Painter
Concepting & Textures	Photoshop
Source Control	Perforce
Documentation	Markdown
Markdown editor	Jetbrains Rider 2024

2.8. PLAYER CONTROLS



3. PROGRAMMING GUIDELINES

Follow **Unity's C# Style Guide** for best practices. The most important rules are outlined below:

3.1. FORMATTING RULES

3.1.1. Indentation style

- Use **Allman brace style** for improved readability.

```
// GOOD EXAMPLE

void DisplayMouseCursor(bool showMouse)
{
     if (!showMouse)
     {
          Cursor.lockState = CursorLockMode.Locked;
          Cursor.visible = false;
     }
}

// BAD EXAMPLE

void DisplayMouseCursor(bool showMouse){
     if (!showMouse) {
          Cursor.lockState = CursorLockMode.Locked;
          Cursor.visible = false;
     }
}
```

- Always include **braces in nested multiline statements**.

```
// GOOD EXAMPLE

for (int i = 0; i < 10; i++)
{
    for (int j = 0; j < 10; j++)
     {
        ExampleAction();
     }
}

// BAD EXAMPLE

for (int i = 0; i < 10; i++)
    for (int j = 0; j < 10; j++)
        ExampleAction();
```

- Setup your editor so you can automatically format your code.

3.2. NAMING CONVENTIONS

More naming conventions and examples can be found on **Unity's C# Style Guide**. A small summary is listed below:

- **PascalCase** for most names (classes , public properties, public/private functions, namespaces).
- **_PascalCase** with an "I" prefix for interfaces. Example: `IExampleInterface`.
- **_camelCase** for private and protected member fields.
- **_camelCase** for private `[SerializeField]` fields.
- `[SerializeField]` GameObjects that hold prefabs should follow `<name>_prefab`. Example: `cannon_prefab`.

3.3. DOCUMENTATION

When implementing more complex systems that require an explanation, write clear comments explaining how the system works and why the system works the way it does.

When writing public properties and/or methods, make sure to write XML documentation for them. Even when the properties and methods may appear simple, to avoid confusion, write a small description about what the exact value represents.

Use descriptive names for methods and properties that require as little documentation as possible.

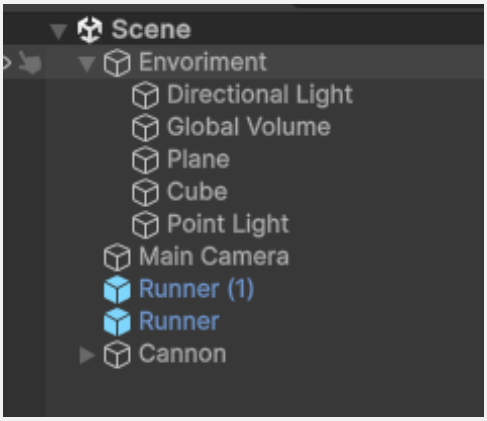
```
/// <summary>
/// Get the currently hooked object
/// </summary>
/// <returns>The object that is currently hooked, null if nothing is hooked</returns>
public GameObject GetHookedObject()
{
    // The target is set the moment the fishing line is cast,
    // however, this does not mean the target has been hooked yet.
    // So while the line is still casting, the target is still null.
    return _casting ? null : _target;
}

/// <summary>
/// Get the GameObject that is currently set as the target for hooking
/// </summary>
/// <returns>The GameObject that is targeted, can be null if no target has been set</
returns>
public GameObject GetCurrentTarget()
{
```

```
    return _target;
}
```

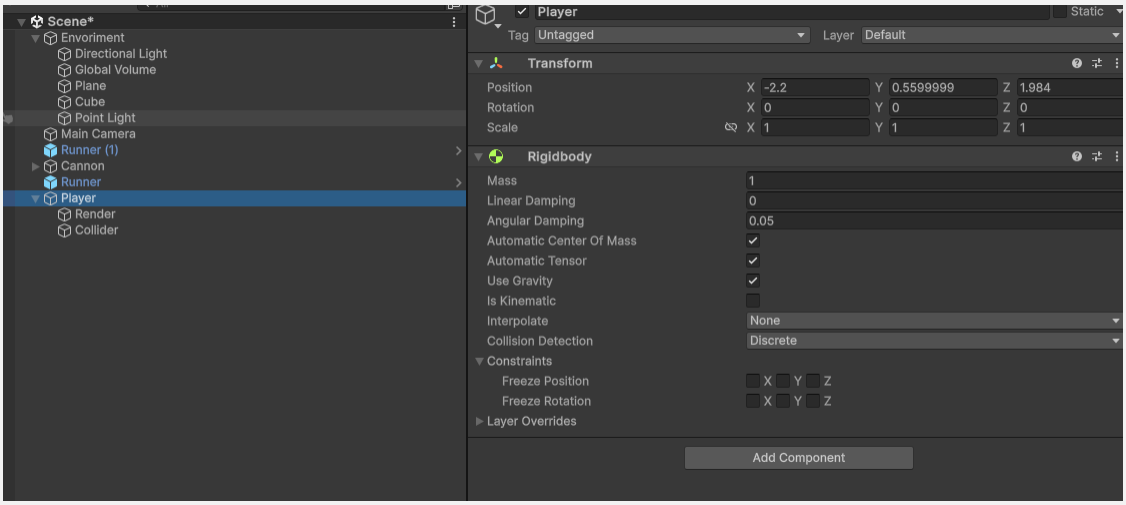
3.4. SCENE HIERARCHY

- All static environment props should be grouped under a **single GameObject called** `Environment`. This simplifies setting them as static. And makes sure the scene doesn't get cluttered



3.5. GAMEOBJECT SETUP

- **Separate rendering and collision components** within player objects. This makes it easier to disable colliders or rendering components individually, reducing clutter in the GameObject components.



3.6. DESIGN PATTERNS

- **State Machine**
  - The State Machine pattern is encouraged and preferred over multiple nested if-statements
- **Singleton**
  - Though controversial, the Singleton pattern works well within unity and can be used
  - It must be added that this does **NOT** mean that **everything should be a singleton**

4. FOLDER STRUCTURE

4.1. GENERAL STRUCTURE

- The first layer of folders is organized by **type** (e.g., Materials, Scenes, Models, Scripts, UI).
- Assets can be further grouped by **feature** within each category.

4.1.1. Perforce main folder structure

```
gamep_group16
    ArtAssets          - contains all source art assets used in
                         the game (NO exports, only source files)
    Design             - contains all design assets and documents
                         (art bible, tech doc, game design doc)
        Resources      - contains images and other resources
                         linked in the .md files
    Dev                - contains one unity project: the main
                         development folder used in the
                         production phase
    Main               - contains a stable version of the game -
                         receives weekly updates from the Dev
                         folder
    Prototype          - contains tests and experiments; will be
                         used mostly during the prototype phase
```

4.1.2. Unity project folder structure

```
Assets
    Scenes
        Levels
        Debug
    Materials
    Textures
    Scenes
    Prefabs
    Models
        Character
        Environment
        Props
```

- PowerUps
- Misc
- Input
- Scripts
  - Interfaces
- UI
  - Menu
  - HUD
  - Settings
- SFX
- RFX

4.1.3. Art assets folder structure

- ArtAssets
  - Materials
  - Textures
  - Scenes
  - Models
    - Character
    - Environment
    - Props
    - PowerUps
    - Misc
  - ConceptArt
    - Character
    - Environment
    - Props
    - PowerUps
    - Misc
  - UI
    - Menu
    - HUD
    - Settings

- `SFX`
- `RFX`

4.2. FILE & FOLDER NAMING CONVENTIONS

- All files and folders within **Perforce** should be named using **PascalCase**.
- **Avoid** spaces, underscores, or lowercase letters.
- Keep the names **short**.
- **Use underscore and postfixes (suffixes) to specify the file's type.**
    - Example: `Explosion_SFX.wav` for sound effects.
    - Example: `Barrel_Prop.blend` for 3D models.
- You don't need to include the file type in the postfix if it's already indicated by the file extension.
    - Example: don't use `MyAsset_PSD.psd`.

4.2.1. List of Postfixes

- **_SFX** for sound effects
- **_RFX** for realtime effects
- **_BGM** for background music
- **_UI** for UI elements (health bar, player icon, buttons, scores, etc.)
- **_Prop** for props (crates, furniture, etc.)
- **_Char** for character models (player, enemies, NPCs)
- **_Env** for non-interactable background environmental models
- **_Tex** for textures
    - **_Tex_BaseColorOpacity** for base maps
    - **_Tex_Normal** for normal maps
    - **_Tex_Emissive** for emissive maps
- **_Mat** for materials

5. TARGET PLATFORM

We chose **Windows** as the primary platform for Reel It In due to its **widespread accessibility** and **compatibility** with local multiplayer setups. Windows provides: - **Broad Hardware Support** – Most PCs can run Reel It In without requiring high-end specs, making it accessible to a large audience. - **Strong Controller Compatibility** – Windows supports a wide range of controllers, which is essential for a couch multiplayer game. - **Ease of Development & Distribution** – Unity offers excellent Windows support, and distributing the game via platforms like Itch.io is straightforward. - **Local Multiplayer Focus** – As a party game, Reel It In is best suited for PC setups with multiple controllers, allowing players to easily plug in and play together.

6. TARGET AUDIENCE

Inspired by party games like Mario Party and Smash Bros, Reel It In prioritizes **fun**, **unpredictability**, and **replayability**, making it ideal for casual and competitive play alike:

- **Casual Gamers & Party Game Fans** – People who enjoy quick, fun, and accessible multiplayer experiences
- **Local Multiplayer Enthusiasts** – Groups of friends or families looking for a competitive yet lighthearted couch co-op experience.
- **Competitive Players** – While the game is simple to pick up, it allows for strategic play through its grappling mechanic, making it appealing to players who enjoy skill-based interactions.
- **All Ages & Skill Levels** – With simple controls and intuitive mechanics, Reel It In is accessible to both beginners and experienced players, ensuring everyone can jump in and have fun regardless of gaming experience.

7. TARGET CONTEXT

Reel It In is designed as a fast-paced, chaotic 1v3 couch multiplayer game that thrives on quick, dynamic interactions. With a focus on simple controls and short play sessions, the game is easy to pick up but allows for creative and strategic play.

The reeling mechanic serves as the core gameplay element, enabling both cooperative and competitive interactions between players. The balance between helping or sabotaging teammates adds depth while keeping the experience accessible and fun.

The game does not aim for realistic physics but instead embraces a cartoonish and exaggerated movement style, using a Bezier curve-based grappling system for smooth and readable reeling mechanics. The game is built with local multiplayer in mind, ensuring responsive controls and clear visual feedback for an engaging couch co-op experience.

8. USEFUL LINKS

8.1. MIRO BOARD

8.2. HACKNPLAN

8.3. ITCH

8.4. DISCORD