Abstract

Ah My 2 Legs is a physics based, 2D platformer, where the players control a prisoner who lost his legs and must escape jail. They do this by launching themselves like a slingshot through levels, to collect a key and reach the end gate.

Ah my 2 Legs  
TECHNICAL DESIGN DOCUMENT

Developer: Daniel Sohler

Contents

[About 4](#_Toc75349083)

[Change Log 4](#_Toc75349084)

[Development Environment 5](#_Toc75349085)

[Software Requirements 5](#_Toc75349086)

[Libraries 5](#_Toc75349087)

[Version Control 6](#_Toc75349088)

[Repository 6](#_Toc75349089)

[Contributors 6](#_Toc75349090)

[Commit Message Format 6](#_Toc75349091)

[Game Overview 7](#_Toc75349092)

[Description 7](#_Toc75349093)

[Genre 7](#_Toc75349094)

[Perspective 7](#_Toc75349095)

[Target Platforms 7](#_Toc75349096)

[Windows / PC 7](#_Toc75349097)

[Windows Limitations 7](#_Toc75349098)

[Minimum PC Specs 7](#_Toc75349099)

[Feature List 7](#_Toc75349100)

[Game Flow & Structure 8](#_Toc75349101)

[Game Modes & Handling 8](#_Toc75349102)

[Game Mode - PrimaryMode 8](#_Toc75349103)

[Description 8](#_Toc75349104)

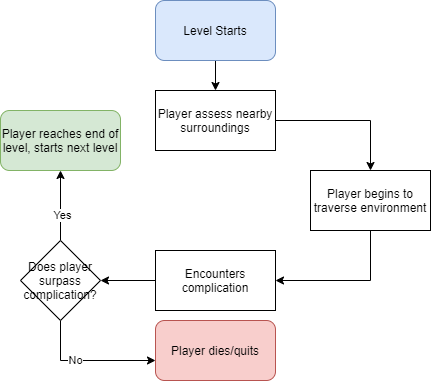
[Objectives 8](#_Toc75349105)

[Objective Tracking 8](#_Toc75349106)

[Mission / Level Structure 8](#_Toc75349107)

[Overview of structure 8](#_Toc75349108)

[Gameplay Loops 9](#_Toc75349109)

[ 9](#_Toc75349110)

[Gameplay Systems 10](#_Toc75349111)

[Controls / Input 10](#_Toc75349112)

[Keyboard / Mouse 10](#_Toc75349113)

[Controller 10](#_Toc75349114)

[Mobile / Touch 10](#_Toc75349115)

[Game Mechanics 10](#_Toc75349116)

[**Slingshot Player Controller** 10](#_Toc75349117)

[**Energy Meter** 10](#_Toc75349118)

[Custom Game Systems 11](#_Toc75349119)

[**Timer Script** 11](#_Toc75349120)

[**Line Draw Script** 11](#_Toc75349121)

[**Camera Follow Script** 11](#_Toc75349122)

[Physics 12](#_Toc75349123)

[Behaviours / AI <if applicable> 12](#_Toc75349124)

[Game Content 13](#_Toc75349125)

[Game Environment 13](#_Toc75349126)

[<Game Content Type X> 13](#_Toc75349127)

[List of <content type> 13](#_Toc75349128)

[Coding Standards 14](#_Toc75349129)

[Coding Standards - Details 14](#_Toc75349130)

[Naming Conventions 14](#_Toc75349131)

[Technical Goals & Risks 15](#_Toc75349132)

[Technical Goals: 15](#_Toc75349133)

[Technical Risks: 15](#_Toc75349134)

[Risk Avoidance: 15](#_Toc75349135)

[Appendix A – Technical choice justifications 16](#_Toc75349136)

[Choice of development engine: 16](#_Toc75349137)

[Choice of scripting language: 16](#_Toc75349138)

[Choice of third-party libraries and content: (if required) 16](#_Toc75349139)

# About

Ah my 2 Legs is a 2D physics-based platformer, where the player is a mutilated prisoner, using a slingshot to escape the prison. Each level they collect keys, while avoid dangerous obstacles, and use the key to open the exit door. The player must also manage an energy meter, which limits how much they can slingshot themselves.

The purpose of this document is to detail and record the development process of creation of scripts and functionality of the game. It also involves specific game information about its availability and practices ect. Version control, Coding Standards, technical guidelines.

# Change Log

Updates made to the document should be described below.

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Date of change | Description |
| 0.0.0 | AIE | DD/MM/202X | Initial Template created |
| 0.0.1 | Daniel Sohler | 16/06/2021 | Added information too ‘Abstract’, ‘About’ |
| 0.0.2 | Daniel Sohler | 23/06/2021 | Finished remaining template sections |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Development Environment

This section outlines the required software and systems required for development of this project.

## Software Requirements

The below table outlines the software requirements for development of this project. Developers contributing to the project are required to use the approved software outlined below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Software | Version | License | Used By | Used For |
| Unity 3D | 2019.3.6f1 | Education | Programmers, Designers, Artists (On Campus) | Development of Game |
| Unity 3D | 2019.3.6f1 | Free | Programmers, Designers, Artists  (At Home) | Development of Game |
| Visual Studios | 2019 | Free | Programmers | Development of Game |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Libraries

Unity/Unreal comes with a default collection of plugins, tools and assets. Its plausible, and often encouraged to pull in additional assets, tools, plugins or scripts etc. developed by a 3rd party. Identify both engine and system libraries used in the project, and especially any 3rd party ones used, including licensing information on its usage.

|  |  |  |
| --- | --- | --- |
| Asset/Library/Package name | License | Used For |
| UnityEngine | Unity | Interacting with Unity elements |
| UnityEngine.UI | Unity | Interacting with UI elements |
| System | Microsoft | Calling system time for timer |
| ProBuilder | Unity | Building game environments |
|  |  |  |

## Version Control

### Repository

<https://github.com/DanielSohler/Game-Prototyping_Fundamental.git>

### Contributors

* Daniel Sohler ([DanielSohler on Git](https://github.com/DanielSohler))

## Commit Message Format

Standard commit message will include the following information:

Please update, add or remove any details that you identify as important to include in commit message.

* **Type**: Represents the type of change, often the “Type” can be inferred based on the associated ticket in your project management tool, which may include: FIX, FEATURE, REFACTOR, DOC, TEST etc.
* **Scope:** Refers to the area of the project being changed, could refer to things like (menu) (inventory) (save\_system) (level) (controls) etc. Scopes may change throughout development but can be broadly identified. Outline the scopes below that seem suitable for your project
  + MENU
  + LEVEL
  + **…**
* **TaskId:** Id of the associated ticket representing the change.
* **Summary:** A short description of what has been changed.

**Format:**

|  |
| --- |
| Type (scope) : TaskId : Summary |

**Examples:**

|  |
| --- |
| Feature (menu) : #1302 : Added Exit button to main menu |
| Fix (menu) : #1395 : Updated button prefab with so that hover works on web builds |
| Feature (sandbox) : #1129 : Added rock asset to test scene, Created Rock prefab |
| <add your own examples> |
| <add your own examples> |
| <add your own examples> |

# Game Overview

## Description

A 2.5d platformer, where the player (with no legs) slingshots themselves around to reach the end. Trying to escape from a prison.

## Genre

The genre it would be is ‘2.5D Physics Platformer’.

## Perspective

The game’s perspective would be ‘2.5D Sidescrolling’.

## Target Platforms

This project will be deployed to the following platforms:

* Windows / PC

## Windows / PC

### Windows Limitations

* Required Inputs are mouse only, any non-mouse users will be unable to play.
* Performance constraints vary depending on the users computer specs.

### Minimum PC Specs

* OS: Windows 7
* Processor” Intel Core i7-4770 CPU @ 3.40GHZ, 3401 Mhz, 4 core
* Graphics Card: GeForce 910m
* Memory: 4gb Ram

## Feature List

The project’s features include:

* Physics / impulse based movement
* System based time calculate (not fully delta-time focused)

# Game Flow & Structure

This section of the document outlines the high-level structure and order of play for the project.

Game Modes & Handling  
The game uses a single game mode, being the play mode. Where the player interacts in a 2.5d level with a sidescrolling viewmode. Modes are not selected; they are just a scene being loaded.

* Primary Mode

Game Mode - PrimaryMode

Description   
The player character will be in the middle of the screen, and the player must maneuver them to an ‘end zone’ of the level (yellow area).

Objectives   
Reach the end of the level, in the shortest time possible, with the lowest number of inputs.

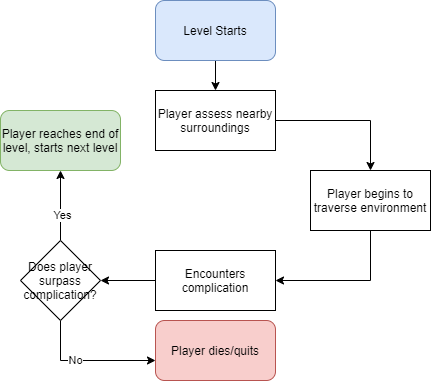
Objective Tracking   
A script will be tracking the time spent in the level at the top right. Below it will be a second script which tracks total player inputs (connected to player controller). Once the player reaches the end zone, the timer stops, and a UI screen will display to the player.

Mission / Level Structure

Overview of structure

* Player enters level at starting zone.
* Player attempts to traverse the level.
* Player reaches end of level.

Gameplay Loops   
  
Primary Gameplay Loop   
This loop encompasses the entire gameplay experience, form starting a level to reaching the end.



# Gameplay Systems

This section of the document provides specifications for the systems that drive the game.

## Controls / Input

The primary control scheme will be entirely mouse controls. Focusing on clicking, dragging, and releasing the mouse. While menus would be interacted in the same way.

### Keyboard / Mouse

With the game being entirely focused on mouse controls, the following inputs will be utilized for gameplay: Clicking, holding the mouse down, releasing the mouse after holding it down, scroll wheel button.

### Controller

N/A

### Mobile / Touch

N/A

## Game Mechanics

All gameplay mechanics are to be covered here. You may format information as you like, but ensure you cover the info for the bullet points listed under the examples below. Keep information concise – This is reference material, not justification, etc.

### **Slingshot Player Controller**

* The ‘Slingshot Player Controller’ is about allowing the player to launch themselves in the direction of their mouse, relative to the player character. This is used to ‘launch’ the character around the level, instead of traditional walking controls.
* How this mechanic will work is by tracking the mouse position on the screen (not relative to the game-world), and the player position (relative to the game-world). When the mouse is initially held down, it will begin to draw a line, and calculate every frame the distance from the player the mouse is. If its too far or too close, nothing will happen when the controls are released. If in a valid position, it will launch the player with an amount of velocity equal to the distance from the player.
* The Inputs would just be either clicking for a quick-launch or dragging and releasing the mouse for a more accurate launch.
* The output would be the value that determines the launch velocity. This is used to consumer meter, to initiate the timer on levels, and to track the amount of player inputs in a level.

### **Energy Meter**

* This meter is a limiter for the player, that is supposed to prevent them from spamming the Slingshot mechanics. It is always visible on the screen, but in a way that doesn’t obstruct player information.
* When the player launches themselves with the Slingshot Player Controller, the output of the launch velocity is multiplied by 2, and then used to subtract from the current amount of meter. When this occurs, the meter will then recharge up to max. Its recharge can be interrupted with another launch. If the meter is consumed too much, and a launch variable would exceed the current amount of meter, the meter prevents launching until it has 30% of its maximum value. After this it will allow the player to launch.
* Mouse input relating to the player controller is the only input.
* Its output is purely visual, giving the player an idea of how much meter is consumed, is regenerating, and if its available/blocked.

## Custom Game Systems

### **Timer Script**

* Tracks amount of time player has spent interacting with level. Can start/stop are based on a bool var that is affected by launching + reaching an end zone. The timer is only in minutes and seconds.
* The script only activates when the player initially launches themselves. From there it counts upwards. It will continue to count upwards until the player reaches an ‘end-zone’, where the collision has a trigger that will pause the timer when interacted with.
* The initial launch is one input, with the second one being colliding with the ‘end-zone’. Both start/stop the timer.
* The output would be visual, with the timer being shown in the top right of the screen, counting upwards.

### **Line Draw Script**

* This script just draws a line between the player character and the mouses current position on the screen. It will be blue if it is considered valid to launch, and red if it isn’t.
* The line draw will be called when mouse is initially held down in ‘Slingshot Player Controller’. It will draw a line, with the start being at the player, and the end being at the mouse. It will be coloured red if it is too close/too far from the player and disable launching. When the player launches successfully, the line will stop being drawn.
* When mouse is initially held down and let go are this system’s inputs.
* The output would be a line being draw between the center of the player and the mouses current position on screen. The colour being dictated by the validity of launching. And stop drawing when successfully/unsuccessfully launching the player.

### **Camera Follow Script**

* This script just makes the camera follow the player character smoothly.
* The camera’s transform is set to the player character, with a lerp attached to the following mechanic to make it smoother. It will not be able to rotate and is a fixed distance from the player at any time.
* The only input would be when the player moves in general, voluntary or not.
* The output would be the camera following the player with a slight delay.

## 

## Physics

The physics of this game is based on Unity’s rigidbody system. It has pre-determined gravity, mass, drag, friction and momentum. The game’s system work around it more than determining it.

## Behaviours / AI <if applicable>

N/A

# Game Content

This section of the document covers content types for the project and provides technical specifications on their usage.

## Game Environment

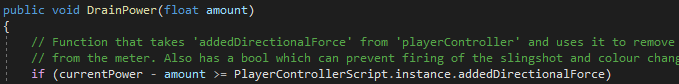
The game takes place throughout multiple environments, that are isolated from each other. They are intended to have a focus on long stretches where the slingshot controls an be used fully. They are 3d, but are viewed in a 2.5D manner. This allows to remove faces and polygons from objects not facing the screen, saving space.

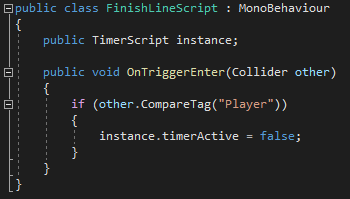
## Game-Level

* Physical environment the player interacts/traverses.
* (ProjectTitle)/Assets/Scenes
* A unity Scene file, requires a .meta, .lighting and .lighting.meta file alongside each scene.

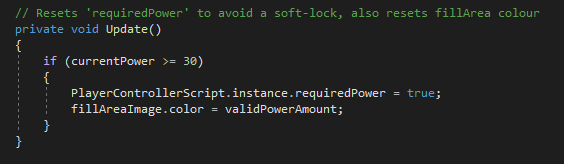
# Coding Standards

## Coding Standards - Details

* When commenting, I try to detail in advance what a function/statement will perform. This is to help with retrospective researching of my code in the future.   
  
* I try and keep everything separated for easy readability, so nested bracks will always sit on their own lines.



* I try to avoid using functions or statements within ‘Update’, since they call said functions every frame, it would be heavy on the CPU. Fixed Update is a slightly better alternative.



## Naming Conventions

* I use camelCase for naming variables, and PascalCase for functions. This just helps when calling certain variables that might have similar names.
* I prefer to start variables with a verb, this helps with reminding myself what each variable performs briefly (i.e addedDirectionForce, has been calculated prior, so when I call it Im aware of its prior functions).
* Avoiding using any terms that might refer to the script name in variables. This is more to prevent general confusion when trying to call other scripts,

# Technical Goals & Risks

* Maintaining 30fps, intended for low-spec computers.
* Having controls be independent from frame rate.
* Having clear and concise mechanics that require little explanation.

## Technical Goals:

* Creating a movement system relying on mouse position relative to the screen, and player position relative to the world.
* Having systems and controls that require little explanation, as to streamline the game experience.

## Technical Risks:

* This is my first attempt at coding a game from the ground up. I lack the necessary experience to ensure everything functions as expected.
* Having to make sure the systems and code can function well for low spec computers.

## Risk Avoidance:

* Keeping goals low enough for me to reach, with room to raise their difficulty.
* Having scripts avoid using CPU heavy practices (i.e using Update to house every function, updating it every frame).

# Appendix A – Technical choice justifications

## Choice of development engine:

* I chose the Unity engine mainly because of my prior education and experience with the engine.
* The game I wanted to make is a 2.5d physics platformer, with the pre-built rigidbody system within Unity, this makes my work easier to compile.
* The Unity also has a large online presence of indie developers, who have attempted similar projects to me. So, I have a large database of prior attempts I can learn/iterate upon.

## Choice of scripting language:

* I chose C# due to its accessibility, and my prior experience in using the programming language for simple games when I was younger.
* C# Tends to be more object-oriented, which works well for my game concept, which would require attaching various modular scripts to multiple in-game objects.

## Choice of third-party libraries and content:

* I decided to also use ‘ProBuilder’, as it allows me to easily create a game level, which I can gauge and fine tune the controls in my game, in a physical environment.