# **SCRIPT DESIGN - Ah My 2 Legs**

**By:** Daniel Sohler

Contents

[SCRIPT DESIGN – Ah My 2 Legs 1](#_Toc69075429)

[Document overview 4](#_Toc69075430)

[Project Script Requirements 4](#_Toc69075431)

[Description 4](#_Toc69075432)

[Feature & Mechanics List 4](#_Toc69075433)

[Key Scripts to Design 4](#_Toc69075434)

[Script Design 5](#_Toc69075435)

[Script 1 - <Name of script/function> 5](#_Toc69075436)

[Required Functionality & Outcomes 5](#_Toc69075437)

[Pseduocode 5](#_Toc69075438)

[Flowchart 5](#_Toc69075439)

[Script 1 Plan feedback 5](#_Toc69075440)

[Pseudocode feedback notes 5](#_Toc69075441)

[Flowchart feedback notes 5](#_Toc69075442)

[Script 1 Revised Plans 5](#_Toc69075443)

[Final pseudocode 5](#_Toc69075444)

[Final flowchart 6](#_Toc69075445)

[Script 2 - <Name of GUI script/function> 6](#_Toc69075446)

[Required Functionality & Outcomes 6](#_Toc69075447)

[Pseduocode 6](#_Toc69075448)

[Flowchart 6](#_Toc69075449)

[Script 2 Plan feedback 6](#_Toc69075450)

[Pseudocode feedback notes 6](#_Toc69075451)

[Flowchart feedback notes 6](#_Toc69075452)

[Script 2 Revised Plans 6](#_Toc69075453)

[Final pseudocode 6](#_Toc69075454)

[Final flowchart 7](#_Toc69075455)

[Script Implementation & Iteration 7](#_Toc69075456)

[Script 1 - <Name of script/function> 7](#_Toc69075457)

[Script(s) generated 7](#_Toc69075458)

[Functionality review 7](#_Toc69075459)

[Implementation feedback notes 7](#_Toc69075460)

[Response to feedback notes 7](#_Toc69075461)

[Script 2 - <Name of GUI script/function> 8](#_Toc69075462)

[Script(s) generated 8](#_Toc69075463)

[Functionality review 8](#_Toc69075464)

[Implementation feedback notes 8](#_Toc69075465)

[Response to feedback notes 8](#_Toc69075466)

Document overview

The purpose of this document is to show the planning, design, implementation of at least two scripts in a single project. The document is broken up into phases to go through in order.

Project Script Requirements

### Description

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.

### Feature & Mechanics List

* **Slingshot Controls** – Main player controller, slings player around level.
* **Energy Meter** – Consumed when player slings themselves, manageable resource.
* **Key Inventory** – Holds keys used to unlock doors in levels.
* **Hazardous floors/walls** – Prevents player from easily manoeuvring terrain.

### Key Scripts to Design

* **Script 1** - Player Controller (Line trajectory script, Camera follow script, Timer script
* **Script 2** - Energy Meter – GUI script

## Script Design

IN THIS SECTION YOU ARE SHOWING YOU CAN ACTUALLY PLAN A SCRIPT FROM NEEDS, AND PROVIDE EVIDENCE YOU ARE TRYING TO IMPROVE THESE OUTCOMES.

### Script 1 - Player Controller

#### Required Functionality & Outcomes

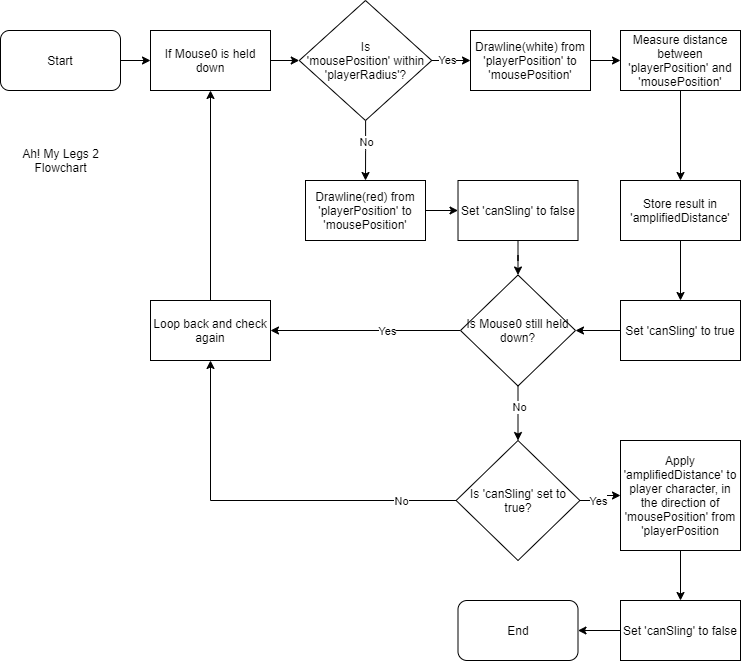
In this script, the controller needs to be able to track the mouse cursors position, and the player’s position. Using these two transforms, they will be used to calculate launch direction and distance. It will also prevent the player from launching if certain parameters aren’t met (right distance, available meter usage).

* Tracks when mouse is initially held down, constantly held down, and let go
* Tracks where on the screen the mouse is, and if its within a ‘safe’ radius of the player
* Draws a line from the player to the cursor to represent launch trajectory
* Prevents a launch if parameters aren’t met

#### Pseudocode

|  |
| --- |
| - Start - If GetMouse0Down is true, then  - If mousePosition is within playerRadius, then  - Drawline(white) from playerPosition to mousePosition, updating every frame  - Calculate distance between playerPosition and mousePosition into store into amplifiedDistance & update value every frame that GetMouse0Down is true.  - Set canSling to true   - else if mousePosition isn’t within playerRadius, then  - Drawline(Red) from playerPosition to mousePosition, updating every frame  - Set canSling to false - If GetMouse0Down was true, and is false and canSling is true   - If canSling is true, then  - Set slingshotFired to true  - Apply amplifiedDistance to player character, in the direction of mousePosition from playerPosition  - Set canSlign to false  - Set slingshotFired to false |

#### Flowchart



### Script 1 Plan feedback

#### Pseudocode feedback notes

* Bools don’t account for ‘was and is’, a separate Boolean would be needed.
* Add a check to see if the player properly releases the button.
* Reads too much like actual code than pseudocode.

#### Flowchart feedback notes

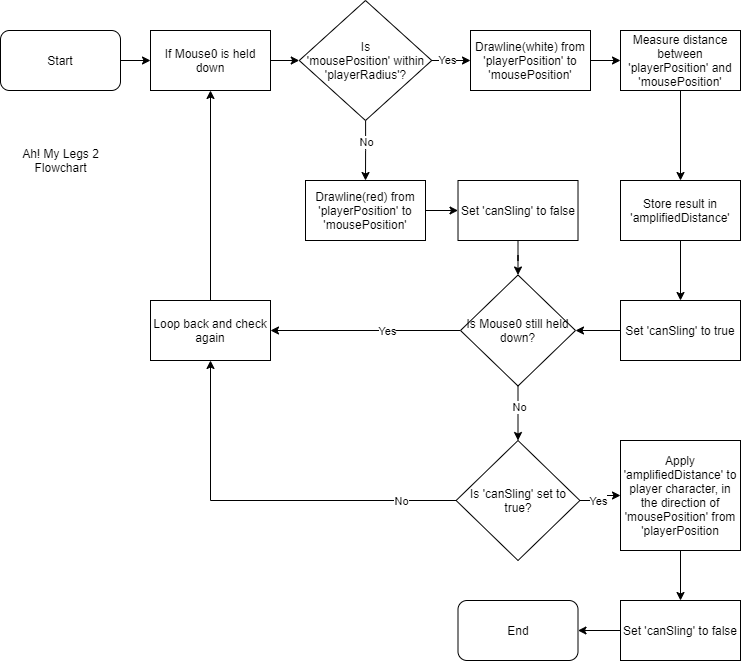
* N/A
* N/A
* N/A

### Script 1 Revised Plans

#### Final pseudocode

|  |
| --- |
| - Start - If GetMouse0Down is true, then  - Set mouseHeld to true,  - If mousePosition is within playerRadius, then  - Drawline(white) from playerPosition to mousePosition, updating every frame  - Calculate distance between playerPosition and mousePosition into store into amplifiedDistance & update value every frame that GetMouse0Down is true.  - Set canSling to true   - else if mousePosition isn’t within playerRadius, then  - Drawline(Red) from playerPosition to mousePosition, updating every frame  - Set canSling to false - If GetMouse0Down is false, mouseHeld is true, and canSling is true   - If canSling is true, then  - Set slingshotFired to true  - Apply amplifiedDistance to player character, in the direction of mousePosition from playerPosition  - Set canSlign to false  - Set slingshotFired to false  - Set mouseHeld to false. |

#### Final flowchart



### Script 2 - Energy Meter

#### Required Functionality & Outcomes

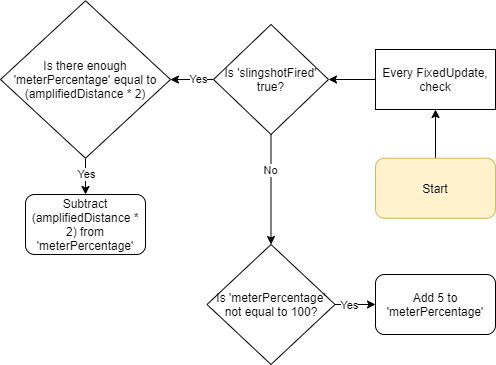
This meter located at the bottom-middle of the screen. This meter will be full at the start of the game. When the player launches themselves with the slingshot, the meter will have some percentage consumed.  
The meter will constantly refill itself over time at a slow rate.  
The meter also won’t be consumed if certain prerequisites aren’t met (available energy, canSling == true ect.)

* Constantly refils if not at a certain value (100 ect.)
* Prevents player controller from launching if lacking meter percentage.
* Consumes meter percentage when launched (correlating with launch velocity)

#### Pseudocode

|  |
| --- |
| - start - on FixedUpdate - If ‘slingshotFired’ is true, then  - If there is enough meterPercentage equal to (amplifiedDistance \* 2)?, then  - Subtract (amplifiedDistance \* 2) from meterPercentage  - If there isn’t enough meterPercentage equal to (amplifiedDistance \* 2), then  - Terminate  - if false, then  - If meterPercentage isn’t equal to 100, then  - Add 5 too meterPercentage every second  - if meterPercentage is equal to 100, then  - Terminate |

#### Flowchart



### Script 2 Plan feedback

#### Pseudocode feedback notes

* There are two decision points that lack a ‘no’ scenario.
* N/A
* N/A

#### Flowchart feedback notes

* N/A
* N/A
* N/A

### Script 2 Revised Plans

#### Final pseudocode

|  |
| --- |
| - start - on FixedUpdate (once every second instead of framerate) - If ‘slingshotFired’ is true, then  - If there is enough meterPercentage equal to (amplifiedDistance \* 2)?, then  - Subtract (amplifiedDistance \* 2) from meterPercentage  - If there isn’t enough meterPercentage equal to (amplifiedDistance \* 2), then  - Terminate  - if false, then  - If meterPercentage isn’t equal to 100, then  - Add 5 too meterPercentage every second  - if meterPercentage is equal to 100, then  - Terminate |

#### Final flowchart

SHOW THE FINAL PSEUDOCDE HERE.

## Script Implementation & Iteration

IN THIS SECTION YOU ARE SHOWING THE OUTCOME OF YOUR ATTEMPTS TO CODE THESE THINGS. SHOW YOUR FIRST WORKING OUTCOME THE

### Script 1 - <Name of script/function>

#### Script(s) generated

SHOW THE CODE THAT WAS GENERATED HERE (PASTE ENTIRE CODE OF SCRIPT)– IT NEEDS TO HAVE AT LEAST MOSTLY ACHIEVED THE GOAL. IF THIS IS OVER MULTIPLE SCRIPT FILES IN IMPLEMENTATION, SHOW BOTH AND MAKE SOME BULLET NOTES ON HOW THE SCRIPTS INTERACT.

#### Functionality review

<State if the script functioned as desired, and also if there were any issues or shortcomings apparent to the developer. Bullet points if this becomes a list.

#### Implementation feedback notes

AFTER SHOWING THE OUTCOME TO OTHERS, COLLECT FEEDBACK NOTES AND LIST THEM HERE

* <First note>
* <Second note>
* <Third note>
* <etc>

#### Response to feedback notes

START BY LISTING YOUR RESPONSES AND THE REASONING BEHIND THEM. THEN PASTE THE FINAL SCRIPT(S) BELOW.

* <First response (“Did X to achieve Y”)>
* <Second response>
* <etc>

FINAL SCRIPT(S) PASTESD HERE .

### Script 2 - <Name of GUI script/function>

#### Script(s) generated

SHOW THE CODE THAT WAS GENERATED HERE (PASTE ENTIRE CODE OF SCRIPT)– IT NEEDS TO HAVE AT LEAST MOSTLY ACHIEVED THE GOAL. IF THIS IS OVER MULTIPLE SCRIPT FILES IN IMPLEMENTATION, SHOW BOTH AND MAKE SOME BULLET NOTES ON HOW THE SCRIPTS INTERACT.

#### Functionality review

<State if the script functioned as desired, and also if there were any issues or shortcomings apparent to the developer. Bullet points if this becomes a list.

#### Implementation feedback notes

AFTER SHOWING THE OUTCOME TO OTHERS, COLLECT FEEDBACK NOTES AND LIST THEM HERE

* <First note>
* <Second note>
* <Third note>
* <etc>

#### Response to feedback notes

START BY LISTING YOUR RESPONSES AND THE REASONING BEHIND THEM. THEN PASTE THE FINAL SCRIPT(S) BELOW.

* <First response (“Did X to achieve Y”)>
* <Second response>
* <etc>

FINAL SCRIPT(S) PASTESD HERE .