Imperial College London

OPEN SESAME

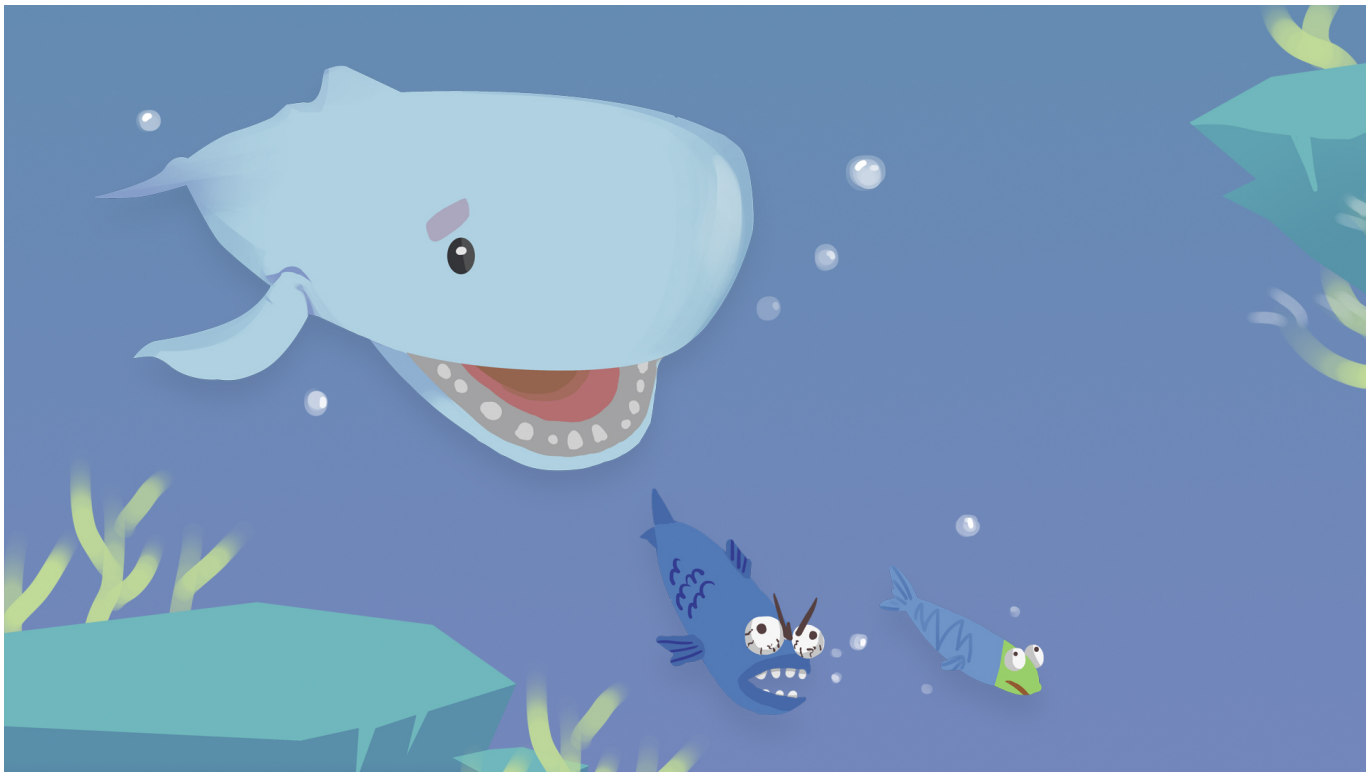
Department of Biomedical Engineering

HCARD Group 3 Final Report

Open Sesame

Authors:
Federica Spinola, Renke Huang, Ziliang Song,
Florence Wu, Junke Yao, Rui Zhou

Date:
April 9, 2019



1 Introduction

Cerebral Palsy (CP) is a very common neurological disorder, with over 3 in $2\ 000$ babies being diagnosed each year worldwide [1]. The hip is a complex ball-and-socket joint, which primary function is to support the body's weight and to perform the movement of the upper legs. The majority of patient with CP develop spasticity. About 35% of CP patients suffer from hips displacement [2], which is caused by the involuntary tightening of the muscle. This places an abnormal force around the hip joint and the neighbour muscles, which could lead to the dislocation of the hip [3]. 1 in 3 patients are affected by hip dislocation and some may need surgery, yet little effective interventions have been developed for lower limb rehabilitation despite its potential of prevention [4]. Conventional rehabilitation of the hip requires the assistance of a specialised physiotherapist and thus is usually carried out only few times a week due to high cost and non-motivating nature of the exercises. In this project a low-cost device with an attractive and fun game interface is designed and developed to help improve lower-limb mobility by guiding the training of the patient's hip joint rotation. The following report describes and discusses in detail the mechanical, the electronic and the software components of the full system, which can be viewed in Figure 1 in Appendix A. All components described in this report were used due to their simplicity, ease of procurement and budget within the scope of this project.

2 Mechanical Design

Open Sesame is a device designed for children with CP aged between four and ten years old. A child has approximately a femur length of 246 to $368\ \mathrm{mm}$ [5], a foot length of 170 to $210\ \mathrm{mm}$ [6] and a foot width of 65 to $79\ \mathrm{mm}$ [7] for a 4 to 10 year old. These mean dimensions are used as constraints for the mechanical design, which features are detailed below.

Rotating Arms

The rotating arms transfer the patients hip joint motion at one end to the potentiometer on the other end. It needs to be robust to sustain the patient's weight and light. To accommodate to the range of the consumers, the design of the arms had to be adaptable. In the prototype, this criteria was achieved by using steel drawer rails that could withstand a maximum load of $45\ \mathrm{kg}$ with a minimum and maximum length of $407\ \mathrm{mm}$ and $800\ \mathrm{mm}$ respectively. These lengths were restricted by both the target audience and foot holder attachment.

A clamp mechanism was implemented to secure the correct length for the consumer. Two $10\ \mathrm{mm}$ x $80\ \mathrm{mm}$ x $45\ \mathrm{mm}$ pieces of ply wood are placed on the top and bottom surfaces of the arm. One end of the clamp is fixed with a M4 nut and bolt assembly, whereas the other end is attached with a quick-release mechanism that facilitates the adjustment of the arm's length. The reader can refer to Figure 2 in Appendix B to view the rotating arm and clamp assembly.

A mechanical fin was designed to connect the rotating arm to the potentiometer to enable an angle measurement which would be inputted into unity by an electrical signal. The fin is fixed at the pivot point of the arms, using cable ties as illustrated in Figure 5 in Appendix B. The detailed presentation of this design process is shown in Figure 4 in Appendix B, with the final design dimensions illustrated in Figure 7 in Appendix B.

Support Structures

Two turn tables of 160 mm in diameter with ball bearings in an internal circular track were used as a pivot point for the rotating arms, as a guided track for the motion required for therapy. The details of the turntables and how the arms are fixed to them by screws are presented in Figure 3 in Appendix B.

The support base of the mechanical device must be stable enough to withstand the forces exerted by users' feet during the movement. The level of stability can be enhanced by either increasing the contact area of the support base or lowering the position of the centre of gravity [8]. Based on the mentioned principles and the resources available, a television stand is used to support the system at one end as shown in Figure 6 in Appendix B.

Foot Holder Design

The foot holder was designed to include the minimum and maximum foot measurements of our target audience as listed above. The design is shown in Figure 8 in Appendix B. For

the sole of the foot holder, the outline was made by eye based on a standard insole insert [9]. An indented platform following the foot shape of depth of $5\ \mathrm{mm}$, approximately $10\ \mathrm{mm}$ inwards from the original outline was formed to insert padding on the bottom of the sole. Two holes of $15\ \mathrm{mm}$ in diameter, corresponding to the screw holes pre-drilled within the arm, were formed to connect the foot holders to the arms. The ankle support of the foot holder was designed to follow a general curve of the calf for a child, with a wider radius of $55\ \mathrm{mm}$ than the standard calf radius at the top, to allow padding for the child's comfort. All connections within the foot holder were filleted to allow extra strength and comfort. The design was then 3D printed using Ultimaker 2+ and Cura software with the printing parameters stated in Table 1 in Appendix B.

Assembly and Mobile Platforms

The foot holders were attached via bolts and nuts to the arm prior to the assembly of the padding. The parts were then attached to mobile platforms. The platforms were made from ply wood of $150\ \mathrm{mm} \times 150\ \mathrm{mm} \times 40\ \mathrm{mm}$ which were mechanical sawed and filed to have rounded corners to minimise the hazards done during the risk assessment. It should be noted that all screws within the device were Type B to reduce hazards. Nine $5\ \mathrm{mm}$ and $3\ \mathrm{mm}$ holes were drilled in the mobile platform to allow the attachment of the arm and two castor wheels as seen in Figure 9 in Appendix B. The castor wheels were chosen to be $70\ \mathrm{mm}$ in height to align with the turn table, to ensure the arm to be level when the foot is placed in the foot holder. The wheels also had brakes to allow one to choose which leg one wanted to train. This was a feature such that, if only one leg was chosen, both legs would still remain at the same height to allow the correct hip placement whilst performing the exercise.

Additional Features

Padding was made from insoles and wound dressings bought and cut to fit the outline of the foot holder. These were attached to the foot holder via super glue. The inner sole padding had holes aligning with the screw holes to allow the surface of the foot holder and screw head to be of the same height. The outer sole padding was attached in the centre only to allow access to the screws for removal and maintenance.

Elastic bands of different elasticities were purchased to be attached to the mobile platforms when one leg mode was performed. This allowed a progressive training programme for the patients to increase the inner-leg muscle strength, which in turn would benefit the control of their hip-joint rotation.

3 Electronics Design

For the whale's movement in the game, three variables are acquired by the electronic system and sent in parallel at each time step. The variables are the two angles of rotation of the patient's legs that control the position and jaw of the whale, and a binary signal that controls the whale's water spray. Two potentiometers were used to detect the rotation of the patient's legs. The three lugs of the potentiometer correspond, from left to right, to power input, signal output and ground, as shown Figure 10 in Appendix C. Rotating the upper part of the potentiometer alters the resistance between the signal output and the power input as shown in Figure 10(middle) in Appendix C. An Arduino M0 board is used to monitor the potentiometer's output as an analog reading in the range of 0 to 1023 from its serial ports A0 and A1, and digitise it. The potentiometer has a rotation range of 270 degrees, thus the potentiometer's output signal, N, can easily be converted into an angle, θ, in degrees using Equation 1.

$$\theta = \frac{270 \times \mathrm{N}}{1023} \tag{1}$$

The potentiometers are screwed in different orientations as the patient's left and right leg rotate anticlockwise and clockwise, respectively. When the two legs are in their initial position (closed and centred), the potentiometer should be at its 0 position. However, since the legs do not have the limitation of the centre position, unlike the potentiometer, an offset was used to counteract this and avoid mechanical damaging. The offsets for two potentiometers were tested and calculated in Arduino to get actual leg angles, shown in Equation 2, where Reference and Actual each indicates θ with and without offset.

$$\theta_{left}^{actual} = \theta_{left}^{reference} - 49 \quad \theta_{right}^{actual} = 229 - \theta_{right}^{reference} \tag{2}$$

The game version requiring both legs; the movement of the right leg corresponds to the opening of the whale's jaw, i.e.

open and close and the movement of the left leg corresponds the whale's position, i.e. up or down. The whale can be in either one of two positions. It will descend towards the bottom position when the angle is larger than a threshold and ascend to the top position when the angle is smaller than another threshold. The difference between the two thresholds was implemented to avoid cheating by swinging the leg across a small range. The whale's jaw angle experiences an error that fluctuates between 0 and 0.5 at steady state and accumulates with time. A move average filter was applied to minimise this fluctuation and avoid error accumulation. The whale's spraying action is triggered by a button. The button is connected to the Arduino's M0 digital input pin. The serial monitor reads a 1 when the button is pressed and a 0 when it is released. The entire electronic connections to the Arduino board presented in Figure 10(right) in Appendix C.

After collecting the signals in Arduino, these need to be sent to Unity to play the game. The signals are printed in a line and separated by comma. In Unity, $System.IO.Ports$ is used to receive strings printed in the port. The data in the serial port is read and split, according to the comma, into a string array containing three strings (left and right potentiometer, and button). The string array is converted to a float type to use them as float numbers.

4 Game Design

The Open Sesame game is designed and built using Unity, and the detailed documentation can be found here: `https://open-sesame-docs.readthedocs.io/en/latest/software_ui/player_control.html`.

Player Control

The whale within the game was designed in two parts, the body, a non-interactive static animation, and the jaw, a part that rotates according to the angle value received. A calibration is initially performed that maps the patient's maximum leg opening range to the whale's maximum jaw aperture of 60 degrees.

To encompass the stationary position of the whale with both independent moving body and jaw parts, the two Prefabs have been wrapped in a parent $GameObject$ of Whale and the script has been executed onto the parent object as well. The implementation of the whale's body movement starts with defining the three plausible states the whale could be in: moving up, moving down and stop. As the up and down movements are continuous, the execution of the up-down movements has been processed simultaneously using Asynchronous Programming. Additionally, when moving up or down, the whale will keep moving for 0.75 s and then switch to a stop posture for a smoother presentation. The boolean function $isMovingDownValid$ has been used to check the current altitude and constrain the movement of the whale such that the whale has not surpassed the limit set, i.e. below the screen or above the water surface.

When the button connected to Arduino is engaged, the $ActivateSplash()$ method is triggered, activating the spraying, which is a predefined object that is hidden by default. A further requirement for the spraying to be activated is the upward position of the whale, which is conveyed through the $Whale$ class data.

Health and Score

To engage the patients to use their legs, a constantly decreasing health point system was used. Points were gained by the eating of two kind of fishes with corresponding reward points. To prevent the patient from maintaining the eat position, a deduction system was employed by the trash element. Adding and deducting health are implemented as functions for the three objects. Each function is called when the jaw collides with the corresponding objects. The health value is normalised between 0 and 1: when the health score is higher than 1, it will be set back to the maximum, when it is lower than 0, the program will display the "Game Over". Other than the health points, there is an independent numerical scoring system. Eating fish and spraying seagulls both affect the scoring. Hitting seagulls does not regenerate health, but is designed to keep the game entertaining.

Object Spawn and Properties

The object spawn action has been conducted onto two parallel scripts: $SpawnManager.cs$ (which takes charge of all items spawn in the sea) and $SpawnSeaGullManager.cs$ (which only takes charge of spawning seagull). The two classes work simultaneously but do not interact with each other. Fish or trash objects will be spawned every two sec-

onds. The kind and altitude of the objects spawned are defined randomly, but constrained depending on the game mode. When playing in "Right-Leg" or "Left-Leg" mode, the objects will only be spawn on the higher altitude. When playing in "Both-Leg" mode, the objects will be spawn on either the top or bottom lanes, each with a 50% chance. The spawn of seagulls has a low frequency and is separated from the fish-trash system by a constant time.

The constant movement of the sea objects observe the following logic. When a new object has been spawned, it is appended to the current spawn manager parent object. Each time the $Update()$ function is called, the programme loops through all the current children of the parent spawn manager object. Finally, a left-pointing vector is applied to every single child in the loop. The child objects of the spawn manager are destroyed after being eaten or after falling outside the screen boundary.

Environment Objects

A function was defined to make various objects such as corals, waves and clouds scroll to the left at different speeds, to create an effect of perspective whilst the whale's relative x-position on the screen is fixed. Three identical wave sprites have been put in a row to simulate the constant flow of waves. When the center of the $WaveEntity$ (containing the three wave sprites) scrolls to one screen width to the left of the original center, the X-position of $WaveEntity$ is reset to create a constant flowing effect. In addition, an oscillating algorithm has been implemented onto the wave entity to mimic the dynamic of real waves. Three layers of wave entities fluctuate according to various periods.

5 Concluding Remarks

Open Sesame's mechanical system was successfully assembled and integrated with the electrical components through a mechanical fin fitting within the potentiometers. The latter provided reading to Arduino, which were converted into the angles necessary to play the game developed in Unity. Open Sesame is a fully functioning device with a stimulating game for the target audience of 4 to 10 year olds. The device also manages to target the hip rotation to help the rehabilitation of patients with CP, particularly those who require hip surgery. Consumer market research would have to be conducted with CP patients and their physicians, and subsequently reviewed to assess the viability and the utility of the device.

This device was a prototype and further improvement with regards to the mechanical design and the software are significant. The mechanical design used internally sourced parts, including ply wood and turn tables, which were decorated with paint and tissue paper to encompass our target audience. To allow mass manufacturing and enhancement of the robust nature of the device, the supporting platforms, mobile platforms and clamps would be manufactured with plastic. This would be chosen due to its low cost, malleability and a variety of colours. Additionally, the mechanism would be designed to be more compact to improve its portability for the consumer. The electrical system could be upgraded to a wireless package, which would communicate to Unity via bluetooth to allow a detachable system that could be used for both traditional physical therapy and within our device for assessment. Software developments include more levels and a more comprehensive analytical assessment of the progression of the degree of hip rotation. This would increase motivation and would be very useful information that could be utilised by physicians to track progress remotely and help inform on future steps to adopt.

References

[1] Benedic Kirby Yeargin-Allsopp Van Naarden Braun Doernberg Arneson, Durkin. Prevalence of cerebral palsy: Autism and developmental disabilities monitoring network. 2009.

[2] Terjesen T. The natural history of hip development in cerebral palsy. 2012.

[3] Anil Agarwal and Indreshwar Verma. Cerebral palsy in children: An overview. pages 77–81, 2012.

[4] Gunnar Hägglund, Henrik Lauge-Pedersen, and Philippe Wagner. Characteristics of children with hip displacement in cerebral palsy. *BMC Musculoskeletal Disorders*, 8(1):101, Oct 2007.

[5] Choi In Youb-Chin Cho Tae-Joon Taek Sung Lee Ho-Seung Park Soo-Sung Lee Hye Oh Chang-Wug Kim Hyun, Jeong. Distribution of lengths of the normal femur and tibia in korean children from three to sixteen years of age. *Journal of Korean medical science*, Nov 2003.

[6] Kids shoe size chart. https://www.blitzresults.com/gb/kids-shoe-size-chart/.

[7] Width chart-kids. https://www.healthyfeetstore.com/width-chart-kids.html.

[8] Stability. http://www.schoolphysics.co.uk/age11-14/Mechanics/Statics/text/Stability_/index.html.

[9] Sigle R. Sigle, J. Foot-supporting insole. US4317293A, 1979.

A Final Product

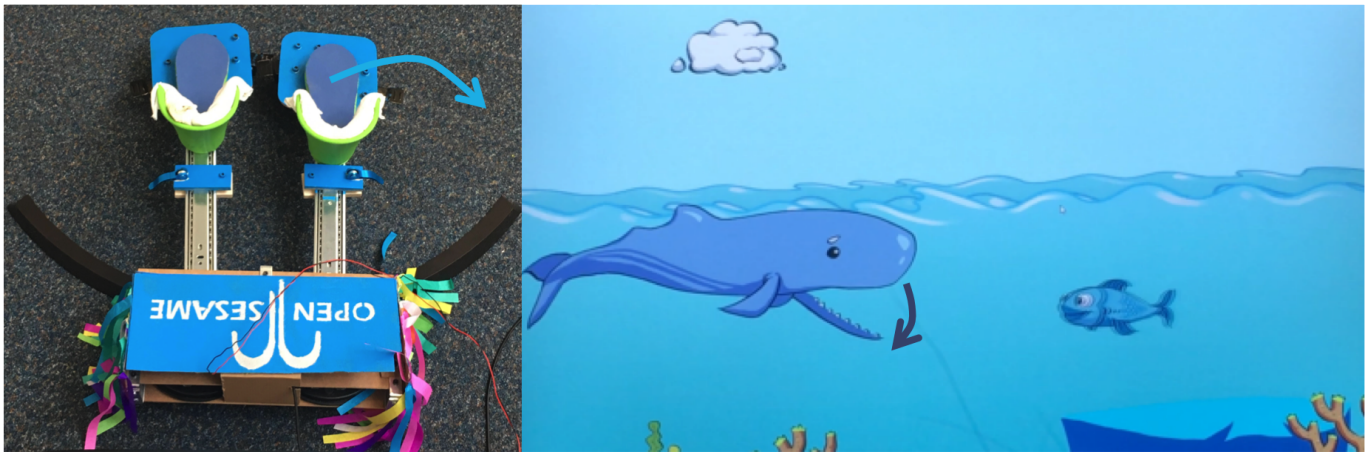


Figure 1: Left: Final assembly of the mechanical and electronic components of the product. Right: Final game interface.

B Mechanical Design

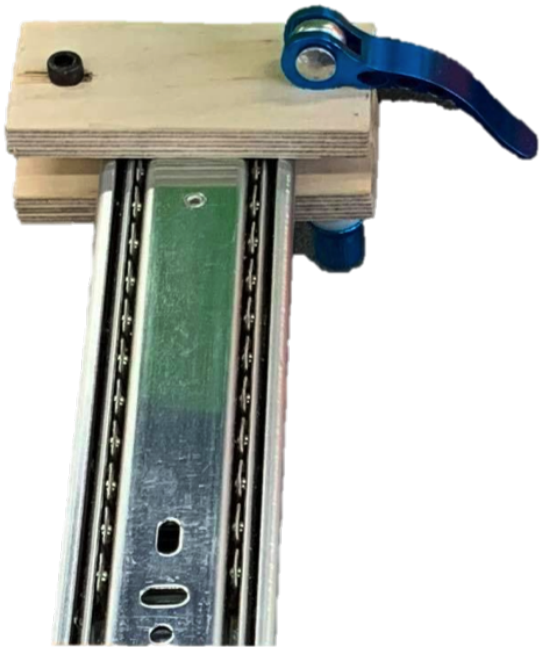


Figure 2: Final assembly of the clamp on the rotating arm.

Figure 3: Left: Turn tables. Right: Turn tables with rotating arms.

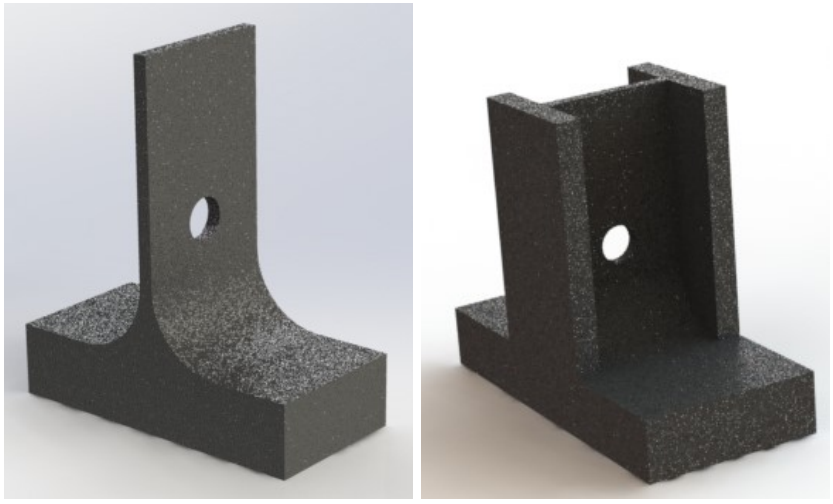


Figure 4: Left: the original design; Right: the final design with I cross-section on the top to make sure the connection between the fin and the potentiometer is always tight. Note the existing hole has the exactly same size as the cable tie used.

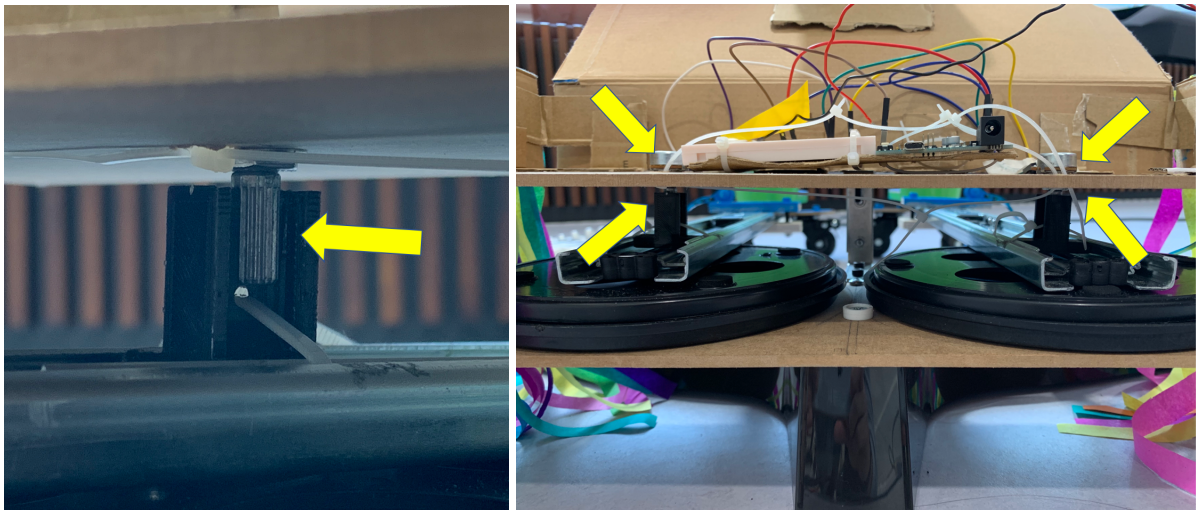


Figure 5: Left: a closed view of how the mechanical fin is connected with the rotating arm and the potentiometer; Right: an overall view of the connection between both potentiometers and mechanical fins.

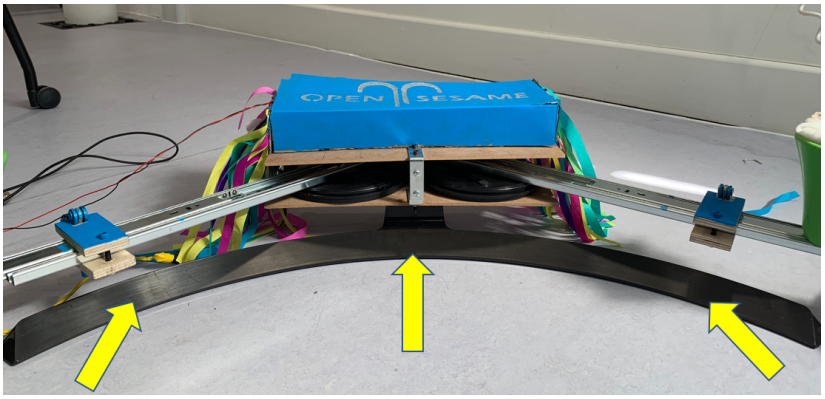


Figure 6: The television stand as the support base of the mechanical device.

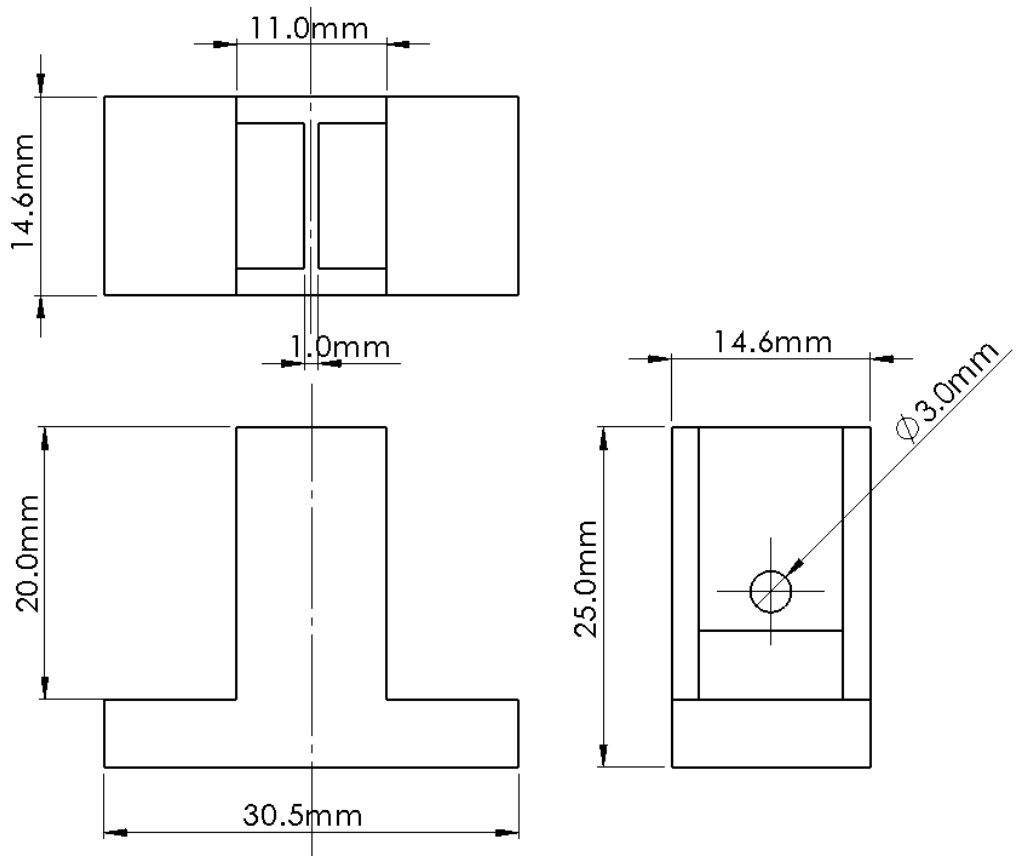


Figure 7: The dimensions about the final mechanical fin design.

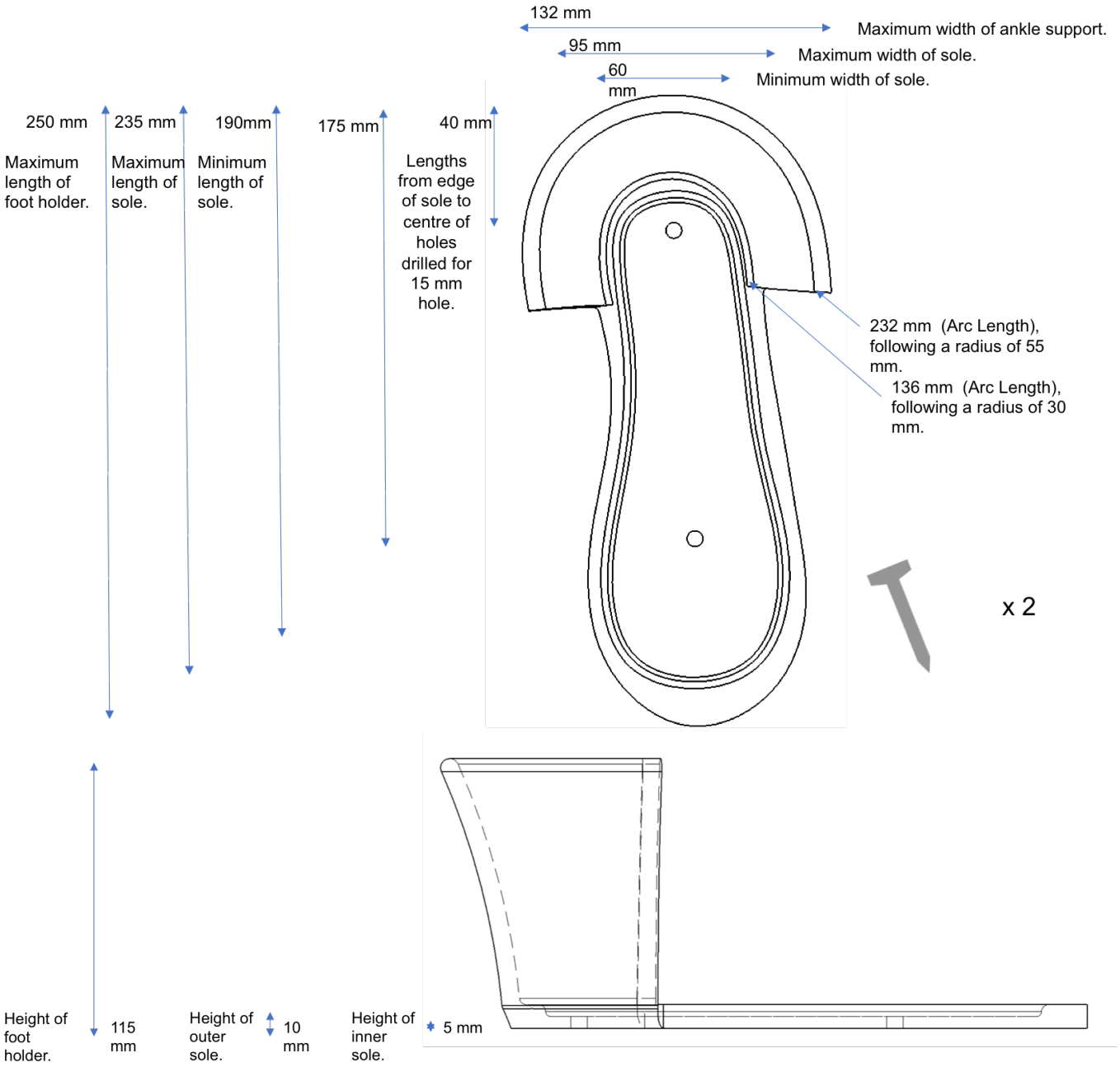


Figure 8: Schematic of the left foot holder with placement of the two 6 mm drilled holes.

Table 1: The printing parameters used for the manufacturing of the foot holder.

Material	**PLA**
Layer Thickness (mm)	0.1
Layer Height (mm)	0.1
Infill Density (%)	100
Print Speed (mm/s)	60
Travel Speed (mm/s)	120
Brim Width (mm)	3
Build Plate Temperature (°C)	60

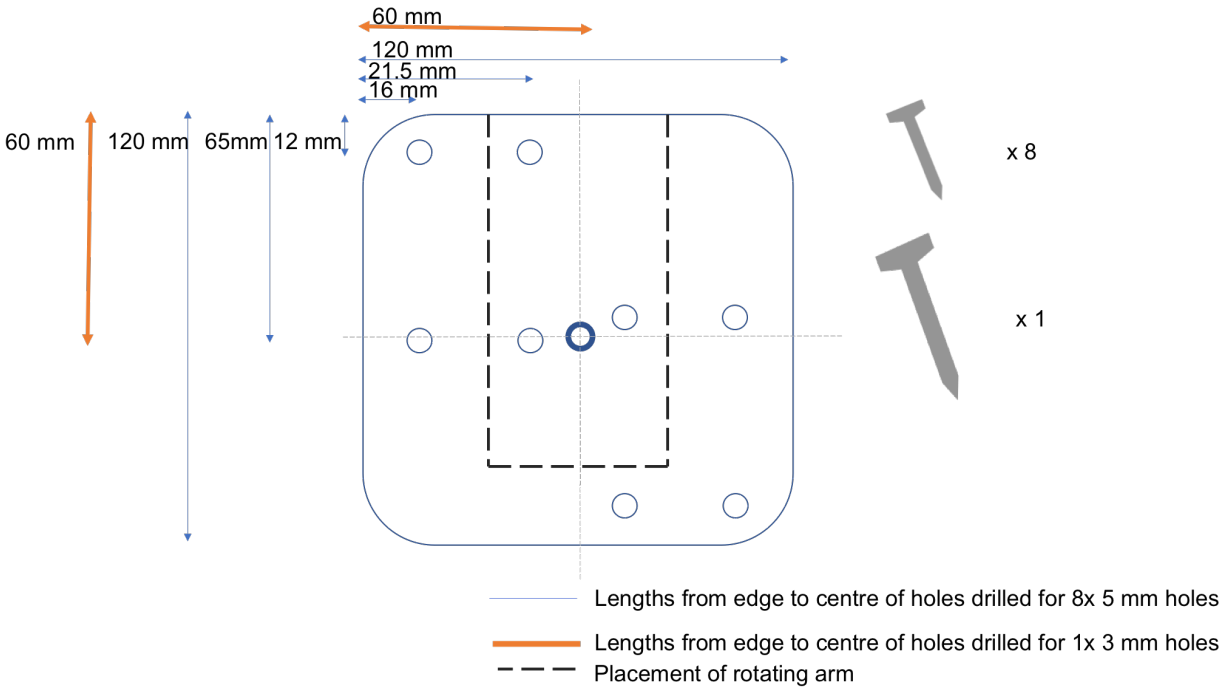


Figure 9: Schematic of the mobile platform with placement of the eight $5\ \mathrm{mm}$ and the one $3\ \mathrm{mm}$ drilled holes.

C Electronics Design

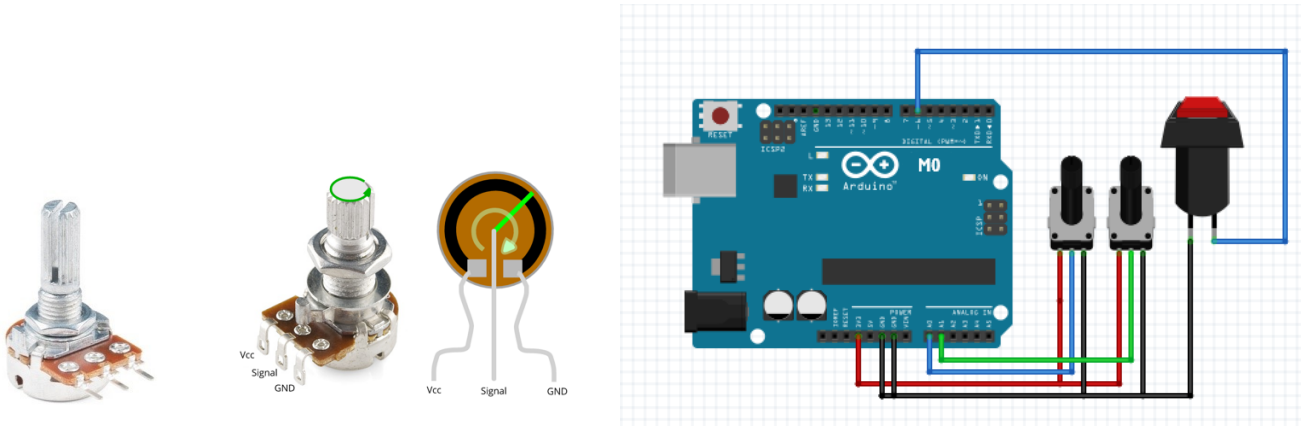


Figure 10: Left: Potentiometer; Middle: Lugs on potentiometer; Right: Full connection of electronic parts.

D Arduino Code

```
/*

  Reads an analog input on pin 0, converts it to angle that potentiometers screwing,
  and prints the result to the Serial Monitor. Graphical representation is available
  using Serial Plotter (Tools > Serial Plotter menu). Attach the center pins of two
  potentiometers to pin A0 and A1, and the outside pins to +3.3V and ground.

*/
#include "MovingAverage.h"
// This is the moving average filter from https://github.com/sofian/MovingAverage
MovingAverage average(20);
const int buttonPin01 = 6;
int buttonPush;

// the setup routine runs once when you press reset:
void setup() {
    // initialize serial communication at 9600 bits per second:
    Serial.begin(9600);
    // initialize digital input and set it as high
    pinMode(buttonPin01, INPUT);
    digitalWrite(buttonPin01, HIGH);

}

// the loop routine runs over and over again forever:
void loop() {
     // when button is not pushed, buttonPush is 0
    buttonPush=0;
    // reset the Δ angle each loop preventing cached reference from the previous loop
    // read the input on analog pin 0:
    int sensorValue_r = analogRead(A0);
    int sensorValue_l = analogRead(A1);
    // Convert the analog reading (which goes from 0 — 1023) to a angle (0 —
    // 270 degree):
    float angle_r = sensorValue_r * (270 / 1023.0);
    float angle_l = sensorValue_l * (270 / 1023.0);
    // when button is pushed, the digital input change to low and buttonPush is asigned to 1
    if (digitalRead(buttonPin01) == LOW)
    {
      buttonPush=1;
    }
    // Get actual angle according to offsetvalue
    float actualAngle_r=angle_r-49;
    float actualAngle_l=229-angle_l;
    // Using filter
    float movingAvg_r = average.update(actualAngle_r);
   // float movingAvg_l = average.update(actualAngle_l);
    SerialUSB.print(movingAvg_r);
    SerialUSB.print(",");
    SerialUSB.print(actualAngle_l);
    SerialUSB.print(",");
    SerialUSB.print(buttonPush);
    SerialUSB.println();
    SerialUSB.flush(); // for completing previous data sending
```

```
55      delay(20); // in case the previous line doesn't work well
56  }
```

E Unity Code(Game Design)

E.1 Jaw-sensor mapping

```
1   // Jaw.cs (... represents other code blocks irrelevant to the current session)
2
3   ...
4
5   [SerializeField] private bool isRightLeg;
6
7   private float angleJaw; // whale jaw open angle controlled by leg open angle
8
9   ...
10
11  void Update () {
12      ...
13
14      if (isRightLeg) {
15          angleJaw = arduinoHelper.angle_r / (calibrationMenu.angleRightConstraint / 60f);
16      }
17      else {
18          angleJaw = arduinoHelper.angle_l / (calibrationMenu.angleRightConstraint / 60f);
19      }
20
21      PotentiometerControl(angleJaw);
22
23      ...
24  }
25
26  ...
27
28  // ------ Arduino Potentiometer Control ------
29
30  void PotentiometerControl (float angle) {
31      transform.localRotation = Quaternion.Euler(0, 0, -angle);
32  }
```

E.2 Whale movement

```
1   // Whale.js
2
3   enum State {
4       movingDown,
5       movingUp,
6       stop
7   }
8
```

```
// ——— Asynchronous Programming: ———

private void MovementHandler() {
    switch (state) {
        case State.movingDown:
            transform.Translate(
                -Vector3.up * speed * Time.ΔTime,
                Space.World);
            break;
        case State.movingUp:
            transform.Translate(
                Vector3.up * speed * Time.ΔTime,
                Space.World);
            break;
        case State.stop:
            // stop the whale by assign the current position to its position
            transform.position = gameObject.transform.position;
            break;
        default:
            transform.position = gameObject.transform.position;
            break;
    }
}

...

// ——— Change Movements by Manipulating States ———

private IEnumerator MoveDown() {
    if (isMovingDownValid) {
        state = State.movingDown;
        yield return new WaitForSeconds(0.75f); // give 0.75s position translation time
        state = State.stop;

        ...
    }
}

private IEnumerator MoveUp() {
    if (!isMovingDownValid) {
        state = State.movingUp;
        yield return new WaitForSeconds(0.75f);
        state = State.stop;

        ...
    }
}

// ——— Boundary Check ———
//(... represents other code blocks irrelevant to the current session)
...

private IEnumerator MoveDown() {
    if (isMovingDownValid) {
        state = State.movingDown;
        yield return new WaitForSeconds(0.75f); // give 0.75s position translation time
        state = State.stop;

```

```
67          // banning the whale from moving further downwards when it's already in lower position
68          isMovingDownValid = false;
69      }
70  }
71
72  private IEnumerator MoveUp() {
73      if (!isMovingDownValid) {
74          state = State.movingUp;
75          yield return new WaitForSeconds(0.75f);
76          state = State.stop;
77
78          // banning the whale from moving further upwards when it's already in higher position
79          isMovingDownValid = true;
80      }
81  }
82
83  // SplashManager.cs (... represents other code blocks irrelevant to the current session)
84
85  ...
86
87  void Start() {
88      ...
89
90      // initially disable the box collider, animator and sprite render and trigger later
91      box2D         = GetComponent<BoxCollider2D>();
92      box2D.enabled = false;
93
94      animator         = GetComponent<Animator>();
95      animator.enabled = false;
96
97      spriteRenderer         = GetComponent<SpriteRenderer>();
98      spriteRenderer.enabled = false;
99
100     // initially set the splash activatable to true
101     isSplashActivatable = true;
102 }
103
104 ...
105 public class SplashManager : MonoBehaviour {
106
107     [SerializeField] private float splashDuration = 0.5f;
108     private int buttonPressed = 0;
109
110     private bool isSplashActivatable;
111
112     ...
113
114     void Start() {
115         ...
116
117         // initially set the splash activatable to true
118         isSplashActivatable = true;
119     }
120
121     ...
122
123     // ———————— Button Control ————————
124
```

```
125     void ButtonControlSplash() {
126         if (buttonPressed == 1) {
127             ActivateSplash();
128 
129             PreventMultipleSplash();
130         }
131     }
132 
133     // ———————— Enable and Disable Splash Activatable to mitigate splash overlay ————————
134 
135     void PreventMultipleSplash() {
136         // prevent the user from splashing various times within short time
137         isSplashActivatable = false;
138 
139         // set the splash activatable property back to true after a short delay
140         Invoke("SplashActivatable", 0.5f);
141     }
142 
143     void SplashActivatable() {
144         isSplashActivatable = true;
145     }
146 
147     // ———————— Splash Manipulations ————————
148 
149     void ActivateSplash() {
150         box2D.enabled          = true;
151         animator.enabled       = true;
152         spriteRenderer.enabled = true;
153         Invoke("DeactivateSplash", splashDuration);
154     }
155 
156     void DeactivateSplash() {
157         box2D.enabled          = false;
158         animator.enabled       = false;
159         spriteRenderer.enabled = false;
160     }
161 }
162 // SplashManager.cs (... represents other code blocks irrelevant to the current session)
163 
164 [SerializeField] private GameObject whaleGameObject;
165 private Whale whale;
166 
167 ...
168 
169 void Start() {
170     ...
171 
172     whale = whaleGameObject.GetComponent<Whale>();
173 
174     ...
175 }
176 
177 void Update() {
178     ...
179 
180     // determine whether the whale altitude and only trigger at higher position
181     if (whale.isMovingDownValid && isSplashActivatable) {
182         KeyboardControlSplash();
```

```
183         ButtonControlSplash();
184     }
185 }
```

E.3 Health and Score

```
1  // PlayerHealth.cs (... represents other code blocks irrelevant to the current session)
2
3  private Transform barMask;
4  private Transform bar;
5
6  ...
7
8  void Awake() {
9      barMask = transform.Find("Green Bar Mask");
10     bar     = transform.Find("Green Bar");
11
12     ...
13 }
14 private void SetSize(float sizeNormalised) {
15     barMask.localScale = new Vector3(sizeNormalised, 1f);
16 }
17
18 // ------ Health Manipulations ------
19
20 private void ConstantHealthDecrease() {
21     if (health > healthMin) {
22         health -= healthDecreaseRate;
23     }
24 }
25
26 // ------ Eaten Behaviour ------
27
28 public void EatSmallFish() {
29     health += 0.2f;
30     ...
31 }
32
33 public void EatBigFish() {
34     health += 0.4f;
35     ...
36 }
37
38 ...
39 ...
40
41 public void EatTrash() {
42     health -= 0.6f;
43 }
44
45 ...
46
47 [SerializeField] private float healthMax = 1f;
48 [SerializeField] private float healthMin = 0f;
49
```

```
50 void Update() {
51     ...
52 
53     ConstantHealthDecrease();
54     SetSize(health);
55 
56     if (health > healthMax) {
57         health = healthMax;
58     } else if (health ≤ healthMin) {
59         // player dead, load game over scene to reload
60         sceneLoader.LoadReloadScene();
61     }
62 }
63 
64 ...
65 // ——— Score ———
66 private int score;
67 [SerializeField] private TextMeshProUGUI scoreText;
68 
69 ...
70 
71 void Update() {
72     scoreText.text = score.ToString();
73 
74     ...
75 }
76 
77 // ——— Eaten Behaviour ———
78 
79 public void EatSmallFish() {
80     ...
81     score += 20;
82 }
83 
84 public void EatBigFish() {
85     ...
86     score += 40;
87 }
88 
89 ...
90 
91 // ——— Splash SeaGull ———
92 
93 public void SplashSeaGull() {
94     score += 60;
95 }
```

E.4 Objects Spawn and Properties

```
1 //——— Spawn ———
2 void Start() {
3     // trigger spawning new object, starting from 2s, with frequency of once each 2s
4     InvokeRepeating("spawnObject", 2.0f, spawnInterval);
5 }
6 // SpawnManager.cs (... represents other code blocks irrelevant to the current session)
```

```
7
8  private void SpawnObject() {
9      ...
10
11     // random 1/3 possibility spawning each of the 3 plausible objects
12     // random 1/2 possibility spawning at each of the 2 plausible altitude
13     Random random = new Random();
14     int randomThresholdObject = random.Next(1, 4); // generate a integer number between 1, 2, 3
15     int randomThresholdPos    = random.Next(1, 3); // generate a integer number between 1, 2
16
17     ...
18 }
19 //----- Play mode constraint -----
20 [SerializeField] private bool isBothLegMode;
21
22 ...
23
24 private void SpawnObject() {
25     ...
26
27     // determine the altitude of the object spawn position, assigning values to spawnPos
28     if (isBothLegMode) {
29         if (randomThresholdPos == 1) {
30             spawnPos = spawnPosHigher;
31         } else if (randomThresholdPos == 2) {
32             spawnPos = spawnPosLower;
33         }
34     } else {
35         spawnPos = spawnPosHigher;
36     }
37
38     ...
39 }
40 //----- Check constraint -----
41 [SerializeField] private GameObject smallFish;
42 [SerializeField] private GameObject bigFish;
43 [SerializeField] private GameObject trash;
44
45 ...
46
47 private void SpawnObject() {
48     ...
49
50     // determine which object will be spawned at the previous defined altitude
51     if (randomThresholdObject == 1) {
52         newSpawn = Instantiate(
53             smallFish,
54             spawnPos,
55             Quaternion.identity);
56         ...
57     } else if (randomThresholdObject == 2) {
58         newSpawn = Instantiate(
59             bigFish,
60             spawnPos,
61             Quaternion.identity);
62         ...
63     } else if (randomThresholdObject == 3) {
64         newSpawn = Instantiate(
```

```
            trash,
            spawnPos,
            Quaternion.Euler(0, 0, -20f)); // beware the trash spawn has rotation angle
        ...
    }
}
//----- Spawn Seagull -----
// SpawnSeaGullManager.cs (... represents other code blocks irrelevant to the current session)

void Start() {
    // trigger spawning new object, starting from 2s, with frequency of once each 2s
    InvokeRepeating("SpawnSeaGull", 2.0f, spawnSeaGullInterval);

    seaGull = seaGullGameObject.GetComponent<SeaGull>();
}

...

// sea gull is not part of the fish-trash system and the spawning rate is very low
// thus doesn't need to be wrapped into the above object spawn-destroy system
void SpawnSeaGull() {
    GameObject newSpawnSeaGull;

    newSpawnSeaGull = Instantiate(
        seaGullGameObject,
        spawnPosSeaGull,
        Quaternion.identity);
    newSpawnSeaGull.transform.parent = transform;
}
...

void Update() {
    float displacement = Time.ΔTime * speed;

    // store all children under Spawn Manager in an array
    Transform[] children = transform.Cast<Transform>().ToArray();

    for (int i = 0; i < children.Length; i++) {
        var child = children[i];
        // beware to add Space.World or otherwise default will be Space.Self
        // where rotation angle of the object will be stored as well
        child.transform.Translate(Vector2.right * displacement, Space.World);
    }
}
```

E.5 Environment Object

```
[SerializeField] private float scrollSpeed = -4f;
[SerializeField] private int resetX = -32;

void Start() {
    // override the start position to its initial sprite position
    startPos = transform.position;
}

```

```
9  void Update() {
10     xPos = transform.position.x;
11     yPos = transform.position.y;
12
13     float displacement = Time.ΔTime * scrollSpeed;
14     transform.Translate(Vector2.right * displacement);
15
16     // when the center of Wave scrolls to one screen width to the left of the original center,
17     // reset the X of the Wave entity to it's original starting position
18     if (xPos < resetX) {
19         transform.position = new Vector3(startPos.x, yPos, startPos.z);
20     }
21
22     ...
23 }
24 //————— Wave —————
25 // for sine periodic oscillation movement implementation
26 [SerializeField] private bool isOscillating = false;
27 [SerializeField] private Vector2 movementVector = new Vector2(0f, 0.25f);
28 [SerializeField] float period = 2f;
29
30 // 0 for not moved, 1 for fully moved.
31 [Range(0, 1)] [SerializeField] private float movementFactor;
32
33 private Vector2 offset;
34
35 ...
36
37 void Update() {
38     ...
39
40     // ———————— oscillation movement implementation ————————
41     // protect against period is zero
42     // period less than or equal to the smallest thing we can represent (as good as 0)
43     if (period ≤ Mathf.Epsilon) { return; }
44     float cycles = Time.time / period; // grows continually from 0
45
46     const float tau = Mathf.PI * 2f; // about 6.28
47     float rawSinWave = Mathf.Sin(cycles * tau); // goes from −1 to +1
48
49     movementFactor = rawSinWave / 2f;
50     offset = movementFactor * movementVector;
51
52     if (isOscillating) {
53         transform.Translate(Vector2.down * offset);
54     }
55 }
```

F Budget Summary

Table 2: Project budget table.

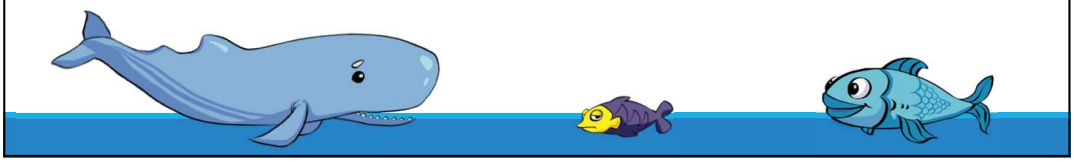
Expense Type	Description	Price (£)
Electronics Part	Micro USB Cable (3m)	6.99
	Arduino Uno	6.99
	2 Rotary Potentiometers*	5.96
	Elegoo 120pcs Dupont wire	6.99
Mechanical Part	Wilko Paint and Brushes	6.75
	Boots Bandages	5.28
	4 Castor Wheels	10.99
	Training Resistance Elastic Bands	7.95
	Insoles (Bts 2pk Scomfort)	3.49
	Screws and Nuts	6.01
	Corner braces	6.44
	Rotating Platforms	30.00
	2 Quick-Release Devices for Bike Seats	9.90
	Pair of Double Fully Extension Ball Bearing Drawer Slide*	10.00
	Television Support Base*	15.00
	Laser-Cutting Wood*	1.50
Total		140.24

* For this project it was not purchased and was sourced internally, and a presumed cost is given.

POWER POINT PRESENTATION SLIDES

OPEN SESAME

HCARD GROUP 3:
FLORENCE WU,
VINCENT SONG,
GEORGE ZHOU,
FEDERICA SPINOLA,
RENKE HUANG,
JUN KE YAO



1

Why?

WHY DID WE FOCUS ON HIP ABDUCTION IN PATIENTS WITH CEREBRAL PALSY?

700 000 are diagnosed with CP each year in the UK

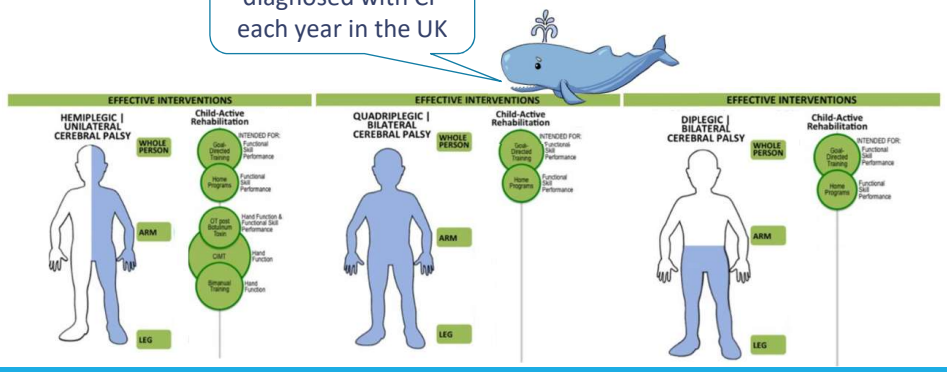


Image from: "Evidence-Based Diagnosis, Health Care, and Rehabilitation for Children With Cerebral Palsy", I. Novak,

2

Why?

WHY DID WE FOCUS ON HIP ABDUCTION IN PATIENTS WITH CEREBRAL PALSY?

1 in 323 babies are affected

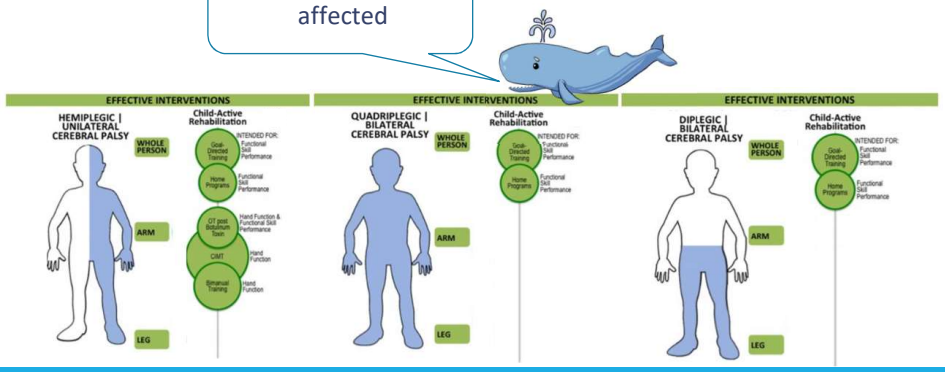


Image from: "Evidence-Based Diagnosis, Health Care, and Rehabilitation for Children With Cerebral Palsy", I. Novak,

3

Why?

WHY DID WE FOCUS ON HIP ABDUCTION IN PATIENTS WITH CEREBRAL PALSY?

1 in 3 have hip displacement

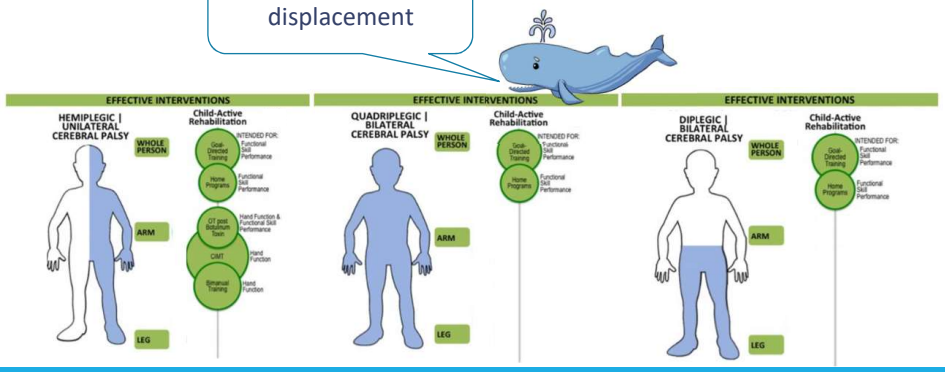


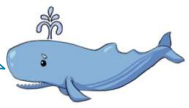
Image from: "Evidence-Based Diagnosis, Health Care, and Rehabilitation for Children With Cerebral Palsy", I. Novak,

4

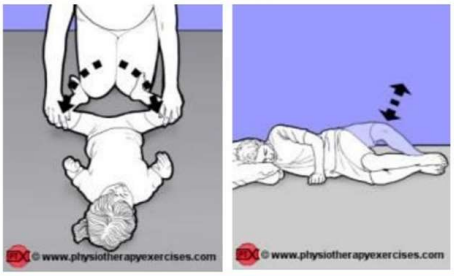
What?

WHAT DID WE DO?

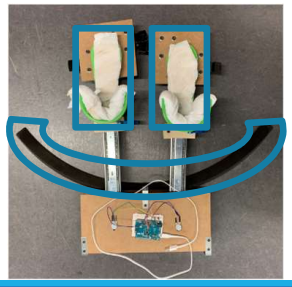
Our Target:
Muscle Group: strengthen and improve hip muscle extensibility
Audience: 4 to 10 years old



Traditional Method
- Requires professional help
- Not fun



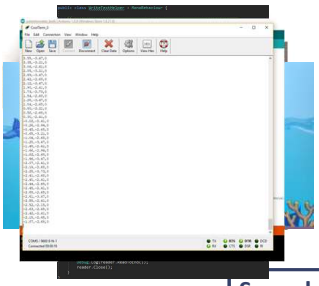
Open Sesame
- Can be done independently
- Fun!



5

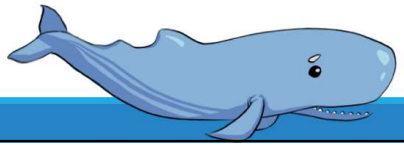
How? System Flowchart

HOW DID WE ACHIEVE OUR AIMS?

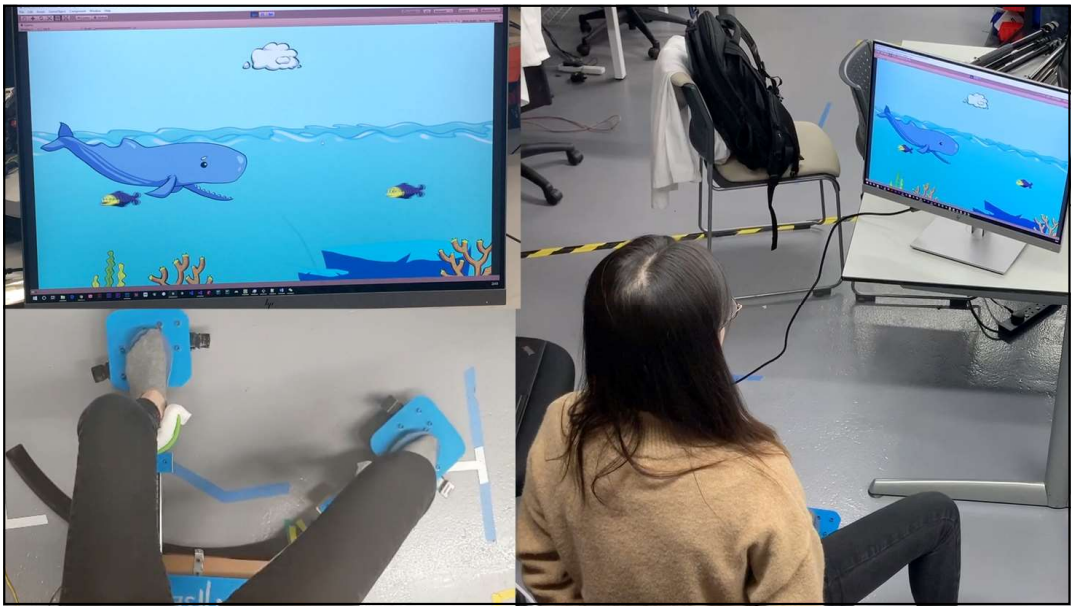


Leg Movement → Arm Rotation → Potentiometer → Arduino → Unity → Save data to track progress

Unity → Control the whale on game



6



7

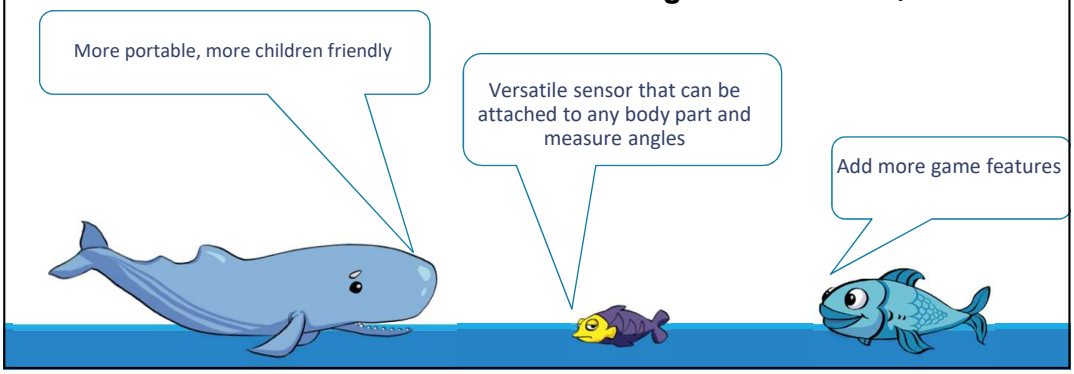
Future Work

Consumer Feedback

‘Such a good idea to motivate young patients’ – Josie, Medic Student

‘So cool’ – Hanna

‘Perfect for children under the age of 11’ – Laura, Playschool volunteer



8

9