SparkAR Tutorial:
Build An Interactive Claw Machine Experience

Overview

In this tutorial, you will learn how to build an interactive claw machine experience in sparkAR. The idea is that users can control the claw by using their face movements instead of their hands. When they feel they get the right position, they can drop the claw by opening the mouth. If the users succeed in catching the toys, they will be rewarded by dressing up the accessories that they caught.
Topics will be covered:

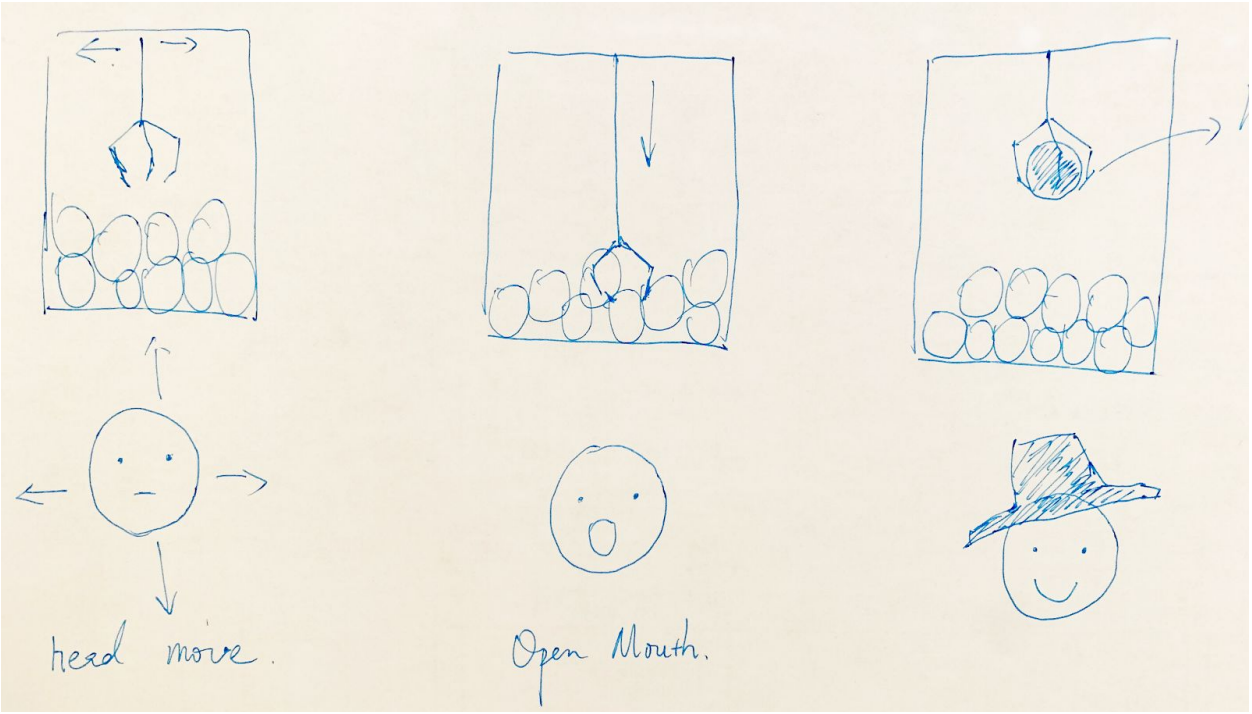
- Design the user interaction flow
- Using face and mouth to control animation
- Design interaction and break down into function module by using sparkAR patch editor
- Set communication between patch editor and the script

Prerequisites

- Download sparkAR editor: https://sparkar.facebook.com/ar-studio/download/
- Download the sample project in github, including both the finished and unfinished versions.

Storyboard

I usually would like to draw a simple storyboard to illustrate the process of the experience, this will help make an initial plan in your mind. As shown below, the experience can be simplified as moving your head to control the claw, open mouth to drop the claw, catch the target toy and drag it up.



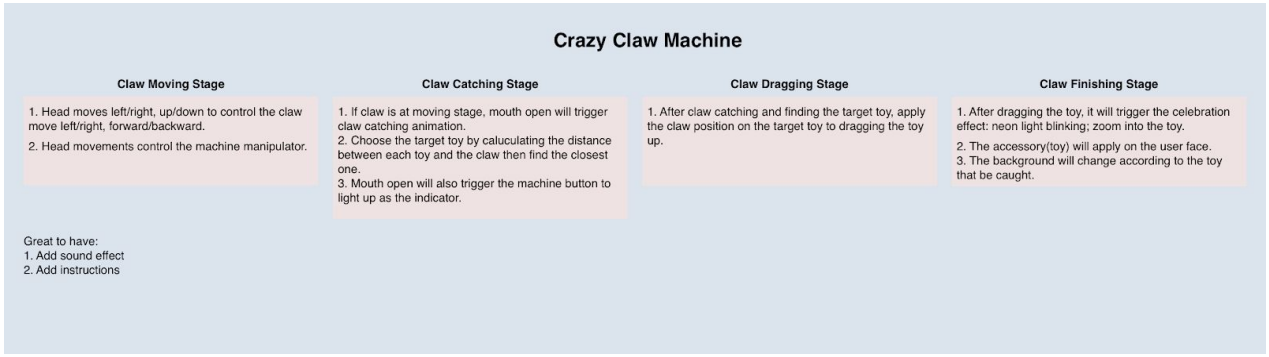
Visual Art

Moreover I made an artboard as the inspiration for the 3d models and visual style. Here is the link to my Pinterest artboard.

https://www.pinterest.com/anyemelody1209/sparkar/

User Flow

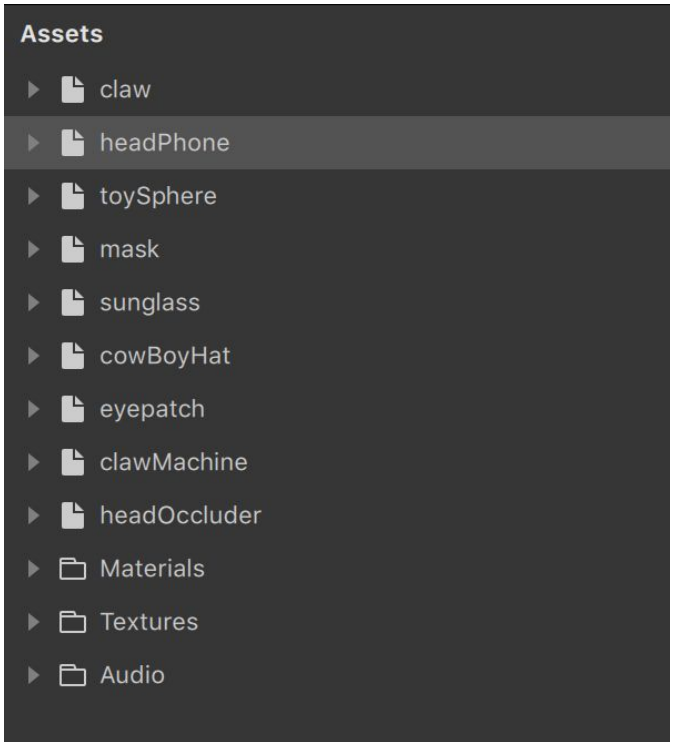
Based on the storyboard, here we design a more detailed user flow. The whole process can be broken down into four stages, and I listed the interactions and behaviors under each stage.



Getting Started in SparkAR

Open the unfinished project from the assets folder. All the needed assets have been already imported into the project, in the assets panel you will find

- clawMachine model
- Claw model with animation clip
- 3D accessories models, including cowBoyHat, eyepatch, headPhone, mask, sunglasses and toySphere.
- A selection of textures
- Audio assets



1. Moving stage

If you refer to the user flow above, the function under this moving stage are:

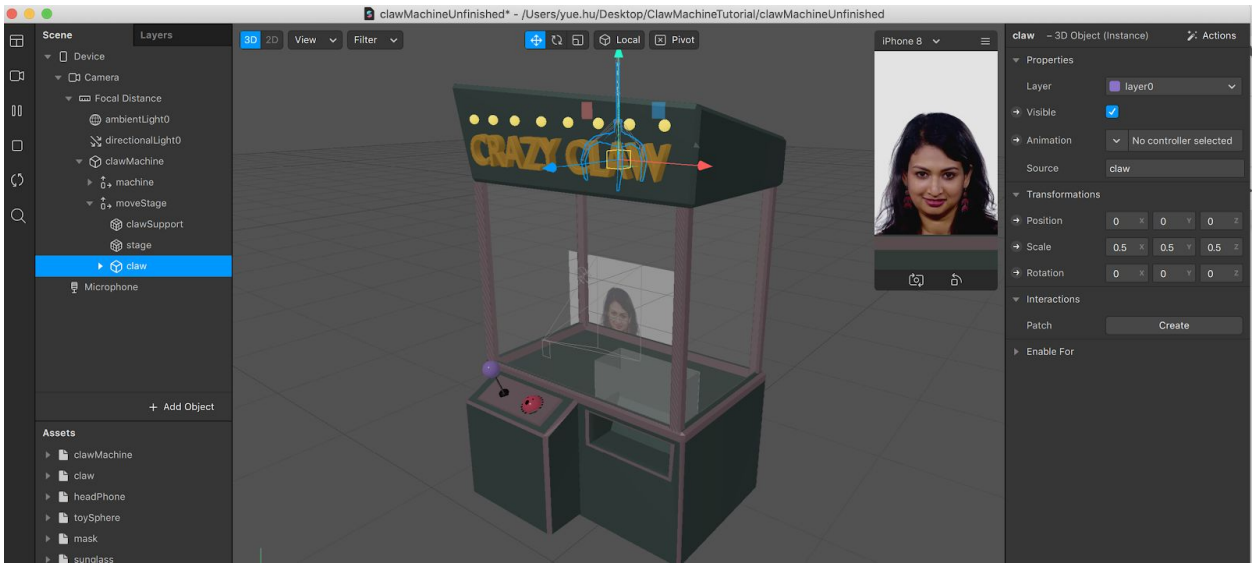
- Head moves left/right, up/down to control the claw move left/right, forward/backward
- Head movements control the machine manipulator

All these interactions are based on the face tracker in SparkAR Studio.

1.1 Setup claw in the scene

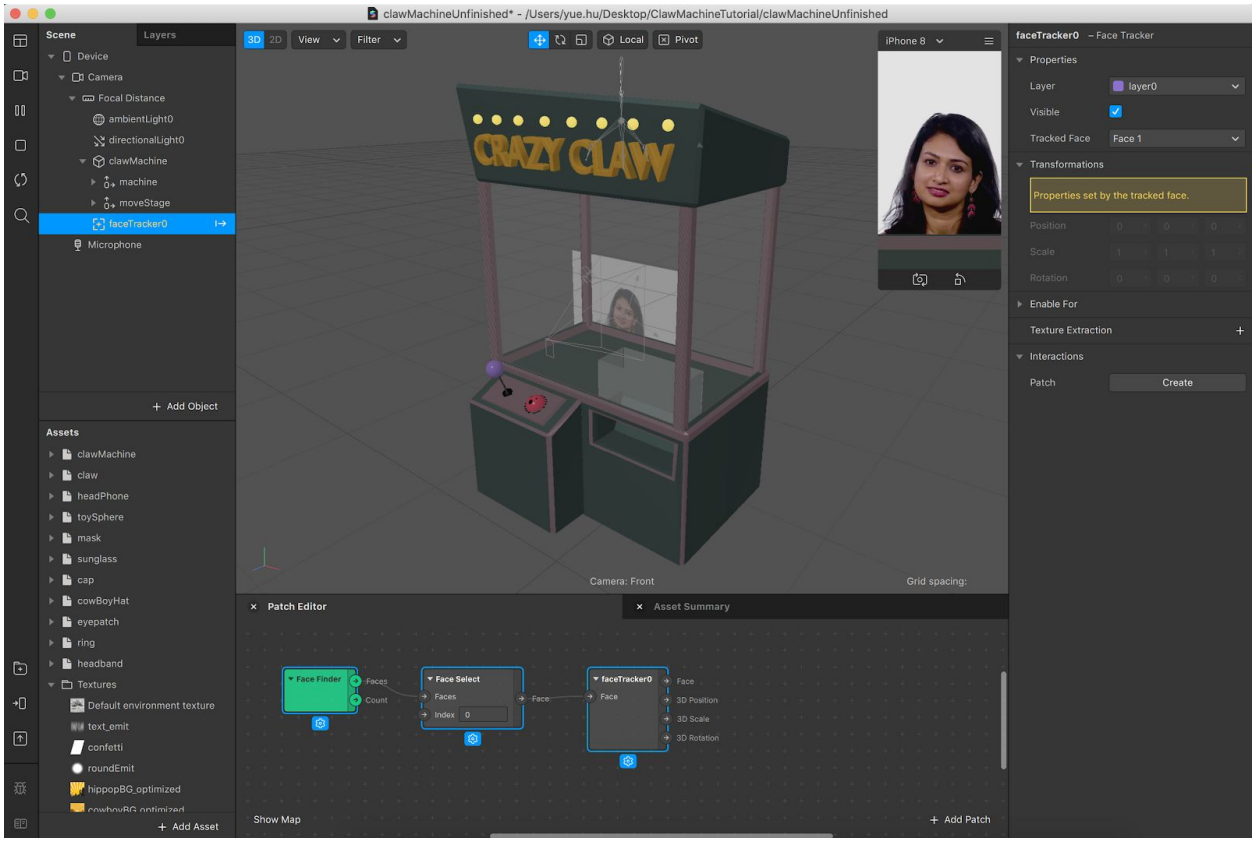
The clawMachine model has already been set up in the scene. Feel free to play with the machine material to customize the appearance.

Drag the claw machine model into the Scene panel,, put it under clawMachine -> moveStage, set claw scale to (0.5, 0.5, 0.5)



1.2 Setup FaceTracker to control the claw

Click **Add Object**, Select **faceTracker**. After you see the faceTracker in the Scene Panel, drag the object into the **patch editor**. (can find the patch editor from **menu -> View -> Show/Hide Patch Editor)**



Drag the lines out from faceTracker 3D Position and 3D Rotation, and apply some math calculations to control the claw moveStage's position. And if you use your own camera for live preview, you will see the claw moveStage starts moving with your head within the claw machine boundary.

1.3 FaceTracker to control machine manipulator

We use the same x and z values that calculated above to control the claw machine rotator as well. The rotator is in clawMachine->machineFrame->manipulator->rotator

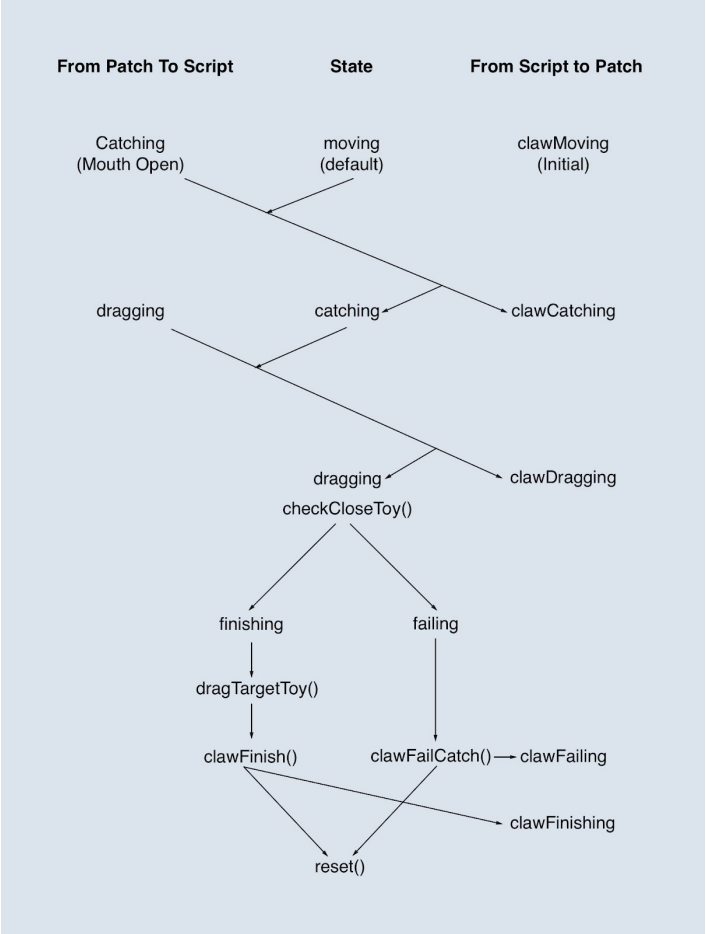
2. Catching Stage

Function be covered in this stage:

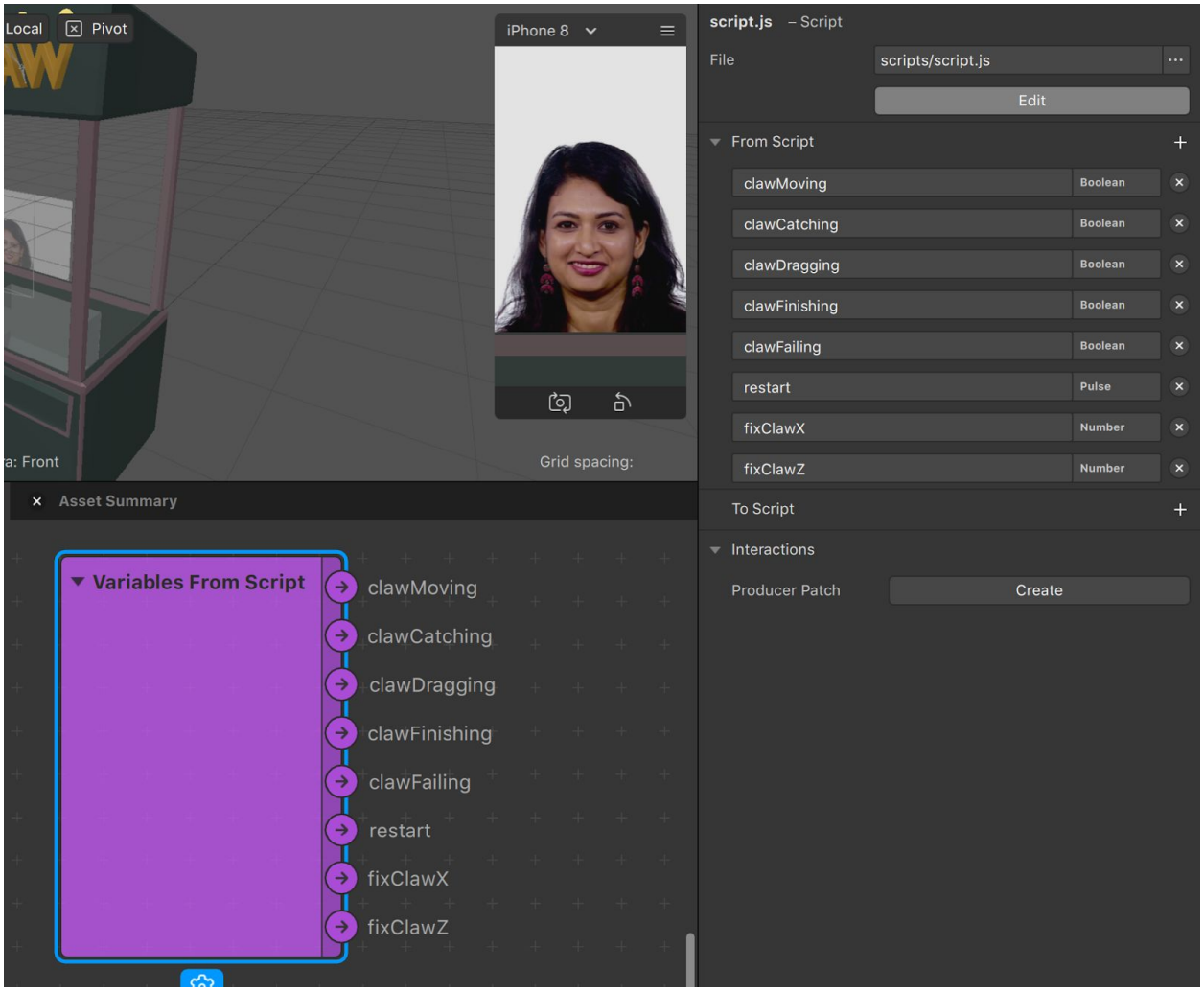
- Stage status will change from moving to catching.
- If it is at the moving stage, then opening the mouth will trigger claw play the catching animation.
- If it is at the moving stage, then opening the mouth will trigger the machine button to light up.
- Choose the target toy by calculating the distance between each toy and the claw to find the closest one.

2.1 Create the Script and the Restart function

Since we need callbacks after each stage finishes and trigger the next stage to begin. We will start to build the script and set variables to help pass the state signals between these stages. For understanding easier and making the logic clearer, I draw this diagram below to help illustrate the communications between the patch editor and script.

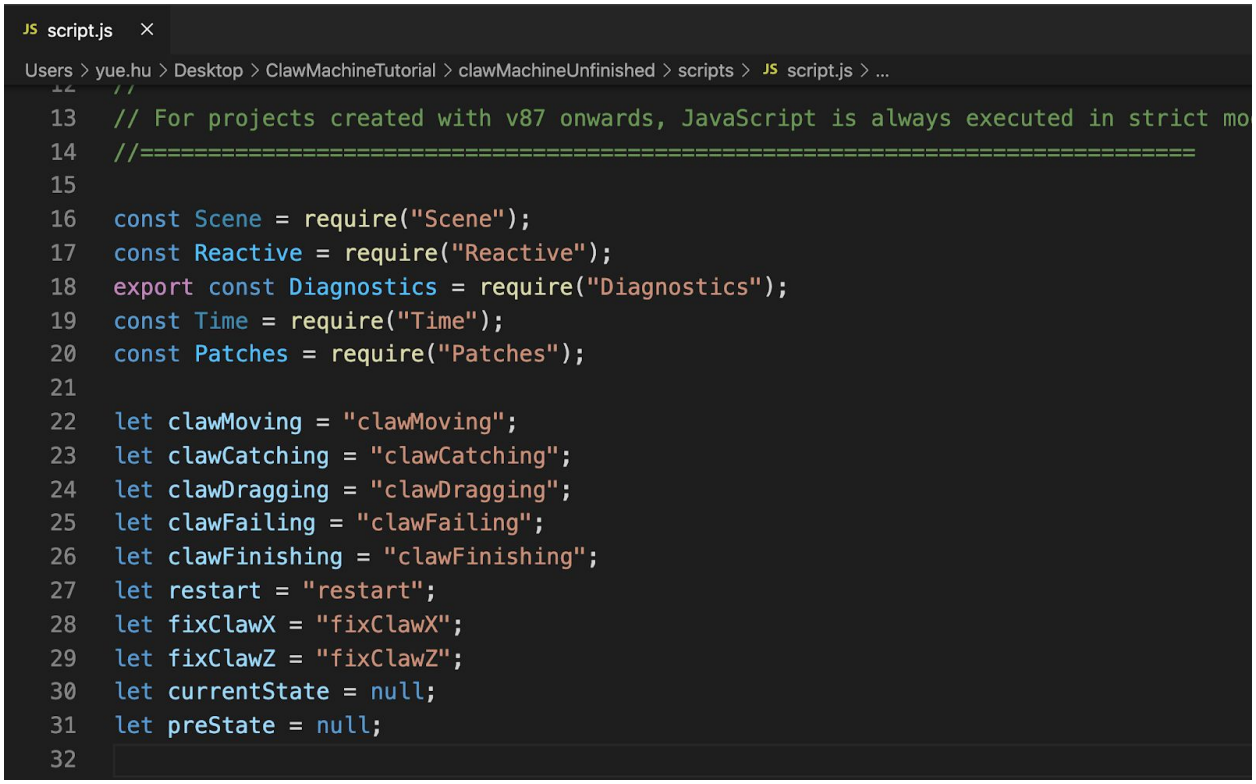


Click **Add Asset → Script,** then you will see a new script be created in your assets. Then, we create 5 **boolean** variables called “clawMoving”, “clawCatching”, “clawDragging”, “clawFailing” and “clawFinishing” to represent the five stages. And a **pulse** variable “restart” to shoot a pulse signal every time starting the experience, and two **Number** variables called “fixClawX” and “fixClawZ” to record the claw position. Then drag the script into the patch editor and you will see the script patch with all the variables that we just created.



After creating these variables, we open the script.js in a code editor like Visual Studio Code, and start writing our first restart function!

We firstly add the needed library in there along with the variables that we created.

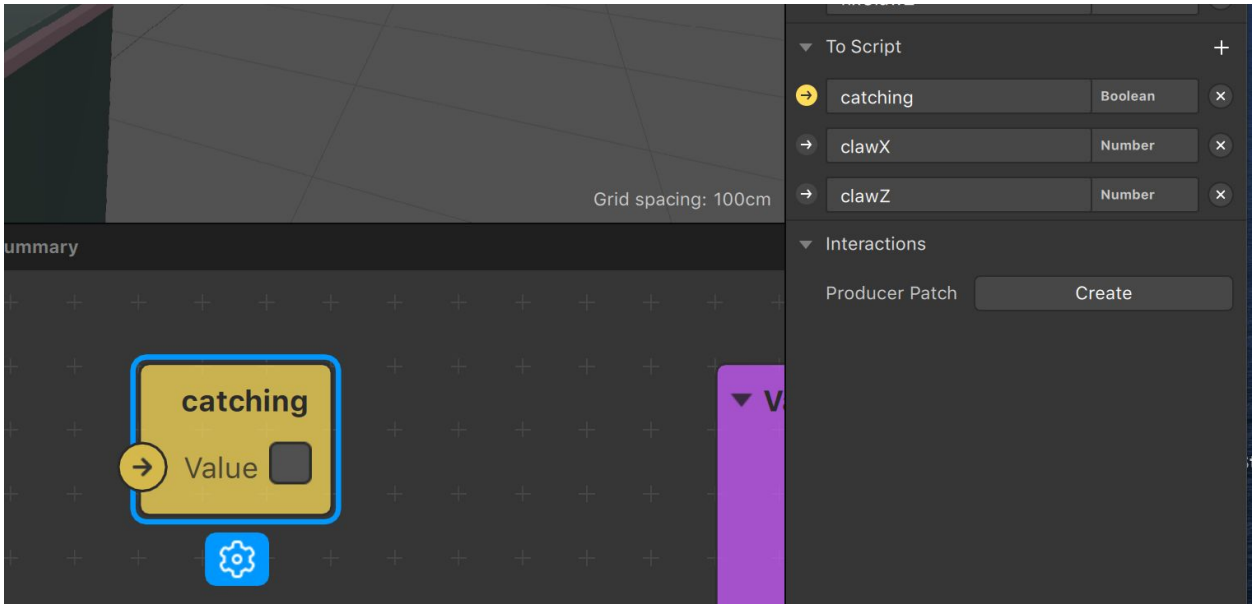


Then we created the function to set the stages' states and the init function to initial these variables at every beginning of the game.

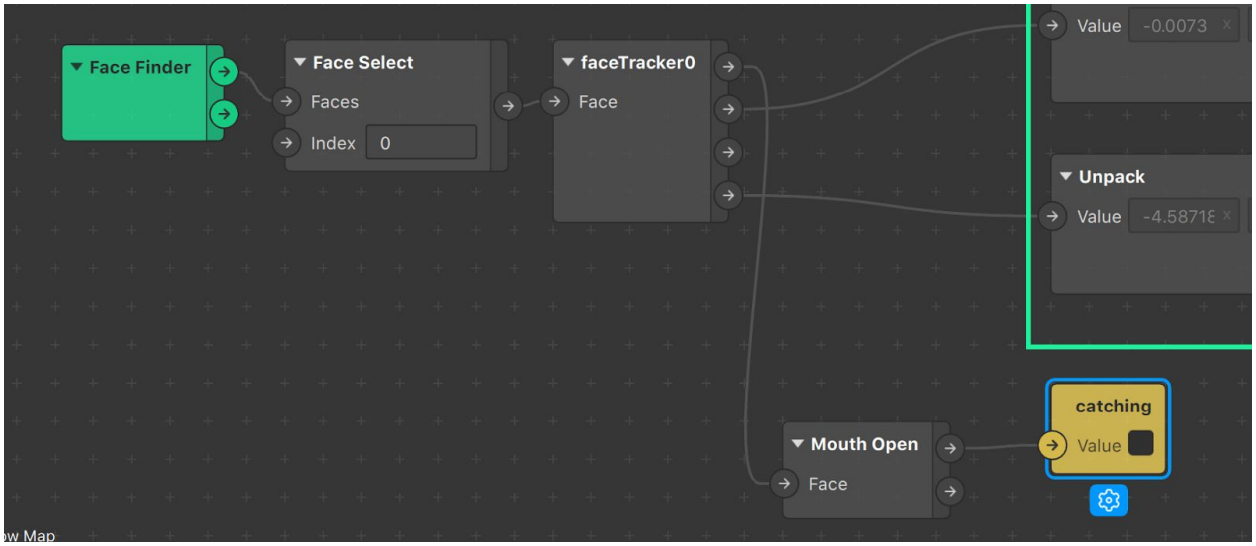
```
/******* function to set the stages ********/
function setPatchState(a, b, c, d, e) {
    Patches.inputs.setBoolean(clawMoving, a);
    Patches.inputs.setBoolean(clawCatching, b);
    Patches.inputs.setBoolean(clawDragging, c);
    Patches.inputs.setBoolean(clawFailing, d);
    Patches.inputs.setBoolean(clawFinishing, e);
  }

  /******* function to init the experience ********/
function init() {
    //shoot pulse when the experience restart
    Patches.inputs.setPulse(restart, Reactive.once());
    //set the active stage to clawMoving at the beginning
    setPatchState(true, false, false, false, false);
    currentState = "moving";
    preState = currentState;
    //reset the claw postion to (0,0)
    Patches.inputs.setScalar(fixClawX, 0);
    Patches.inputs.setScalar(fixClawZ, 0);
  }

  init()
```

2.2 Using Mouth Open to Control the “Catching” Stage

After setting up the script, let's see the “catching” stage and interactions. As it described in the above diagram, we will create a **boolean** “catching” from patch to editor to get the “Open Mouth” signal and send it to script. Then activate the customized variable by clicking the arrow next to it.



Drag the “Mouth Open” patch from the **faceTracker’s** Face node, and connect the “Mouth Open” with our customized patch “catching”.



Then we pass the real time “catching” signal from patch to the script by writing the function in the script.

```
//update the openMouth realtime from the patch//
let openMouth;
  Patches.outputs.getBoolean("catching").then((event) => {
    event.monitor().subscribe(function (values) {
      openMouth = values.newValue;
    });
  });
```

2.3 Tracking the claw position

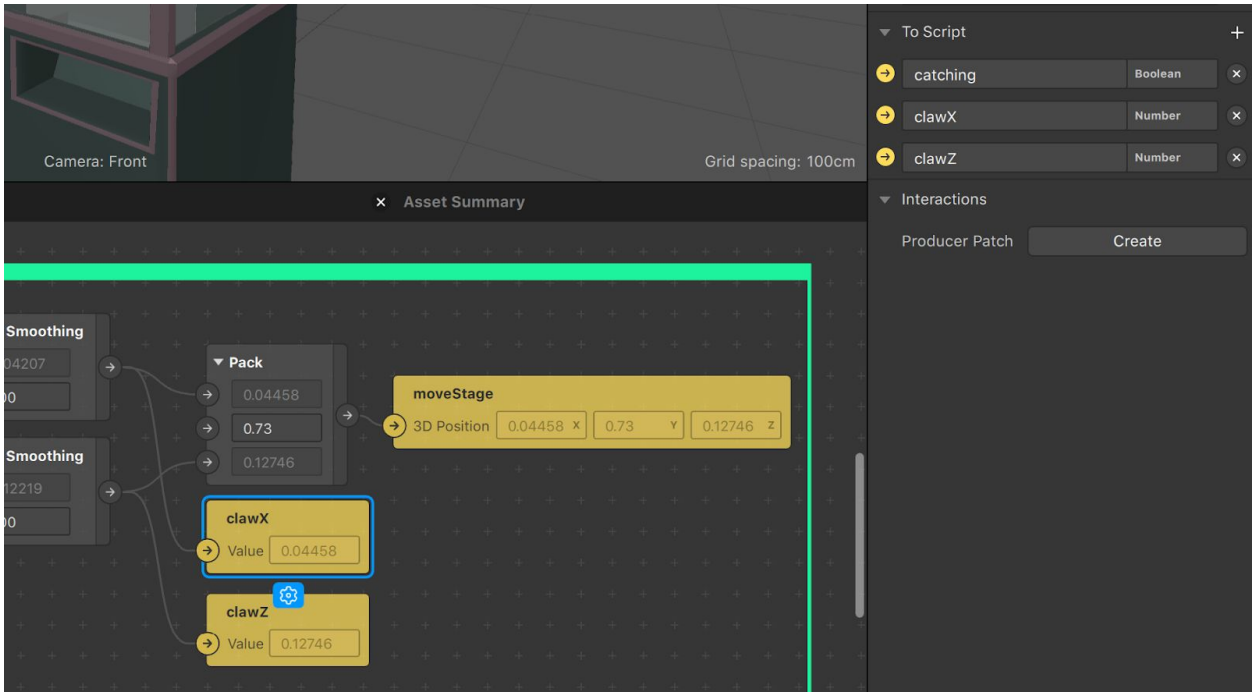
Then we watch the claw posX and posZ values from the patches.

```
 //get the clawX and clawZ from the patch//
 let posX, posZ;
Patches.outputs.getScalar("clawX").then((event)=>{
 event.monitor().subscribe(function (value) {
   posX = value.newValue;
 });
});
Patches.outputs.getScalar("clawZ").then((event)=>{
 event.monitor().subscribe(function (value) {
   posZ = value.newValue;
 });
});
```

When it comes to the “catching” stage, we pass the claw X and Z position as the fixed x and z position for the claw before it drops down and plays the catch animation.

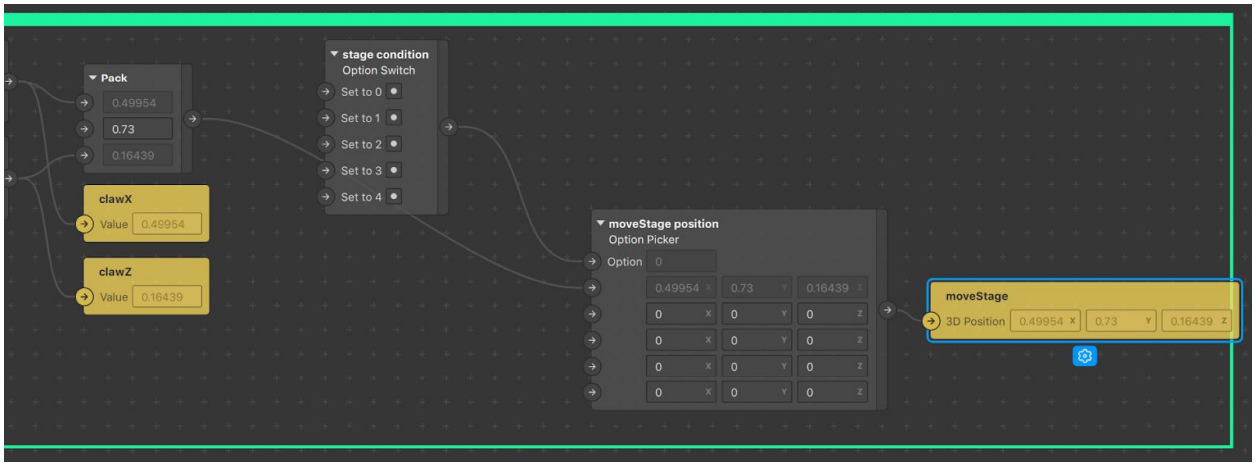
```
function update(){
  //check the state then switch to catching//
  if (openMouth && currentState === "moving") {
    setPatchState(false, true, false, false, false);
    currentState = "catching";
  }
  //lock the claw x and z position
  if (currentState === "catching" && preState === "moving") {
    // let posX = Patches.outputs.getScalar("clawX");
    // let posZ = Patches.outputs.getScalar("clawZ");
    preState = currentState;
    Patches.inputs.setScalar(fixClawX, posX);
    Patches.inputs.setScalar(fixClawZ, posZ);
  }

  Time.setTimeout(update, 50);
}

init()
update()
```

At the same time, we connect the customized clawX and clawZ values with the moveStage in the patch editor to get the values.

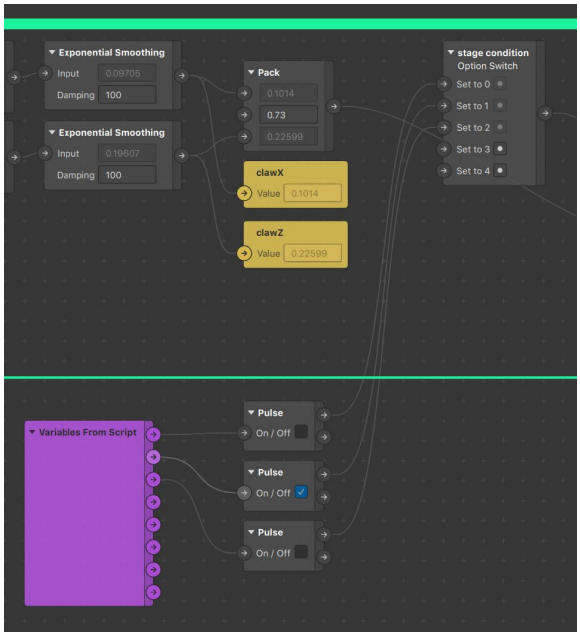


2.4 Set the MovingStage position at different stages

We insert the **Option Switch** and **Option Picker** to switch the moveStage position under different stages' condition.

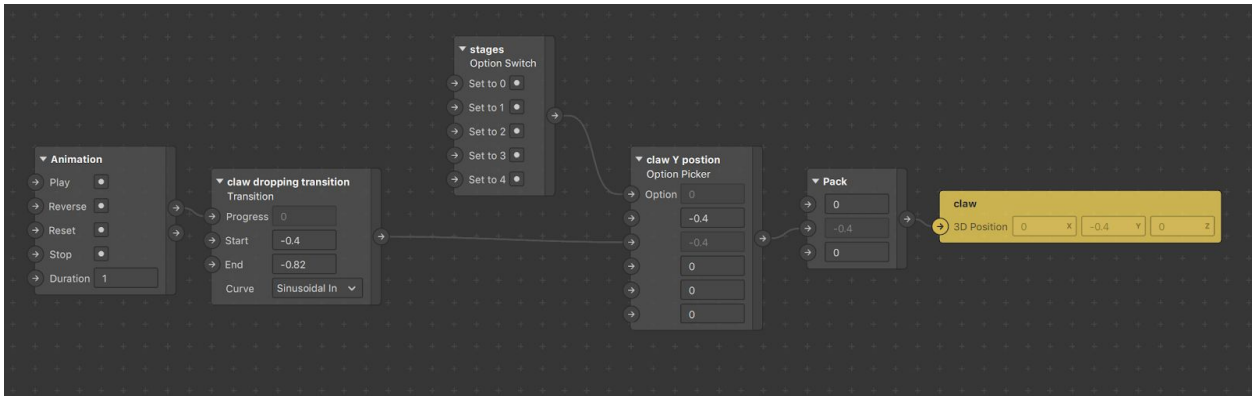


Connect the stages' condition from script to set the option one by one.

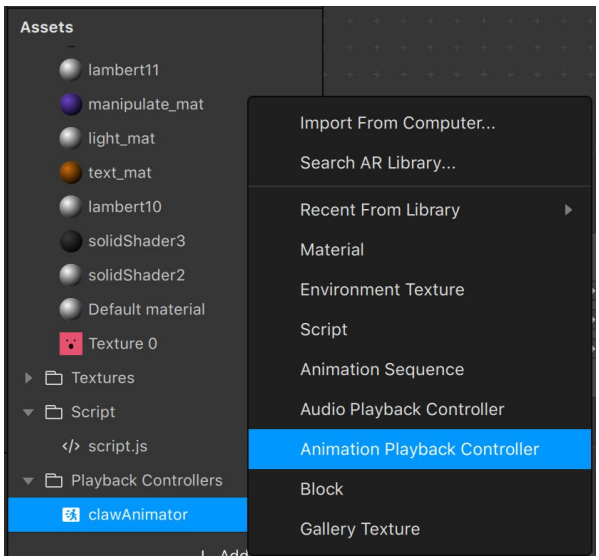


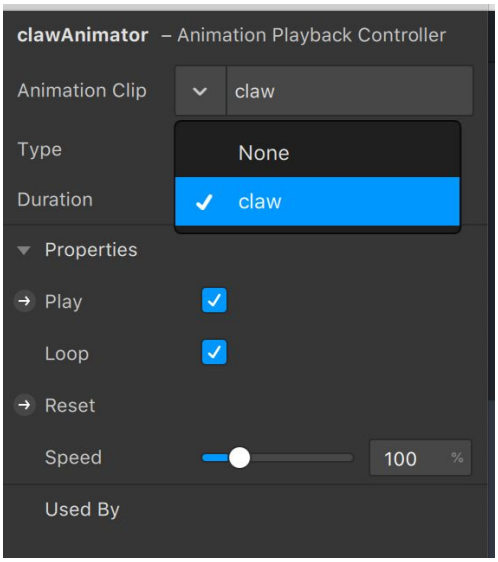
2.5 Setup the Claw Catching Action

After the perapation, let's build the first action after the stage changes to catching. We firstly animate the claw object to drop along the Y axis from -0.4 to -0.82.

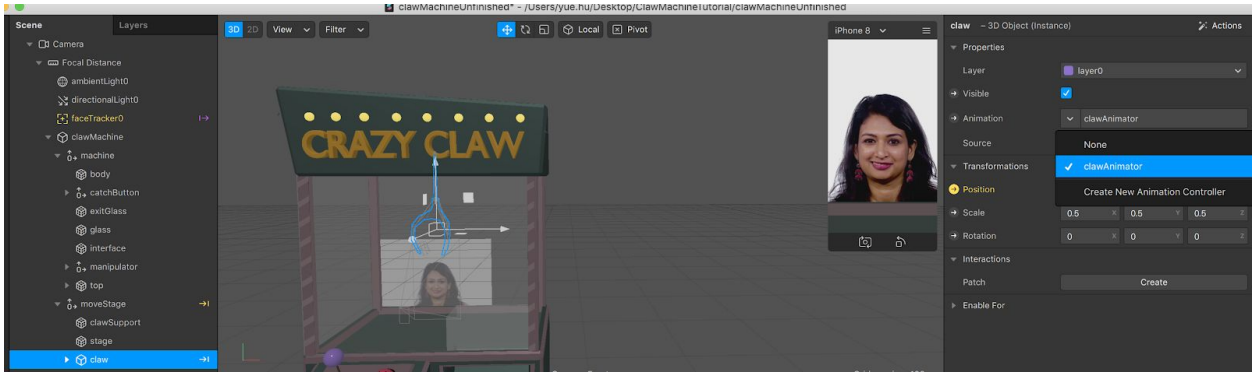


Then we add a new animation controller and named it "clawAnimator", assign the "claw" clip to this animator.

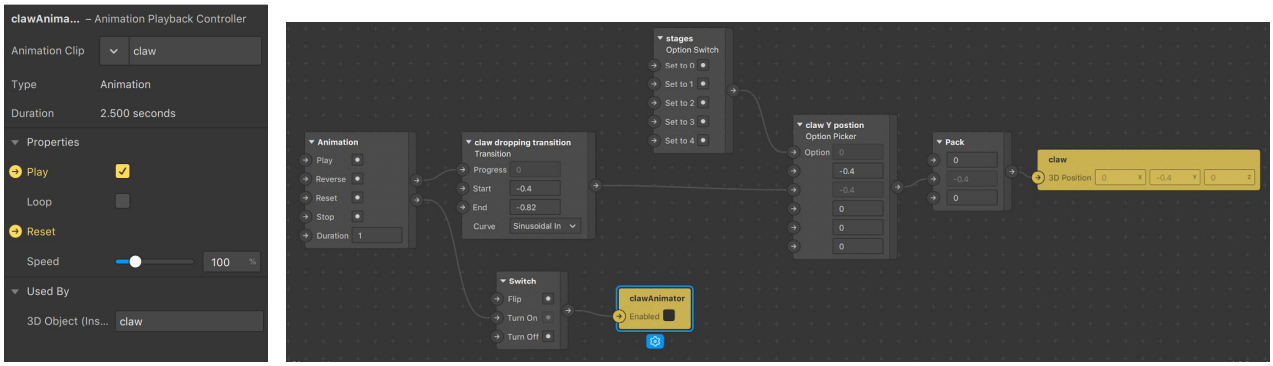




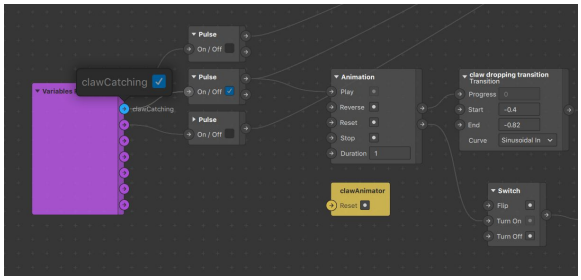
After setuping the animator, we want to assign it to the claw object in the Scene. You will see the claw is playing the catching animation.

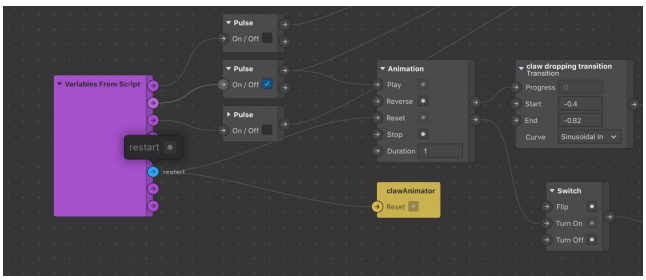


Next, let's navigate to the clawAnimator and bring **Reset** and **Play** into the patch editor by clicking the arrow next to the function, don't forget to disable the loop function since we just the animation to play once every time catching.
And we want the claw catching animation after it drops, so we **switch on** the **Play** after the dropping animation is completed.



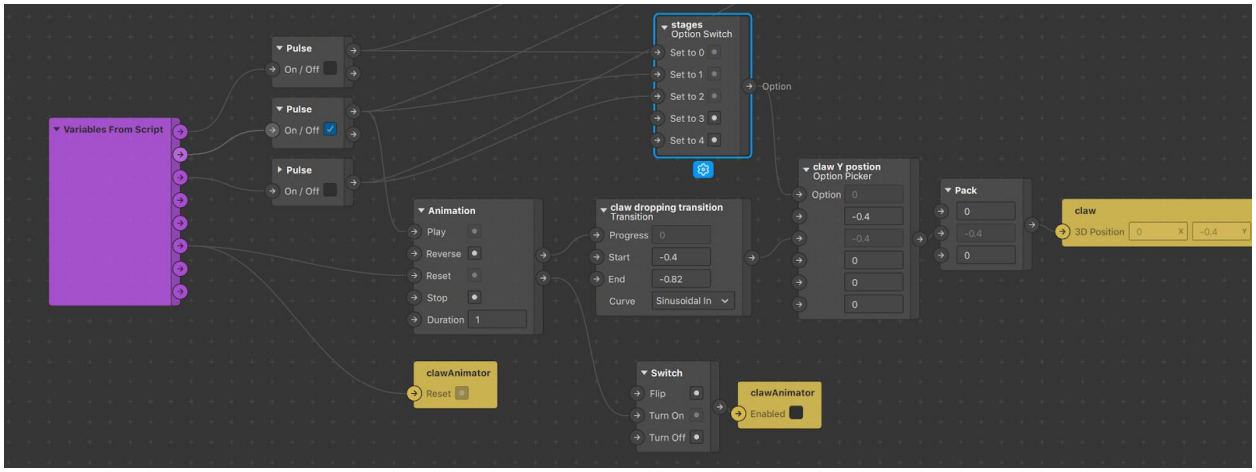
Connect the **clawCatching** condition from the script to **Play** the dropping animation, and **Restart** condition to **Reset** both the dropping animation and **clawAnimator**.





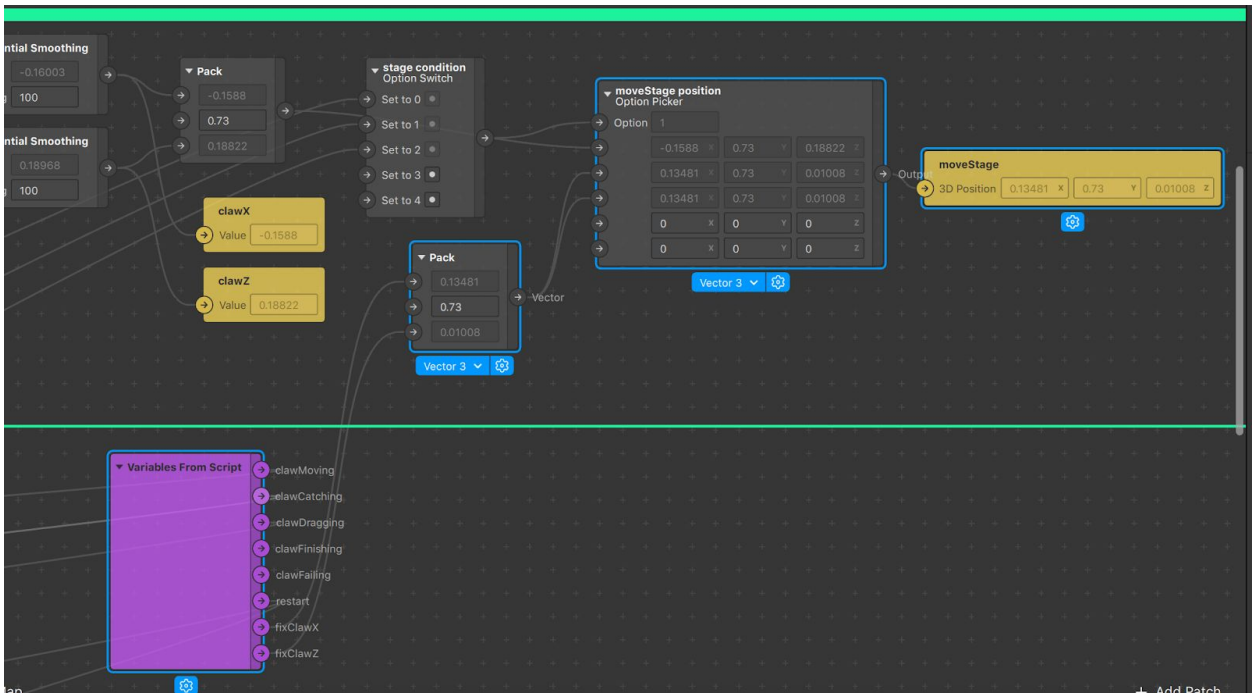
2.6 Connect Signals from the Script to Control Catching Stage

Connect the three stages from the script to switch the options for the claw Y position. If you preview the effect, you will find now the claw is moving following your face, and when you open your mouth, the claw will drop and play the catching animation. However, you will also notice that the movingStage is dropping with the claw, so we're gonna fix that problem in the next step.



2.7 Lock the moveStage position when start catching

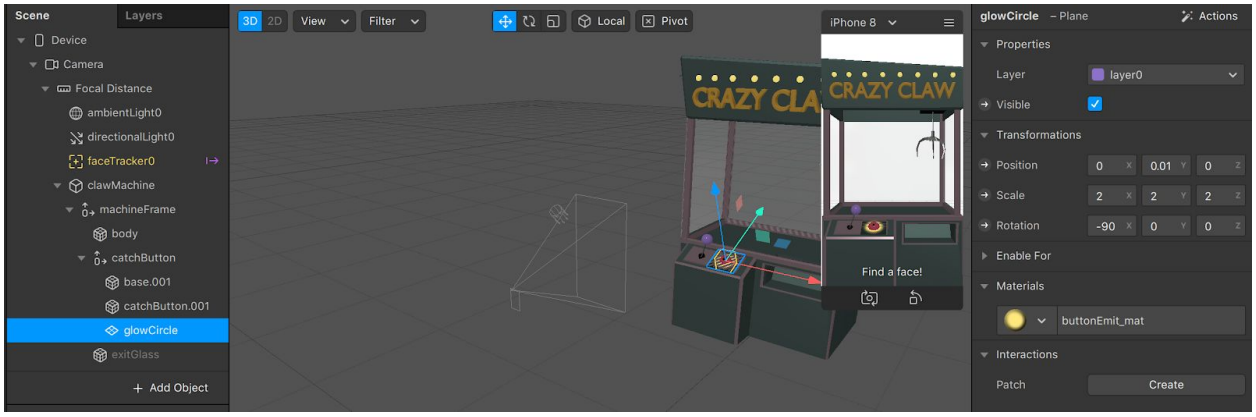
The moveStage's position needs to be locked when the user chooses a good spot and opens his mouth to trigger drop the claw. That is why we need variables **fixClawX** and **fixClawZ** to record the position. Since we already pass the real time clawX and clawZ patch values to the **fixClawX** and **fixClawZ** in the script function**,** what we need to do is connect the two values from script and set it as the moveStage position in the patch editor. And now if you try in the preview, you will find the moveStage is locked at the position where you stopped and dropped the claw.



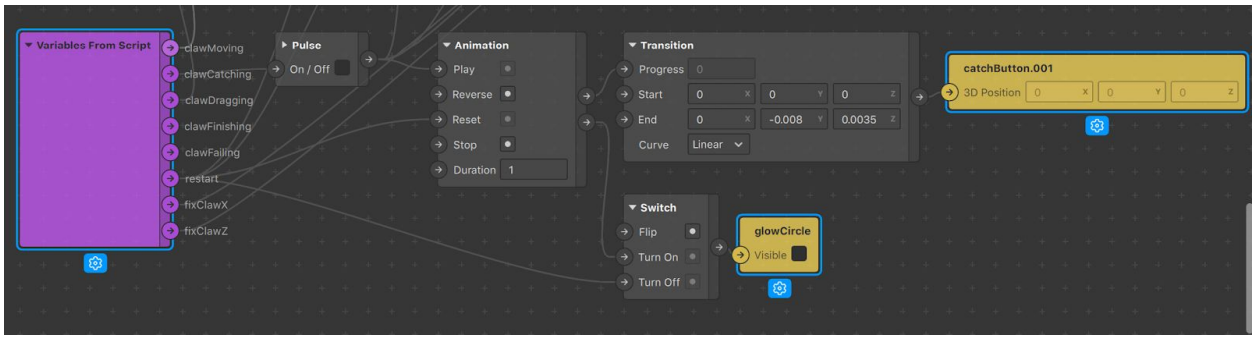
2.8 Play Button light up

Let's add a new **plane** object in the scene, rename it to **glowCircle**, and put it inside of clawMachine -> machineFrame -> catchButton, tweak the transformation (parameters are in the screenshot) to match the catchButton position. And also build a material by using

the roundEmit texture for it.



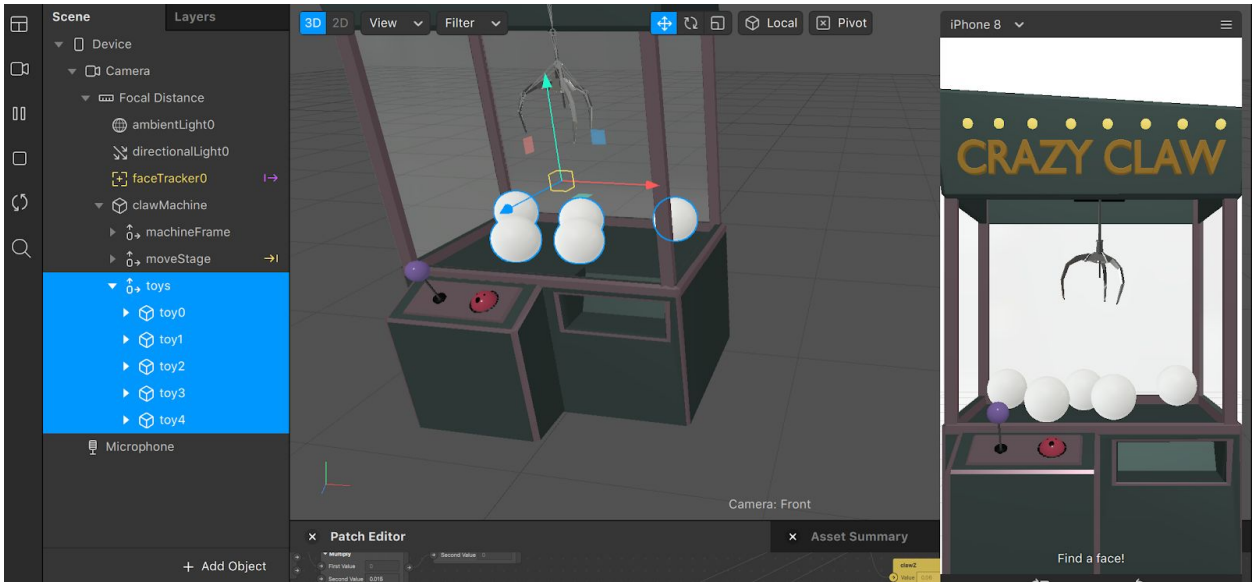
Then, we set the animation in the patch editor to make the button be pushed down and enable the glow object when the stage goes to catching and dropping the claw.



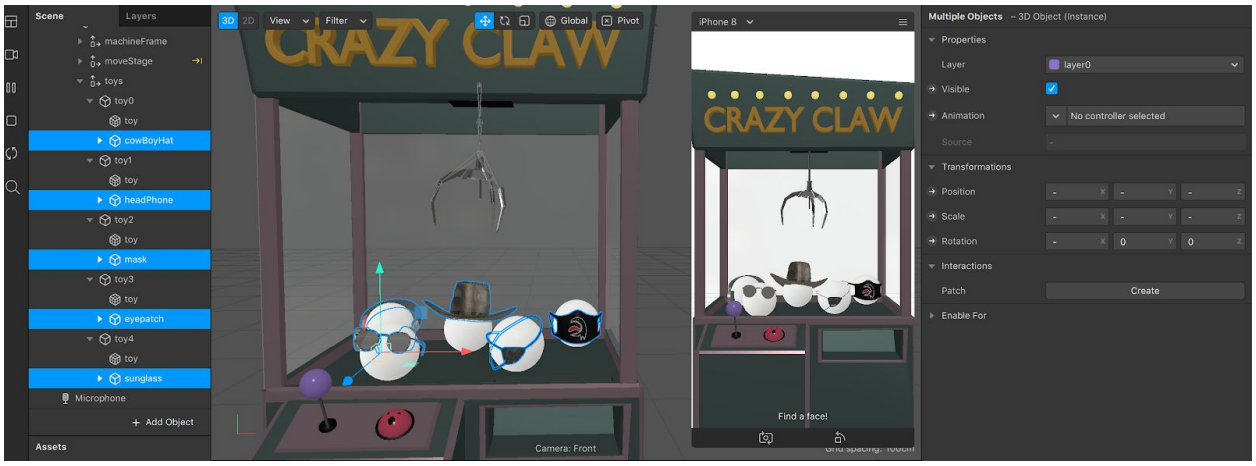
2.9 Drag toys into the scene

We know after dropping the claw, we actually want to catch our target toy. So firstly, let's set up the toys in the scene.

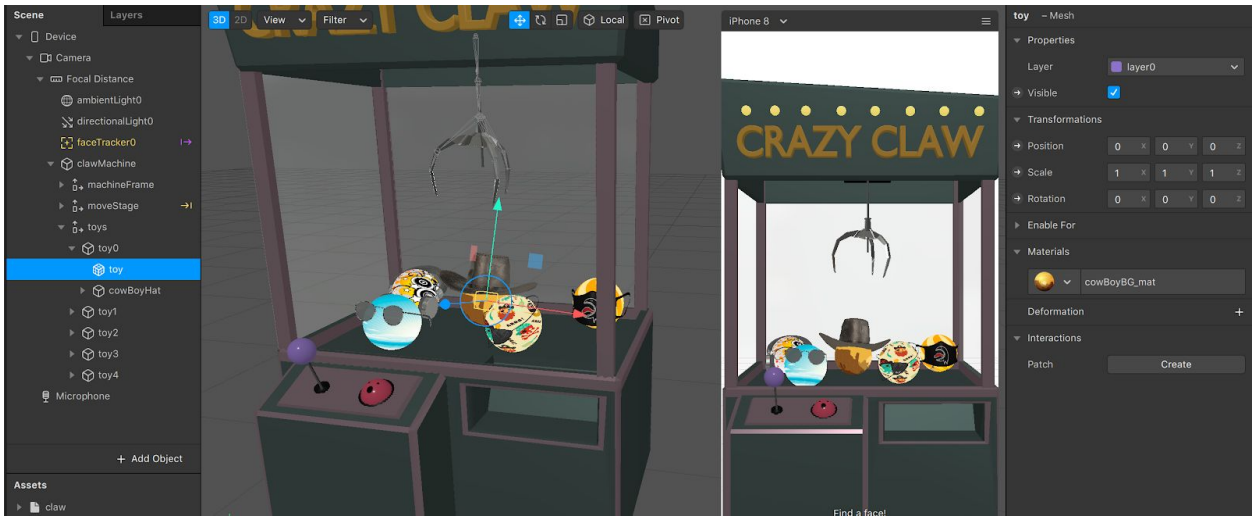
We add a **Null Object** and rename it to **toys**, put it under the clawMachine object. Then we drag the **toySphere** from the **Assets** to the scene and put it under **toys**. These spheres are holders of the accessories, we are gonna place five different toys in the machine, so let's repeat and place five **toySphere** in there. And we rename them followed by toy0, toy1, toy2, toy3, and toy4. Set each toySphere scale to (0.5, 0.5, 0.5), and tweak their positions to place them in the machine properly.



Choose accessories from the assets and insert them into each sphere holder. Adjust the transformation of the accessories to make them suit perfectly on the toySphere.



After that, we create corresponding materials for each sphere.



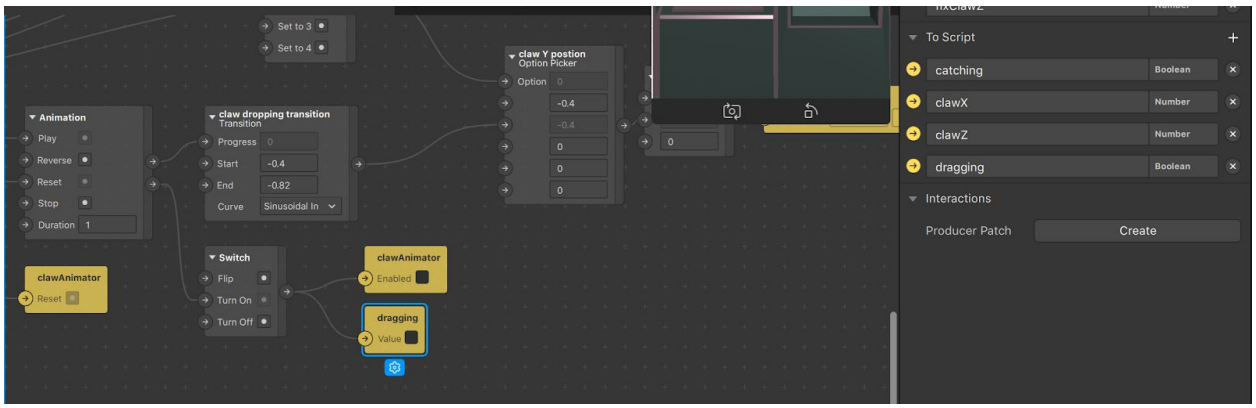
3. Dragging Stage

Function be covered in this stage:

- Write a script function to find the closest toy,
- Set minimum threshold to judge if succeed catching the toy or not
- Return the target toy index and drag the toy with the claw going up if succeed catching
- Return none if fail to catch a toy and the claw will go up alone

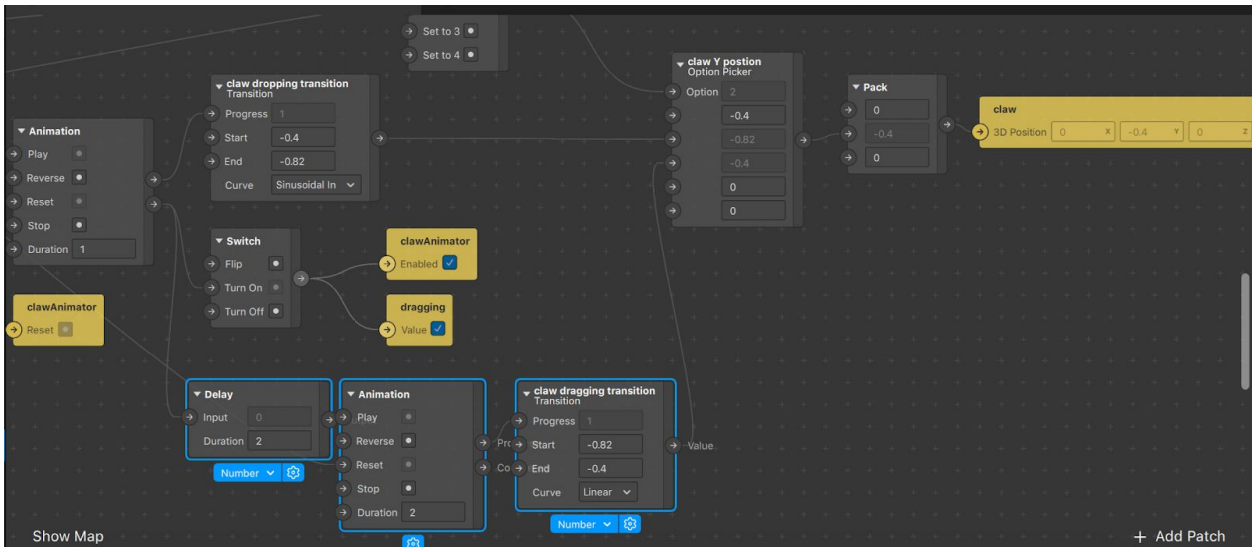
3.1 Create Dragging variable and send to script

Let's create a boolean variable called **Dragging** under the script.js **To Script**. Then we create the variable patch by clicking the arrow on the left. And we connect it in the patch editor to make it be triggered at the same time as the clawAnimator is enabled.



3.2 Set Claw dragging back motion

In the patch editor, set the animation to drag the claw up when it comes to the dragging stage. If you preview the effect, you will find the claw will go down and play the catching animation, and drag up after finishing the animation.



3.3 Get the Dragging signal from the Patch to the Script

Let's write some codes in the script.js to find the closest toy when the stage comes to dragging.

Get the dragging signal from the patch, and create a variable to hold the signal.

```
//get the clawDrag signal from the patch//
let clawDrag;
Patches.outputs.getBoolean("dragging").then((event) => {
  event.monitor().subscribe(function (values) {
    clawDrag = values.newValue;
  });
});
```

In the update function, write the module that is executed when it comes to the dragging stage.

```
function update(){
  //check the state then switch to catching//
  if (openMouth && currentState === "moving") {…
  }
  //lock the claw x and z position
  if (currentState === "catching" && preState === "movir
  }
  //go to claw Drag stage
  if (clawDrag && currentState === "catching") {
    setPatchState(false, false, true, false, false);
    checkCloseToy();
    preState = currentState;
    currentState = "dragging";
  }


  Time.setTimeout(update, 50);
}
```

3.4 Function to check the closest toy

Before writing the function to check the closest toy, define the global variables that will be needed in the function.

```
//variables for toys
let toys = [];
let targetToyNum = "targetToyNum";
let targetToy = null;
let originTargetToyY = 0;
let startClawHeight = 0;
let failToCatch = null;
```

Clarify the const that navigate to the objects in the scene.

```
/******* find object from the scene ******/
let clawMachine;
Scene.root.findFirst("clawMachine").then((result)=>{
  clawMachine = result;
})
let moveStage;
Scene.root.findFirst("moveStage").then((result)=>{
  moveStage = result;
})
let claw;
Scene.root.findFirst("claw").then((result)=>{
  claw = result;
});
```

Write the function to check the closest toy by calculating the distance between the claw position and each of the toys and find the closest one.

```
function checkCloseToy() {
  let minDis = 100;
  let moveStageX = moveStage.transform.x.pinLastValue();
  let moveStageZ = moveStage.transform.z.pinLastValue();
  let dis = 0;
  for (let i = 0; i < toys.length; i++) {
    let toyPosX = toys[i].transform.x.pinLastValue();
    let toyPosZ = toys[i].transform.z.pinLastValue();
    dis = Math.pow(moveStageX - toyPosX, 2) + Math.pow(moveStageZ - toyPosZ, 2);
    dis = Math.sqrt(dis);
    if (dis < minDis) {
      targetToy = i;
      minDis = dis;
    }
  }
  //judge if the closet toy is too far
  minDis < 0.08 && !toys[targetToy].hidden.pinLastValue()
    ? (failToCatch = false)
    : (failToCatch = true);
  Diagnostics.log(minDis + " " + failToCatch);
  if (!failToCatch) {
    toys[targetToy].transform.x = moveStageX;
    toys[targetToy].transform.z = moveStageZ;
    originTargetToyY = toys[targetToy].transform.y.pinLastValue();
    startClawHeight = claw.transform.y.pinLastValue();
    checkCaughtToy();
  }
}
```

3.5 Set and check the toy type

Since our accessories include the types that we wear on head, eyes and mouth, in order to not overlap the same type of accessories we classify them at the beginning and check the type everytime catch a new toy.

Define the global variables that will be needed in the function.

```
//varibles for checking repeat type of accessory
let usedHead, usedEye, usedFace;
let preType = null;
let removeHead = "removeHead";
let removeFace = "removeFace";
let removeEye = "removeEye";
```

Write the function to set the toy type one by one. Then execute it inside of **init** function

```
function setToys() {
  for (let i = 0; i < 5; i++) {
    let name = "toy" + i;
    let toy;
    Scene.root.findFirst(name).then((t)=>{
      toy = t;
      if (i == 0 || i == 1) {
      toy.type = "head";
    } else if (i == 2) {
      toy.type = "face";
    } else if (i == 3 || i == 4) {
      toy.type = "eye";
    }
    toys.push(toy);
    })
  }
  usedEye = false;
  usedFace = false;
  usedHead = false;
  preType = null;
}
```

```
  /******* function to init the experience ********/
function init() {
    //shoot pulse when the experience restart
    Patches.inputs.setPulse(restart, Reactive.once());
    //set the active stage to clawMoving at the beginning
    setPatchState(true, false, false, false, false);
    currentState = "moving";
    preState = currentState;
    //reset the claw postion to (0,0)
    Patches.inputs.setScalar(fixClawX, 0);
    Patches.inputs.setScalar(fixClawZ, 0);
    //set toy types
    setToys();
  }
```

Write the function to check the previous toy type

```
function checkCaughtToy() {
  if (toys[targetToy].type == preType) {
    switch (toys[targetToy].type) {
      case "head":
        Patches.inputs.setBoolean(removeHead, true);
        break;
      case "face":
        Patches.inputs.setBoolean(removeFace, true);
        break;
      case "eye":
        Patches.inputs.setBoolean(removeEye, true);
        break;
    }
  } else if (toys[targetToy].type != preType) {
    switch (toys[targetToy].type) {
      case "head":
        if (usedHead) Patches.inputs.setBoolean(removeHead, true);
        break;
      case "face":
        if (usedFace) Patches.inputs.setBoolean(removeFace, true);
        break;
      case "eye":
        if (usedEye) Patches.inputs.setBoolean(removeEye, true);
        break;
    }
  }
  if (toys[targetToy].type == "head") usedHead = true;
  if (toys[targetToy].type == "eye") usedEye = true;
  if (toys[targetToy].type == "face") usedFace = true;
  preType = toys[targetToy].type;
}
```

3.6 Drag the target toy

According to the above function, we will find the closest toy, then write the function to drag the target toy by mapping the claw position with the toy's position.

```
function dragTargetToy() {
  Patches.inputs.setScalar(targetToyNum, targetToy);
  let deltaHeight = claw.transform.y.pinLastValue() - startClawHeight;
  toys[targetToy].transform.y = originTargetToyY + deltaHeight + 0.03;
}
```

3.7 Update the status/stage

In the update function, write the module to go to the next stage after checking the closest toy.

```
function update(){
  //check the state then switch to catching//
  if (openMouth && currentState === "moving") {…
  }
  //lock the claw x and z position
  if (currentState === "catching" && preState === "moving") {…
  }
  //go to claw Drag stage
  if (clawDrag && currentState === "catching") {
    setPatchState(false, false, true, false, false);
    checkCloseToy();
    preState = currentState;
    currentState = "dragging";
  }
  //go to finishing state
  if (currentState === "dragging" && !failToCatch) {
    preState = currentState;
    currentState = "finishing";
  }
  //go to failing state
  if (currentState === "dragging" && failToCatch) {
    preState = currentState;
    currentState = "failing";
  }
  //drag the target toy
  if (currentState === "finishing" && preState === "dragging") {
    Time.setTimeout(dragTargetToy, 2000);
  }
```

3.8 Achievement

After these steps, when you try in the preview, you will find the target toy can be caught when it's close enough, otherwise, you will fail and catch nothing. Next it will come to our final stage setting, thank you for sticking till now, we are almost there!

4. Finishing Stage

Function be covered in this stage:

- Setup functions for both succeed and fail conditions in the script
- Improve the init function to reset the variables
- Setup the on face accessories in the scene
- Setup the background in the scene
- Get the caught toy index
- Choose the corresponding accessory according to the toy index and apply on face
- Choose and apply the corresponding background according to the toy index
- Set up the game restart function

4.1 Finishing Stage condition setup

In this finishing stage, let's firstly write two functions to set all the stages' states after it succeeds or fails to catch a toy. Then set up the finishing and failing condition inside the **update** to execute these two functions separately.

```
function clawFinish() {
  setPatchState(false, false, false, false, true);
  preState = currentState;
}

function clawFailCatch() {
  setPatchState(false, false, false, true, false);
  Time.setTimeout(init, 2000);
}
```

```
function update(){
  //check the state then switch to catching//
  if (openMouth && currentState === "moving") {…
  }
  //lock the claw x and z position
  if (currentState === "catching" && preState === "moving") {…
  }
  //go to claw Drag stage
  if (clawDrag && currentState === "catching") {…
  }
  //go to finishing state
  if (currentState === "dragging" && !failToCatch) {…
  }
  //go to failing state
  if (currentState === "dragging" && failToCatch) {…
  }
  //drag the target toy
  if (currentState === "finishing" && preState === "dragging") {
    Time.setTimeout(dragTargetToy, 2000);
    Time.setTimeout(clawFinish, 4000);
  }

  if (currentState === "failing" && preState === "dragging") {
    preState = currentState;
    Time.setTimeout(clawFailCatch, 4000);
  }
```

4.2 Improve the init function

Add codes to reset parameters when restarting the experience.

```
function init() {
    //shoot pulse when the experience restart
    Patches.inputs.setPulse(restart, Reactive.once());
    //set the active stage to clawMoving at the beginning
    setPatchState(true, false, false, false, false);
    currentState = "moving";
    preState = currentState;
    //reset the claw postion to (0,0)
    Patches.inputs.setScalar(fixClawX, 0);
    Patches.inputs.setScalar(fixClawZ, 0);
    //set toy types
    setToys();
    //put the target toy back
    if (targetToy !== null && !failToCatch) {
    //hide last time target
    toys[targetToy].hidden = true;
    targetToy = null;
    }
    originTargetToyY = 0;
    startClawHeight = 0;
    Patches.inputs.setScalar(targetToyNum, 100);
    failToCatch = null;
    //reset these variables
    Patches.inputs.setBoolean(removeHead, false);
    Patches.inputs.setBoolean(removeFace, false);
    Patches.inputs.setBoolean(removeEye, false);
}
```

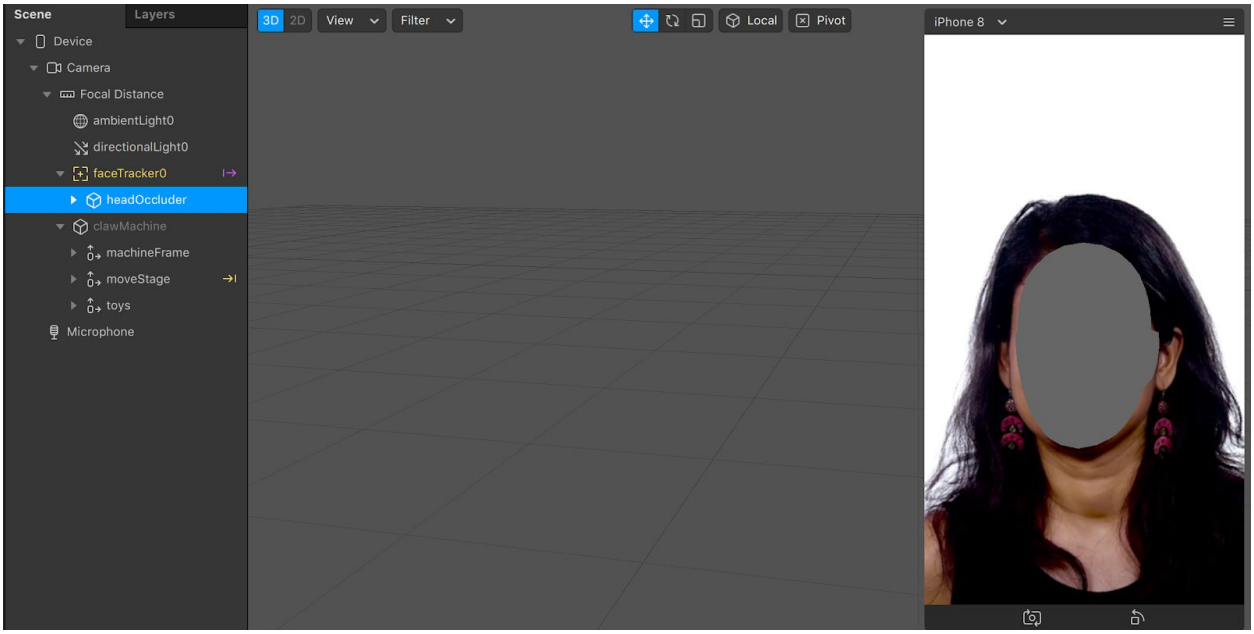
4.3 Set the accessories on face

The accessories we set above are for catching, and next we will set the accessories to apply on the face.

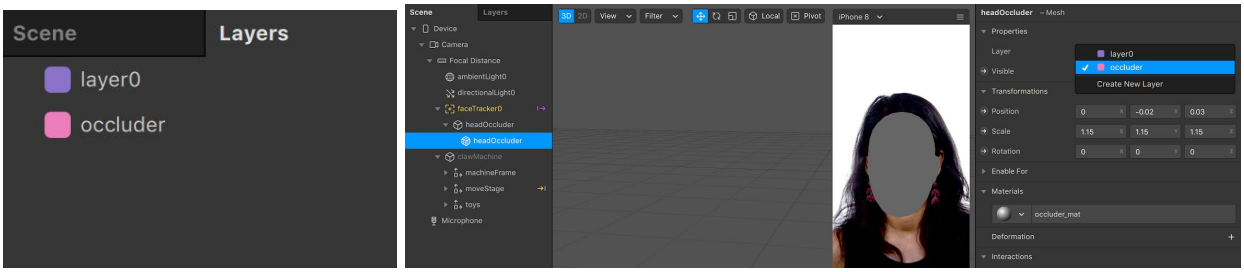
4.3.1 Add headOccluder

Firstly, try to download the headOccluder from the sparkAR reference assets.(https://sparkar.facebook.com/ar-studio/learn/articles/people-tracking/face-reference-assets#whats-included-in-the-face-reference-assets) and bring it in the assets.

Drag the headOccluder under the **faceTracker0** in the scene

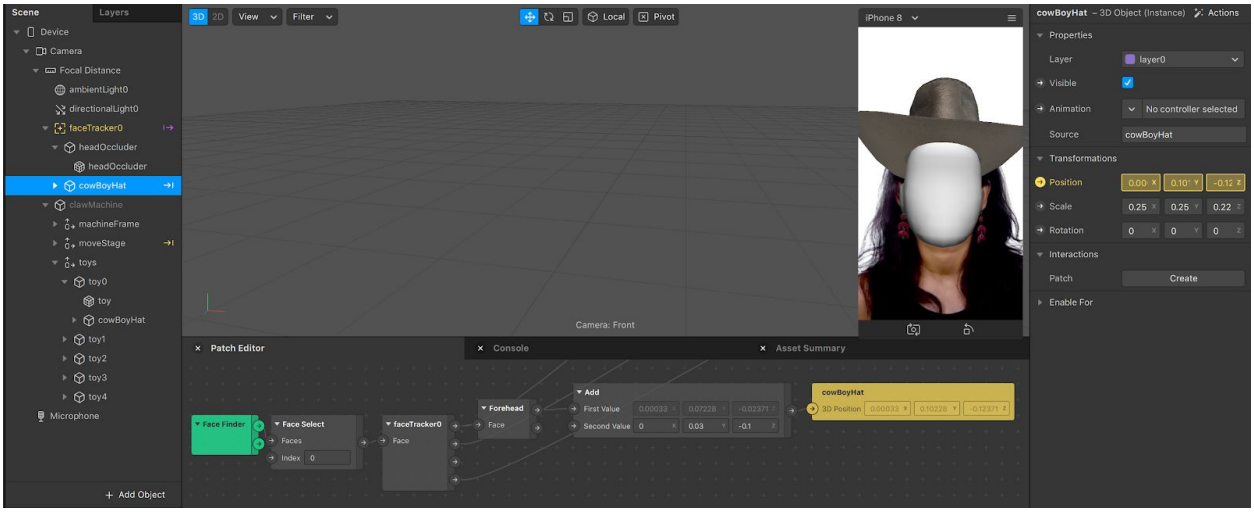


Then we build a new layer called occluder, put this layer underneath layer0, and assign the headOccluder layer to occluder. And you can tweak the headOccluder transformation to make it match more seamlessly with your head.

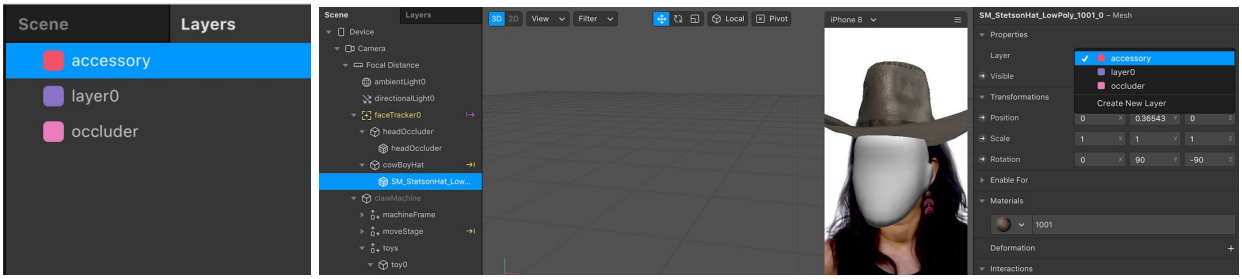


4.3.2 Add and Apply Accessory on Face

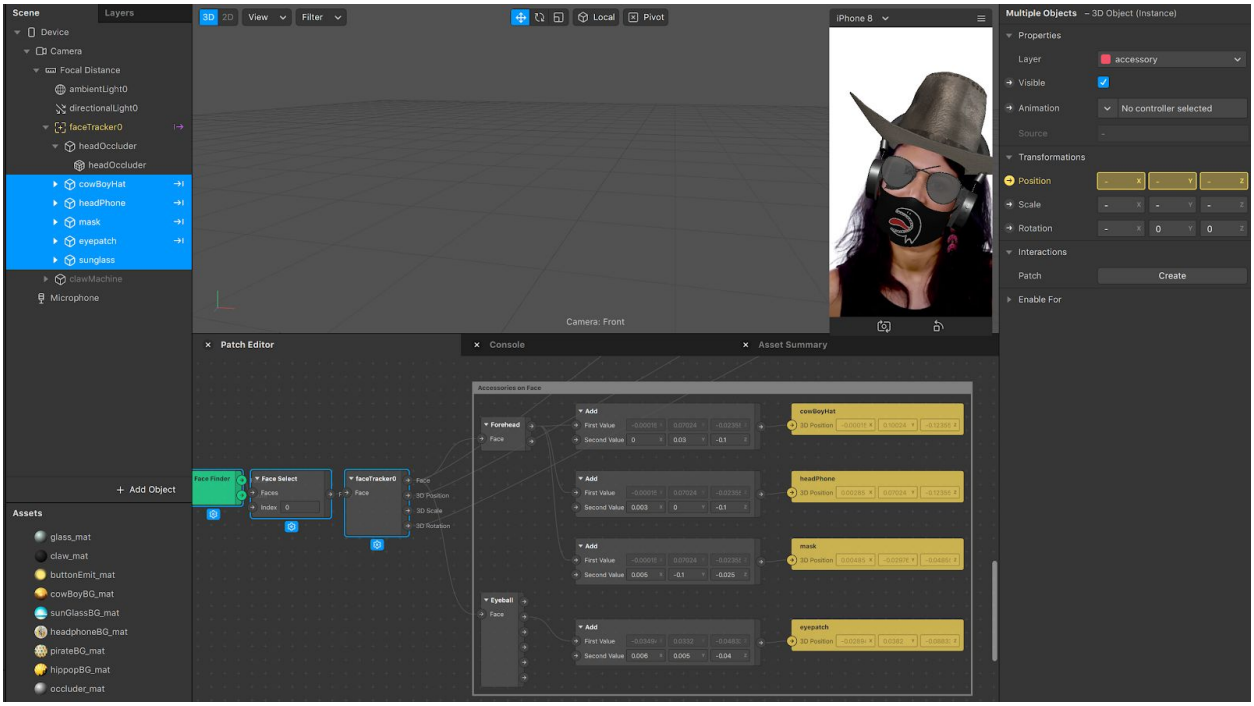
What we do next is bring the accessories that we put under the machine into the scene again under the **faceTracker0.** For example, we put the cowboy hat under the first toySphere, so we bring another cowboy hat from the asset into the scene under **faceTracker0**. Adjust the scale to (0.25,0.25,0.22). Then connect and apply some offset of the position with faceTracker's forehead in the patch editor.



Create a new **layer** called **accessory**, move the layer above the **layer0**. Then assign the cowboyHat object under this layer.

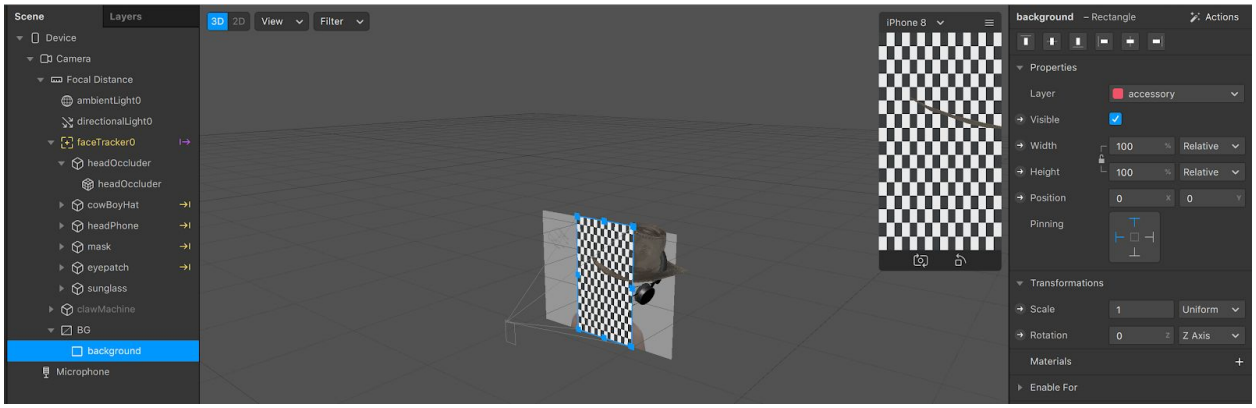


Repeat step 4.3.2, bring the rest 5 accessories into the scene under the faceTracker0, adjust the transformation to fit on the face properly and assign them under the accessory layer. After you feel them in good position, you can tweak the opacity of the headOccluder's material to 0.

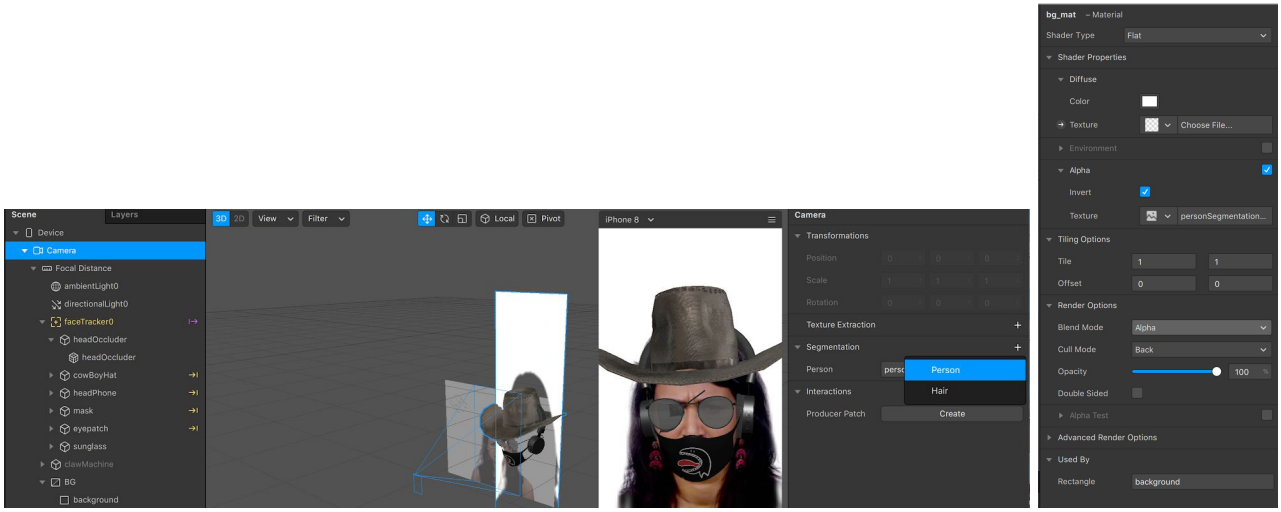


4.4 Add background

Add a **Rectangle** object in the scene, you will see a canvas with a rectangle be created. Let's rename the canvas as BG and the rectangle as background. Set the canvas to World space mode, change position to (0,0,-0.25), scale to(0.0011, 0.0011, 0.0011). Then set the background size as Fill Width and Fill Height.

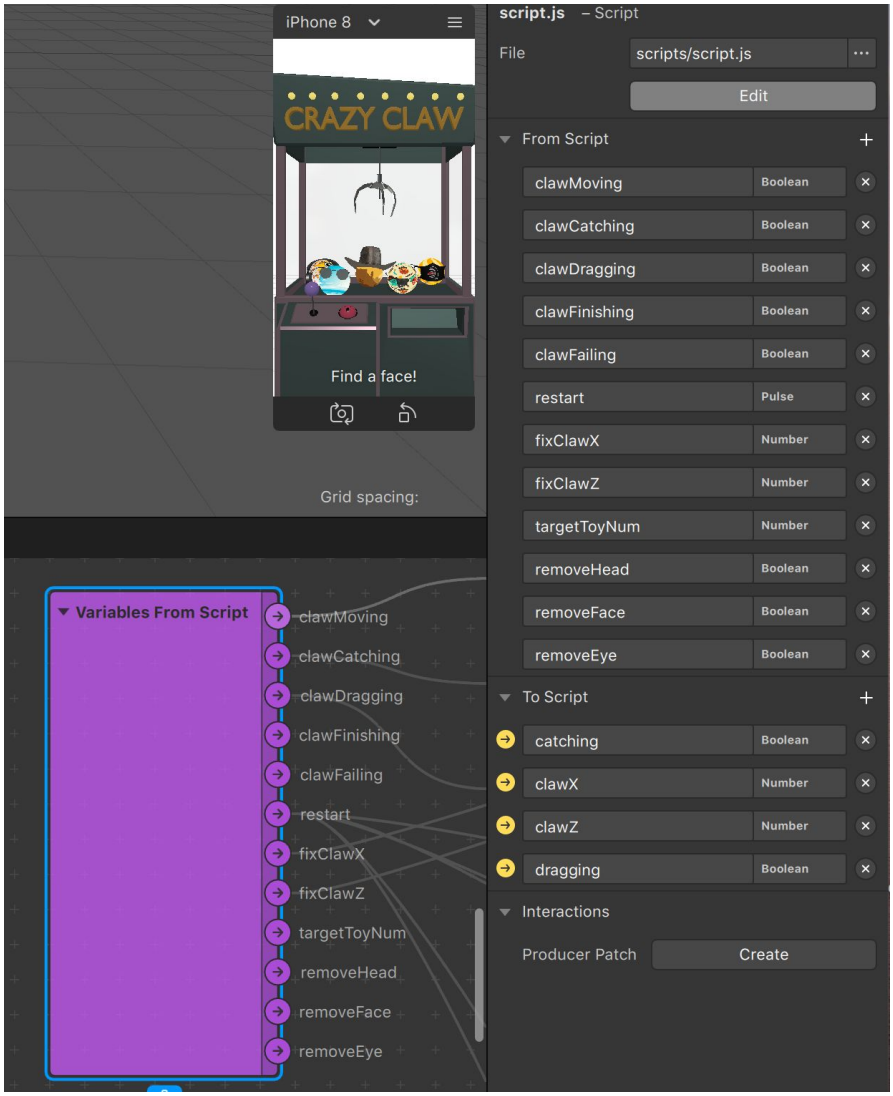


Click at the Camera object and generate a **personSegmentationMaskTexture** from it. Then create a new material for the background, set the shader type to flat and enable Alpha and invert. Assign the **personSegmentationMaskTexture** to the alpha texture.



4.5 Get the closest toy index From script

Create a **Number** variable called **targetToyNum** under the script.js **From Script**. And three **boolean** variables named **removeHead**, **removeFace** and **removeEye**. You will find these variables shown at the script patch.

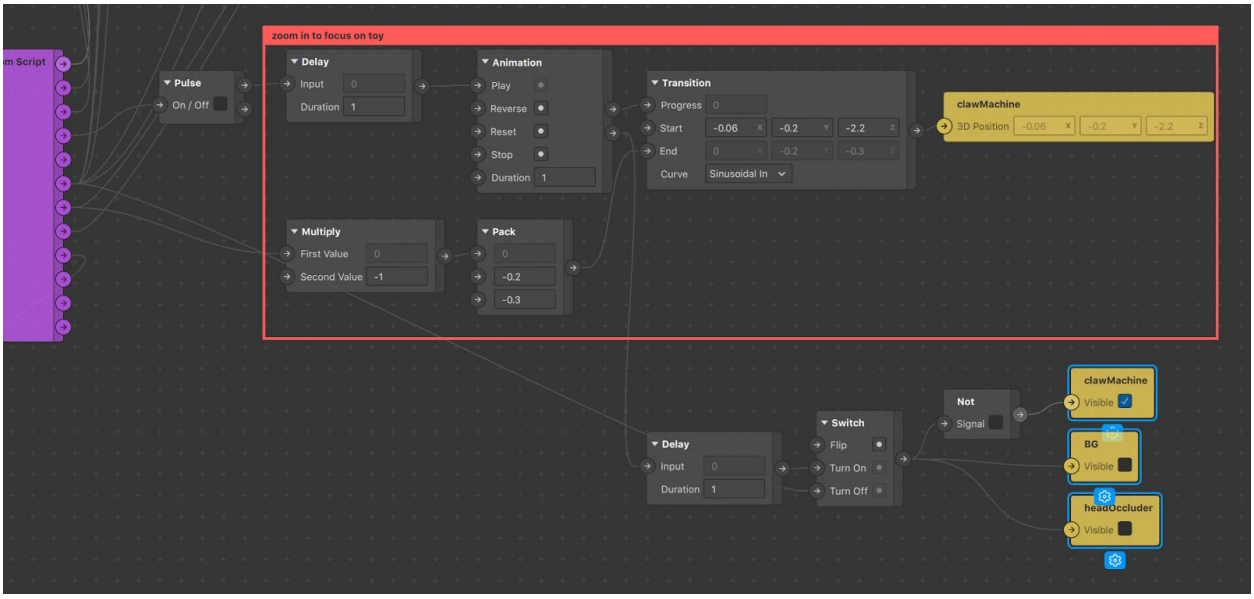


4.6 Build the logic to choose accessory and apply on face

4.6.1 Zoom in to focus on the toy

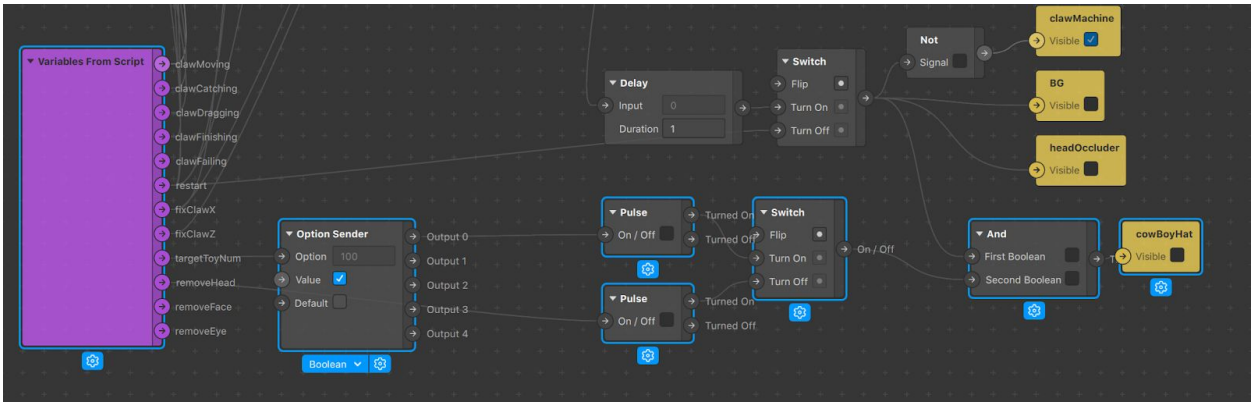
We use the finishing signal from the script to zoom in the claw machine position and make the camera focus on the toy that is caught.

Disable claw machine and enable BG and headOccluder after zoom in to focus on the caught toy.



4.6.2 Choose accessory and apply on Face

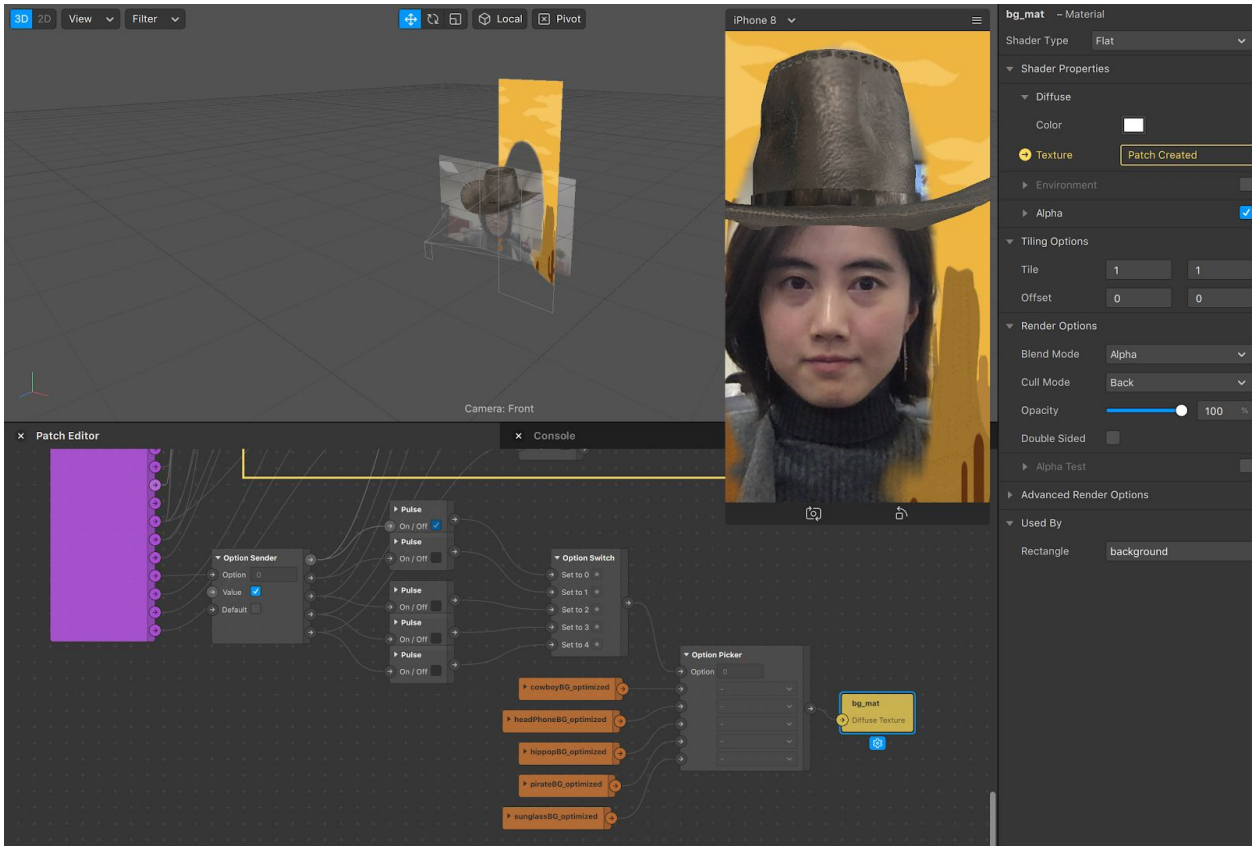
In order to enable the correct accessory and apply on the face, we need to check two main conditions. Firstly, if the headOccluder is enabled, secondly, the **targetToyNum**. For example, if the user catches the toy0, then the **targetToyNum** will be 0, the toy assigned under it is cowBoyHat. If the headOccluder is enabled also, then we can enable the corresponding toy cowBoyHat accessory on the face. Moreover, we need to check the **removeHead** condition from the script to determine if the accessory needed to be disabled because of repeating type.



Follow the rules above and we can finish setting the rest four accessories.

4.7 Choose environment texture and apply on the background

We also want the background texture to be changed according to the toy sphere's material that is caught. So we set an option picker for the background material texture, and switch between these texture options according to the **targetToyNum** from the script.

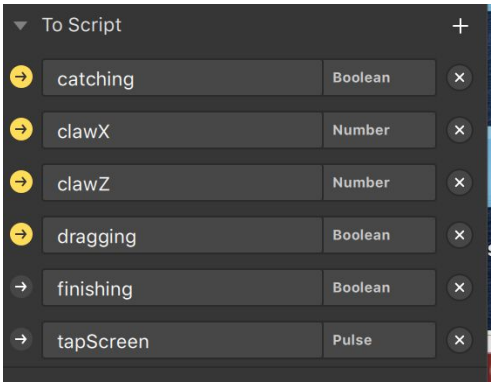


4.8 Game Finish and Restart

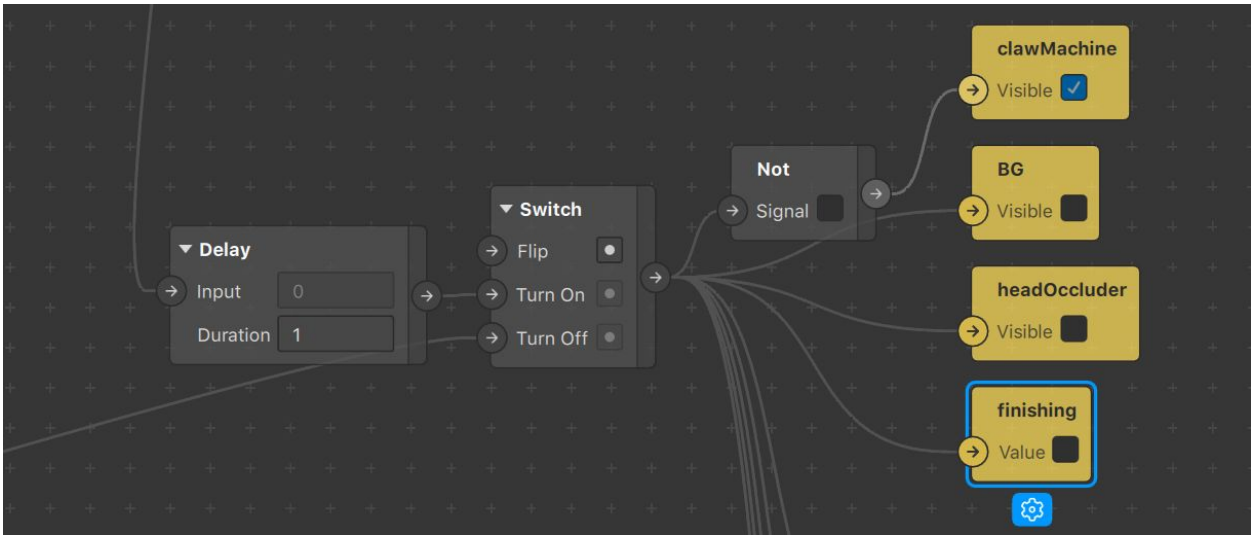
Until now, you have already finished one complete claw machine experience! Try in the preview to catch a toy and see how the accessory you caught is applied on your face. If you try more time, you will notice there seem to be some bugs for restarting the game, so we are gonna create some functions to complete the restart function.

4.8.1 Create variables send to Script

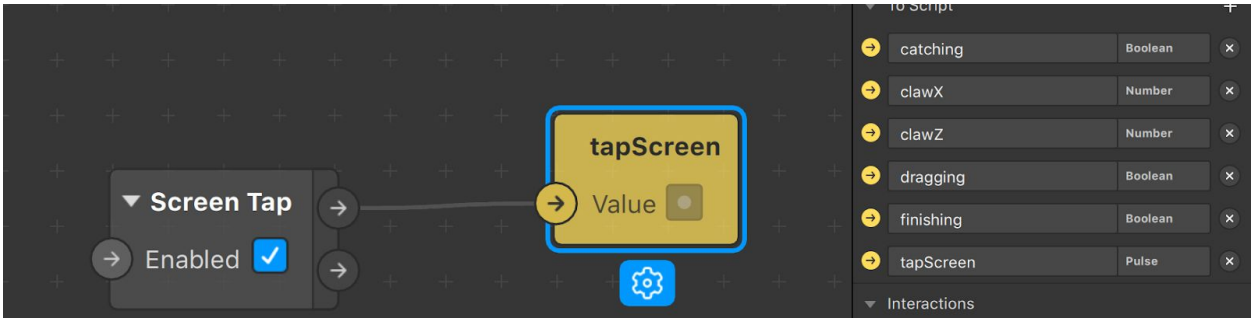
Click script.js in **Assets**, and in the inspector **To Script** panel, create a **Boolean** variable called **finishing** and a Pulse **boolean** named **tapScreen**.



Click the arrow at the left of the **finishing** variable and create a patch in the editor. And connect it with the same place when it turns on the headOccluder and the BG object.



Click the arrow at the left of the **tapScreen** variable and create the patch in the editor. Right click in the patch editor to find a patch called **Screen Tap.** Connect **Screen Tap** -> **Gesture State** with **tapScreen** variable patch.



4.8.2 Set function in Script for restart

```
//get the finishing signal from the patch
let gameFinish;
Patches.outputs.getBoolean("finishing").then((event) => {
  event.monitor().subscribe(function (values) {
    gameFinish = values.newValue;
  });
});

//get the tapScreen signal from the patch and detect if the game is finished
Patches.outputs.getPulse("tapScreen").then((event) => {
  event.subscribe(() => {
    if (gameFinish) {
      init();
    }
  });
});
```

What to go Next

Congratulations! If you've made it this far you are already a sparkAR expert! You will get how to plan a complete project from ideation to the development stage. This tutorial helps to realize the main function and interactions of the claw machine experience. In the finish project, I also add the background music along with the sound effects to indicate success and failure. Try yourself to add instructions, replace or add more toys, or some other interesting rewards or failure effects. Have fun and enjoy the journey!