NovCon Game Beta Writeup

Gravity Affect

Austin M, Sathwik

2.13.2022

Summary

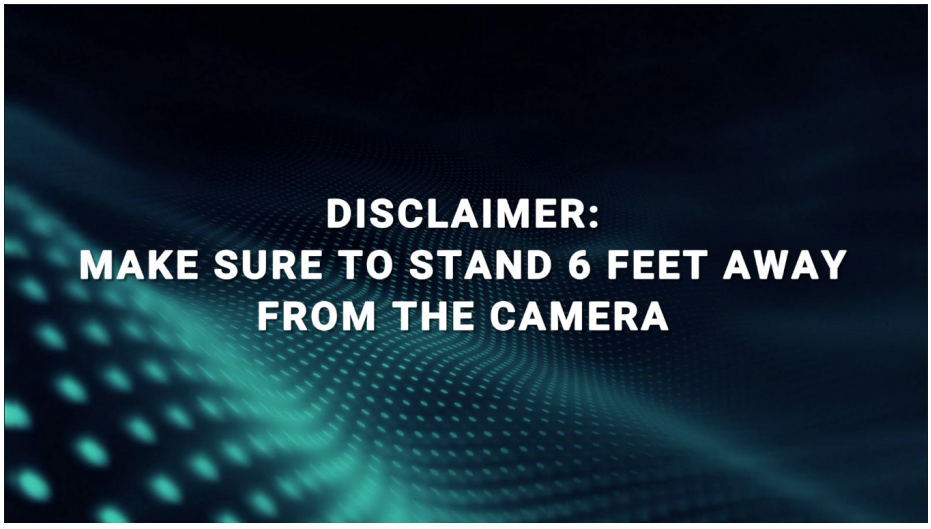
Gravity Effect is a top down 3D maze and strategy game with a futuristic neon theme. The player must control their self-illuminating ball by tilting the maze with their hands (moving the gravity affected ball) as they traverse the maze and attempt to complete the three levels, avoiding the incorrect pathways so as to complete the maze with haste. With this, the player will also have an element of exploration as different sections of the maze will be completely dark when the player is not moving through the corridors of the maze. As each level of the maze is completed, the maze progressively gets more challenging, offering more narrow corridors for the player to move through as they move their hands to rotate the maze within the world.

Functions

Disclaimer

Throughout the development of the game, Austin and I ran into a reoccurring problem of distance mapping with the Kinect Controller. We finally found the perfect distance from the Kinect Controller that a person had to stand in order for the depth sensors to function correctly. We found that distance to be 6 feet away from the Kinect controller. In order to make sure that the player of our game would have the most optimable and enjoyable experience, we decided to add a Disclaimer screen to our game that loads in before the title screen to ensure the player is calibrated correctly to the game.

The picture in the bottom shows our Disclaimer screen in out game



Title

Our Title Screen features the standard Play, Options, and Quit buttons. The screen is very simple and straightforward so the player can choose what they want to do very quickly and get to playing our game. The title screen was difficult to code at first, in which we wanted the player to have the ability to move their right hand and select an option (Play, Options, and Quit), but quickly found that our preferred method of closing the hand to confirm the player's selection was outdated with the Kinect SDK version which we were using for the Unity project. To adjust to this issue, we changed the method of selection to be the player moving their hand forward (until about 6 feet away from the Kinect); however, this also came with its issues.
The largest issue which we found while testing was that it was difficult to distinguish which button was selected before the player pushed their hand inwards to confirm selection. To fix this, we wrote code which locks the rightHand ball in place next to the option which the right hand dot (separate from the rightHand ball, which is transformed to move in relation to the raw right hand dot's position) hovers over. When the button we want is pushed, we activate and deactivate game objects. Even though our method of switching is abstract, the efficiency in performance is unmatched as the game does not have to load in all assets in different scenes and build those scenes.

The following code snippet below highlights the way our title screen works

```
private void UpdateBodyObject(Body body, GameObject bodyObject)
{
 bool leftFound = false;
 bool rightFound = false;

 Vector3 leftDot = new Vector3(0,0,0);
 Vector3 rightDot = new Vector3(0,0,0);
 foreach (JointType _joint in _joints)
 {
  // Get new target position
   Joint sourceJoint = body.Joints[_joint];
   Vector3 targetPosition = GetVector3FromJoint(sourceJoint);
   Vector3 targetPosition2D = GetVector3FromJoint(sourceJoint);
   targetPosition2D.z = 0;

   // Get joint, set new position
   Transform jointObject = bodyObject.transform.Find(_joint.ToString());
   jointObject.position = targetPosition2D;
```

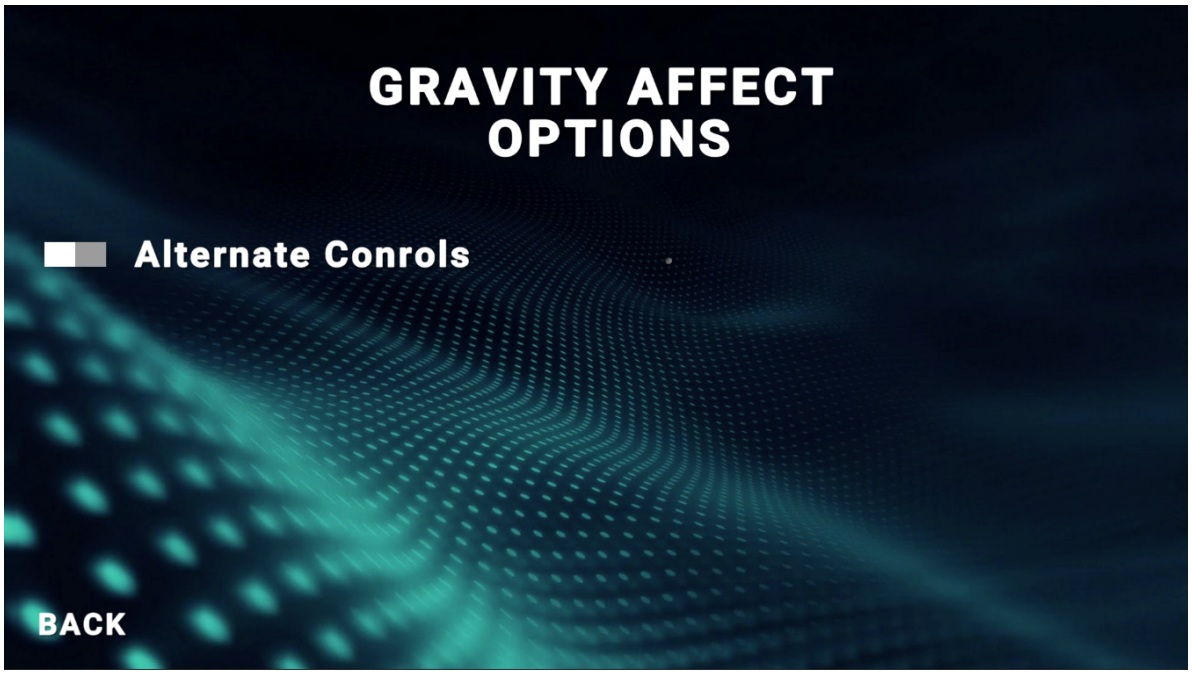
```
  if (_joint.ToString() == "HandLeft")
  {
   leftFound = true;
   leftDot = targetPosition;
   }
  if (_joint.ToString() == "HandRight") {
     rightFound = true;
     rightDot = targetPosition;
     }

     if (leftFound && rightFound)
     {
      rightHand.transform.position = new Vector3(rightDot.x * multiplier, rightDot.z + 3, rightDot.y *
multiplier);
      //leftHand.transform.position = new Vector3(leftDot.x * multiplier, leftDot.z + 3, leftDot.y * multiplier);
//disables the movement of the left hand
      //Debug.Log(rightDot.y * multiplier);
      if((rightDot.x * multiplier) > 10f && (rightDot.x * multiplier < 28f)){
      if(rightDot.y * multiplier > -4f && rightDot.y * multiplier < 5f){
      rightHand.transform.position = new Vector3(21f, 10f, 2.5f);
       if((rightDot.z+3) <= 19f){
        if(!tooSoon){
         manager.GetComponent<LoadingScript>().startTutorial();
        }
        //SceneManager.LoadScene("TutorialDown");
       }
      }
      else if(rightDot.y * multiplier > -12f && rightDot.y * multiplier < 4f){
      rightHand.transform.position = new Vector3(17.5f, 10f, -8f);
       if((rightDot.z+3) <= 19f){
        if(!tooSoon){
        manager.GetComponent<LoadingScript>().loadOptions();
        }
```

```
      //tutorialDown.active = true;
      //manager.SetActive(false);
     }
    }
    else if(rightDot.y * multiplier > -22f && rightDot.y * multiplier < -12f){
    rightHand.transform.position = new Vector3(21f, 10f, -18.5f);
    if((rightDot.z+3) <= 19f){
     if(!tooSoon){
      Application.Quit();
     }
    }
   }
  }
  /*Debug.Log(rightDot.y);
  Debug.Log(rightDot.z);*/
 }
 else if(!rightFound){ //called if onne of the hands is not found
 //change the position of the right hand to the center of the screen
 rightHand.transform.position = new Vector3(0, 3, 0);
 }
}
}
```

Options

Our Options menu does not look too complex at first. However, our options menu has a very complex system in place known as DOTween. This extension allows an object to translate betwee two different states of being without an animate in place and this improves the speed of our workflow as well as the speed of the game because it does not need to load in a different animation. Instead, the toggle switch can simply be two objects, the background and the actual button image, and with the help of DOTween, our switch can act as a switch without animation. We integrated this state toggle switch with our alternate method of movement option (which is explored in more depth in the next section) so that the player can choose with type of movement he or she wants. We also have a back button that just switches the scene back to the title scene. We hope to add extra functionality, such as volume level, in the future of the game.

Below is our Options Menu



Tilting Methods (Both types)

Overall, the game has come along very well, showing full completion of the main movement of the board and the main menu selection screen. Together, Sathwik and I have completed all four movement types for the maze board, allowing the player to move the ball throughout the entire maze. This was completed through tangent methods measuring the depth between the player's hands and a constant multiplied by the average distance the two hands may be raised or lowered out of a dead zone(where the hands would register no movement).

Initially, we experienced an issue with the code when trying to make the board more sensitive to movements the player makes (so as not to tire the player while playing the game), but this came with an increasing number of errors due to the large amount of unsimplified code - making it more difficult to read if there was a need to adjust the code. To avoid this, we restructured the entire code so we could set bounds easily and test how smoothly the board moved through the adjustments - finding a working solution within two hours of implementing this.

Within Gravity Affect, the game features two core types of movement - which can be toggled to the user desires. The alternate controls use a different perspective on the coordinate plane to create the same four movements of the normal controls. Moving the left hand up and out of the deadzone and moving the right hand down and out of the deadzone moves the maze up, and vice versa. The controls that turn the maze side to side stay put as those controls match perfectly withthe intended movements. The alternate controls are only used for layers who think that out normal controls are not sufficient enough for their gaming experience so they would prefer alternate controls.

Below is a snippet of the tilting code

```
  void FixedUpdate(){

    bool bothHandsInDZ = leftHand.transform.position.z > DZBottom && leftHand.transform.position.z < DZTop && rightHand.transform.position.z > DZBottom && rightHand.transform.position.z > DZTop;
    bool bothHandsUp = leftHand.transform.position.z >= DZTop && rightHand.transform.position.z >= DZTop;
    bool bothHandsDown = leftHand.transform.position.z <= DZBottom && rightHand.transform.position.z <= DZBottom;
    bool rightHandForwardOnly = leftHand.transform.position.y >= minDepth && rightHand.transform.position.y <= depthToSetOffForward;
    bool leftHandForwardOnly = rightHand.transform.position.y >= minDepth && leftHand.transform.position.y <= depthToSetOffForward;

    bool rightHandUp = rightHand.transform.position.z >= DZTop && leftHand.transform.position.z <= DZBottom;
    bool leftHandUp = leftHand.transform.position.z >= DZTop && rightHand.transform.position.z <= DZBottom;

    if(!alternateControl && bothHandsUp){ //both up
      float turnDegree = -50f + (40f * ((leftHand.transform.position.z + rightHand.transform.position.z) / 2f))-112f;
      if(turnDegree < -45){ //turn to degree if not too large
        EntireMaze.transform.rotation = Quaternion.Euler(turnDegree, 0, 180);
      }
      else{ //if turn too large, set to max
        EntireMaze.transform.rotation = Quaternion.Euler(-45, 0, 180);
      }
    }

    else if(!alternateControl && bothHandsDown){ //both down
      float turnDegree = 400f - (40f * (1f - ((leftHand.transform.position.z + rightHand.transform.position.z) / 2f)));
```

```
        if(turnDegree > 225){ //turn to degree if not too large
            EntireMaze.transform.rotation = Quaternion.Euler(turnDegree, 0, 180);
        }
        else{ //if turn too large, set to max
            EntireMaze.transform.rotation = Quaternion.Euler(225, 0, 180);
        }
    }

    else if(!alternateControl && ((leftHand.transform.position.z > -4f && leftHand.transform.position.z < 2.8f) && leftHand.transform.position.y <= 19f) && (rightHand.transform.position.y > .9f && (rightHand.transform.position.z > -4f && rightHand.transform.position.z < 2.8f))){
        float turnDegree = 2f * (230f + Mathf.Rad2Deg*Mathf.Atan((leftHand.transform.position.x - rightHand.transform.position.x) / (leftHand.transform.position.y - rightHand.transform.position.y)));
        if(turnDegree > 575){
            EntireMaze.transform.rotation = Quaternion.Euler(turnDegree+10, 90, 90);
        }
        else{
            EntireMaze.transform.rotation = Quaternion.Euler(225, 90, 90);
        }
        Debug.Log(turnDegree);
    }

    else if(!alternateControl && ((rightHand.transform.position.z > DZBottom && rightHand.transform.position.z < DZTop) && rightHand.transform.position.y <= 19f) && (leftHand.transform.position.y > .9f && (leftHand.transform.position.z > -4f && leftHand.transform.position.z < 2.8f))){
        float turnDegree = 2f * (45f + Mathf.Rad2Deg*Mathf.Atan((leftHand.transform.position.x - rightHand.transform.position.x) / (leftHand.transform.position.y - rightHand.transform.position.y)));
        if(turnDegree < -45){
            EntireMaze.transform.rotation = Quaternion.Euler(turnDegree-10, 90, 90);
        }
        else{
            EntireMaze.transform.rotation = Quaternion.Euler(-45, 90, 90);
        }
```

```
        //Debug.Log("Tilt Right");
    }

    else if(alternateControl && rightHandUp){ //tilt left - alternate
        float turnDegree = -3f * (225 + Mathf.Rad2Deg*Mathf.Atan((rightHand.transform.position.z -
leftHand.transform.position.z) / (rightHand.transform.position.x - leftHand.transform.position.x)));
        if(turnDegree > -835){
            EntireMaze.transform.rotation = Quaternion.Euler(turnDegree-10, 90, 90);
        }
        else{
            EntireMaze.transform.rotation = Quaternion.Euler(225, 90, 90);
        }
        Debug.Log(turnDegree);
    }

    else if(alternateControl && leftHandUp){ //tilt right - alternate
        float turnDegree = -3f * (Mathf.Rad2Deg*Mathf.Atan((rightHand.transform.position.z -
leftHand.transform.position.z) / (rightHand.transform.position.x - leftHand.transform.position.x)));
        if(turnDegree < 135){
            EntireMaze.transform.rotation = Quaternion.Euler(turnDegree+180, 90, 90);
        }
        else{
            EntireMaze.transform.rotation = Quaternion.Euler(-45, 90, 90);
        }
        Debug.Log(turnDegree);
    }

    else if(alternateControl && ((rightHand.transform.position.z > DZBottom &&
rightHand.transform.position.z < DZTop) && rightHand.transform.position.y <= 19f) &&
(leftHand.transform.position.y > .9f && (leftHand.transform.position.z > -4f && leftHand.transform.position.z
< 2.8f))){
        //tilt up - alternate
        float turnDegree = 2f * (230f + Mathf.Rad2Deg*Mathf.Atan((leftHand.transform.position.x -
rightHand.transform.position.x) / (leftHand.transform.position.y - rightHand.transform.position.y)));
```

```
        if(turnDegree < -45){ //turn to degree if not too large
            EntireMaze.transform.rotation = Quaternion.Euler(turnDegree, 0, 180);
        }
        else{ //if turn too large, set to max
            EntireMaze.transform.rotation = Quaternion.Euler(-45, 0, 180);
        }
        Debug.Log("Tilt up");
    }

    else if(alternateControl && ((leftHand.transform.position.z > -4f && leftHand.transform.position.z < 2.8f)
&& leftHand.transform.position.y <= 19f) && (rightHand.transform.position.y > .9f &&
(rightHand.transform.position.z > -4f && rightHand.transform.position.z < 2.8f))){
        //tilt down - alternate
        float turnDegree = 3f * (45f + Mathf.Rad2Deg*Mathf.Atan((leftHand.transform.position.x -
rightHand.transform.position.x) / (leftHand.transform.position.y - rightHand.transform.position.y)));
        if(turnDegree > 310){ //turn to degree if not too large
            EntireMaze.transform.rotation = Quaternion.Euler(turnDegree-80, 0, 180);
        }
        else{ //if turn too large, set to max
            EntireMaze.transform.rotation = Quaternion.Euler(225, 0, 180);
        }
        Debug.Log(turnDegree);
        Debug.Log("Tilt Down");
    }
    else{
        EntireMaze.transform.rotation = Quaternion.Euler(-90, 90, 90);
        //Debug.Log("DeadZone - set rotation to zero");
    }
    EntireMaze.transform.position = new Vector3(0, 1, 0); //locks around single point of rotation
    //Debug.Log(leftHand.transform.position.y); //15.6 resting
  }
```

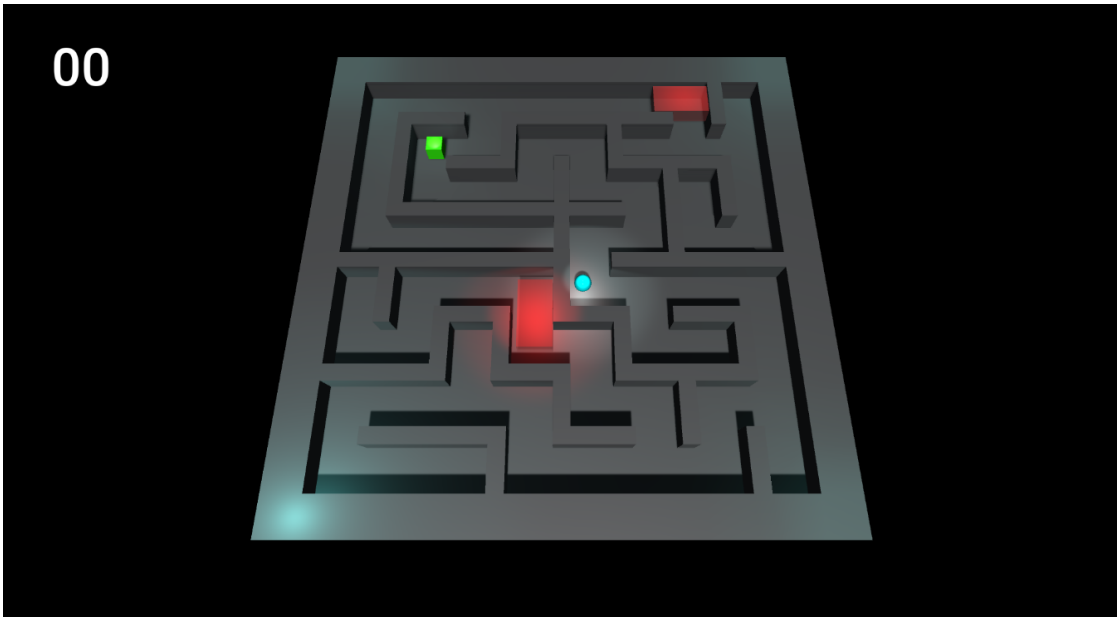
Tutorial Levels

To start the game, the player must move through different tutorial levels so they can understand the controls of the game before being thrown into the actual game levels immediately. In order to do allow this learning stage, we created four different levels to demonstrate the four different styles of movement the player can make: Up, Down, Left, and Right. These scenes are fully illuminated, allowing the player to see the entire tutorial board so they know the exact outputs of their inputs.

Level Doors/Exits

Within every level is a door: an object which the ball must collide with in order to move to the next level. Although simple, this is one of the most important features of the entire game as the player cannot be stuck on a single level for the project to be considered a game. To make the door more apparent, we made it glow green within the maze, giving the player a general sense as to where the door is without explicitly providing a direct route to the object.

Below is an image of the actual door within the first level.



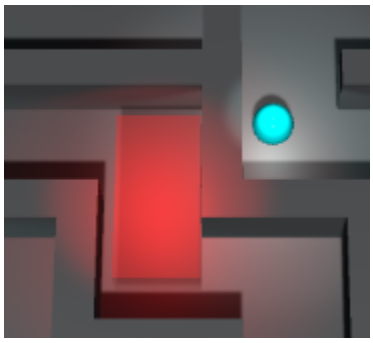
Timer

Our Timer is variable from level to level. The timer in level one is 20 seconds, the timer in level two is 45 seconds, and the timer in level three is 60 seconds. The timer uses Time.deltaTime to iterate through the seconds and print on the canvas of each level. Instead of ending the game when the timer hits zero, we decided to restart the level and the timer on level one, and restart the previous level on level two and onwards, and reset the level timer. This adds a second dimension to the challenge as no one wants to restart a level or lose progress of the levels they have just gone through. The timer itself adds an increase in the presence of mind because the player cannot just go through the levels without any challenge whatsoever.

Danger Zones

As we completed the rest of the maze, we added an element to the game which prevents the game from being a trivial completion of three mazes. In order to change the boring gameplay, we planned to add lighted zones which the player must try to avoid, causing them to lose if the player moves the ball into the zone for too long. This was implemented by creating a cube 3D object, making it partially transparent, then adding an OnTriggerEnter and OnTriggerExit method to start a timer to see if the ball is within the zone for too long.

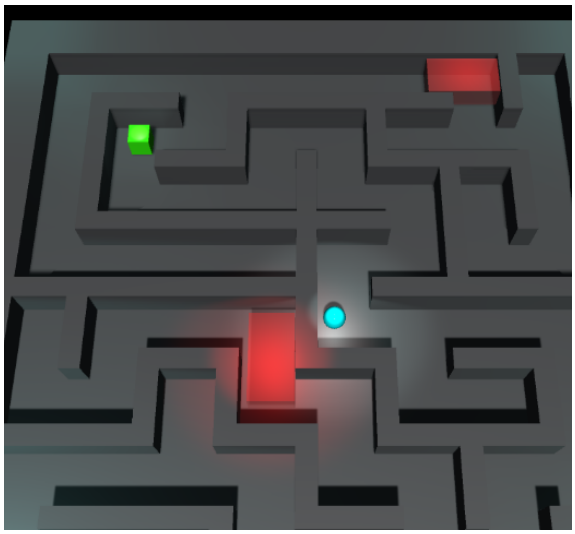
Below is a danzer zone



Lighting

Our lighting uses a mixed baked lightmap that saves GPU power and makes the performance efficient and streamlined. The lighting we used sets up a cool dark theme that is highlighted in the summary. The game uses a baked lightmap because we did not want the GPU intensive game to be even mor intensive just to make the lights work. Baked lightmaps ensure that the light is always similarly baked to the original lightmap, so the data of the realtime updating lightmap would not really make a difference since the lights are static. We have global illuminations everywhere. From the door, ball, and the danzer zones, to the actual maze, we have static global illumination providing lighting for the game.

Below is an image of the lighting



Technical References

Setting up Kinect with Windows (Youtube)

https://www.youtube.com/watch?v=GehUgGG9Z-U

Unity with Kinect Tutorials (Youtube)

https://www.youtube.com/watch?v=aHGlLxh6a88&list=PLmc6GPFDyfw-gF4aGw4Etgo0hJSWQcrYQ
https://www.youtube.com/watch?v=B7T0XTNk-Vg&list=PLmc6GPFDyfw-gF4aGw4Etgo0hJSWQcrYQ&index=2
https://www.youtube.com/watch?v=hKDaI_E7rDg&list=PLmc6GPFDyfw-gF4aGw4Etgo0hJSWQcrYQ&index=3
https://www.youtube.com/watch?v=fePCENFMYRc&list=PLmc6GPFDyfw-gF4aGw4Etgo0hJSWQcrYQ&index=4

Develop a game using Unity3D with Microsoft Kinect v2

https://andreasassetti.wordpress.com/2015/11/02/develop-a-game-using-unity3d-with-microsoft-kinect-v2/

Unity Manual Emissive Materials

https://docs.unity3d.com/Manual/lighting-emissive-materials.html

Detecting Closed Hands on the Kinect in Unity (None worked but these were the sources we used)

https://stackoverflow.com/questions/18729142/how-to-detect-open-closed-hand-using-microsoft-kinect-for-windows-sdk-ver-1-7-c
https://nevzatarman.com/2015/07/13/kinect-hand-cursor-for-unity3d/
https://www.youtube.com/watch?v=tlLschoMhuE