Avatar Demo

Description:

The Avatar demo needs to be intuitive to use, as it displays an avatar module that resemble the image of the player as well mimic the movements done by the player and showcase the body parts that are being used in the player's movements.
In this demo the Kinect Sensor will detect the various dots of player and forms a cloud of dots and uses this to form a skeleton that will be used for tracking the position and the rotation of the player as well as tracking the movements that the player is making.

This demo can be divided in three parts: Creating the Avatar, Mimic the Player's Movements, Showcase Body Parts;
In the 1st part we create the Avatar using the Blender program. First we make the 3D drawing of the humanoid and apply a texture onto the model.
Then we create the skeleton, creating the different bones and joints that are going to be used for the character rigging and name them. These bones have to follow a certain hierarchy starting from a root bone in order to specify which bones are going to move in the character's movement. Once we have completed the bone hierarchy we then move on to the character rigging in which we connect the various bones to 3D drawing and in which we specify which regions of the body should move according to the bones that those are being referenced to.
Finally, once we are done doing the character rigging we then move on to Pose Mode in order to test the different poses of the avatar and that everything is in order;
In the 2nd part we export the avatar we created in Blender as an .fbx file and we import it to our unity project. After that is done, we check if everything is in order like see if all the joints of the avatar are in right place and if everything is named right. The objective of this part is that we need to apply an avatar controller to our avatar and see if all bones respond accordingly with the players movements and they follow the player's position and rotation, displaying all the movements counter-clockwise;
Finally the last part we need to showcase the various body parts that are being used in the character's movements and in order to do that we first need to create the textures referencing the body parts with a different color in Blender and then we need to apply them in our avatar model using our avatar controller script. The end result should be an avatar that responds accordingly with the player's movements and showcases(painting the body parts with a different color - red) the body parts that are being used in said movements;

Execution:

To execute this demo you only need to:

- Connect the Kinect Windows V1 Sensor to your computer and have all the drivers and SDK that you need installed.
- Play the scene in Unity or run the AvatarDemo.exe file.
- To close the scene you either click on the play button again in Unity or click the “Esc” button if you are running the AvatarDemo.exe file

Scripts:

- Avatar Controller:
  - Awake() -> inits the bones array, get the initial rotations and directions of the bones, map the bones to the points the kinect tracks and get the initial renderer for the avatar;
  - UpdateAvatar(uint userID) -> update the avatar model in each frame. Get the KinectManager instance, get the 1st player, if the kinect sensor cannot detect the 1st player, reset the pointman position and rotation, move the avatar to its Kinect position;
  - ResetToInitialPosition() -> for each bone that was defined, reset to initial position and restore the offset's position and rotation;
  - SuccesfulCalibration(uint userID) -> invoke on the successful calibration of a player and recalibrate the the position offset;
  - TransformBone(uint userId, KinectWrapper.NuiSkeletonPositionIndex joint, int boneIndex, bool flip) -> Apply the rotations tracked by kinect to the joints;
  - TransformSpecialBone(uint userId, KinectWrapper.NuiSkeletonPositionIndex joint, KinectWrapper.NuiSkeletonPositionIndex jointParent, int boneIndex, Vector3 baseDir, bool flip) -> Apply the rotations tracked by kinect to a special joint;
  - MoveAvatar(uint UserID) -> Moves the avatar in 3D space - pulls the tracked position of the spine and applies it to root;
  - MapBones() -> If the bones to be mapped have been declared, map that bone to the model;
  - GetInitialRotations() -> Capture the initial rotations of the bones;
  - Kinect2AvatarRot(Quaternion jointRotation, int boneIndex) -> Converts kinect joint rotation to avatar joint rotation, depending on joint initial rotation and offset rotation;
  - Kinect2AvatarPos(Vector3 jointPosition, bool bMoveVertically) -> Converts Kinect position to avatar skeleton position, depending on initial position, mirroring and move rate;
  - changeColorOfAvatar(Transform[] bones) -> change the color of the avatar according to the bones position and rotation;
- Cubeman Controller:
  - Start() -> store the bones in a list for easier access, store the name of the bones, create an array holding the skeleton lines;
  - Update() -> update function is called once by frame, get the 1st player, if the kinect sensor cannot detect the 1st player reset the pointman position and rotation, set the user position in space, store the initial position, update the local position and local rotation of the bones;
  - changeColorCube(Material mat, int boneIndex) -> change the color of the material of the joints of the Cubeman skeleton;
  - changeColorLine(Material mat, int lineIndex) -> change the color of the material of the lines of the Cubeman skeleton;

- TextScript:
    - Script that changes the value of the String of the textbox in the "Canvas-Panel".
    - It gets its value from the global variable with the respective value that comes from the changeColorLine() function in the CubemanController.cs script.

Notes:

Here are some notes that you need to know:

- The Avatar module only responds to basic movements and it does not move away from the initial spot.
- In order for the Kinect Sensor to detect the user he has to stay still for a few seconds.
- We have assigned texture maps according to the user's movements so in order to change them the user has to move slowly and sometimes stay in the same position for a few seconds in order for the Kinect Sensor to detect the user's movements and different texture map to be assigned.

Requirements:

Windows, MacOs or Linux;
DirectX (to run Unity apps);
Blender;
Kinect Windows V1;
Kinect Windows V1
SDK1.8-https://www.microsoft.com/en-us/download/confirmation.aspx?id=40277;

Website:

https://xcoa.av.it.pt/~pei2016-2017_g8/
http://code.ua.pt/projects/pei-2016-2017-g08/wiki