Aperture Bioscience Laboratories

Progress Report 8 for NICView: A Virtual NICU Simulation

Due: 3/20/15

Professor Walker

Amy Young, Caitlin Li, Jennifer Duan, Pamela Wu, Lindsey Sumners

I. Background of the project

Considering the new regulations that limit work hours for medical residents, these residents do not have the same real case experience as in the past. Therefore, this reduces patient safety, because the residents do not experience a large and varied amount of cases. In order to solve this problem, our project will involve developing a simulation game that can be played at the resident's home. By playing this game, the residents will be able to gain better conceptual knowledge by going through different scenarios, which addresses the issues of volume and variety of cases and patient safety. By having this knowledge, the residents will be able to use the time in the hospital in order to gain experience with the physical actions in medicine.

II. Achievements since the last report

Since the last project report, the team has made progress in two areas. The first area of progress is that the team has received the pre and post questions for each scenario. There are six questions for each scenario, and they are short answer. The questions will be asked before the scenario, but the correct answers will not be given. The resident will proceed and be asked the same questions after completion of the scenario. This will allow Dr. Krakauer and the team to see if the residents are gaining knowledge from the game. Also, one of the questions asks the resident to rate his/her comfort with the skill sets that the scenario is testing. Therefore, this question will allow the team and Dr. Krakauer to determine if the residents feel that they are learning concepts from the game. This perception is important to the value of the game, and this feedback will allow for improvements to be made to the game.

The second area of progress is in programming. The first scenario has been completed; it has the points system and sequential collision code integrated into the game. The code for these steps are shown below.

```
	//BEGINNING OF SEQUENTIAL COLLISIONS CODE
	using UnityEngine;
	using System.Collections;

	public class BabyScene2 : MonoBehaviour {


		//Initialize variables
		private Rect buttonSizeV = new Rect(Screen.width/2 - 15,Screen.height - 
600,100,30);
		private string buttonTextV = "Continue";
		private string boxTextV = "RR=30 with gasping; Heart Rate=120;
Saturation=80% in foot; Physical: Pale, limp, no murmur, lungs clear, slow respiratory rate.";
		private bool showBool = false;
		public bool isTiming = false;
		public static int stepOrder;
		public static int scoreVar;

		// Use this for initialization
		void Start () {
		}

		void Update () {
		}


		// OnGUI Function to Display Text and Button
		public void OnGUI () {
			GUIStyle labelStyleV = new GUIStyle (GUI.skin.label);
			labelStyleV.fontSize = 16;
			labelStyleV.normal.textColor = Color.black;

			//Tests whether the Vitals? button has been clicked
			if (showBool) {
				if (GUI.Button (buttonSizeV, buttonTextV)) {
					//If the button has been clicked, change the value of
showBool, isTiming, and stepOrder
					showBool = false;
					isTiming = true;
					UpdateStep ();
					UpdateScore (10);

				} else {
					GUI.Label (new Rect (Screen.width/2 - 200,
Screen.height/2 - 30, 400, 600), boxTextV, labelStyleV);
				}

			}

		}

		//Runs when Vitals? button has been clicked
		public void Clicked () {
			showBool = true;
		}


		//Called when a collision occurs
		public void OnTriggerEnter(Collider Baby) {
			//Test for collision with a gameObject that is tagged "CPAP"
			Debug.Log ("The dfsdsf value of stepOrder is " + stepOrder);
			Debug.Log ("Total dfdsf points is currently " + scoreVar);
			if (stepOrder == 1) {
				if (Baby.gameObject.tag == "CPAP") {
					//If collision is detected
					Debug.Log ("Collision between baby and CPAP mask
detected");
					Application.LoadLevel ("WinScenario");
				}
			} else {
				Application.LoadLevel ("DeadBaby");
			}
		}

		// Update is called once per frame
		void UpdateStep () {
			stepOrder++;
			Debug.Log ("The value of stepOrder is " + stepOrder);
		}

		void UpdateScore (int scoreVal) {
			scoreVar = scoreVar + scoreVal;
			Debug.Log ("Total points is currently " + scoreVar);
		}
	}
```

The other area of progress in programming has been the medical inventory. The medical inventory code has been completed and is functional. The code for the inventory is displayed below.

```
	//INVENTORY

	using UnityEngine;
	using System.Collections;
	using System.Collections.Generic; //gives access to List type

	//create variables
	public class Inventory : MonoBehaviour {
		public List<Item> slots = new List<Item>(); //slots list
		public int slotsX, slotsY, xpos, ypos, xsize, ysize, offset; //number of slots and
position and size of slots
		public int bxpos, bypos, bxsize, bysize; //position and size of button
		private bool showInventory; //whether or not inventory appears
```

```
        private bool showTooltip; //whether or not to show pop up box when hovering
over item in inventory
        private string Tooltip; //text that shows up in said box
        public Item CPAP = new Item ("CPAP", "Continuous Positive Airway Pressure",
Item.ItemType.Airway); //CPAP mask
        public Texture tool = Resources.Load<Texture2D> ("tool"); //background image
for inventory

        string CreateTooltip(Item item) //create the text in the tooltip box
        {
            Tooltip = "<color=454545>" + item.itemName + "</color>\n\n" +
"<color=473344>" + item.itemDesc + "</color>";
            return Tooltip;
        }

        //Populate slots with empty items
        void Start ()
        {
            for (int i = 0; i < (slotsX * slotsY); i++) //looping through number of slots
            {
                slots.Add(new Item()); //creating a list of empty items as long as
number of slots
            }
            AddItem (CPAP); //Add CPAP to slots
        }

        void OnGUI()
        {
            if (GUI.Button (new Rect (bxpos,bypos,bxsize,bysize), "Crash Cart"))
            {
                showInventory = !showInventory;
            }
            Tooltip = ""; //set text to empty
            if (showInventory) { //if show inventory is true, draw the inventory
                DrawInventory ();
            }
            if (showTooltip)
            { //if showtooltip is true, draw a box containing the tooltip
                GUI.Box (new Rect (Event.current.mousePosition.x + 15f,
Event.current.mousePosition.y, 200, 200), Tooltip);
            }
        }

        void DrawInventory() //define draw inventory
        {
            int i = 0;
```

```
                for (int y = 0; y < slotsY; y++)
                {
                        for (int x = 0; x < slotsX; x++) //loop through indices of grid
                        {
                                Rect slotRect = new Rect(xpos+x*xsize+offset, 
ypos+y*ysize, xsize, ysize); //rectangle
                                GUI.Box(new Rect(xpos+x*xsize+offset, 
ypos+y*ysize,xsize,ysize),tool); //background image
                                if(slots[i].itemName != null) //if the slot is not empty, draw 
the item icon in the corresponding box
                                {
                                        GUI.DrawTexture(slotRect,slots[i].itemIcon);
                                        if(slotRect.Contains (Event.current.mousePosition)) 
//additionally if the mouse is over it, show tooltip
                                        {
                                                CreateTooltip(slots[i]);
                                                Tooltip = CreateTooltip(slots[i]);
                                                showTooltip = true;
                                        }
                                }
                                if(Tooltip=="") //if the tooltip is empty, don't show the 
tooltip box
                                {
                                        showTooltip = false;
                                }
                                i++;
                        }
                }
        }

        void AddItem(Item device) //create a method for adding items to the inventory
        {
                for (int i = 0; i < slotsX*slotsY; i++)
                {
                        if(slots[i].itemName==null)
                        {
                                slots[i] = device;
                                break;
                        }
                }
        }
    }

    //Used AwfulMedia inventory tutorial, only slots, because inventory is not dynamic

    //ITEM
```

```
using UnityEngine;
using System.Collections;

[System.Serializable] //makes it so that all attributes are listed under item
public class Item {
	public string itemName;
	public string itemDesc;
	public Texture2D itemIcon;
	public ItemType itemType;

	public enum ItemType {
		Airway,
		IV,
		Medication,
		Other
	}
	//two constructors for creating items: one with attributes and one empty
	public Item(string name, string desc, ItemType type)
	{
		itemName = name;
		itemDesc = desc;
		itemIcon = Resources.Load<Texture2D>("Icons/" + name);
		itemType = type;
	}
	public Item()
	{

	}
}

//AwfulMedia tutorial, only removed item ID designation and changed types
```

III. Deviation from the plan and corrective action

Looking at the work plan, the team has made significant progress and is back on track to complete the project on time. In order to maintain this progress and based on the recommendation of Dr. Walker, the team has stopped work on graphic design and will focus purely on programming until all of the scenarios are completed.

IV. Plan for Next Week

Over the next week, the team plans to focus on completing the second scenario and having it be fully functional, which requires the team to integrate the timer system and medical inventory into the game. For the medical inventory, the images of the needed items will be added to the game. Then, a universal sprite will be used and dragged to the baby in place of the image. Finally, the code for the vitals monitor display will be completed and integrated into the game. After completing the second scenario, the team will begin coding scenario three.

V. Assessment of Progress

Considering the current progress, the team is back on track with the work schedule. As long as the team continues this level of progress, the project will be able to be completed on time.