Dungeon 2: The Charming Choreographer (Inverse Kinematics, **25g of total 100g**). 1g may be exchanged for a human percentage point in COMP5823M.

For the birthday celebration of Lord Spinhead, the Great Dancer Lady Anklesprainer is the master choreographer. She has been designing a glorious but difficult dance show. After breaking some ankles, she decides to use simulated characters to try some of her moves.

You, my loyal geniuses, will provide your assistance. You will build an Inverse Kinematics system to animate human characters. It should come with a GUI which can load/save BVH files. It should be able to play an animation clip from the BVH file and allow the user to control the positions of the joints by dragging.

For your hard work, Lady Anklesprainer has kindly agreed to give you rewards for:

1. A GUI to load and play the BVH files in 3D. (**3g**)
2. The same GUI to write BVH files. This should write an animation clip into a BVH file. (**5g**)
3. A basic Inverse Kinematics (with pseudo-inverse) system which can handle positional constraints for any single joint to update a posture (e.g. drag the right wrist and use IK to update the whole body from the root). (**5g**)
4. A basic Inverse Kinematics (with pseudo-inverse) system which can handle positional constraints for all combinations of joints to update a posture (e.g. position both hands and the neck simultaneously to reach given target positions). (**6g**)
5. A damped Inverse Kinematics system that can accomplish 3 and 4. (**3g**)
6. An Inverse Kinematics implementation with control on joints (refer to the slides for 'Adding control' to IK). (**3g**)

The total reward is **3g + 5g + 5g + 6g + 3g + 3g** = **25g**.

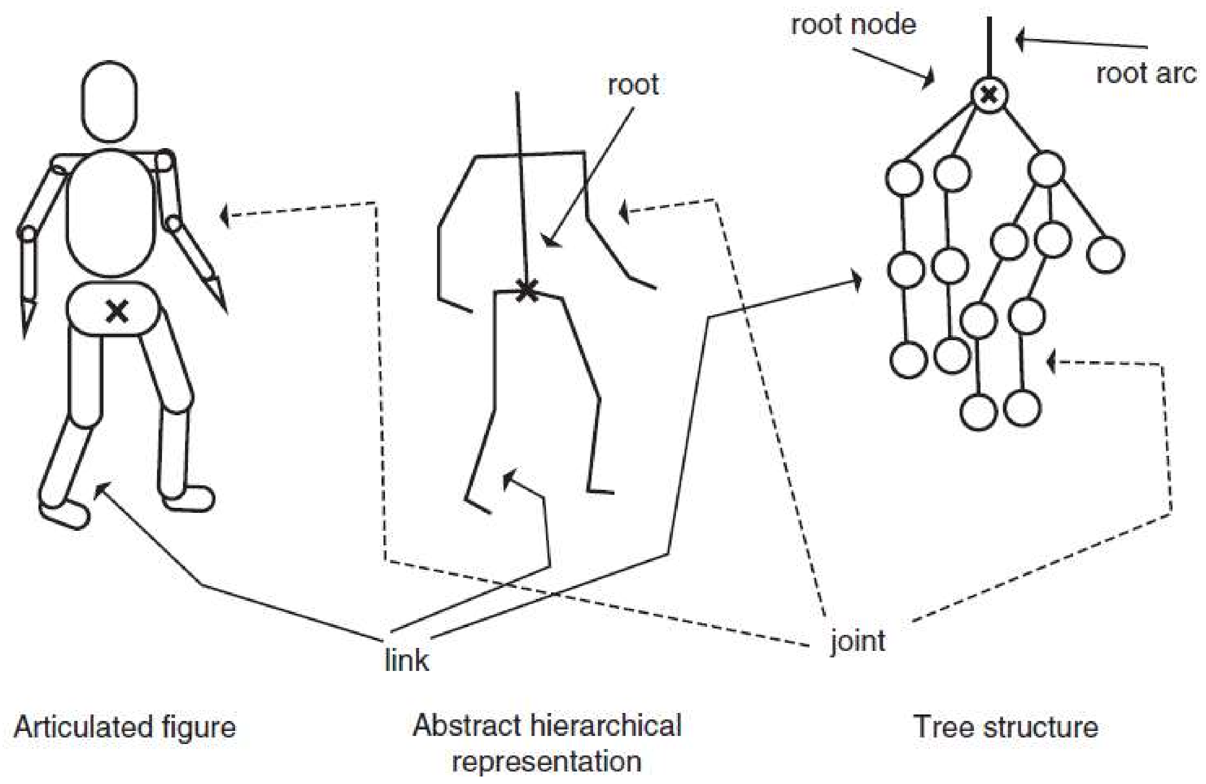
Hint 1: There are several places in town where you can find assistance. (You can find the specs of BVH files at https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html and some sample code and BVH files in Minerva under Lab Resources)

Hint 2: You can use other BVH software to help you debug and test your system (BVH player https://sites.google.com/a/cgspeed.com/cgspeed/bvhplay, BVH hacker http://www.bvhacker.com/)

The whole submission needs to be implemented in Visual Studio C++. It should be your own work except for any third-party libraries. Make sure that you submit a zip file containing the *whole* Visual Studio solution. You are free to use third-party libraries for GUI, rendering, triangulation, video-capture and linear algebra operations, etc. But IT SHOULD BE STANDALONE SO THAT I CAN RUN IT ON MY WINDOWS COMPUTER.

Some further hints below:

A human body skeleton can be represented as follow:



We already know that each joint includes an offset and a rotation (refer to the BVH file). We can stack all the rotations (**joint angles**) into a long vector. If you have 25 joints in the skeleton and each has three rotational angles, you can stack them into a 75x1 vector. Let's call this vector θ. Now assuming that you want to control the **3D position** of **one** joint, e.g the right hand. Its position can be represented by a 3x1 vector Y = [p_x, p_y, p_z]. Assume that the current position of the right hand is **c** and the joint angles are θ_1. The target position of the right hand is ***t*** and the IK aims to compute new joint angles θ_2 so that the right hand will be at ***t***, by ***t*** – ***c*** = ***J*** ($\theta_2 - \theta_1$). Here both ***t*** and ***c*** are 3x1 vectors, and θ_1 and θ_2 are both 75x1 vectors. Naturally, the Jacobian matrix ***J*** is a 3x75 matrix. The only unknown here is θ_2. ***t***, ***c*** and θ_1 are known. We still need to compute ***J***, by computing each entry of it. Let's take a look at the structure of ***J***:

$$J = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} & \cdots & \frac{\partial p_x}{\partial \theta_n} \\ \frac{\partial p_y}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_2} & \cdots & \frac{\partial p_y}{\partial \theta_n} \\ \cdots & \cdots & \cdots & \cdots \\ & & & \end{bmatrix}$$

To compute an entry, we look at one example $\frac{\partial p_x}{\partial \theta_1}$. It can be computed by:

$$\frac{\partial px}{\partial \theta 1} \approx \frac{\Delta px}{\Delta \theta 1} = \frac{p2-p1}{\Delta \theta 1}$$

Assuming θ_1 is your root joint angle indicating its rotation around the x-axis, this entry essentially means if the root rotates around the x-axis by a small amount say $\Delta\theta_1 = 0.001$, the x coordinate of right hand will change from p_1 to p_2 . So the only unknown here is p_2. One way to compute it is to change θ_1 by $\Delta\theta_1$, then do Forward Kinematics once to compute p_2. You should already know how to do Forward Kinematics (You already used it when you render the BVH file. Look at the code I shared for labs.).

Now we know how to compute the Jacobian ***J***. The rest is the specific IK schemes you want to implement for task 3-6. One change about controlling more than one joint is that Y is not a 3x1 vector anymore. If you want to control two joints regarding their 3D positions only, Y will be a 6x1 vector. If you want to control both the orientation and position of one joint, Y will also be a 6x1 vector.