RUTGERS

Rutgers - Computer Graphics - Assignment 3

Unity: Animating Characters

This is a group assignment.

Git Repo for Unity

Please clone/fork this repository which contains all the resources (and more) that you will need for this assignment. These can be found in the “Animations” and “Models” folders.

Although we will enumerate resources for you to investigate and learn, I strongly suggest you to read all the requirements thoroughly before heading staring into the tutorials, for these, might cover things which you and your team will not have to mandatorily implement. Having said that, extras are always welcome.

Unity Official Tutorials: One stop for all. They have been adding plenty of documentation for you to get started: Animate Anything with Mecanim (video tutorials on Unity’s animation engine). Here you can find the documentation.

Here you can find the basic Controller/Navigation Setup.

Online community made: These are more targeted to specific aspects of the animation, DO NOT get into these before understanding the ABCs of the Mecanim engine.

Simple built-in IK

Part I - Animation [7 points]

Before starting, we want to emphasize that as long as you comply with the basic assignment’s requirements, we encourage you to try different models, animations etc. You are not stuck to a particular character or set of animations. Let creativity flow!

The **main goal** of this part, is for the team to create a fully **animated user-controllable character**.

Meet Mr. Manny Bones:

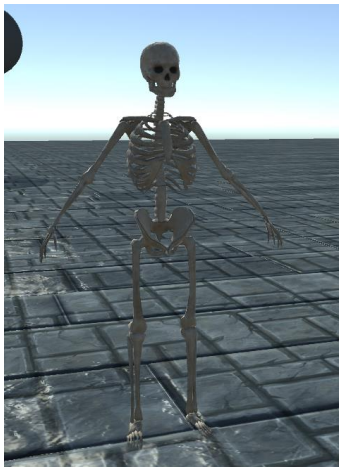
This is just one of the model you can use to start your project.

Feel free to investigate the 20 + models we added!

Background

Manny can **walk**, **run**, **jump** while idle, jump while running, jump while walking (all three different types of jumps of course), **strafe**, **turn**, **walk backwards** and **strafe while backing up**. He is very lively for a dead guy.
To create such a character, you basically need to:

1) Select and adjust the model (Avatar) you will use, in this case I chose Manny. A skeleton Humanoid.
2) Add the animator controller which will manage all the different states of the playing animation at any given time.
3) Script the user's IO and how to handle this with respect to the actual animations.



At the beginning you will have nothing but the model

You will complete this model with all the components you have been working with so far, including physics, NavMeshAgents so on so forth.

You can see a demo of part I expected result in this link.

Requirements - P I

1. A simple third-person or over-the-shoulder camera view to appreciate your model while being animated. Orbital cameras (Witcher-like) are not required. I strongly suggest you to expand the free-flight one from the previous assignment.
2. Fully **animated and WASD-controllable character** which can perform the following actions:
    a. Walk forward and backwards. Turn while walking forward. Strafe / Turn while walking backwards.
    b. Run. Turn while running forward.
    c. Jump while running, walking and standing.
    d. Turn on the spot.
    e. Be able to jump over small obstacles. (Physics such as rigid bodies and colliders will be involved here)

    Please use Horizontal and Vertical input axes that are mapped to WASD. This will make Part 2 easier.
3. The animator controller must have at least the following components:
    a. At least different **states** for representing idle and locomotion.
    b. The locomotion state must use “2D Blend Trees” and at least two “2D Freeform Cartesian” **trees**.
4. At least all the locomotion **transitions must be smoothly blended** (parameters are key)

PART II - Animated Navigation [5 points]

Now that your animated character is ready, it is time to start working in making the agent to move autonomously throughout your walkable areas. This means that you no longer need the WASD-controller, but it will still be useful.

In the last assignment, you utilize a component called a NavMeshAgent, which pretty much handled all the aspect of navigation on your behalf. As opposed to particles (capsules, voids…), when it comes to humanoid figures, there are certain aspects which we will need to deal with, for example, handling animations while traversing the map, and of course, making these look good.

For Part 2, you will use what was developed on Part 1 along with the techniques learnt on B1 (Navigation Basics), so that you can **create a prefab animated humanoid**, capable of navigating meshes, **to populate an animated crowd**.

Requirements - P II

1. Implement a simple isometric camera which navigates with WASD, right click selects, left click acts and wheel zooms (changes world-y up/down) in and out.
2. The animated character you created on Part I will now need to be ALSO able to navigate with point-n-click I/O (Starcraft like). The controlled navigation from the previous part is not required for this.
3. Characters should jump when navigating off mesh links.
4. User should be able to set the desired speed of the character(s) and they should transition from running/walking (e.g. by multiple clicks on a target.)

PART III - Crowd [3 points]

Finally, just as you did on the “Navigation Basics” assignments, you will create a nice scenario and utilize the prefab you created to populate a crowd.

Requirements - P III

1. You might reuse the scenario you created for B1, making sure it is well adapted for the new humanoid type of agent and includes the requirements (4).
2. There must be at least 10 agents on the scene at any given time.
3. Agents must implement the breaking mechanics you used for B2
4. On play, at the main scene, agents start in a “room” like location and evacuate through at most 3 different exists to a given point (this is defined before starting the simulation)

Assignment Extra Credit [10 pts]

Animation [5 pts]

- Inverse Kinematics using Unity’s built in agent
  - Look at objects (easy)
  - Reach and grab (medium)
  - Footing (advanced)

Navigation [5 pts]

Animation-physics managed navigation - desired result demo here.

- The NavMeshAgent is disabled on the prefab, the animations’ physics do all the work, following a list of points (target locations towards an end goal)
- You do NOT need to implement any pathfinding-related code. You can use the NavMeshAgent component of the prefab to resolve the best path to a target location.

This can be achieved very easily the following way:

```
NavMeshAgent.enabled = true;
NavMeshPath navMeshPath = new NavMeshPath();
NavMeshAgent.CalculatePath(target,navMeshPath);
List<Vector3> path = new List<Vector3>(navMeshPath.corners);
NavMeshAgent.enabled = false;
```

Splines [5 pts]

Use a centripetal Catmull-Rom spline to interpolate between the points on the navigation path and have the agent race along the path while adjusting speed with banking (where the agent tilts into the curve of its motion and adjusts speed to maintain this angle).

Submission

Submit the following in Sakai for grading:

1. Your Unity project in a zip file which contains the Assets/ folder and includes all your C# scripts.
   ONLY SUBMIT RELEVANT FILES, NO UNNEEDED OVERHEAD ASSETS (models and animations you haven't used)
2. Zip with WebGL builds of your 3 demos (part1.zip, part2.zip, part3.zip). Remember always to test your builds.
3. Brief README about your project. Clearly describe how your project works and your extra credit attempts. This is your guide for the user. For unmentioned/unclear aspects of your project, you won't get any mark. So please write your documentation brief, clear, and complete. Please also include 3 links to at least 3 youtube videos (at least one for each of the three parts).