Low resolution pictures

lingoltechart.tumblr.com

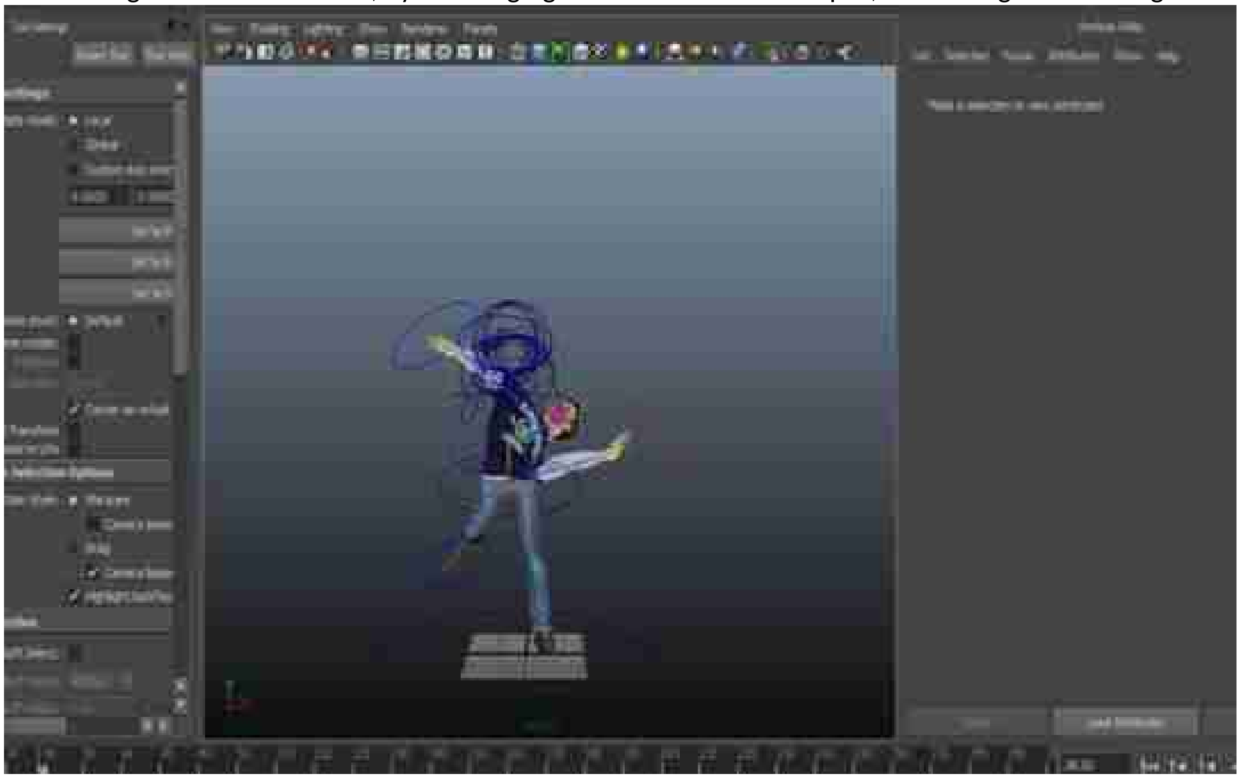
Contents

1. 2015

1.1 October

Week 1- Animating Grunt (2015-10-15 7:21)

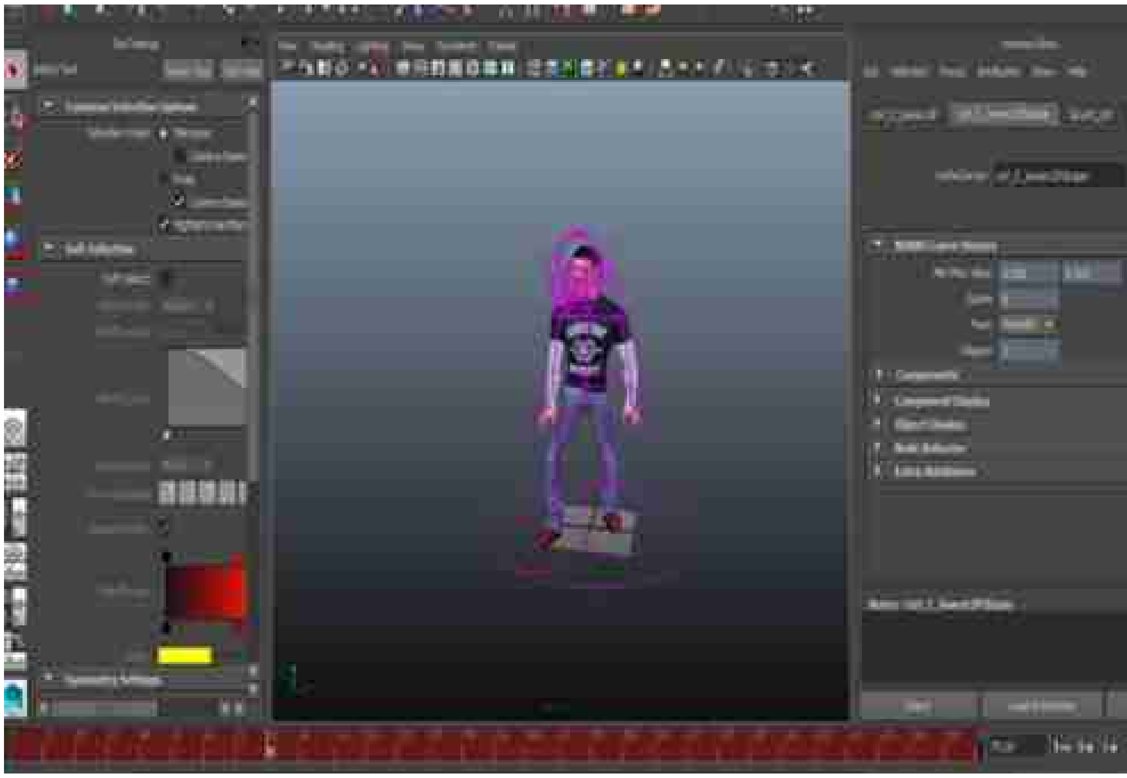
In our first week we were introduced to the Grunt model, a rigged character with many control shapes and a skeleton. Our first task was to get familiar with him, by rearranging him into ridiculous shapes, and seeing what this rig was capable of.



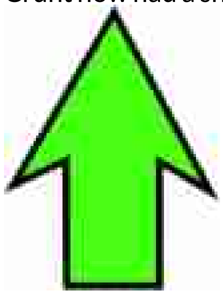
Although this pose is fantastic from a comical standpoint, it wouldn't exactly be very useful when it comes to games. Thankfully we were equipped with a reset script. Since this script would need to be repeated in the future, the script was saved to a shelf. This shelf could then be edited to make it more user friendly.

Next it was finally time to animate. The frame rate had to first be set up in preference settings to ensure the frame rate was appropriate for a game setting. This was set to 60fps and in real time play-back speed.

Animations were created using key-frames. Moving the model using a control shape into the best position, then saving it as a key-frame would place the model in its starting position for the animation. This could then be repeated for every end movement the user wanted their character to make, and the key-frames would fill in the movements in between. It is vital that every control shape has key-frames if the user wanted the full body to move. Key-frames could be moved by using click, shift and dragging on the timeline.



Grunt now had a short movement:



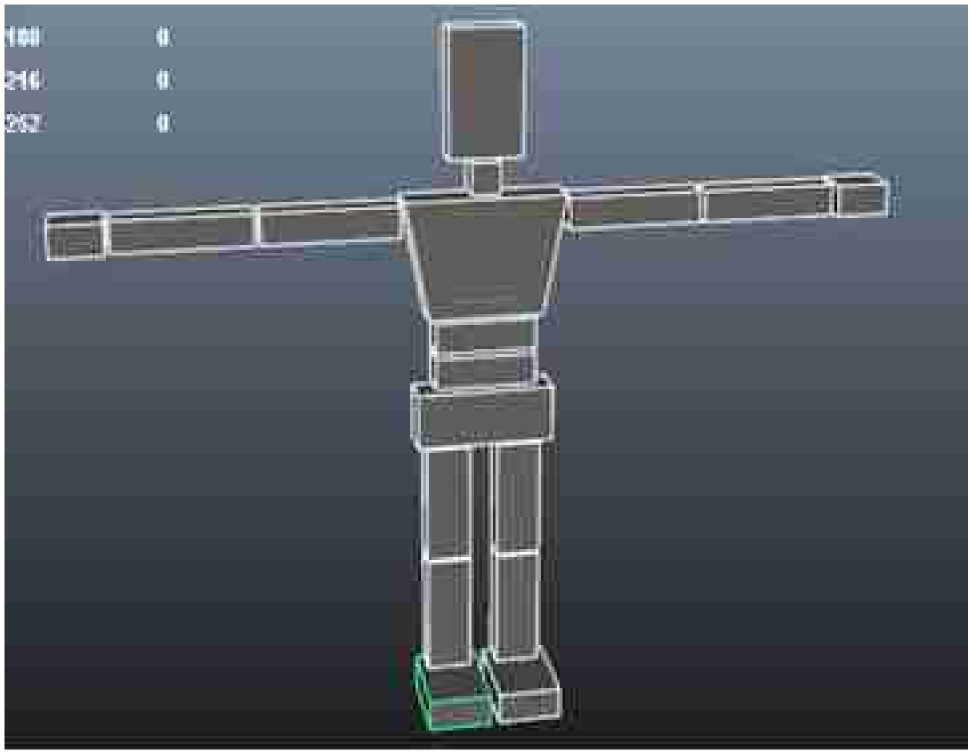
These small red lines represent the key-frames. There are so many because this is after baking, the next step. This is necessary so the animation can be used in a game engine. First, the root joint (which highlights the entire skeleton) is right-clicked and then the 'joint hierarchy' is chosen. From here the edit menu is selected, then 'keys' and finally 'bake simulation'. Only the joint hierarchy and the mesh need to be exported, so these have to be selected before the "Export Selection" button is pressed. The file has to be exported in .FBX format, so the FBX plug-in must be switched on.

The "animation" box within the .fbx option must be ticked to ensure the animation is saved.

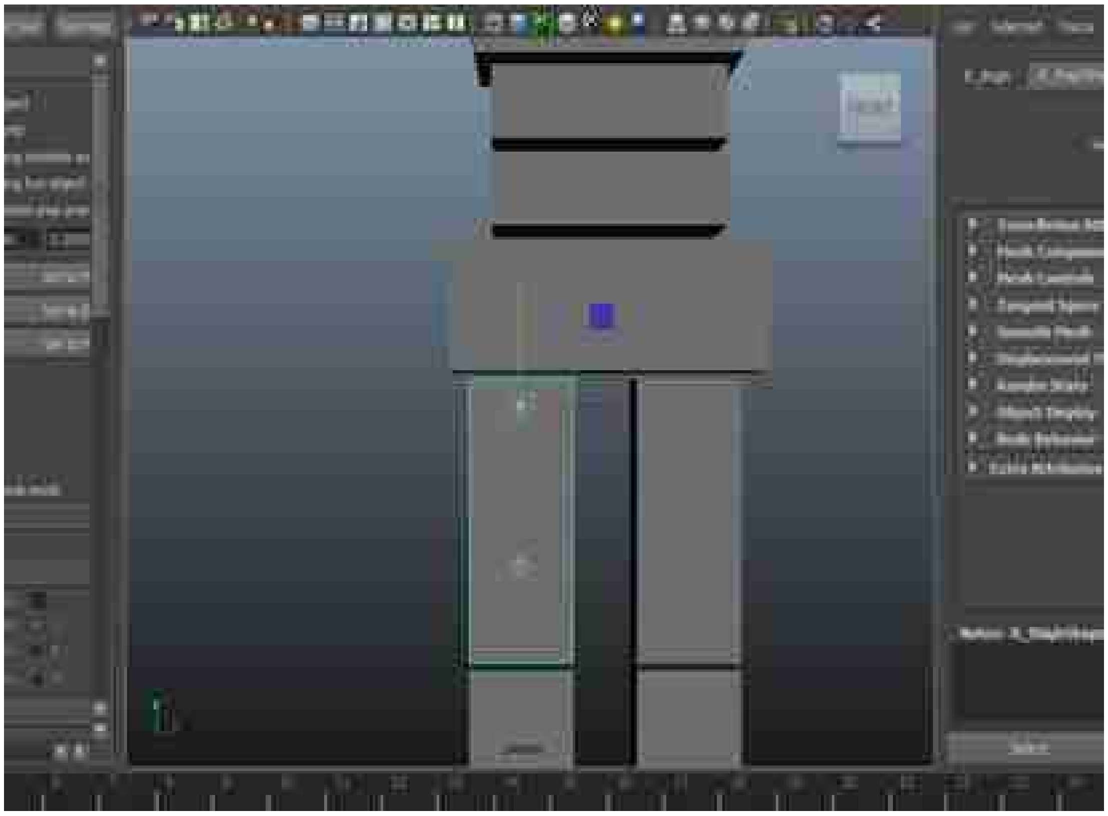
Finally the animation and model were imported into Unity.

Overall this was a very fun task and extremely useful to know how to create animations in Maya, how to appropriately export them and how to successfully import them into a game engine. I will find this very useful in the future when creating games, including my professional project.

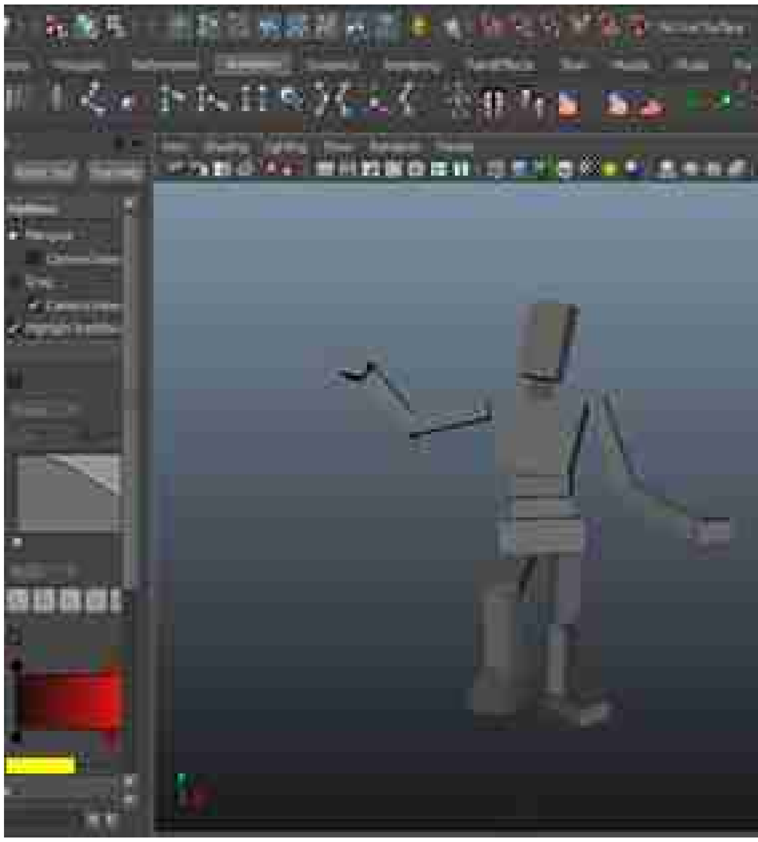
Week 2 (Part One)- Pivots (2015-10-15 7:49)



Say hello to “Cube man”. This was the model used to experiment with pivot points. Pivots are vital to “Cube Man”, since his body should only rotate in certain ways. For example, we wouldn’t want the poor cubed fellow to have calves that rotated from the foot upwards, separating his calf from his knee.

First the object was selected, eg. the lower leg, and the move tool was chosen. The insert key was pressed, creating a pivot point. This was then moved into the best position for the object to be rotated appropriately. In the case below, the pivot point was placed on the top of the calf (just below the knee). This allowed the lower leg to be rotated from the knee. 

Next the object hierarchy had to be set up. This would ensure that if the, for example, the thigh was moved, the rest of the leg would move with it, whereas moving the foot would only move the foot. To do this, parents had to be set up. The child object was selected (eg. the lower arm), the shift button was held down and the parent object (the upper arm) was also selected. The shift button was released and the ‘P’ key was pressed, asserting the upper arm as the parent of the lower arm.

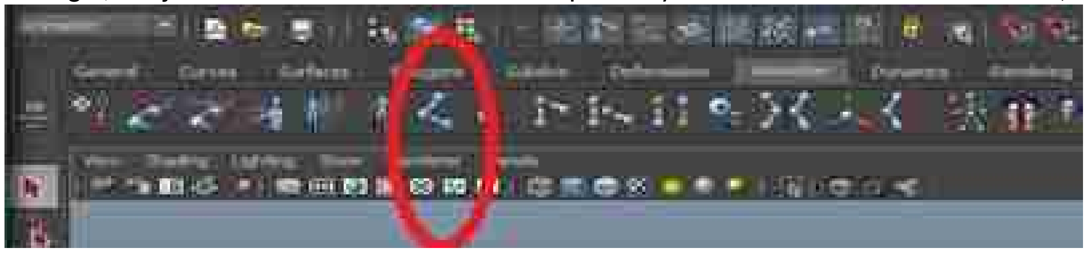


When the hierarchy is set up, “Cube Man” should be ready to pose:

Week 2 (Part 2)- Skeleton (2015-10-15 8:44)

Building a skeleton proved to be quite an easy, but potentially frustrating, task, and to test out our skeletons we said hello to Grunt again. This time his rig and skeleton had been removed-poor guy- and his model was ready for a new rig.

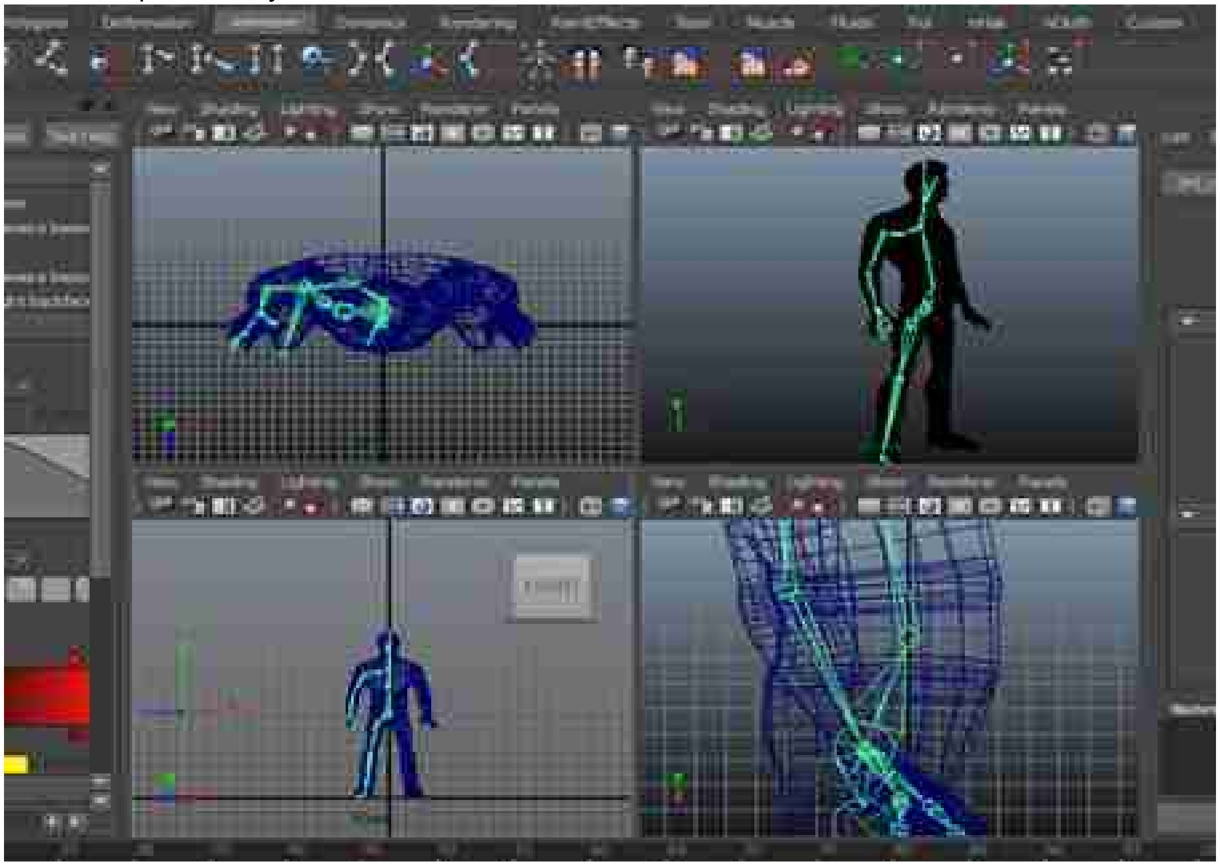
To begin, the joint tool was selected from the top of Maya underneath the animation tab (as seen below).



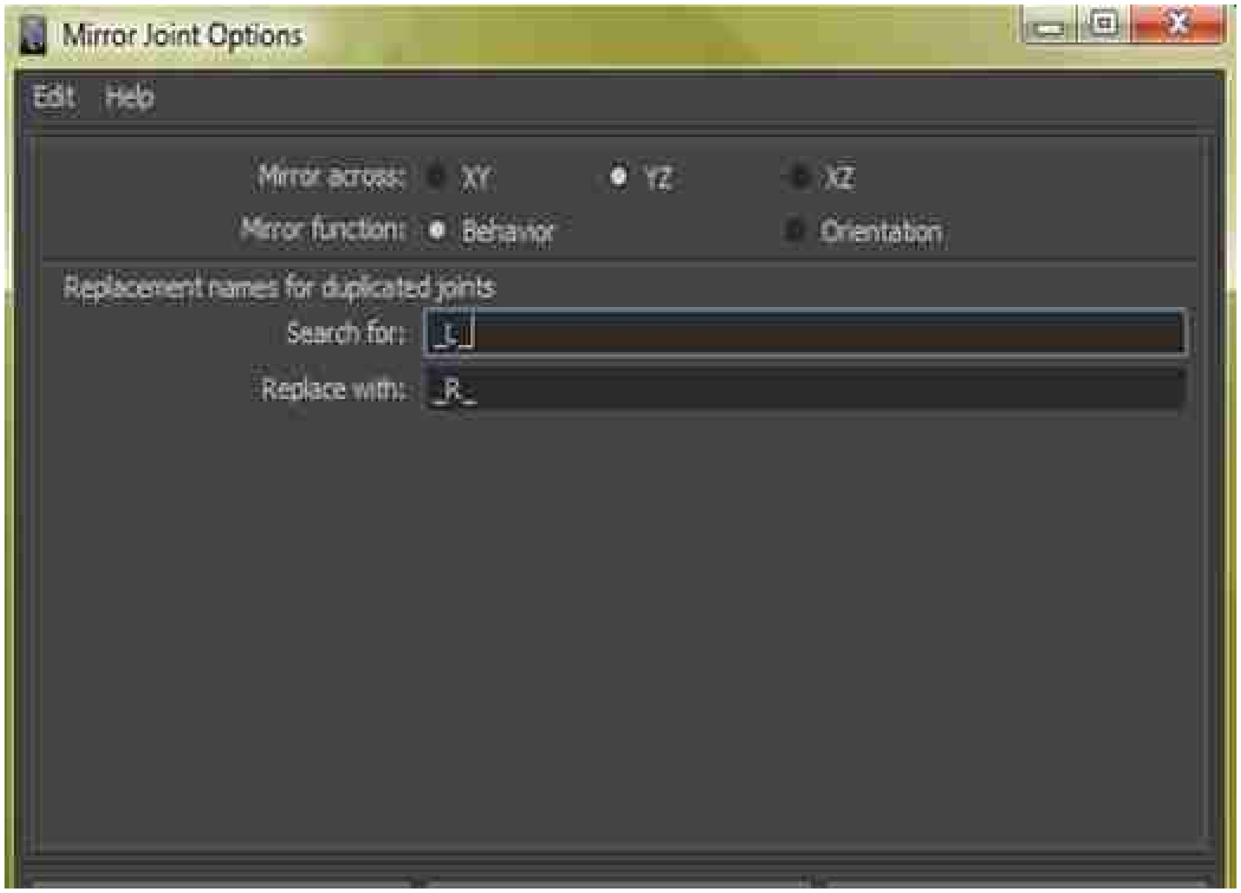
The first joint was placed in the groin area. This would be the root joint. From there the spine could be created using about 5 joints until the neck was reached. Then the root joint was selected again and from there the left leg could be created. The root joint had to be the parent of the first left and right leg joints (this could be tested by clicking on the root joint. If the left leg and the spine are both highlighted, they are parented by the root joint). Only one side of the skeleton had to be created as these could be mirrored at a later stage. The last spine joint before the neck joint was selected in order to create the shoulder and arm joints. The neck joint would become the parent to the jaw, eyes and the top of the head joints.

It is highly recommended to have many view-ports open in order to see the placement of the joints from all sides. The joints should usually be placed in the middle of the mesh (exceptions can be made for knees and elbows). The joints should follow the curves of the body, especially in the spine. Also the joints should be named appropriately, and joints on the left side should be called something with “L” in the name e.g. “Jnt _L _arm”.

Once one half of the skeleton is done, the Local Rotation Axis of each joint should be edited to ensure every joint rotates properly. This is vital in defining how much freedom the joints have. This is done by choosing the “Skeleton” menu and then “Orient Joint”. This menu allows the user to choose what axis points up the joint group and which one faces up from the joint.



Since the left side has been completed and the joints are all oriented, certain joints should be mirrored to the opposite side. The hip joint was clicked (which highlighted the whole leg) and the skeleton menu was selected. Next “Mirror Joint” was clicked and another menu was opened. This is when it comes in handy to have named the joints with “L” in it, since this tool replaces these names with “R”s instead:


Mirror Joint Options

Edit Help

Mirror across: XY [✓ YZ] XZ

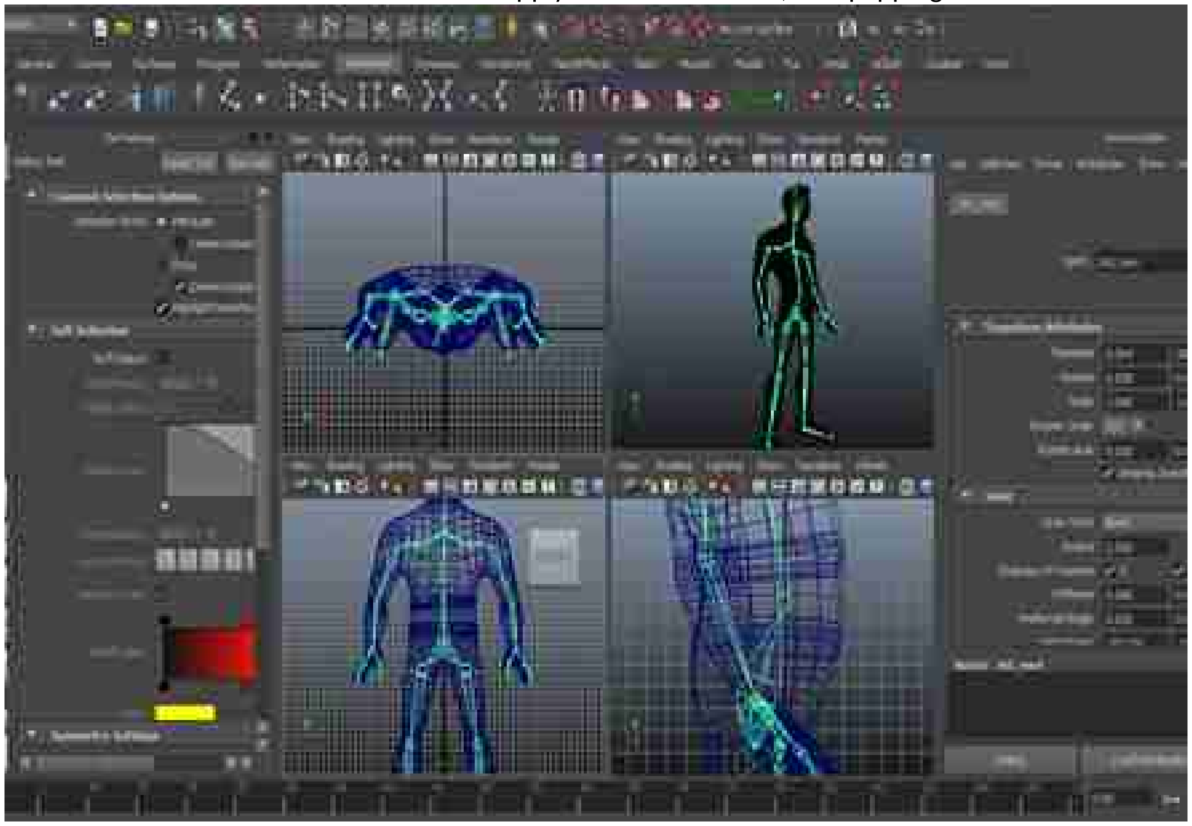
Mirror function: [✓ Behavior] Orientation

Replacement names for duplicated joints

Search for: _L_

Replace with: _R_

The joints can be mirrored across in a different axis, and either the joint's behaviour or orientation can be mirrored. 'Apply' was selected, equipping Grunt with a full skeleton:

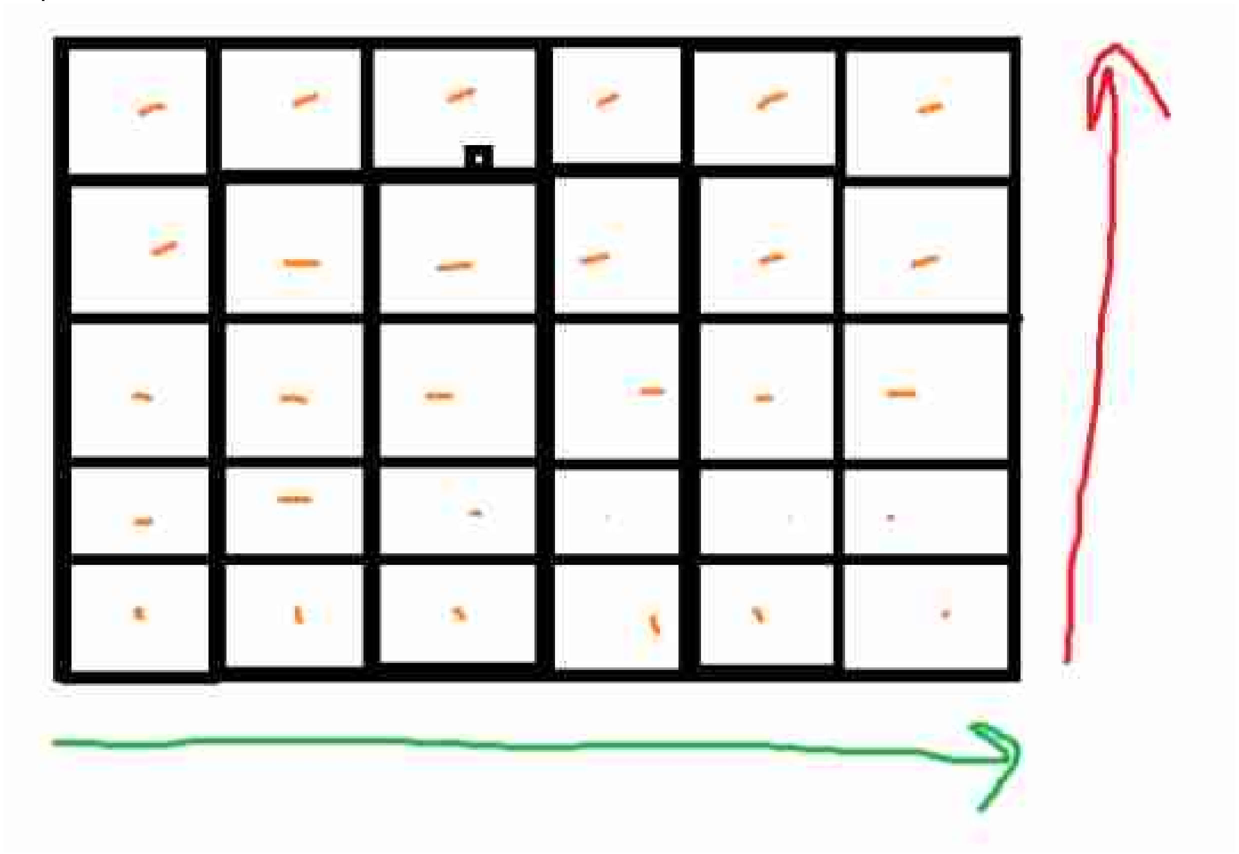


Overall this task was extremely useful. It was interesting to find out how the beginning of a model's rig was made up and how powerful a tool Maya is. One part I struggled with was the Local Rotation Axis, as I found it quite difficult to wrap my head around what axis each joint would rotate in, but I feel I have grasped it.

Week 3- Rise of the Cube Army (2015-10-15 9:51)

Apparently "Cube Man" from last week has been up to no good. The only way to stop him? An army of his brethren. But this army would not be created by creating sphere upon sphere manually. There is clearly not enough time! Therefor, this would have to been done using script.

First, to create this army, there had to be a moment to think about how this code would translate well on-screen. Questions like, "How many rows?", "How many columns?" and "What changes?" had to be asked. Diagrams could help this:

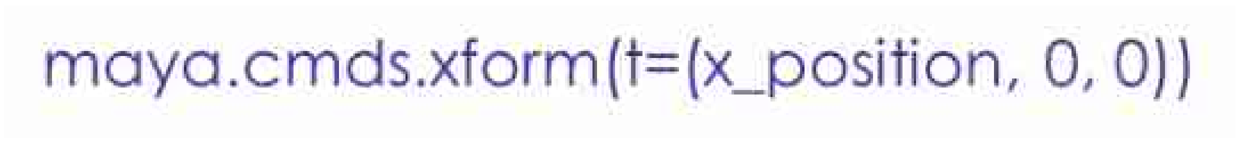


This- very poorly drawn- diagram shows how the only thing changing about the cubes are their positions, creating columns and rows of cubes. In code, for each of these cubes, a cube had to be created and then positioned.

First to create a cube, the command "maya.cmds.polyCube" was used. The values of the cube such as the height and width could be altered by including them in the brackets - e.g. cmds.polyCube(w=3) which creates a cube of width 3 units.

Next to create a group of cubes, the planned amount of cubes were entered into a for loop e.g. if we wanted 30 cubes like above in one position, the code would be "for x _position in range (30)" then create a cube of height 2 units. However, this creates 30 cubes in only one position.

To change the position of a cube, this code needed to be incorporated in some way:

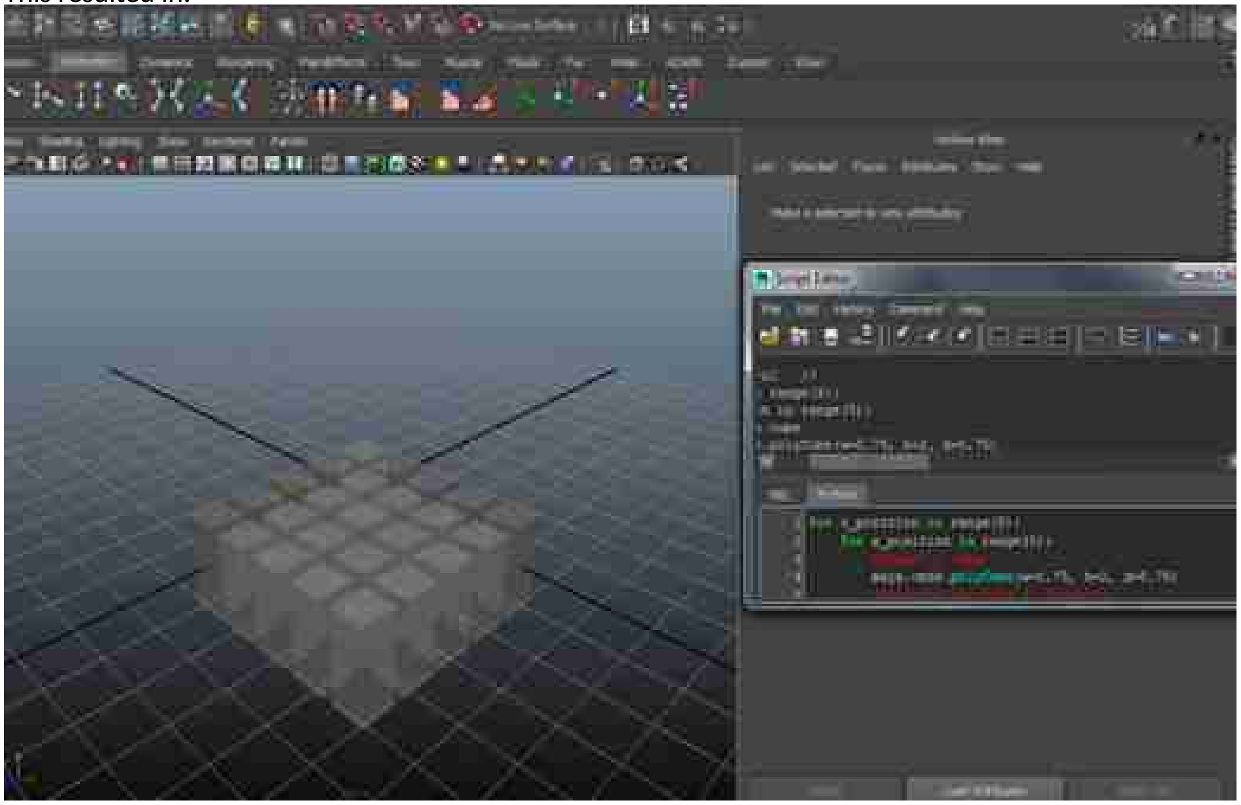


```
maya.cmds.xform(t=(x_position, 0, 0))
```

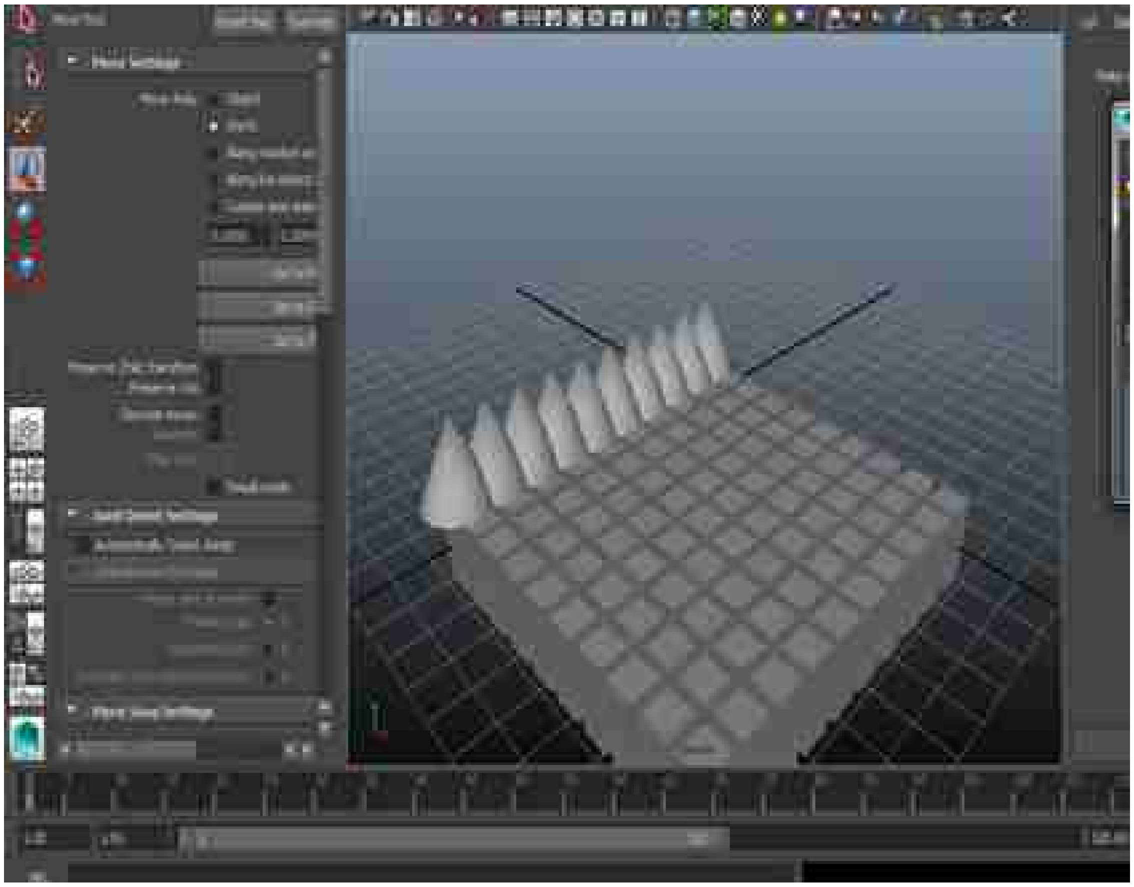
So, to change the position of each cube, the position also had to be inserted into a for loop. The code looked like below, each line will be explained in #:

```
for z _position in range(5): #How many cubes in column (and z-position)
for x _position in range(5): #How many cubes in row (and x-position)
#Create a cube
maya.cmds.polyCube(w=0.75, h=2, d=0.75) #re-sizes cube so no squashing
#Position the cube
maya.cmds.xform(t=(x _position, 0, z _position)) #puts each cube in different position
```

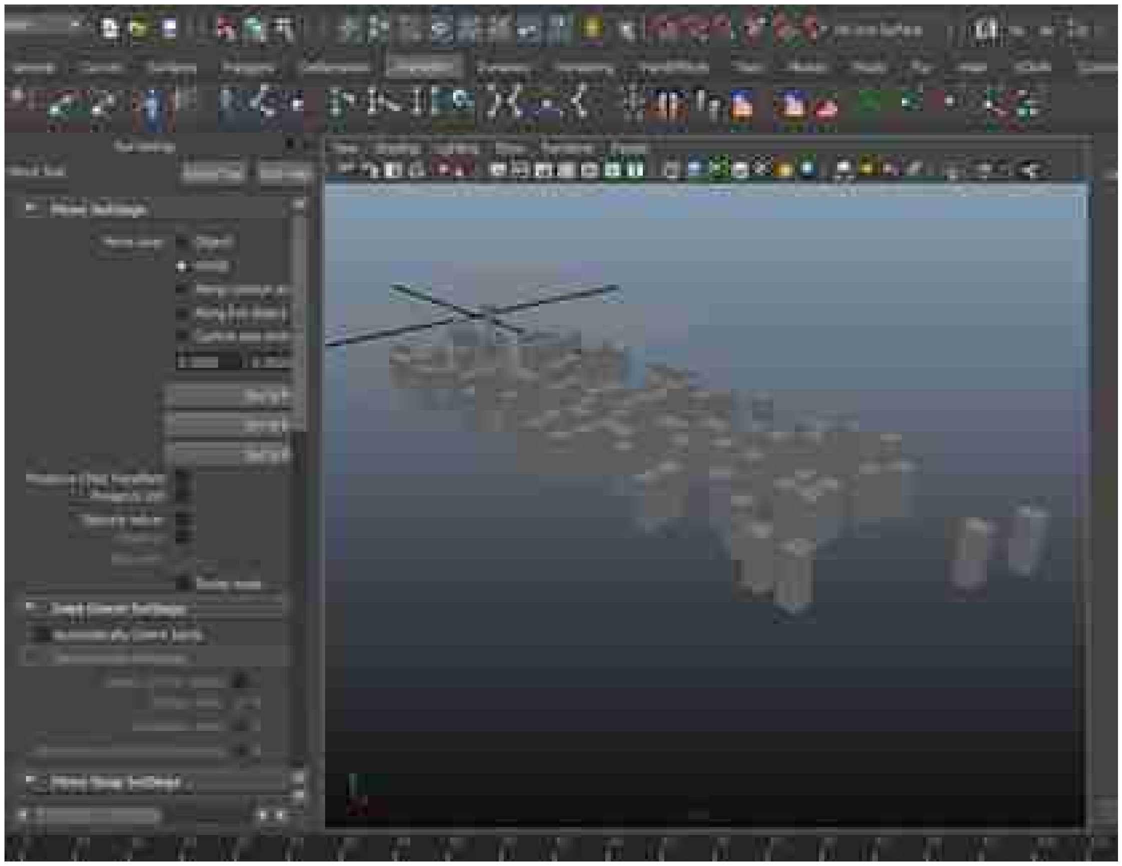
This resulted in:



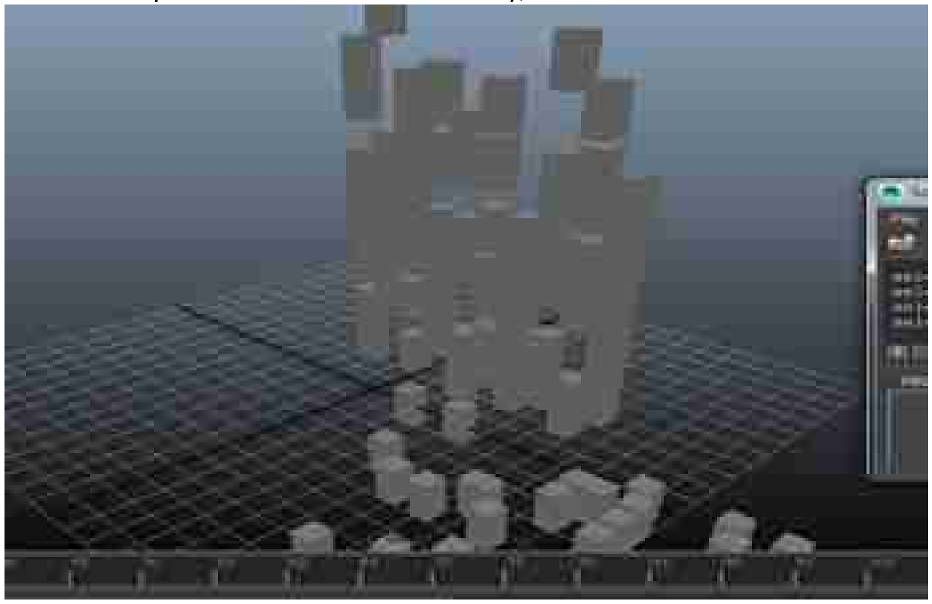
25 cubes, ready for the army. If statements were also experimented with in Maya. The image below displays how a cone will appear above a cube IF they are positioned at 0 in the for loop:



The random() function was also experimented with. "random.uniform" was used to create the image below. The range intended can be implemented using this e.g. "random.uniform(0,30)"- between 0 and 30.



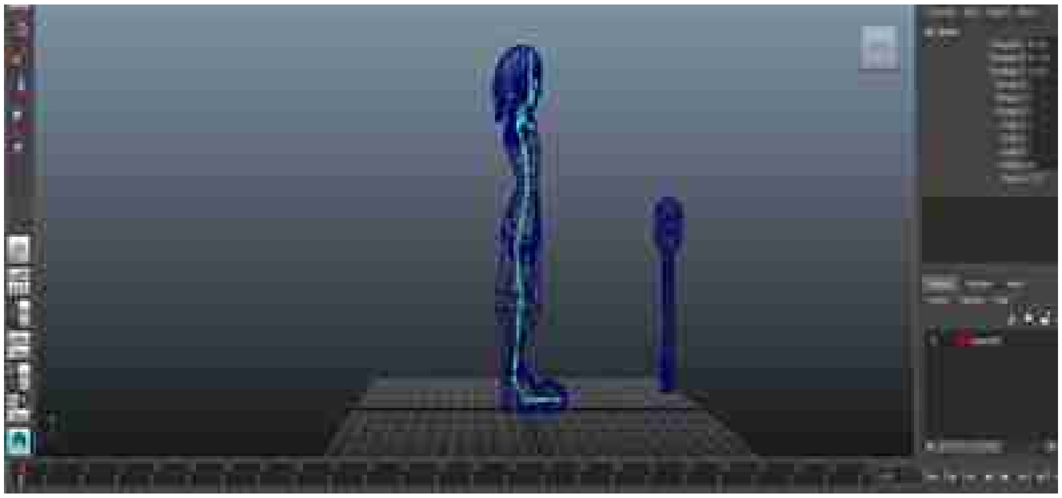
Overall, scripting to create objects proved to be a worthwhile lesson and a fun one too. It shows that scripting can save a large amount of time and can work effectively as well as efficiently. I will continue to experiment with the various placements of this cube army, and research into the numerous functions that can aid the process.



1.2 November

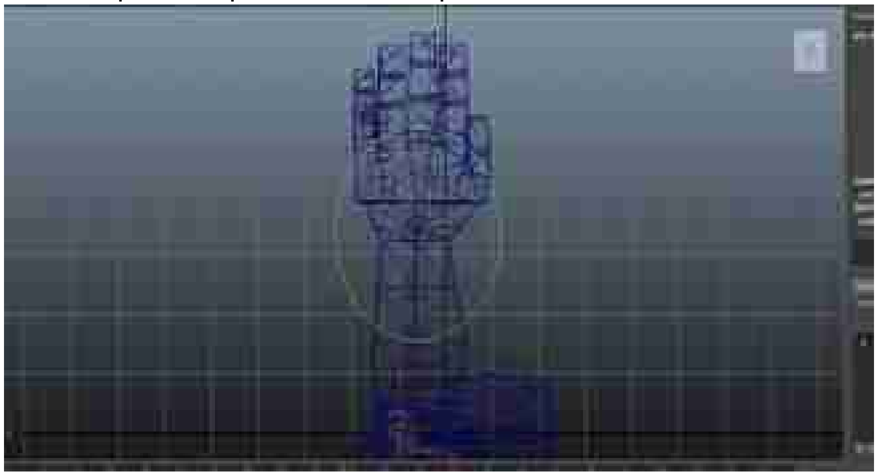
Week 5- Control Shapes & IK Handles (2015-11-12 11:53)

First off, it was time to build a new skeleton. Please refer back to week two for details.



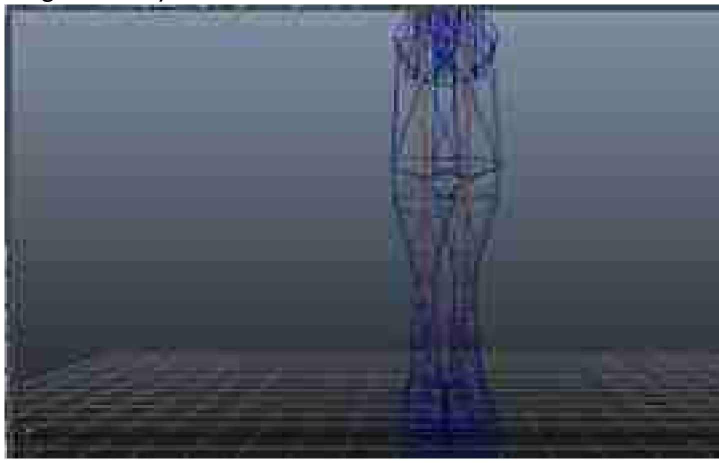
After rebuilding a skeleton, it was time to input the control shapes. These are necessary in order to control our joints easily as well as our IK handles for us. The model must be in its' default pose aka, all the model's transformations and rotations are set to zero. They also have to have the same orientation as the joints they are supposed to control.

First a NURBs circle was created and placed at the origin, as well as being named sensibly, e.g. "ctrl _R _wrist". This circle was then grouped and the group was also named appropriately. The group was selected and then by holding shift, the joint it was going to control was selected also. The group and joint were then turned into parent and child (by pressing 'p') and the transformations and rotations for the circle were zeroed out. This caused the control shape to appear around the joint area. Then the circle was un-parented from the joint, with the shape still in place. This was repeated for both wrists and both feet.



IK handles were created now. IK handles define how a child joint such as the foot can control the movement of the parent joints such as the knee and thigh. To create this, the "IK handle" option was selected in the animation tab, which looks like this:

After this was selected, the parent joint we wanted to be moved by the child was selected i.e. the thigh joint, which was then followed by one of its' child joints i.e. the foot joint. This would mean that if the foot joint were to move, the knee joint in between would move appropriately without being manually



moved too. This process was repeated with both legs and upper arms.

Overall, this lesson was very worthwhile and it is satisfying to see our rig taking more of a form. One thing that could be improved is the repetitiveness and slow nature of the process; however this could be aided next week when scripting is introduced to this process.

Week 6- Speeding Up the Process (2015-11-12 11:59)

Last week the control shapes and IK handles were added to the rig manually, which proved to be a highly repetitive and slow task to do. This week there was a solution to this. We could solve this with script.

A completely new project was opened and a lonely joint was placed at the origin as the starting point. First we had to find the joint using script, which could be done by giving it a name and searching for it. This could be done by using the code:

```
selected _joints=maya.cmds.ls(selection=True,
type="joint")
```

Next, a control shape had to be made and named using script. This could be handled by using strings. The name of the joint would be taken and altered by accessing only parts of the string. For example, the important part of the joint name would be the part that would be used for the joint, control shape and control group I.e. " _R _Wrist", removing the "jnt" from the start. In order to access the vital part of the name, the variable 'joint _name' was created and was defined as 'jnt _R _Wrist', or the joint name. Next 'base _name' was created to catch the vital part mentioned above, by selecting the characters ' _R _Wrist', which looked like this:

base _name
= joint _name[4:]

Now, in
order to name the control shape, a new variable was created named
‘control _shape _name’. This would
take the base name created above and the string “ctrl” would be added to it,
which could be coded like this:

control _shape _name = “ctrl _”+base _name

This would
return ‘ctrl _R _Wrist’.

The print
command could be used to ensure the name was created properly since it would
output in the script editor history panel.

Now we had
to figure out how to create a NURBS circle using code, which was done by
reading the MEL script.

A NURBs
circle could be created by coding, ‘maya.cmds.circle()’.

A group
could be created by using the code, ‘maya.cmds.group()’ and inserting the
objects needed to be grouped within the brackets, followed by the group name.

To parent an
object the parent() command could be called by using, ‘maya.cmds.parent()’ and
inserting the child, then the parent objects into the brackets separated by
commas.

Unparenting

Since the
name could be searched and the objects could be created, it was time to modify
the objects. The joint could be moved using code. This was done by knowing the
joint name and accessing the joint’s attributes in order to change its
position.

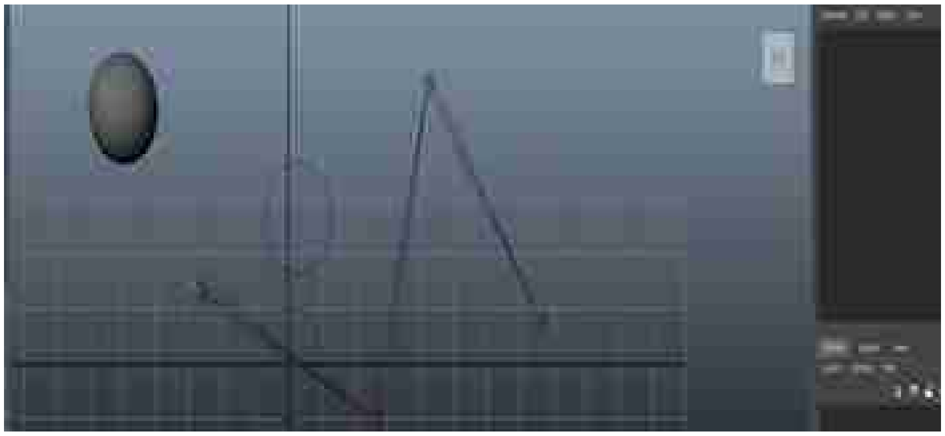
maya.cmds.setAttr(“jnt _root.translateX”, 0)

maya.cmds.setAttr(“jnt _root.translateY”, 0)

maya.cmds.setAttr(“jnt _root.translateZ”, 0)

This code

translated the joint to zero in the X,Y and Z axis.



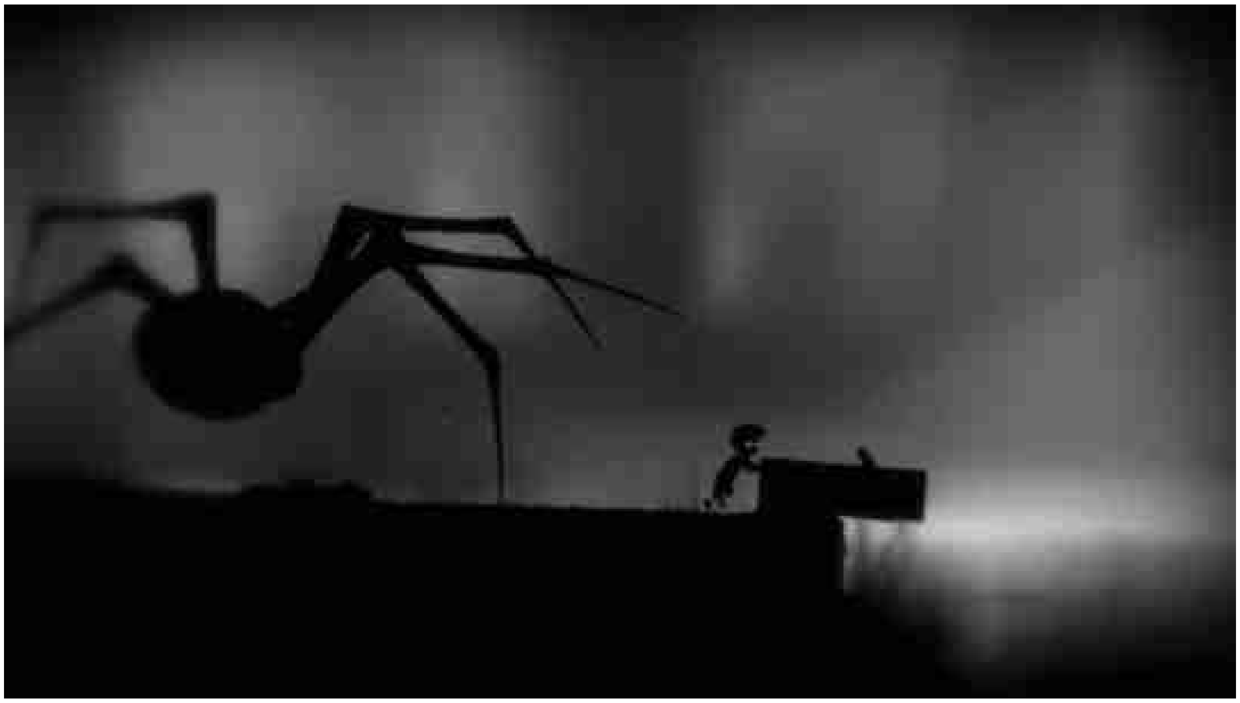
These
joints, spheres and circles were created using a script. With a bit more understanding, this could speed up the process of rigging since control shapes and joints are necessary in rigging.

1.3 December

Platform (2015-12-10 13:47)

One thought I had while choosing my model was choosing a model from a game on an older platform and using it on a lower-power current generation platform. Since my model was originally from the PS2 era, I looked into platforms with lower needs. I decided to specialize in tablet games, after looking at some of the games currently available for it, a lot of them being remakes from other platforms. While researching, I was able to find some games from the PS2 era being remade onto the tablet platform. This includes PS2 classics such as Portal and Half Life 2. Plus, I was able to find a few emulators for the tablet to enable it to play PS2 games, so the platform seemed appropriate for the model.

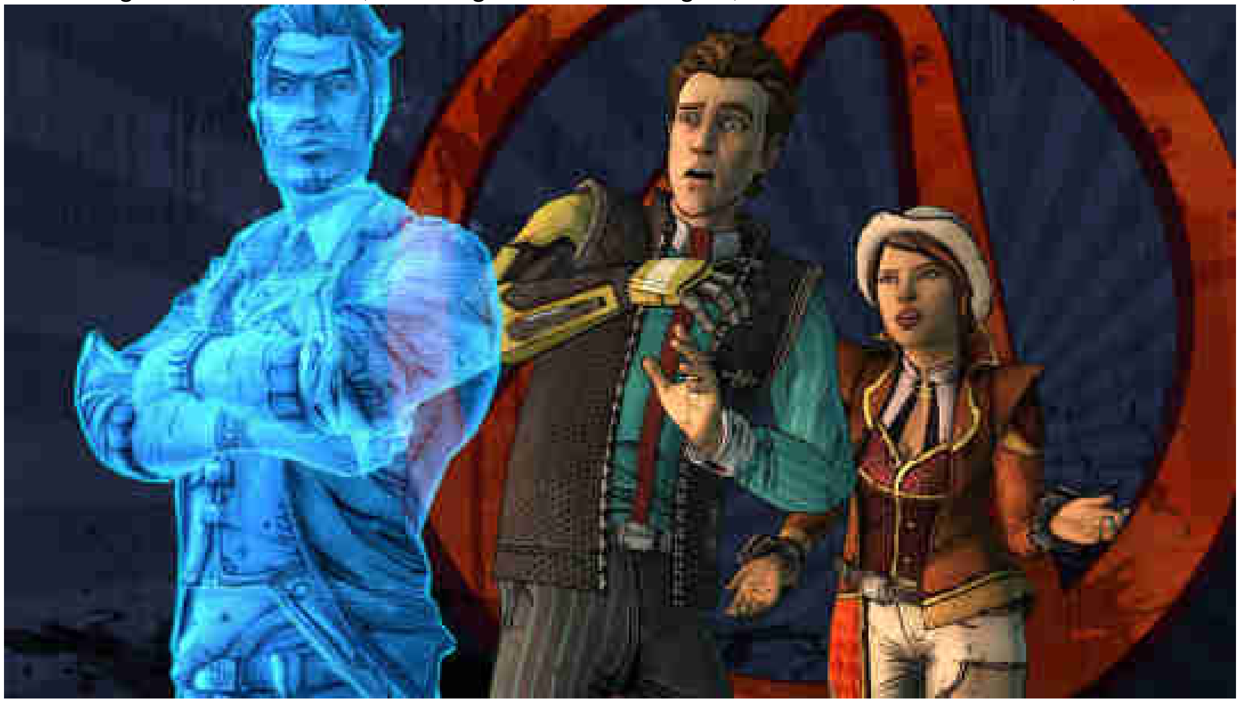
The first few games I looked at were mostly 2D platformer games such as Badland, Limbo, and 3d platformer Monument Valley, that all were examples of great games that ran quite fluidly and had little issue with performance.


(Shoemaker, 2010)

All these games would take little rigging of characters, as their faces and hands need very little rigging since they don't usually face the screen. Also, in Monument Valley, it is mostly the legs that are animated so if I were to do a character for a game like this, most of the bones would probably be situated in the legs. This would keep the bone-count down, performance up but could still result in a visually-stunning game. For rigging for a platformer like Limbo or Badland, most of the focus could just be on one side of a character, since that's the only side facing the screen. These games are mostly about the gameplay, so there'd be little need for too many bones.

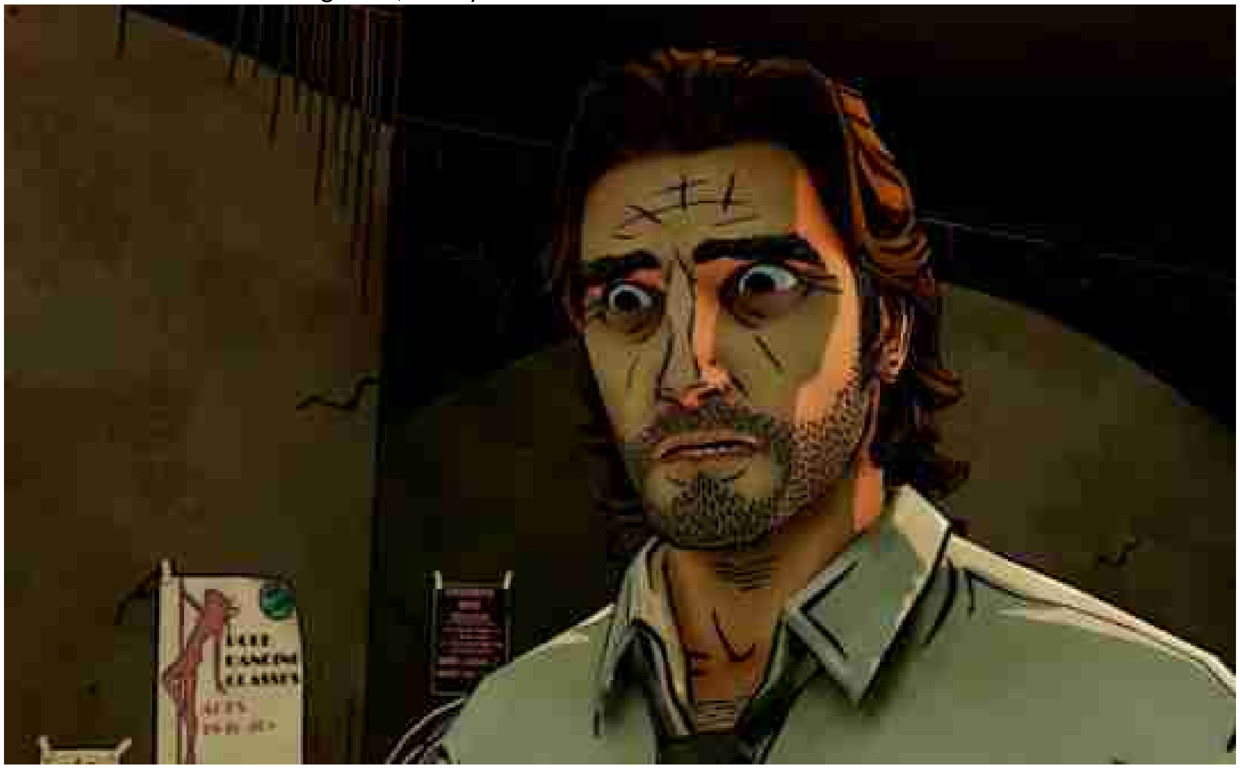
(*Badland-Screenshot, 2015*)

However, there are also games on the other end of the spectrum. Story-driven game specialists TellTale Games made versions of their games for the tablet, including The Wolf Among Us, Tales from the Borderlands, and The Walking Dead.



(The Escapist Staff, 2015)

These are examples of games that required a larger amount of bones to do facial expressions, detailed cut-scenes and quick-time events. However, because the gameplay was mostly quick-time events for these games, they could afford to use more animation and bones in their games.

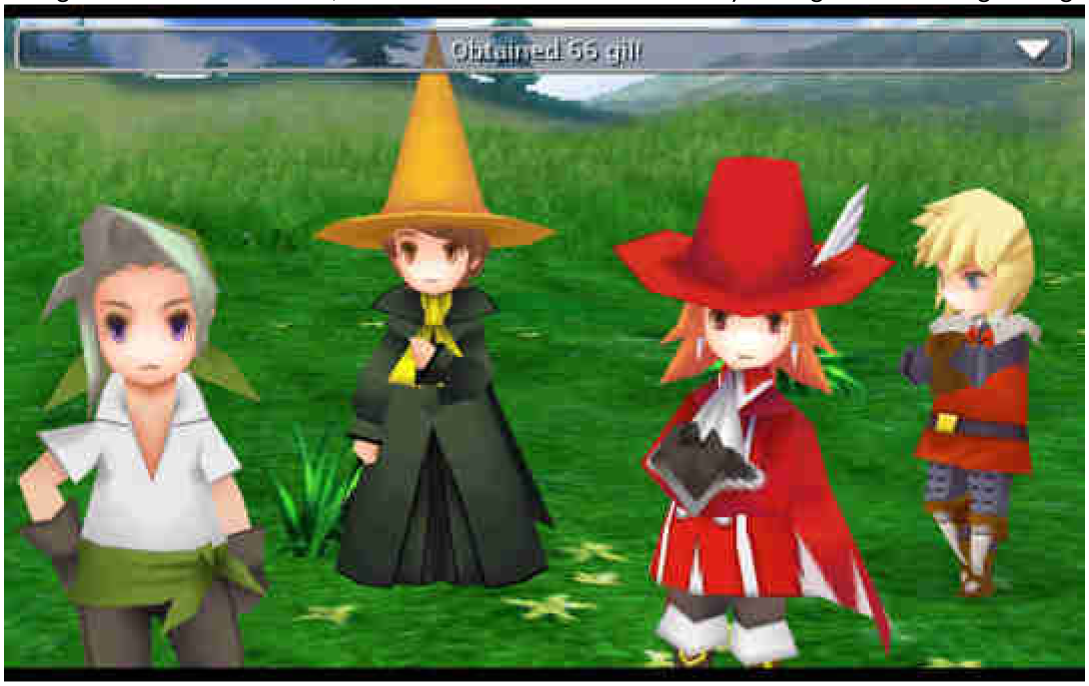


(LeatherWolf89, 2014)

Then there were games like XCOM:Enemy Unknown, which shows a pretty good mix of good graphics and more indepth gameplay, being a mostly turn-based strategy game. It is known to be using android to its full capabilities.

(Kelly, 2012)

Also on android are a few of the early Final Fantasy games remade for android devices. These would have much simpler animations and rigging compared to XCOM or the Telltale games since these games were first released on the SNES or EARLIER consoles. However the gameplay is great, being a turn-based RPG, is well known for its story-telling and has good graphics for their time.



(Clatworthy, 2015)

It was then decided to mix some of these as a target type of game.

References

Clatworthy.2015.final _fantasy _iii _android _04. [Online Image]. Available From: http://gamers-haven.org/haven-review-final-fantasy-iii/ [Acc essed 16 November 2015]

Escapist Staff.2015.*Tales From The Borderlands Logo Large*. [online Image].Available From: http://www.escapistmagazine.com/articles/view/video-games/walkthroug hs/14833-Tales-From-the-Borderlands-Episode-5-Mystery-Vault-Hunter -Guide . [Accessed 17 November 2015].

GooglePlay. 2015.*Badland-screenshot*.[online Image]. Available From: https://play.google.com/store/apps/detai-ls?id=com.frogmind.badland . [Accessed 17 November 2015].

Kelly.2012. *XCOM: Enemy Unknown Screenshot 17*. [online Image].Available From: http://www.videogamer.com/xbox360/xcom _enemy _unknown/screenshot-17.html . [Accessed 17 November 2015]

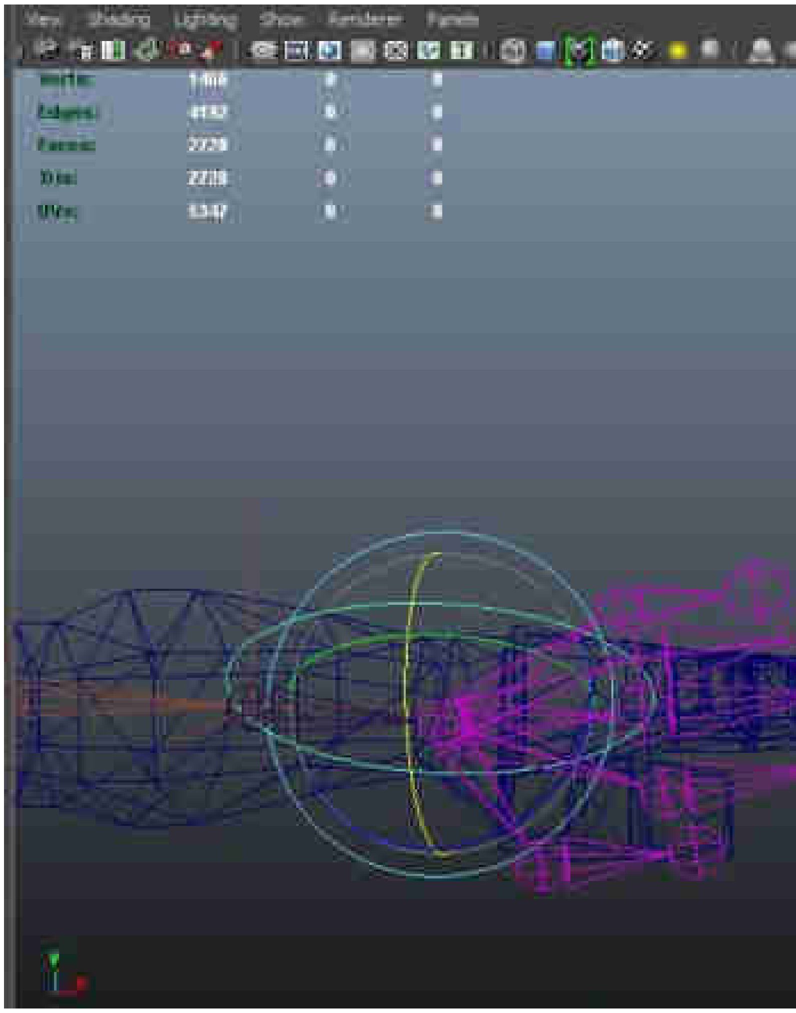
LeatherWolf89.2014.*Favourite Wolf Among Us Expressions?*. [online Image]. Available From: https://www.telltalegames.com/community/discussion/62619/favorite-wo lf-among-us-expressions/p2 . [Ac-cessed 17 November 2015].

Shoemaker.2010.*You Will Die*, *A Lot*.[online Image]. Available From: http://www.giantbomb.com/reviews/limbo-review/1900-301/ [Acc essed 29 October 2015].

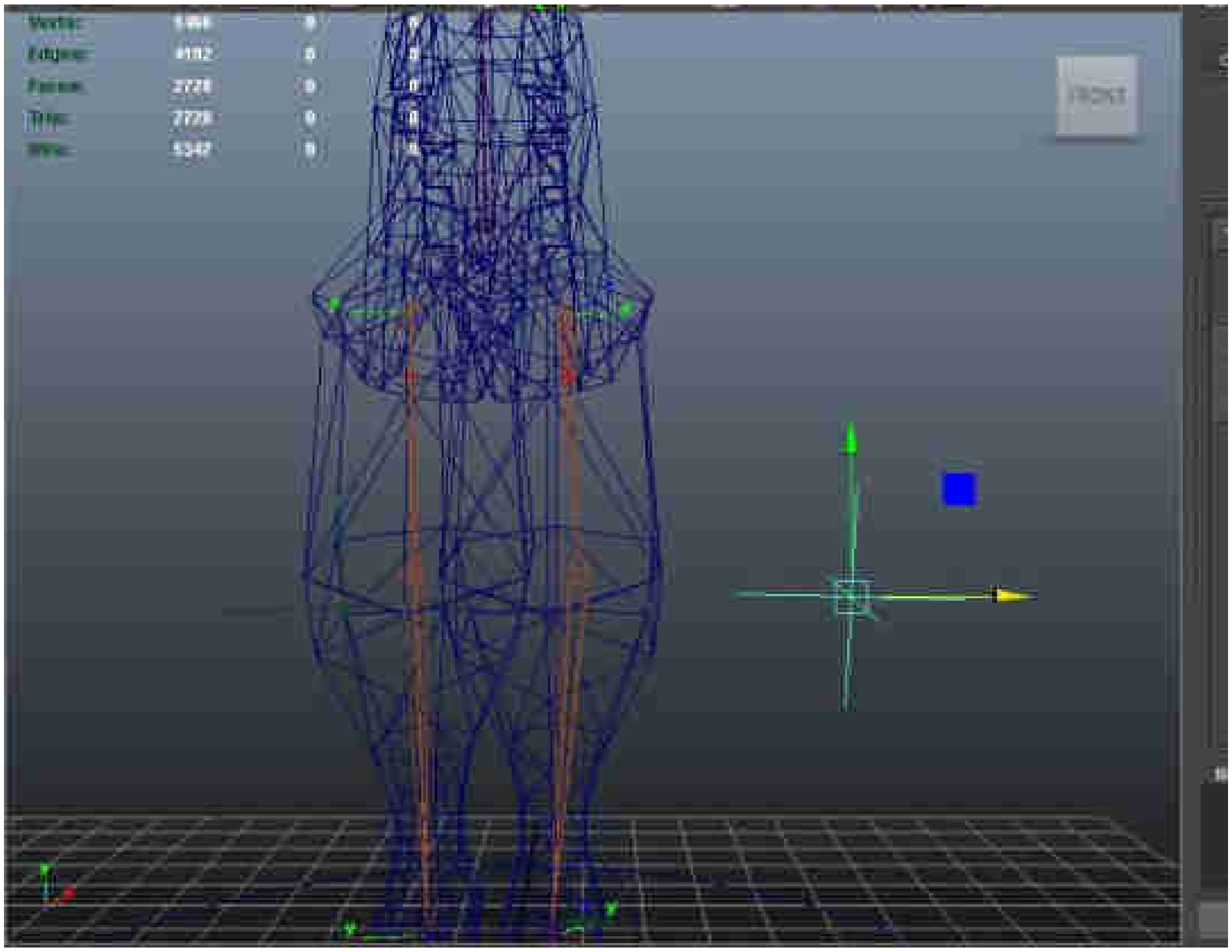
Week 8- Constraints & Attributes (2015-12-10 15:40)

This week we had to apply some constraints to our model.
This would allow the model to act a little more like a human body, especially
the knees and elbows. First we started with the wrist by clicking on its
control shape. Then the “Constrain” option was chosen at the top and then the “Orient”
menu was opened from here. Once this was set up, the wrist joint rotated with

the control shape. The same was done to the other wrist.



Next up were the knees. To move the knees appropriately, I
added locators to the scene which can be set up with a pole vector. The locator
was set up in the same way as a control shape, with a group and being named
appropriately. Then it was moved towards the knee but was pulled out in front
of the knee so it was visible. After the locator was in position, it was
selected, the shift button was pressed and the ankle IK handle was selected
too. Now, the constraint menu was opened again and the "Pole Vector" option was
chosen. The knee now pointed towards the locator. The same was done for the

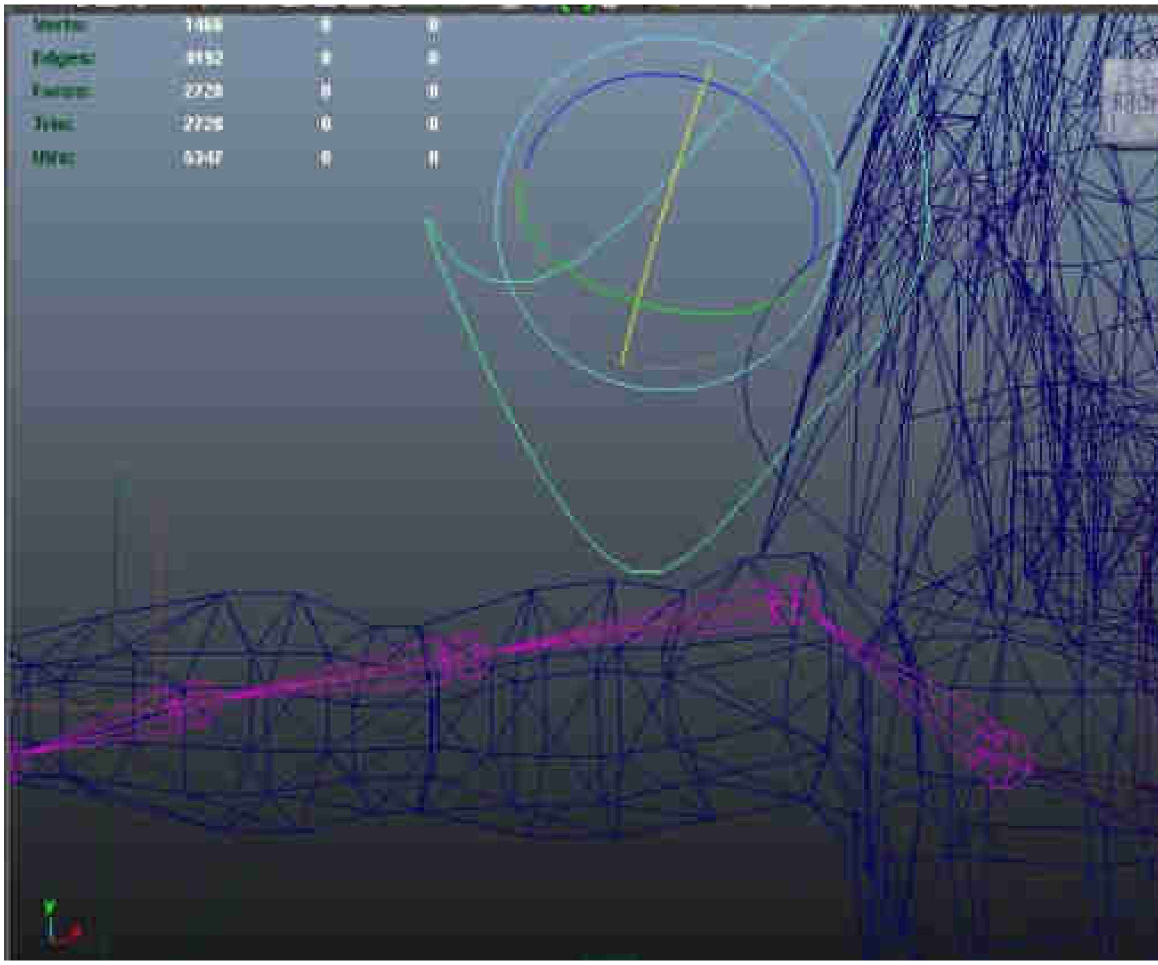
other knee. 

```
Verts:    5466   0
Edges:    8182   0
Faces:    2728   0
Tris:     2728   0
UVs:      5347   0
```

The same type of constraint
was used for the elbow.

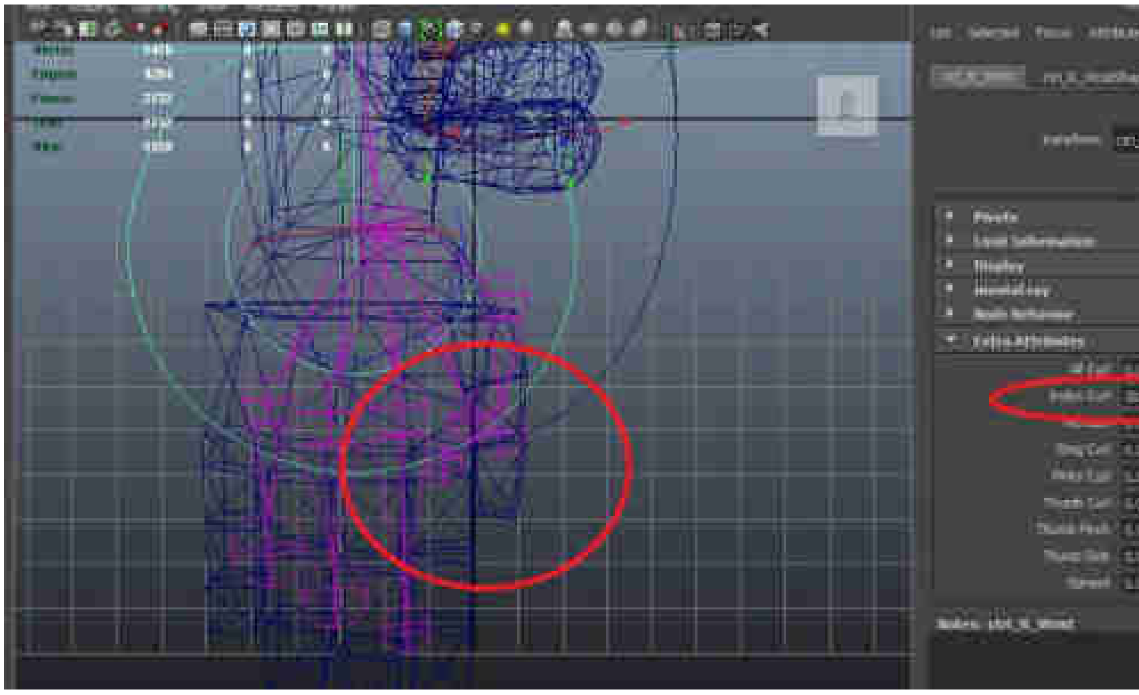
Next
the shoulders were set up. I used an orient constraint on the shoulders. The
control shape was placed outside the mesh and reshaped a bit to more resemble

the curve of the shoulder.



The
hands were next to be worked on, as the fingers needed to know how to bend and how far to do it. This would be achieved by using the set driven key tool provided by Maya. First the wrist’s control shape was selected and the Modify menu from the top was selected. From here “Add Attribute” was chosen and the new attributes were named, describing what finger(s) was/were being moved. After all attributes were added, the “Set driven key” under the animate menu was chosen. From here, the “Load driver” button was pressed and then the wrist controller was selected. Then the new attribute “Index _curl” was chosen in the driver window. Now the index finger joints were selected in the viewport and then the “Set driven key” button was pressed. The attribute to be set here was the Z axis, which would rotate the finger appropriately.

It
was time to hit the “Key” button in the menu, before visiting the attribute editor for the wrist controller. After this, the “Index _curl” was set to 20, and then the finger joints were rotated (in the z axis) to the maximum point that I desired. Next, when the finger was bent correctly, the key button was pressed again. This would set this finger bend as the maximum of the “Index _curl” attribute, and the starting point as the minimum. The same was done for all the

other fingers and their attributes. 

The neck and head joints were given orient constraints towards their respective control shapes. The jaw's group was made the child of the head joint, and the neck control shape was made the parent of the head joint.

The effector of the forearm was moved further down towards the wrist to try and lesson the candy-wrapper effect on the wrist.

Specialization (2015-12-10 7:39)

I chose to specialize in video game rigging, since I felt it would be most helpful for my future aspirations. I could be able to understand the steps taken by the Technical artist in optimizing a model for video games.

A major difference between rigging for video games and for animation/film, is the limitations you are faced with. Video Games need the optimum performance for the player, so too many bones can disturb this good performance, depending on what is done with them. It is costly because, when the character is animated, the data for each bone has to be saved, and the more bones mean the more data needs to be saved. This along with player interaction (the player controlling the character), important calculations taking place in the environment (collision detection, physics, AI etc.) as well as other art assets, can take a large toll on the performance. Time and money can also hinder the final product.

However some games can occasionally break the rules. For example, the visually stunning "The Last of Us" had characters with 98 bones in the face alone. Even with this though, the game was very focused on story and

each expression in the face was vital in telling the story.

Although it was
extremely well done, there were some performance issues, with the frame rate only being 30 FPS in the PlayStation 3 version of the game. The designer even explained that they were trying to milk the PlayStation's capabilities at the time until it could do no more. This was improved however, with the re-release on the Playstation 4, which was more powerful and could handle more. They also used their own engine to handle the characters, so was more customized to their needs. Engines like Unity would be more limiting in development.

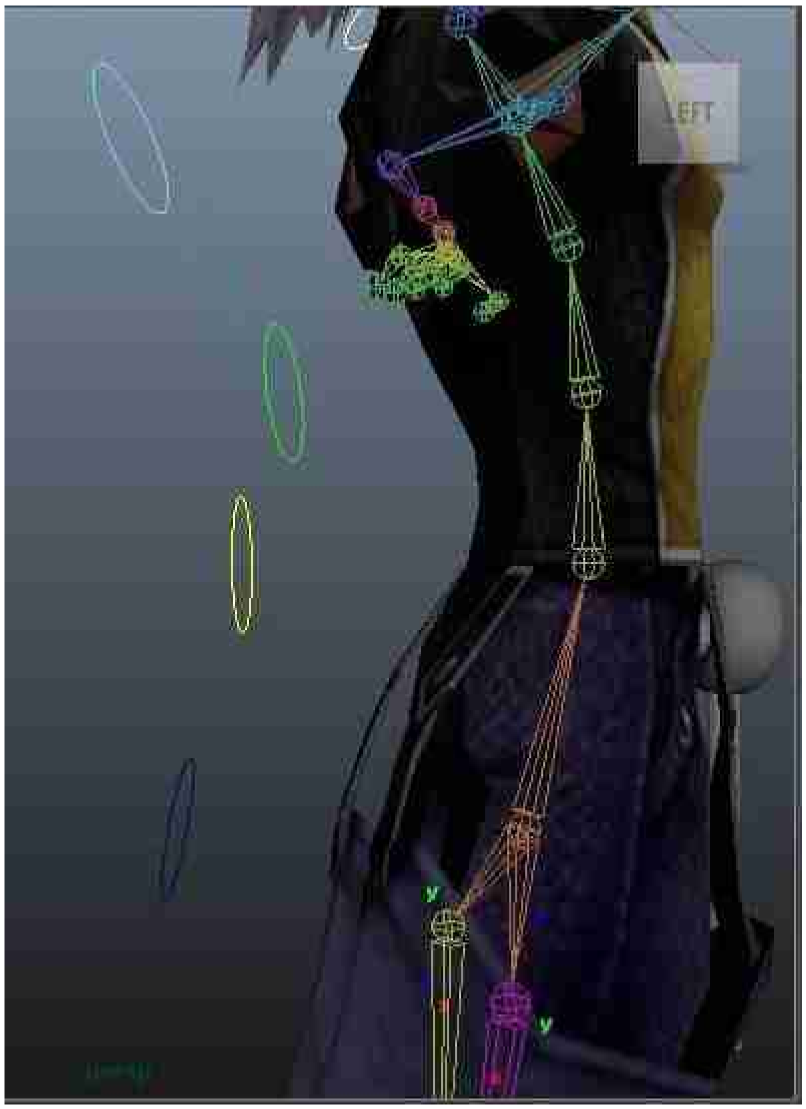
Another limitation for game rigging is the platform being
developed for. Next generation consoles would be a lot more capable of handling high-detailed, perfectly rigged models than a mobile would, as they have more processing power. Whereas platforms like mobile have a lot less power, and are recommended to have under 30 bones per mesh according to Unity's website.

Week 9- Spine and Face (2015-12-13 20:07)

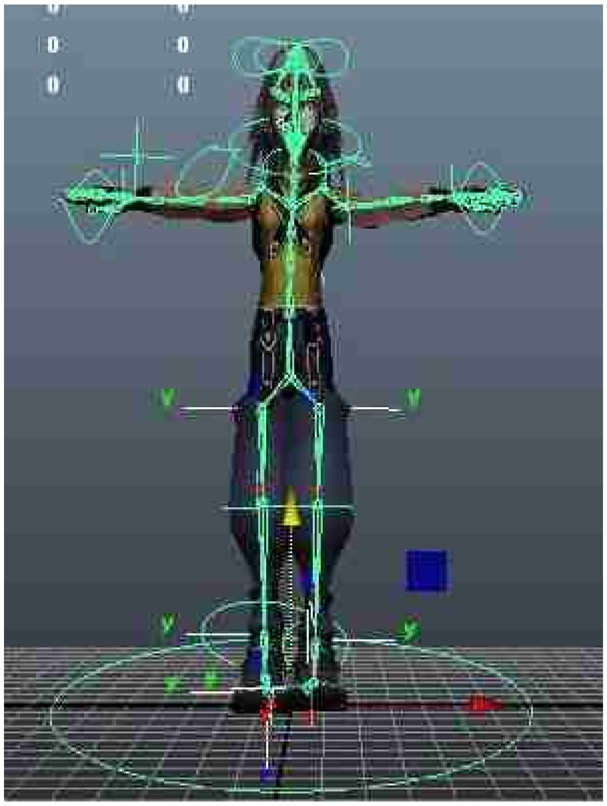
This week we learned how to control the spine. We were given the option of either creating an IK spine or an FK spine. The IK spine involves using the spline IK tools which, if set up correctly, would give the character more flexibility. However, because the platform target and the fact that the model is a humanoid character, it seemed like the better choice to go for the FK spine.

The FK spine revolved a lot around using control shapes and parenting them to the appropriate joints. First, control shapes were made for every joint inside the spine and snapping them to the appropriate joint's position. The shapes were then repositioned to be outside the body in order for the user to easily select them. Now constraints for each joint had to be created. To do this, the root joint's control shape was selected, followed by the root joint. Then I dropped down the "Constraint" menu at the top of Maya, and went to "Parent Constraint". This would move the root joint and rotate it whenever the control shape was moved or rotated (different from the orient constraint as orient constraints only allow rotation of joints using control shapes). Now the root joint had a control shape and would move whenever the control shape did.

Although normally the other spine joints would only need an orient constraint placed on them as rotation is usually all that is needed with the higher spine joints, I found it a little frustrating not being able to translate my joints in certain situations. Although I didn't want the torso to stretch too much, I preferred to use the parent constraint on all of the spine joints. I tried to combat torso stretching by locking some of the attributes, only allowing the user a certain amount of freedom when both translating and rotating the joints. Therefore, the exact same method was used on all my spine joints. The shapes were also coloured to

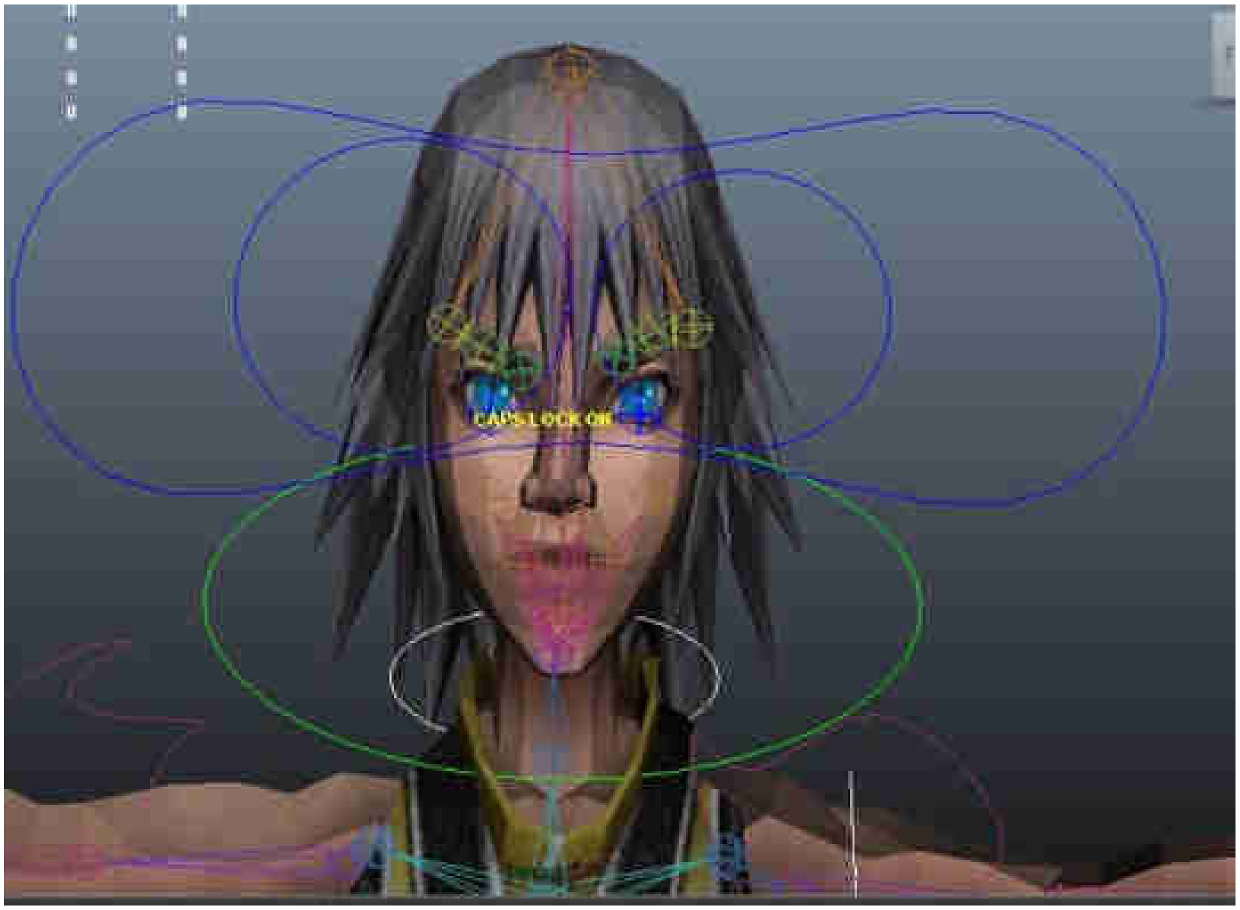


Next the hierarchies were sorted out to ensure the correct joints were controlling others. To make this happen, the root joint control shape became the parent of the first spine joint group, which became the parent of the second, and the second became the parent of the third (by highlighting both joints and pressing ‘P’). The rest of the body was set up this way. This was finished off by creating a master control shape which would move and rotate the entire hierarchy.



For the face I decided to use joints because blend shapes seemed like a more costly option for the game engine. I decided that the face in this game would probably be a bit of a waste of performance, since it is intended to be for a platformer and mostly based on game-play and little to no 3D model cutscenes, however I wanted to show how the face might be set up. This model was not given eyes or a gap for the mouth.

The face I constructed consisted mostly of the mouth, creating five joints for each corner and important parts of the lips. There were also joints for cheeks, all of which could make the character smile, and possibly talk if given a mouth sock. Also the character was given joints for the eyebrows, stretching from the top of the head.This was given an orient constraint. I felt that considering the platform and the gameplay-focussed game being targeted, the eyebrows and the mouth were enough to convey enough emotion. Most of these joints were locked to ensure the user didn’t translate or rotate them too far.

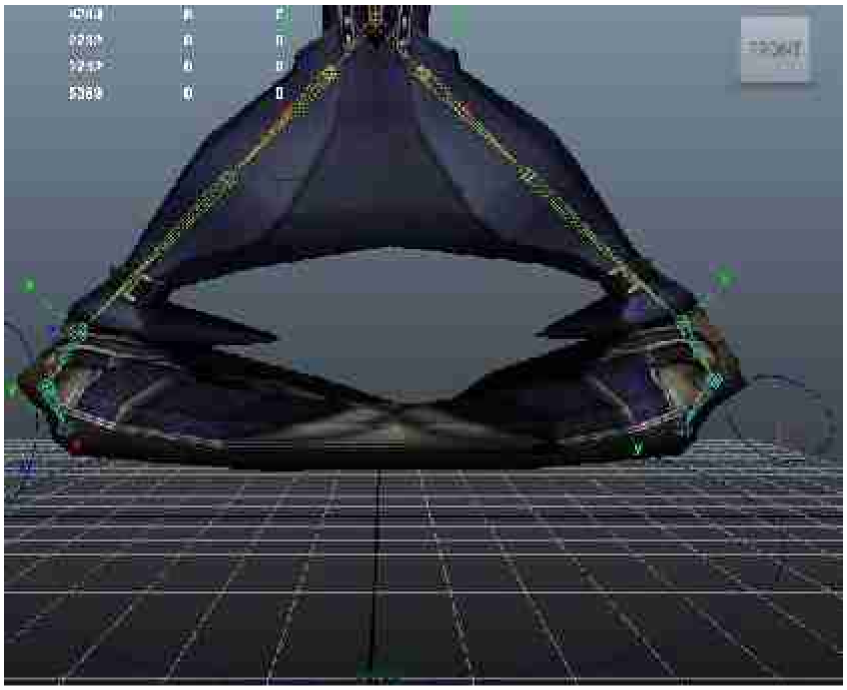


Overall, this lesson was very useful and gave new life to the models being created.

Week 10- Skinning **(2015-12-14 10:06)**

Here I tackled what I would say was the most frustrating part of rigging- skinning the model. This is the stage where having made a good model choice with good topology etc. can pay off. My first mistake was choosing a model that was not very suitable.

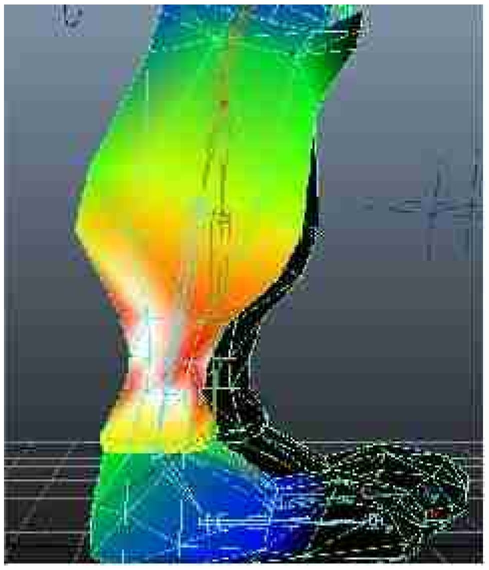
First of all, don't choose a model whose legs are joined together. This was my first problem because when it came to skinning, the model looked like the image below.



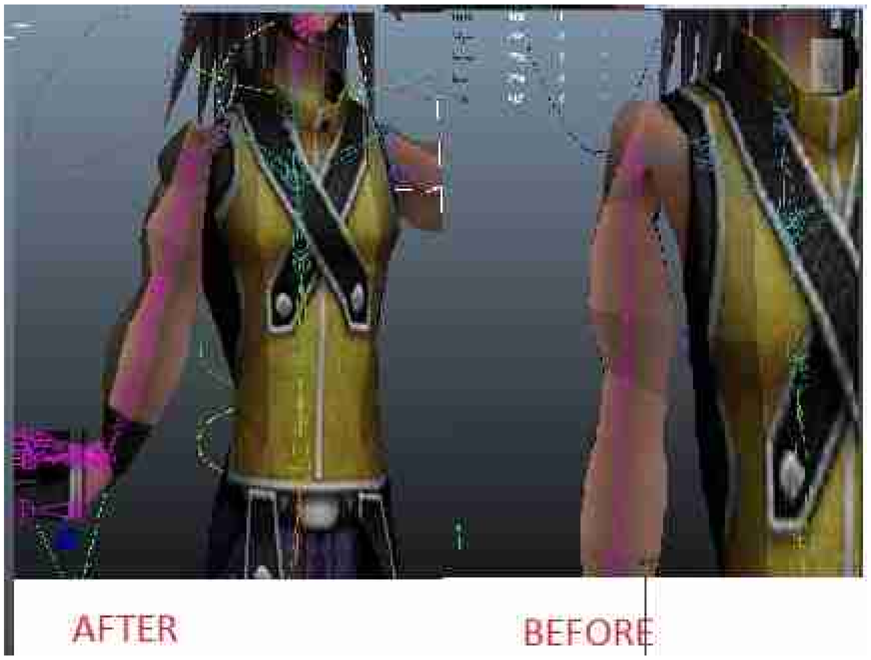
This was not fun to pick apart. Another frustrating part about the model I chose were the clothes. The bagginess of the trousers and the braces attached to them were extremely frustrating to get to work in a way I wanted them to.

In order to skin the model, all the joints were selected along with the mesh and then the “skin” menu at the top of Maya was visited. From here I selected “Smooth bind” and ensured the vertices limit was set to 4 as this was the limit for some game engines. This attached certain parts of the mesh to the nearest joints, so if a joint were moved a nearby part of the mesh would too.

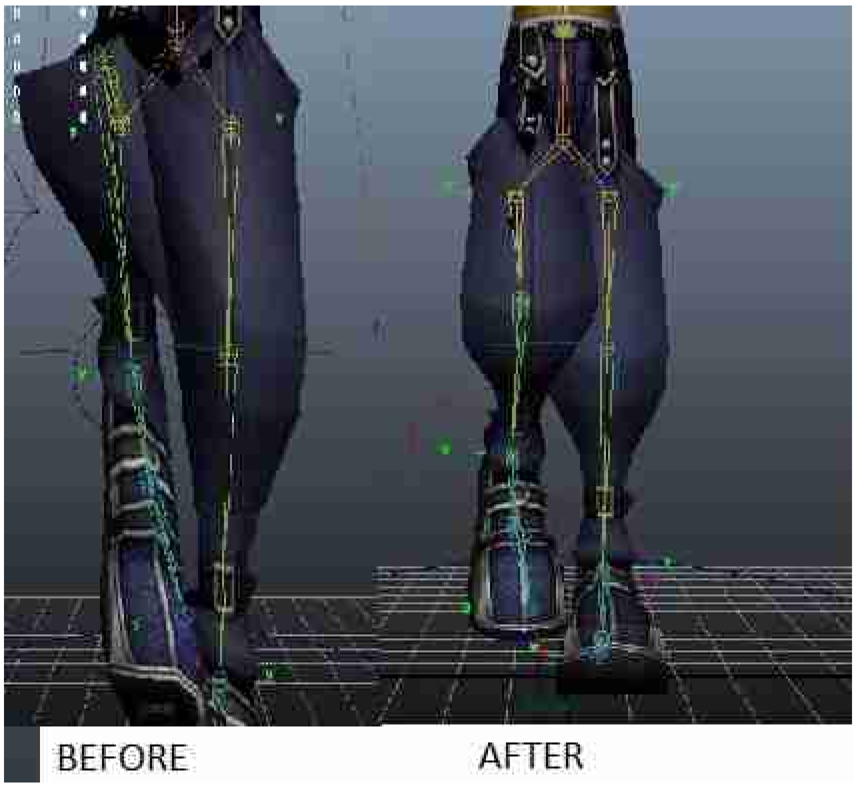
Next was the frustrating part. Weight painting. In the same menu as before, we select the “Paint Skin Weights” tool. This came up with a new tool that showed how much each vertex was being influenced by the joints surrounding it. How this was shown could be changed, either being a grey-scale paint, or a colour spectrum paint.



Some of the more difficult areas not mentioned above were the shoulder and knee bends. It was very easy to have some parts of the mesh being affected too much, so the knees and arms would lose a lot of volume which isn't desirable. Although the skinning wasn't great for the shoulder, you can see some improvement to ensure the shoulder didn't lose too much volume.



Similarly the knees were not skinned perfectly but there was a giant improvement



Although skinning was extremely frustrating, it was a really interesting and vital lesson in rigging, and was the final step in bringing the model to life.

Game Type (2015-12-14 10:23)

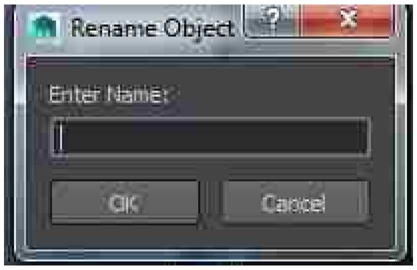
I decided a few weeks ago the model would be for a 3D platformer, therefore the game-play is the most important part. The facial structures have only been included to show how the control shapes may be set out, and if developed more could allow the character to do simple talking.

This is also why the fingers have only been given two bones each instead of three. I was going to give him just one bone for all the fingers (mitten effect) to lessen bone count, however I wanted to give more freedom to the character. The fingers give the option of allowing the character to wield a keyblade.

Scripting (2015-12-14 10:25)

For the script, I wanted to create something that tackled some of the more monotonous tasks I faced during rigging. One of the most repetitive tasks was the creation and grouping of control shapes.

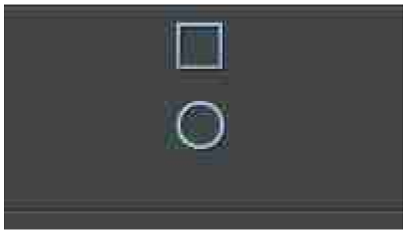
My script would first enable the user to choose between a circle control shape or a square shape. Then it would ask the



user to give the shape a name, by using a "prompt dialog" window.

The user would no longer have to write out the the prefixes and suffixes of the shape or group, as the tool would add in the "ctrl _" and "grp _... _offset" on its own. All the user would have to do now was parent the group to a joint and re-position it on their own. I found this to be a useful little shortcut, and helped ensure all the shapes were named appropriately.

The UI design for this was decided as the buttons being pictures. This would allow the user to easily recognise what the tool was going to create, plus would eliminate the need to localize this part of the tool.



Scripting - Picker (2015-12-14 11:02)

Another part of the script I wanted to add was another problem I faced in the earlier stages of rigging- finding the control shapes. Sometimes it can be difficult to see these shapes if they are not in the optimum position or if they are being viewed from the wrong side. This is why I needed to include a picker in my script.

A picker was created by first loading in an image of the model as the background of the window. A form layout was used, which gives a lot of freedom for placing buttons and images, allowing the user to position them in an exact position instead of being in a column or row. This code was used to use the image in the layout. This would start the image at the top left, and would stretch depending on the image initialization earlier in the code.

**maya.cmds.formLayout(layout, edit=True,
attachForm=[[image _name, "top", 0], [image _name, "left",
0]])**

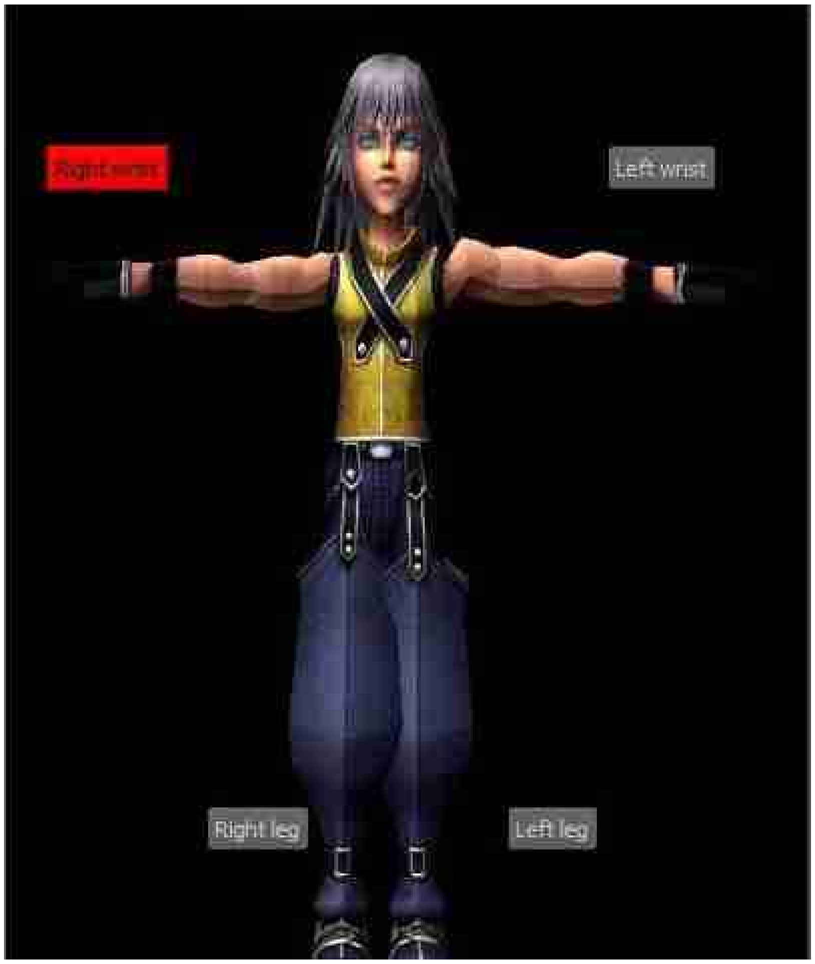
Now the buttons had to be placed appropriately on the body. First the buttons had to be set up as usual. The button was named, and was given a background colour to make it stand out more.

**r _wrist _button = maya.cmds.button(backgroundColor =
[1.0,0.0,0.0],label="Right wrist")**

Now the layout had to place the button in the correct position. This is very similar to how the image was added. The form layout allowed the words "top", "bottom", "left" and "right". These would just signify where the object would be placed, for example ("top", 20) would simply mean 20 from the top.

```
maya.cmds.formLayout(child3, edit=True,
attachForm=[[r _wrist _button, ”top”, 70], [r _wrist _button,
”left”, 20]])
```

This would set the button near where the wrist is situated in the picture as shown below. The same was done for the other buttons too.



Scripting- Animation (2015-12-14 11:40)

While looking at my rig for the fingers and the attributes created for them, I thought that I may be able to make it easier to see the fingers being moved instead of typing in the position, since this can involve a bit of trial and error when looking for the best finger position. I thought that a slider would make more sense here, since the animator would be able to see the fingers move in real time. Also, it helped me when I was skinning so I could easily animate them and be able to tell where the skinning had gone wrong.

To do this the script would need access to the attributes. These could be accessed by using the “setAttr” function.

Setting up a slider involved using a float slider group. This was set up like so:

moo = cmds.floatSliderGrp(label = ’All Right fingers’,field=True, min=0, max=20, value=0, dc=MoveFinger)

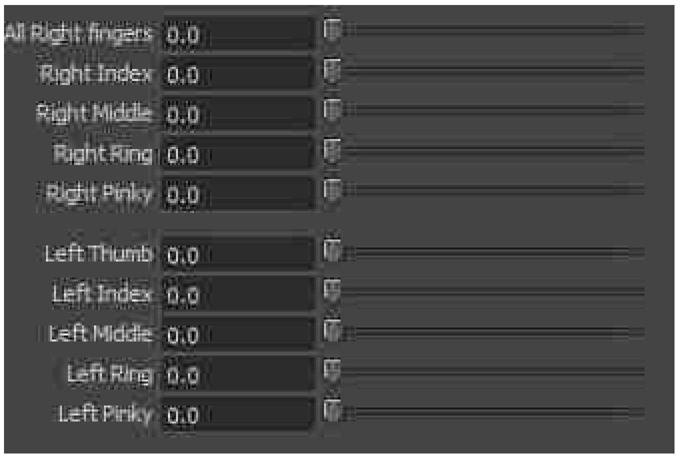
This code would eventually move all the fingers on the right hand, had a maximum of 20, a minimum of 0 and would execute the “MoveFinger” function as its command.

The vital part though was how the function was set up. This time the slider would be updating positions with every movement instead of just feeding the function one answer. Therefore, the function had to be fed a parameter. This would get the number from the slider and update it as it is moved.

def MoveFinger(*args):

cmds.setAttr(”ctrl _R _Wrist.All _Curl”,*args)

Here, “args” would be the number read from the slider and would update as the slider moved, resulting in moving fingers.



All Right fingers 0.0
Right Index 0.0
Right Middle 0.0
Right Ring 0.0
Right Pinky 0.0
Left Thumb 0.0
Left Index 0.0
Left Middle 0.0
Left Ring 0.0
Left Pinky 0.0

The same method was used for every finger.

Scripting - Joints (2015-12-14 14:06)

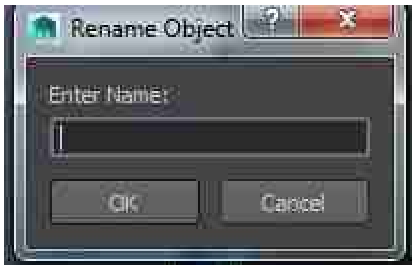
Another piece of functionality I wanted to add to the script were the addition of adding joints, and positioning them in the user’s specified place. This would allow the user to add the joint wherever they wanted, by inserting the position into the fields.

To do this I used the floatFieldGrp function. First I set the number of fields to 3 (aka the x, y, and z positions), by using this code:

jointfields = cmds.floatFieldGrp(numberOfFields = 3)

Next, a button was made to run the command that would decide what the input in these fields would mean. This would be called “placejoint()”.

In the placejoint() function, a new promptdialog window would be created, similar to the control shape maker.



After the user accepts, this will place the joint and rename it with the prefix “jnt _”. However, the function isn’t finished yet. We still have to make the fields place the joint in the right position. Thew following code was written:

if result2 == ’OK’:

jointname =
cmds.promptDialog(query=True, text=True)

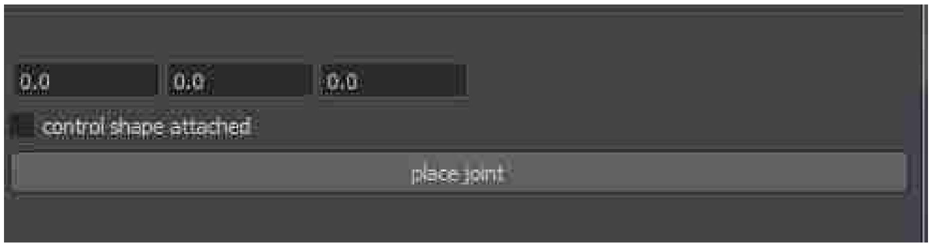
jointX =
cmds.floatFieldGrp(jointfields, query = True, value1 = True)

jointY =
cmds.floatFieldGrp(jointfields, query=True, value2 = True)

jointZ =
cmds.floatFieldGrp(jointfields, query=True, value2 = True)

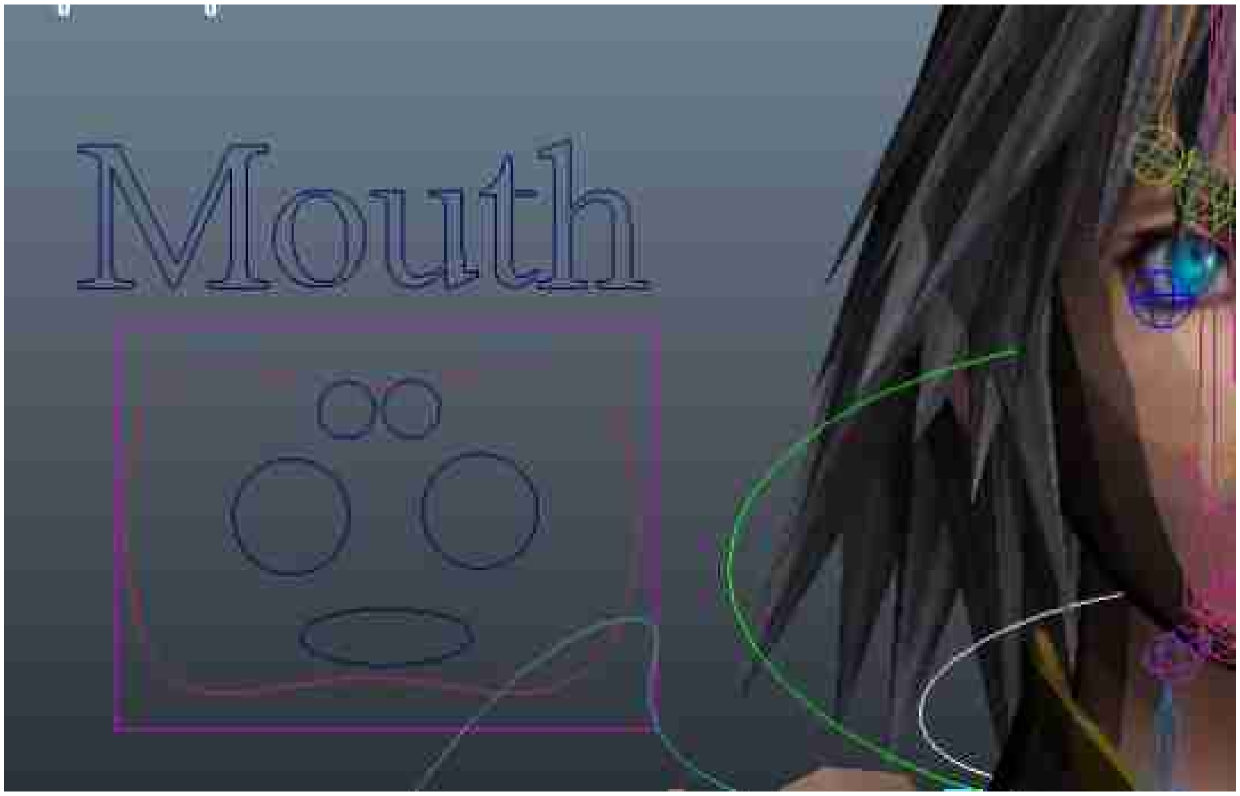
jointmade =
cmds.joint(name=’jnt _’+ jointname ,p=(jointX, jointY, jointZ))

This means that if the user presses the ok button, the name written in would be kept as a variable. Then each field is assigned to the variables jointX, Y and Z (value1 is the first field, value2 the second and value3 the third). Jointx would become the x position of the joint, jointY the y position and jointZ the z position. Lastly, the final joint is created with the name made by the user and the position also chosen by the user.



Making Things More User Friendly - Rig (2015-12-14 15:04)

After doing so much to the rig and the script, the interface looked a bit of a mess and was still quite difficult for the user to use well. Some of these could be aided a little after some extra steps were taken. We’ll start with the rig.



1. Locked a lot of the attributes on some of the control shapes. This would ensure that the user wouldn't accidentally stretch the mesh too far. This was evident in the spine controls, stopping them from twisting too far, but also in the legs, face and shoulders.

2. Added a few set driven keys. This allows easier posing by just typing in a number. These were done in the fingers and also in the bending of the legs.

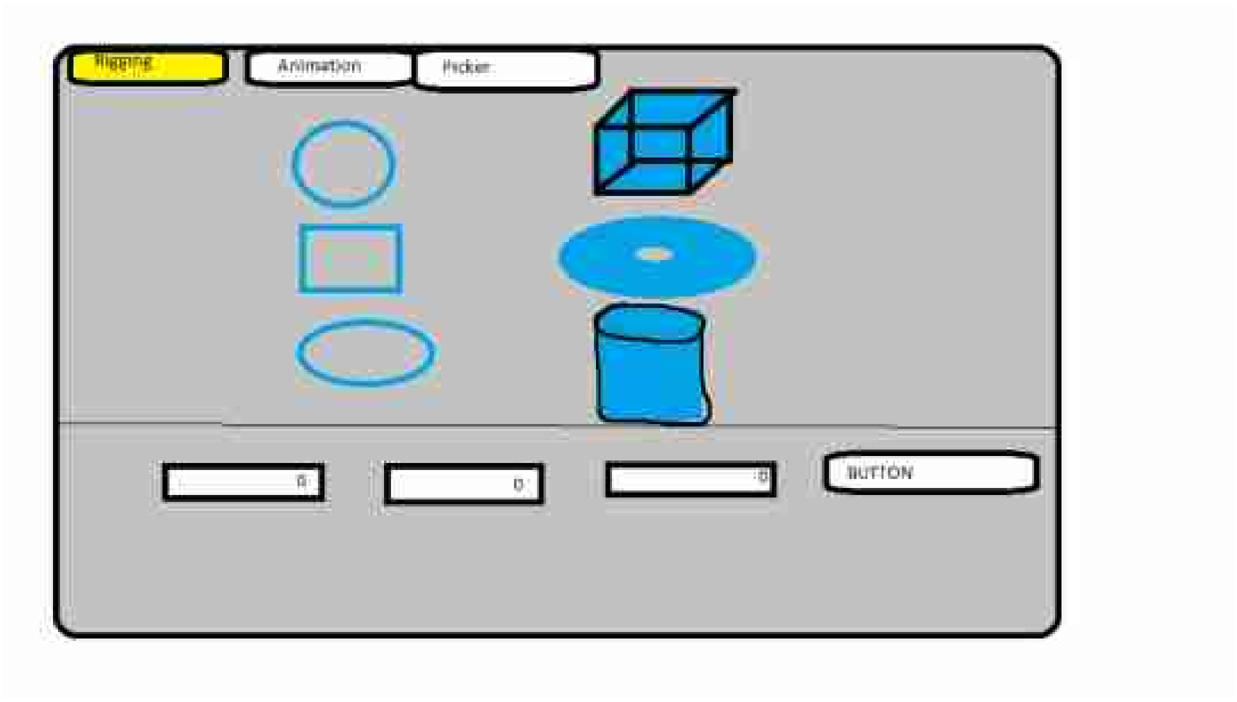
3. Changed the colours of the control shapes to make each more visible and diverse from the ones around them.

4. Changed the shapes of the control shapes into odd shapes to make them stand out from the mesh. This is visible in the eyes (although not fully working eyes but shows how the control shapes would be placed). Also the shoulders are shaped oddly too. Similarly, the mouth controls were placed outside the rig and was clearly labelled as shown above.

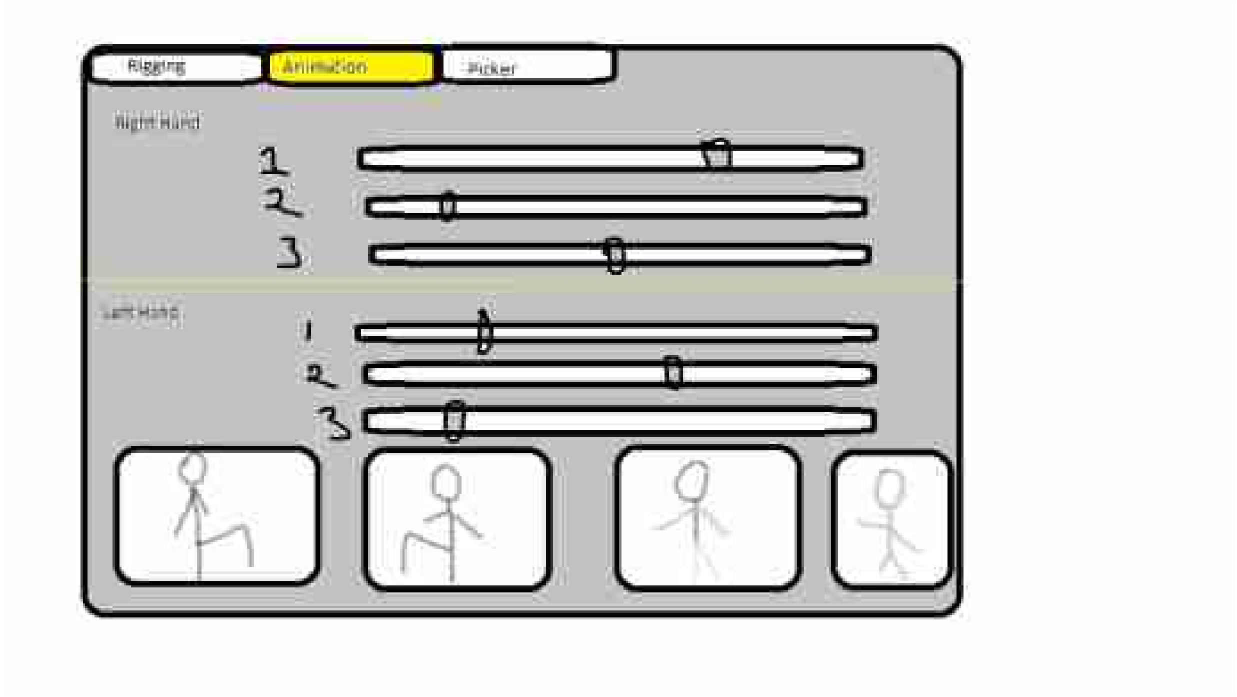
Making Things More User Friendly- Script **(2015-12-14 15:46)**

At the moment my script contained all the features in one window. These made no sense all together, the window was too big and cluttered and it wasn't very attractive. Since I had a tool for both riggers and animators, I decided the tool would look better organised into tabs. From there the rigger or animator could go to their section and use the tools designed for them.

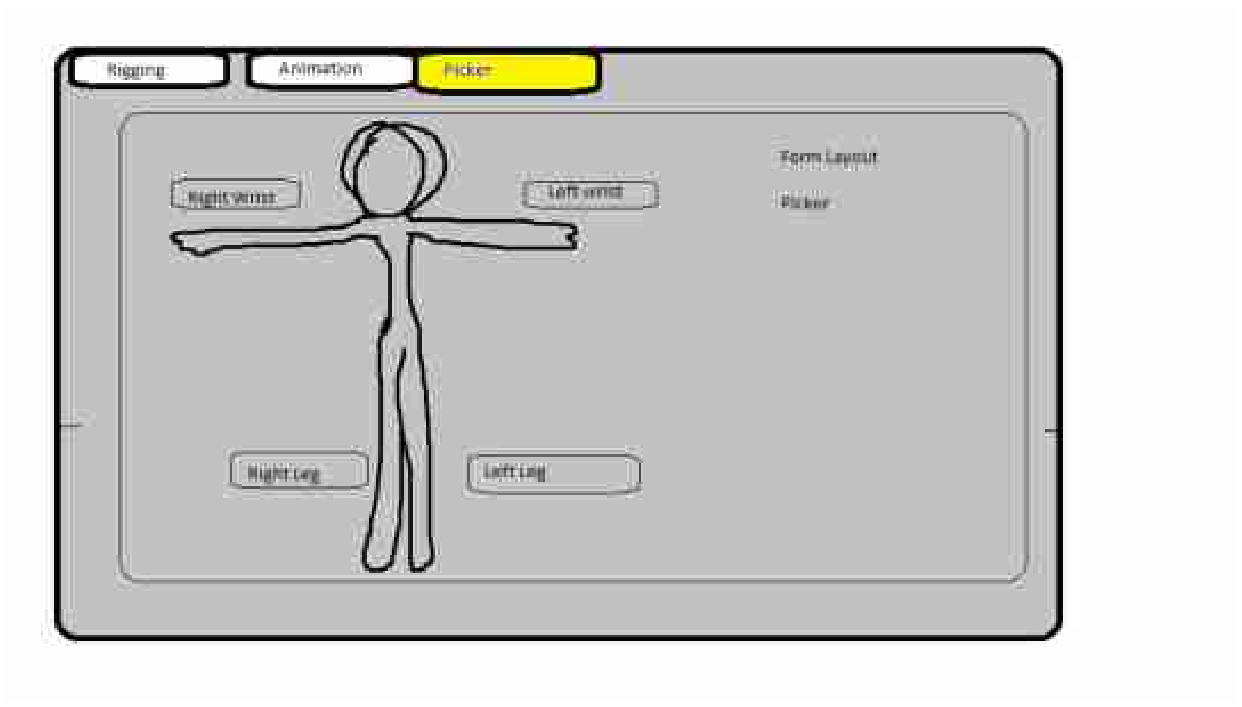
Before coding the final layout, I drew out the design I had in mind. These can be seen below.



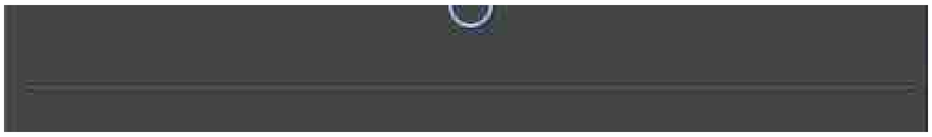
This would be my first tab for the rigger, which allows them to place and rename control shapes, place normal shapes that may be used to create a keyblade. (This could help with rigging the hands as they may need to re-work the hands to make it fit). Also the user can place a joint by putting the position in the fields. To achieve this layout the tab was set to a row/column layout.



This was the animator tab and allows the animator to curl the fingers using the sliders. Also the symbol buttons at the bottom put the character in the positions depicted. The layout was a form layout to ensure the buttons were in approriate places and so the text and separators could b placed where they were needed.



And finally the picker. This was also a form layout so the buttons could be moved to the perfect position.

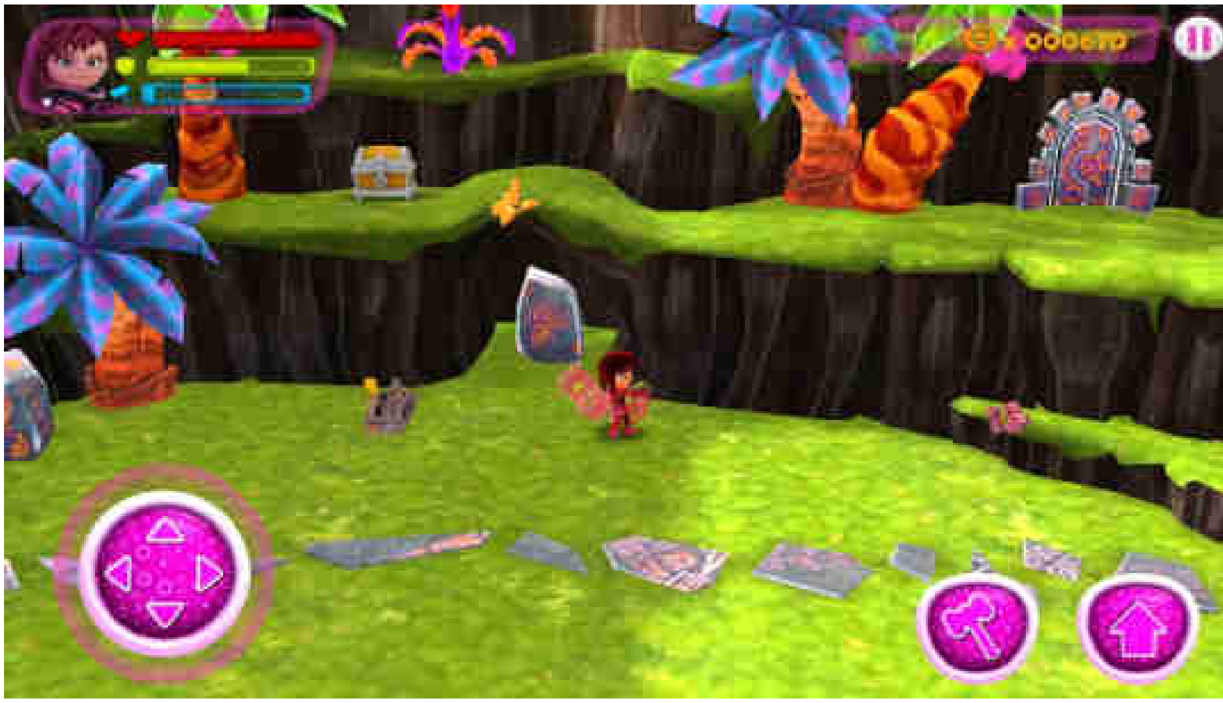
Other features I added to help with the UI were separators that blocked off certain parts from each other. These came in different styles. 

Also, text was added to certain sections such as the finger slider section, to allow the user to know that the first group of sliders were for the right hand, and the second group for the left hand.

Games **(2015-12-15 14:49)**

I decided to have more of a look at android games. Here are some images of the games I looked at.

First there were some 3D platformers.

(Glenn, 2011, Cordy) 

(Blyudov, 2012, SiliBili)

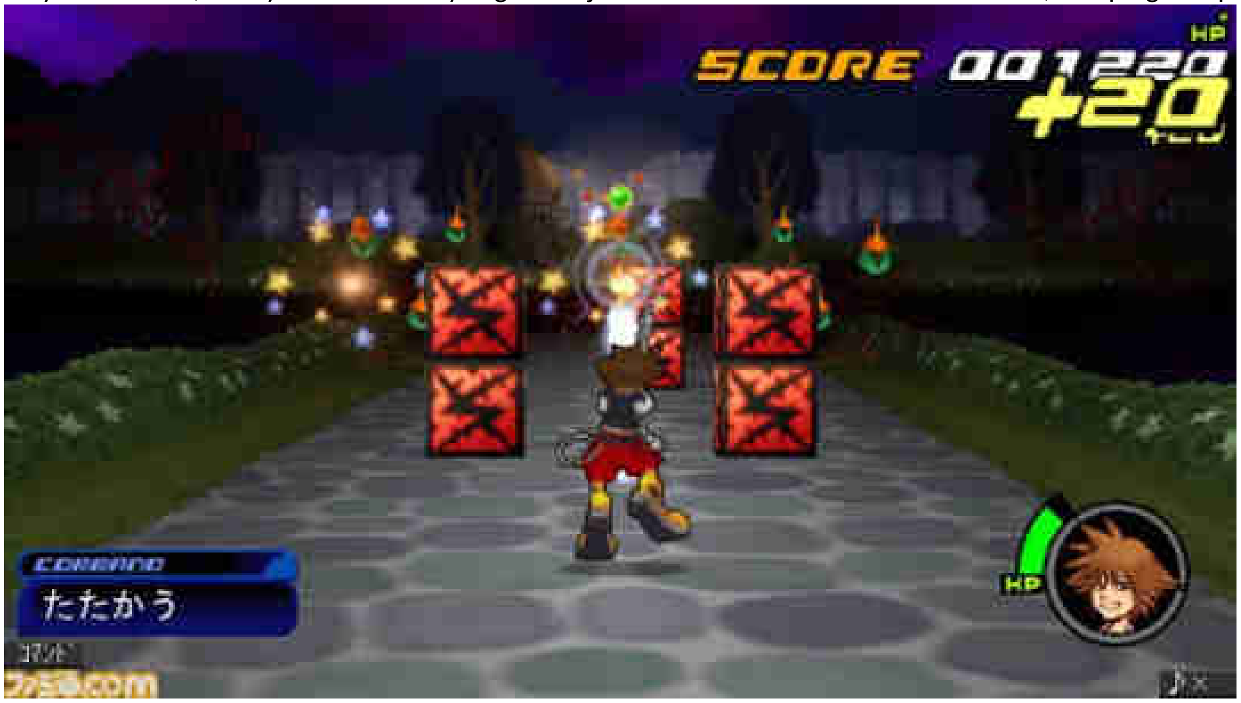
This last one in particular inspired me. You can see there would be very little need for rigging in the face, however the facial expressions could be used in the status bar up at the top.

However, a similarity these share are the facts that these characters are on the lower end of bone counts. To make my model fit in with these, I would get rid of the finger joints altogether, as well as the face joints. I'd also make the spine have less joints and possibly the feet.

Looking at these games reminded me of a game from the series this character belongs to. The game was Kingdom Hearts Re:Coded, recreated for the Nintendo DS and was originally for a Japanese mobile. It would be quite interesting to see how mobile devices would handle the game nowadays.

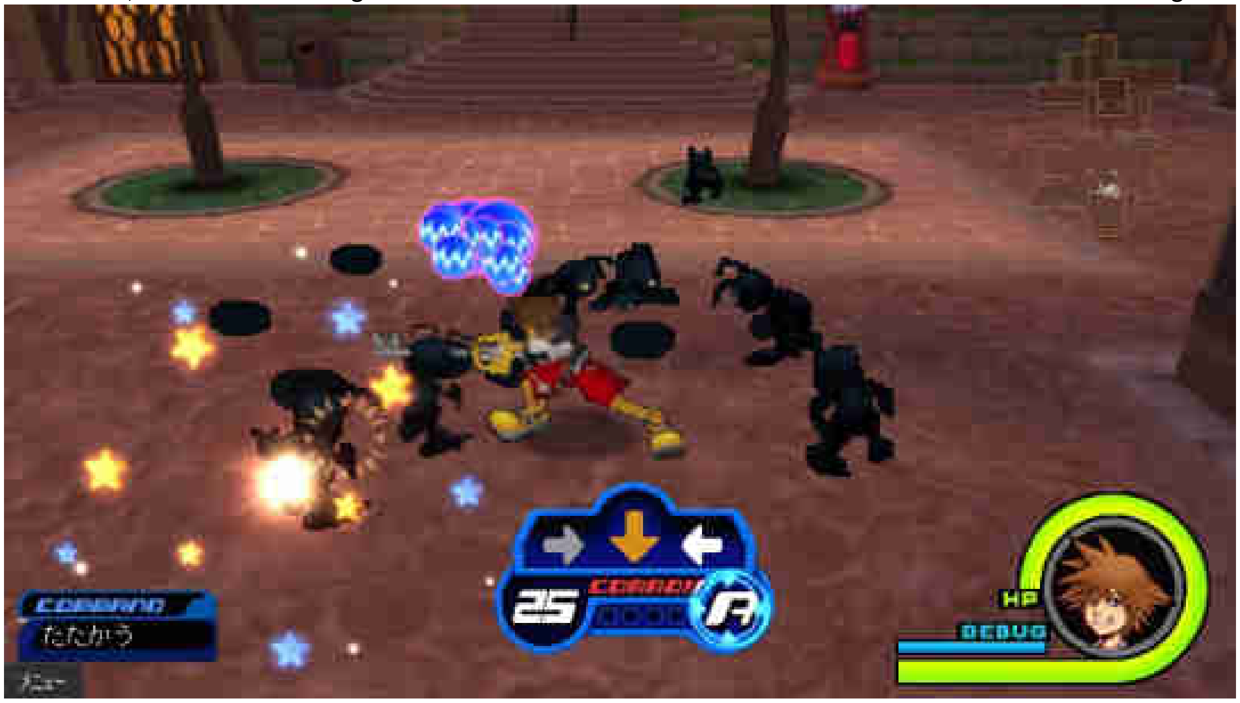
(Ric, 2009, *khplanet.com*)

Kingdom hearts Re:Coded was a game that was made up of a lot of different genres. One level would be a platformer, the next a first person shooter and even a turn-based RPG. The battles were also very controlled, as you could only fight major enemies while in certain areas, helping in performance.



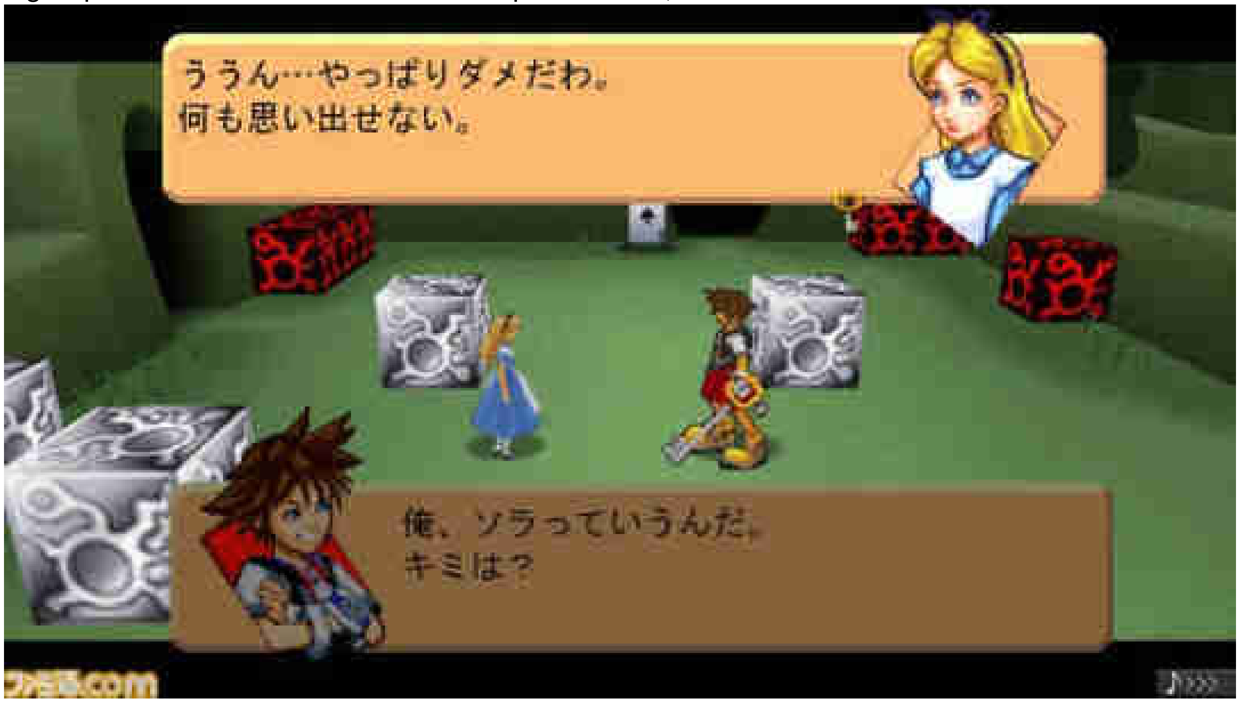
(tapcash, 2015, diendan.vietgiaitri.com)

Also to help in performance, the game was released in chapters, the game-play only needing to keep up for the length of a level, while still having pretty good graphics for a mobile game especially for its time (These were being released between 2008 and 2010 and the DS version being out in 2011).



(tapcash, 2015 diendan.vietgiaitri.com)

Cut-scenes were kept mostly to text scenes, with only stills of the characters showing up. This would also save performance, and have less need for a lot of animations.


うう ん…やっぱりダメだわ。
何も思い出せない。

俺、ソラっていうんだ。
キミは？

(Avengelho, 2011, videogamewriters.com)

I decided that it might be quite interesting to design the rig for a remake of a remake, where the game Coded or Re:Coded is remade back onto a mobile device (or more likely a tablet).

Cut-scenes would be kept as text scenes with character stills (so the model's body would be posed appropriately while the face is drawn in 2D).

References

Avengelho.2011.Kingdom Hearts Coded. [online Image]. Available From: http://videogamewriters.com/review-kingdom-hearts-recoded-1928 . [Accessed 6 December 2015]

Blyudov.2012.SiliBili.[online Image].Available From: http://app4smart.com/en/819-silibili.html . [Accessed 29 November]

Glenn.2011.Cordy.[online Image].Available From: http://www.androidauthority.com/360-security-review-a-lot-of-bang-for-zero-bucks-652964/ . [Accessed 5 December 2015]

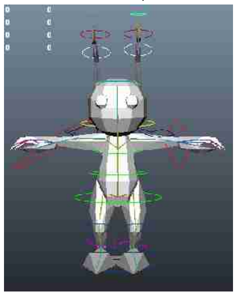
Ric.2009.Here's a Picture of the Cell. [online Image].Available From: http://www.khplanet.com/forum/threads/kingdom-hearts-coded.358 2/ . [Accessed 10 December 2015]

Tapcash.2015.Kingdom Hearts Coded. [online Image]. Available From: http://diendan.vietgiaitri.com/chu-de/kingdom-hearts-39nguoi-anh-em3 9-cua-final-fantasy-se-co-mat-tren-di-dong-1603905.vgt .[Accessed 30 November 2015]

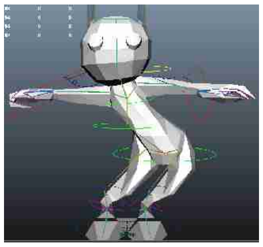
Enemy character **(2015-12-15 15:12)**

I decided I would have a go at creating an enemy that may be shown in the game. Since the enemy would have a lot of copies, it was important that they have less bones than the main character.

One of the most recognizable enemies in the Kingdom Hearts series is the Shadow Heartless. I gave one model an unfinished rig to give an idea of how an enemy might be rigged. As can be seen, the control shapes were appropriately placed and easy to identify as they were coloured and reshaped.

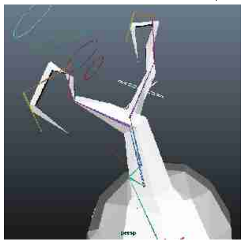


First of all, the heartless was given less spine joints and arm joints than the Riku model. He was also given absolutely no face joints, and only had three "fingers". The rig was also given no constraints either, giving the model full flexibility. This could be useful as these enemies are known to deforming down into puddles.

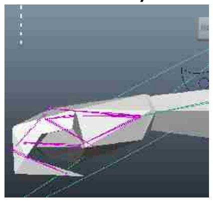


The rig was not finished due to time constraints (the arms and legs are not properly set up) but it gives a good idea of what they may look like.

A couple of big differences from the Riku model were the antenna and the feet. The antenna were set up using IK handles, to make them bend in a more realistic way. If I were to get rid of any bones I could remove some of them from the antenna, however I thought i'd give them more because they are a distinguishing feature of the Heartless.

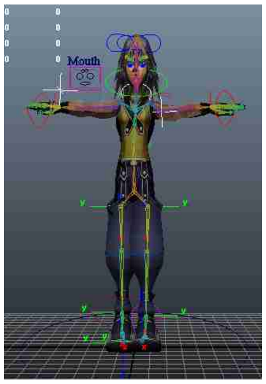


The feet were also set up using an IK handle.

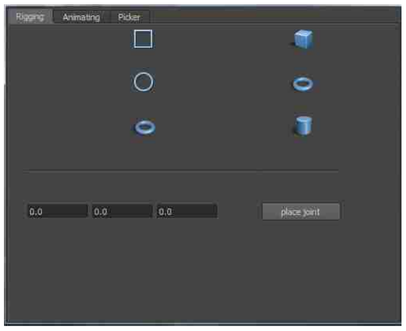
The claws were set up the same way as Riku's hands, with the set driven key tool. These could also be scrapped, but I decided to keep them since it is what they use to attack with. 

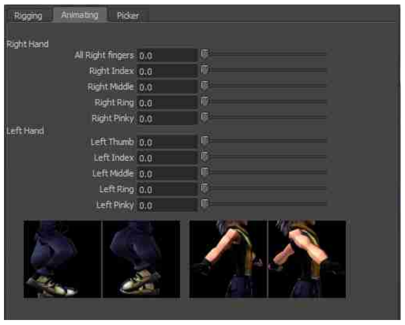
I'd like to finish this rig eventually and have him properly set up, but for now I feel I've shown the more important and unique features.

Rig and Tool (2015-12-15 15:18)

The Rig: 

The Tool:



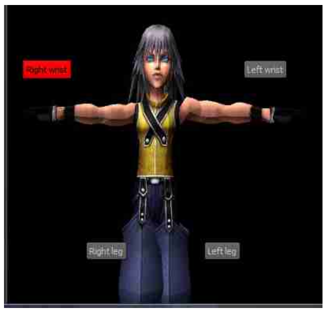


Right Hand

All Right fingers	0.0
Right Index	0.0
Right Middle	0.0
Right Ring	0.0
Right Pinky	0.0

Left Hand

Left Thumb	0.0
Left Index	0.0
Left Middle	0.0
Left Ring	0.0
Left Pinky	0.0



1301186 **(2015-12-16 13:23)**

AG0932A – Technical Art Applications

gads

BlogBook v0.8,
LaTeX 2_ε & GNU/Linux.
`http://www.blogbooker.com`

Edited: December 16, 2015