Building a 3-D Game in Multiple Environments

Sam Snodgrass

Department of Mathematics and Computer Science

Mentor: Dr. April Kontostathis

Spring 2012

Submitted to the faculty of Ursinus College in fulfillment of the requirements for

Distinguished Honors in Computer Science

Distinguished Honors Signature Page

Sam Snodgrass

Building a 3-D Game in Multiple Environments

Advisor: _____

April Kontostathis

Committee Members:

_____

April Kontostathis

_____

Akshaye Dhawan

_____

Matthew Kozusko

Outside Evaluator:

_____

John P. Dougherty

Approved:

_____

Mohammed Yahdi

Abstract

Choosing a tool for the graphical modeling that is required for game development is a complex task. There are many tools available, but we were primarily interested in Blender and 3D Studio Max. Both are widely used by digital animation companies as well as by game developers. This thesis describes the process by which we compared these tools, and the pros and cons of each tool, before coming to a conclusion about the better tool for our project.

Our goal is to construct a Chess-based game, and enhance this game with an artificial intelligence engine, as well as RPG (role-playing game) elements. After comparing Blender and 3D Studio Max, we decided to use Blender to build our game. We made use of the objects we built in Blender, as well as the Python programming language and Blender's unique logic bricks to implement game rules and movements. In this thesis we describe our progress to date. We currently have implemented a graphical version of chess, including designing and building the pieces, and scripting the rules for movements (according to the rules of chess).

Contents

Chapter 1: Introduction

In this thesis we examine the differences between Blender and 3D Studio Max, two 3-dimensional (3-D), graphical modeling tools often used for modeling 3-D objects for use in video games. We also discuss how we use Python along with Blender to add functionality to our game. Most game developers prefer a visual approach to modeling, because building objects and environments using a programming language and a graphics library, such as OpenGL, is more time consuming, and requires a great deal of technical expertise. We began building our chess-based role-playing game (RPG) as a means to compare the usability, texture application, model aesthetics, and usefulness as a game development tool of these two products. We accomplish this by building many game objects in both tools and comparing the user experience as well as the finished products.

Chapter 2: Background

The process of designing a 3-D game is more rigorous than for a 2-D game, because 3-D games can be viewed from all sides. This forces the designer to make sure that every object and environment looks appealing from every angle. Alternatively, in a 2-D game, the designer only needs to consider the view that is presented to the player.

2.1 Modeling Tools

A modeling program is a valuable tool for a designer who is building objects for use in a 3-D game. A modeling tool with a graphical interface, like Blender or 3D Studio Max, allows the user to create and edit objects and see the results in real time. This is accomplished through a viewport that displays the object as the user edits it. Designers use this viewport to see exactly how the changes they make are affecting the object. The viewport is particularly helpful for objects being built for 3-D games, because the viewport allows the designer to see the objects from many different angles, ensuring the finished product is designed precisely as intended.

2.1.1 Blender

Blender is an open source 3-D modeling software package. It was initially released in 2002 and large community of designers and programmers participate on

the forums as well as in the development of the software itself. New versions are released every few years, and minor updates and bug-fixes are released more frequently. The version we use is 2.59. This version was released in 2011 [3]. Blender is a popular tool, used mostly by small teams and individuals, because it is remarkably powerful despite being open-source.

2.1.2 3D Studio Max

3D Studio Max is a commercial 3-D modeling tool. 3D Studio has an established place in the market. It has been in existence for over 20 years, and the most recent version was released in 2011 by Autodesk [1]. 3D Studio Max is used by many game and entertainment companies, because it is a very powerful tool that is maintained by professionals. However, it is expensive and the cost may be prohibitive for small companies or individual developers.

Chapter 3: Related Works

We found three articles that are related to our thesis. Two of the articles put forth methods for comparing modeling and simulation programs. The third article recounts a user evaluation study of a graphical modeling program. In this chapter we discuss these three articles and explain how they influence our thesis. We also examine how our thesis is different from the three articles.

3.1 Selecting Simulation Software

In [2] the authors assert certain criteria that should be considered when choosing a simulation modeling tool. Before they discuss the points of comparison, they give a few warnings. The authors warn people who are looking for simulation software to know what is needed for their project or company. They also warn not to judge a product's features based on "yes" or "no," but to use a scaling metric. After giving these warnings, the authors begin discussing the criteria with which to judge potential simulation modeling programs.

The first category is "Input Considerations." Within this category, only a few points apply to our comparison. One criterion is point and click capability. Though this feature could be applied to our comparison, it would be unhelpful in distinguishing Blender and 3D Studio Max because both programs have point and click capability. The next concept that could be used in our comparison is the ability

to import and export files. This also would do little to differentiate Blender and 3D Studio Max, however, because both have this capability.

The next category is "Processing Considerations." The criteria within this category that are useful in our comparison are programming and portability. Being able to program in a modeling tool allows for more exact and extensive customization. Portability allows the user to run the created model on any computer without changing the program.

The third category is "Output Considerations." Mentioned in this category are different simulation reports and the ability of the program to write these reports to an output file. None of the criteria in this category are relevant to our comparison of Blender and 3D Studio Max.

The next category is "Environment Considerations." This category has several concepts that are applicable to our comparison. We consider ease of use and ease of learning as a major point of comparison for Blender and 3D Studio Max. Animation capability, which, in our case, includes the game animations, is also a criterion that we consider in our comparison.

"Vendor Considerations" is the fifth category. It includes measures such as support, stability, and history. Of these criteria we compare the support of both Blender and 3D Studio Max. This includes update releases, new versions, and community support with forums and tutorials. We combine support and ease of use into one point in our comparison.

The last category is “Cost Considerations.” We mention this in our comparison but it is not a distinct criterion. Blender is open-source and we have a free “student trial” of 3D Studio Max.

The authors of this article put forth a set of points of comparison to consider when choosing a simulation modeling tool. The authors do not, however, put these methods into practice. They merely describe the criteria for use by others. We use several of the concepts mentioned by the author for comparing Blender and 3D Studio Max, including the ability to program within the tool, portability, support, and animation capability.

3.2 User Evaluation of a Graphical Modeling Tool

The authors of [5] conducted a user evaluation of the Graphical Learning Modeler (GLM), which is a graphical implementation of IMS Learning Design (IMS LD). IMS LD is used to model learning processes. Other implementations of IMS LD used XML as their means of modeling. This made it so that anyone who wanted to use IMS LD needed to become proficient with XML. The GLM allows the user to bypass learning XML by including a drag and drop interface. The GLM simplifies the number of options available to the user. It also leaves out some more complex actions, which are available in previous implementations. These qualities make the GLM more easily accessible to new users and those without programming experience.

The authors conduct two types of user evaluation. The first allows people who are familiar with IMS LD evaluate it. This group includes three sections: pedagogical experts, PROLIX test bed partners, and IMS LD tool developers. All within this group give positive feedback on the GLM, with a few ideas for improvement. Some ideas include templates that cover a longer time frame, allowing for better recovery from errors, and including more of the IMS LD features that are not included in this version.

The next user evaluation comes from professors at the University of Vienna. The authors allow professors of any age and teaching experience, provided they had some experience with using technology for teaching, to participate in the evaluation. The evaluation includes twenty-one professors from varying fields. They are broken into two groups, one group of fourteen and one group of seven. The group of fourteen is given an in-person introduction and presentation on the program. The group of seven is shown a short flash animation, and then given time for Q&A. All of the participants are given a project to create, and instructed to create a project of their own as well. Half of the participants are told to work on the given project first, while the other half are told to work on their own projects first. After the allotted time for the projects, the participants are given a short questionnaire to complete.

The results of the evaluation are promising. Thirty-seven of the projects are deemed "complete" or "nearly complete." "Nearly complete" means a small part of the project is missing, such as a connection between activities. The questionnaires

provided positive feedback for the tool, but the participants also mention areas in need of improvement. The professors express dissatisfaction with some of the naming conventions in the program, such as labeling some tools as an "environment." These "environments" are easily confused with the modeling environment.

The authors consider the input from the professors and the input from the people familiar with IMS LD before making updates to the GLM. Some of which include changing the names of the objects and options. They change "interactions" to "add-ons," and "environment" to "tools & materials." The authors also include a wizard to guide new users through using more complex elements.

The authors acquire a significant amount of input from both professionals and first time users before making changes to the GLM. We have no intention of making changes to either Blender or 3D Studio Max. We are interested in which program is more suited to our needs. It would not be beneficial for us to conduct an evaluation as comprehensive as the authors'.

3.3 Simulation Software Evaluation and Selection

The authors of [7] describe many measures to consider when choosing simulation modeling software, as well as explain the steps that should be taken when choosing a program. The authors break the criteria into four major groups: "Hardware and Software Considerations," "Modeling Capabilities," "Simulation Capabilities," and "Input/Output Issues." Each of these categories is further broken

into subcategories which contain the criteria. We gloss over the subcategories and summarize the points in each category.

The "Hardware and Software Considerations" category has several concepts that are applicable to our comparison. *Sources of information about the package* is a criterion in this section. We associate this with general support and the community of the programs. Having a *built-in logic builder* and *global variables* is a point of comparison under software considerations. This is also useful in our comparison of Blender and 3D Studio Max. The quality and amount of *discussion groups* is the other point in this section that we use in our evaluation.

The next category, "Modeling Capabilities," has criteria that coincide with our own. This section includes concepts such as; *ease of use*, *ease of learning*, and *user friendliness*. Another feature that interests us is the inclusion of *model libraries*. Other measures in this section relate to debugging, which is not a pressing concern of ours when modeling.

The third section is "Simulation Capabilities." This category is heavily involved with the actual practice of modeling. The applicable criteria from this section are the quality of *shape libraries, level of detail*, and *editing partially developed models*. The other points in this section consider testing the validity of a simulation as well as error checking. These have little effect on our choice of modeling tools, because we are not modeling simulations.

"Input/Output issues" is the last category of criteria. This section's conditions relate to outputting information from simulation, analyzing information, and creating graphs with the information produced. However, we are not using these, because we are not modeling simulations.

After describing the criteria, the authors explain the stages of selecting a simulation modeling program. The first step is establishing a need for the software. The second step is an initial software survey. This entails compiling a list of qualities that are needed in a program, and finding a few programs to explore more fully. The next step is evaluation; this step involves judging each program on the features the authors suggest. The criteria will be weighted according to the needs listed in the previous step. The fourth step is to select a program based on the evaluation. The final two steps are negotiating a contract and purchasing the software.

Much like [2], the authors of [7] put forth criteria and a process for selecting a simulation modeling software. Several of the concepts in this article align with those we use in our comparison. Some examples are: *ease of use and learning*, *quality of model libraries*, and *general support*. The authors in this paper do not test their methods. The authors suggest the criteria and the method of implementing them, but they do not make use of the ideas or process they outline.

Chapter 4: Methodology

In this chapter we describe the game we set out to create and the criteria we use in our comparison of Blender and 3D Studio Max. We also explain why we decided to use each criterion.

4.1 The Game

It is important to understand the game we are building in order to understand how we are comparing Blender and 3D Studio Max. We are building a two player chess game. The rules, movements, and pieces are modeled after real world chess. A rook in our game has the same movement capabilities as a rook in standard chess. The win condition remains unchanged as well: capture the opponent's king. In our game the white player always makes the first move.

In the future, we plan to add RPG elements, such as a turn-based battle system, that activates when a piece is captured, and piece exclusive skills that can help with a piece's movement or provide an advantage in the turn-based battles. Because there are many of basic chess games on the market, it is important to set our game apart. The RPG elements mentioned above will provide a niche for our game within the market flooded with chess games.

4.2 Criteria for Comparison

We develop several objects for our chess game in both 3D Studio Max and Blender. We start by building the pawn, the simplest piece, in order to test the basic features of each program. Next we apply a texture to the pawn in both programs in order to test how easily and quickly the user can apply textures. Afterwards, we build more complex pieces, including the knight and rook, in both programs. This allows us to test the programs' more advanced features, and compare the finished product for complicated objects.

4.2.1 Ease of Use

Ease of use is an important aspect when choosing a modeling tool [7]. Ease of use includes how intuitive the controls are, how easily a user can learn the controls including hot keys and menu navigation, and how quickly and accurately the user can manipulate an object. A high ease of use shortens the learning curve of the tool. Using a modeling tool with a short learning curve will expedite the modeling process, allowing the user to be more efficient.

4.2.2 Textures

Textures are an important part of most games [4], although our game does not make much use of them. They add depth to otherwise plain objects, and easily make a simple object more interesting. A modeling tool that allows for easy application of

textures can save the user time. This is especially true for designers who require a large number of textures for their models and environments.

4.2.3 Finished Product

The finished product is the culmination of the modeling process. It represents what is ultimately achievable with the modeling tool. A high quality finished product suggests a high quality program. Ease of use and texture application both affect the finished product. A higher ease of use can lead to a better finished product in less time. This is an important concept when choosing an object modeling program because designers should be able to create models that fit their design in a reasonable amount of time.

4.2.4 Use as a Game Development Tool

The use of Blender and 3D Studio Max as game development environments can play heavily into a designer's choice of tool. Some designers need a strong modeling tool that is able to transition seamlessly into a game development tool. An ideal situation would be one where the program can function as both a modeling tool and as a game development tool, to avoid the need for a supplemental program. This removes the overhead associated with importing and exporting objects between a modeling tool and game engine.

Chapter 5: Comparison

In this chapter we explore how Blender and 3D Studio Max compare when judged using our criteria.

5.1 Ease of Use

Any tool will have a learning curve. A short learning curve and easy usability afterwards are preferable in a modeling tool.

5.1.1 Blender

Blender offers an intuitive environment for modeling objects and building games. The modeling process is efficient because of the inclusion of hotkeys and shortcuts. Making changes to an object is easy. The interface allows the user to select faces, edges, or vertices and then can scale, rotate, or shift them in any direction. Using the hotkeys, the user can lock the direction in which to manipulate an object. This feature lets the user edit models quickly and easily.

The viewport in the Blender interface is easily controlled with hotkeys or menus, allowing the user to view the object from many angles and distances. Simple and comprehensive camera control ensures that mistakes can be found more easily.

Blender's installed base adds to its ease of use. There are many online sources for help, and both professionals and hobbyists contribute to these sites. There

are forums, threads, videos, and blogs all dedicated to helping people become more proficient with Blender. If there is something a user does not know how to do, one of these sources often provides a solution.

5.1.2 3D Studio Max

3D Studio Max provides a less intuitive environment than Blender. It relies heavily on menus which makes it easier to navigate initially, but leads to slower progress further into the project. Editing an object in 3D Studio is much like editing an object in Blender; however the extrude function is unintuitive. With an extrude function, the user selects a face on an object. The function copies the face and connects it to the original face with edges from matching vertices. The user is then able to move, scale, and rotate the new face. The problem we had with the extrude function in 3D Studio Max occurred when we were working on the horns for the knight piece. Our horns start with a cylinder as a base and then, through several extrusions, we build it taller and thinner until it comes to a point. The extrude function tends to polygonize the faces of the starting cylinder, changing it from a cylinder to a blocky-looking object (see Figure 1, inset). The visual quality of the horn suffers as a result. This effect can be seen in the upper chest of the knight in 3D Studio Max as well.

3D Studio has four viewports initially, and more can be added. Viewports can also be removed. Having multiple viewports allows the user to see objects from several angles as the user edits them. This method can be quicker than Blender's

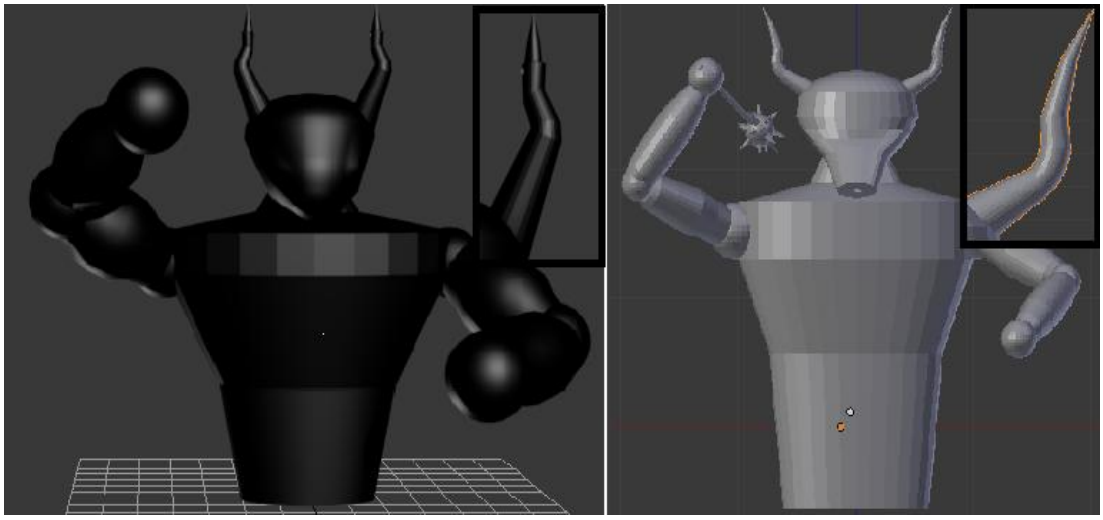
Figure 1. Comparison of a knight piece built in 3D Studio Max (left) and Blender (right).

method of editing an object and then looking from different angles. The drawback of the multiple viewports in 3D Studio Max is controlling them. There is a navigation cube in each viewport (see Figure 2). The feature is designed for the user to click somewhere on the cube and for the viewport to respond by rotating to focus on the point in the field corresponding to the point clicked. This is an interesting idea in theory, but in practice it is unwieldy and difficult to use. The user is able to shift the view in other ways, but if the user shifts the view then rotates using the cube, it often removes the object from view.

5.2 Textures

Textures give depth and substance to objects that may otherwise look bland. To test the use of textures in Blender and 3D Studio Max we apply a simple scale armor texture to a pawn in both programs.

Figure 2. The navigation cube from 3D Studio Max.

5.2.1 Blender

Applying textures in Blender is difficult as a first time user; the process is unintuitive. Selecting an area to texture, and selecting the texture to add, results in a warped texture pattern in some places and an unaligned tiling in others. We reapply the texture using a technique that involves "unwrapping" the faces of the object. Unwrapping means Blender lays the faces out side-by-side instead of in the usual 3-dimensional way. This technique lets us position the texture in a certain way over the faces of the object. This does not solve our problem, however, as the layout of the faces is unpredictable. Eventually, we align the faces in a way that allows us to place the texture across the "unwrapped" faces. Unfortunately, we still notice a tiling effect on the faces. We modify each face individually to make the tiling look seamless, but the result is not perfect. The results of this effort are unsatisfactory (see Figure 3, right). In the end, applying a texture to a simple object took much longer than we

anticipated. Applying multiple textures to many different pieces would be very time consuming.

5.2.2 3D Studio Max

3D Studio Max provides an environment for textures that is simple and efficient. Selecting the appropriate faces on an object, and then selecting a texture gives an excellent result (see Figure 3, left). The texture is not noticeably tiled. Instead, it is pieced together in an elegant way and then stretched across the selected faces. 3D Studio Max provides options for increasing and decreasing the size of the texture tile. The color of the texture can also easily be changed. In summary, applying a texture in 3D Studio Max is easier and produces a better result when compared to applying textures in Blender.

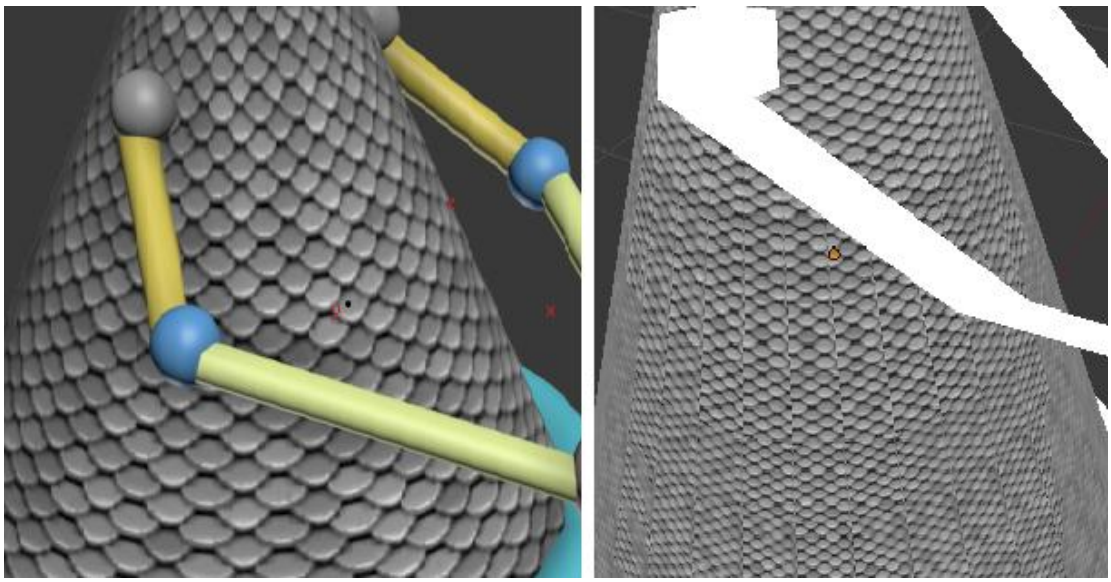

Figure 3. A scale texture applied a pawn in 3D Studio Max (left) and Blender (right).

5.3 Finished Product

Initially, the pieces built using 3D Studio appear to be of a higher quality than those built in Blender (see Figure 4). This continues as a trend through several pieces, such as the knight (see Figure 1). However, as development continues, the difference in the finished objects becomes less noticeable. This is exemplified in the rook piece which looks nearly identical in both programs (see Figure 5). This may have to do with the ease of use of the two tools. With Blender, we are able to learn the controls more quickly and become proficient sooner. With 3D Studio Max, we are able to produce high quality finished products initially, but our skills quickly plateaued because of the unintuitive controls. As we became more proficient in Blender the difference in quality between the objects built in Blender and 3D Studio Max lessened.

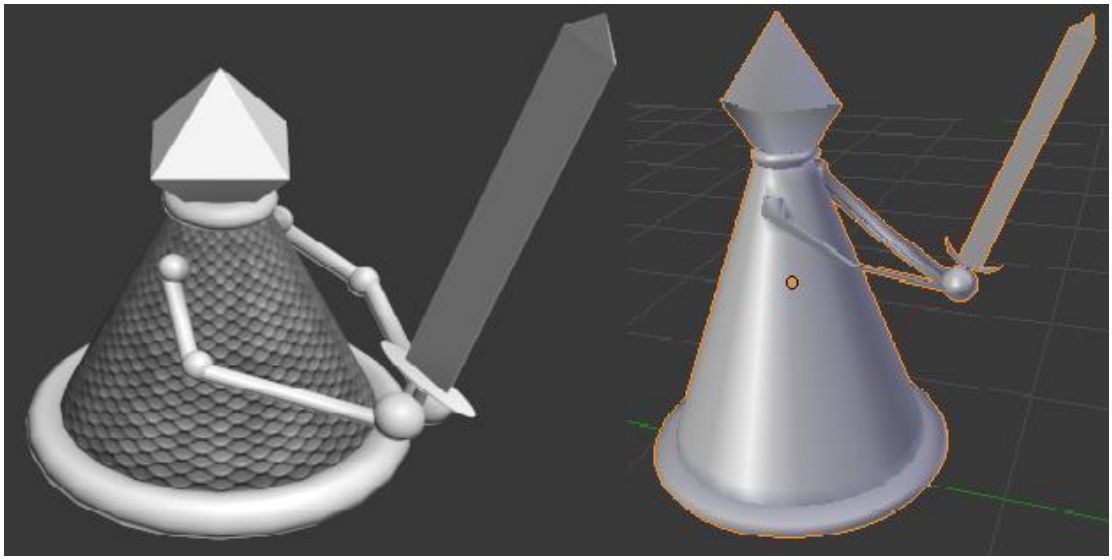

Figure 4. Pawn piece modeled in 3D Studio Max (left) and Blender (right).

5.4 Use as a Game Development Tool

Blender has a significant advantage over 3D Studio Max as a game development tool. Blender has a game engine built into it. This lets the user model all of the game pieces and build the game using the same tool. Doing so allows the developer to avoid the overhead of exporting objects and environments from Blender and importing them into a different program. Furthermore, Blender uses Python as its game scripting language. Python is a high level programming language. This means that Python is closer to human language than many other programming languages and is, therefore, easier to learn [6]. We discuss Python in more detail in Chapter 6. In addition to these advantages, Blender uses something called "logic bricks (see Figure 6)." These are used for simple movements and controls instead of Python.

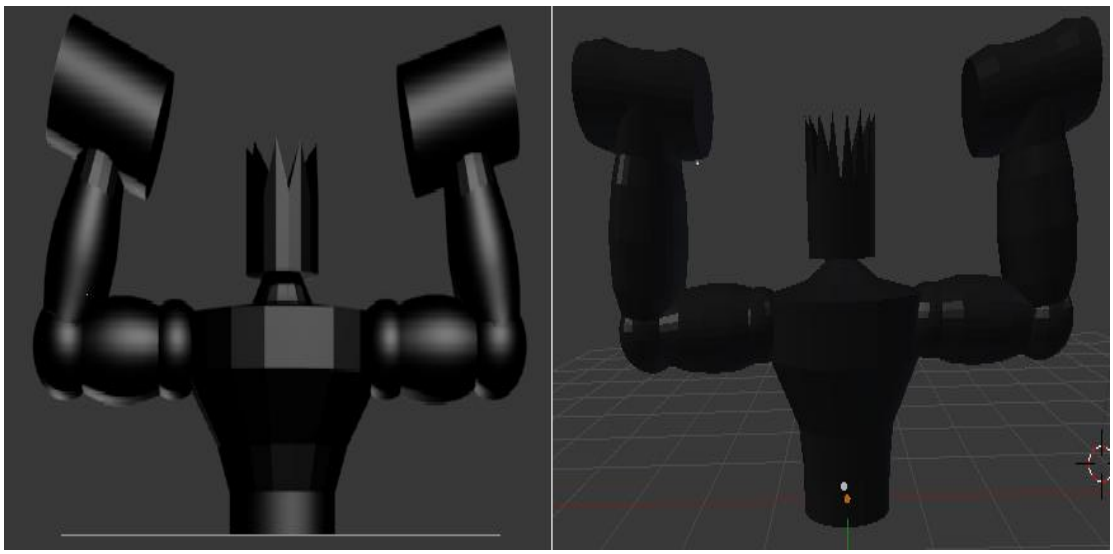

Figure 5. Rook piece built in 3D Studio Max (left) and Blender (right).

3D Studio Max does not have a game engine built into it, so it is necessary to get a game programming tool that will accommodate objects built in 3D Studio Max. We use the Unity game engine with 3D Studio Max. To test this process, we export our models from 3D Studio Max and into Unity. Unity relies entirely on scripting for the implementation of game rules and movements. Only using scripts can make building game rules and logic more tedious, especially when only simple movements or rules are needed. Unity alleviates this disadvantage by allowing for the use of several different programming languages; including JavaScript, C#, and Boo (see Figure 7).

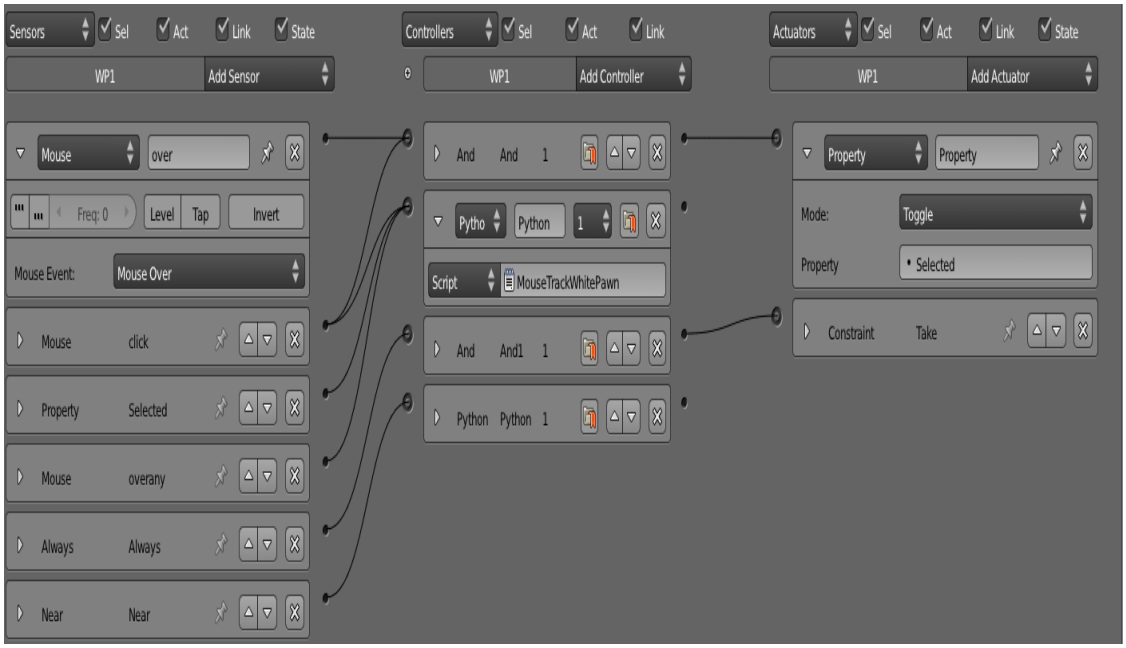

Figure 6. A set of logic bricks applied to a pawn. Some bricks expanded for clarity.

```
var Speed:float = 1.0;
function Update () {

    if(Input.GetButton("Left"))
        transform.position += transform.forward*Speed*Time.deltaTime;

    if(Input.GetButton("Right"))
        transform.position += transform.forward*Speed*Time.deltaTime;

    if(Input.GetButton("Forward"))
        transform.position += transform.forward*Speed*Time.deltaTime;

    if(Input.GetButton("Backward"))
        transform.position += transform.forward*Speed*Time.deltaTime;

}
```

Figure 7. A simple movement script from Unity written in Javascript.

5.5 Discussion

We decided to use Blender as our modeling program and game engine for building the rest of our game. Through building pieces of varying complexity in both Blender and 3D Studio Max, we discovered that Blender is more user-friendly and has a larger support community. The fact that Blender has a built in game engine whereas 3D Studio Max requires a supplemental program also weighed heavily into our decision. Textures are much easier to use in 3D Studio Max, but do not appear in our game frequently enough for texture support to be a deciding factor. Both tools can be used to develop high quality products. For a small team an open-source tool is preferred over an expensive commercial tool.

There are situations in which 3D Studio Max may be preferable. If the team is already familiar with modeling tools, then learning how to use 3D Studio Max effectively would be easier [2]. Such a team would also benefit from the power of 3D

Studio Max. Furthermore, if a project requires the use of many textures, 3D Studio Max would be the more logical choice.

Chapter 6: Logic Bricks & Python Scripts

Logic bricks are a unique property of Blender that allows the user to create simple rules without scripts (see Figure 6). Python is a high-level programming language [6]. Being a high-level language, it is closer to human language than other programming languages, such as C++ and Java. This, along with its fairly simple syntax, makes Python an easy language to learn.

6.1 Logic Bricks

Logic bricks in Blender can be used to control simple movements, put restrictions on objects, and control when a Python script should execute. Each object has its own set of logic bricks associated with it. This allows the user to program different pieces to behave in distinct ways. Each object can also have a set of properties attached to it. A property can be a Boolean, an integer, or a float type. The properties have no real functionality unless they are incorporated into the logic bricks or into Python scripts. There are three types of logic bricks: sensors, controllers, and actuators. Each brick has a different job. Sensors bricks detect certain events, and are linked to controller bricks. Controller bricks act as basic logic gates, and are linked to actuator bricks, which perform specified actions when they receive the right signals from the controller bricks.

6.1.1 Sensors

The sensor brick waits for an event to happen. There are many different kinds of sensor bricks. A sensor can be set to detect user input, such as from the keyboard or from the mouse. The sensor can be set to detect when the object it is associated with is near another object, or near an object with a specified property. There is also the option of setting the sensor to always behave as though it's activated. These are the three types of sensors we made use of the most when creating our game, but there are many other options. When a sensor detects what it is set to detect, it sends a positive pulse to the controller brick it is linked to. Otherwise, it consistently sends out a negative pulse.

6.1.2 Controllers

Controller bricks wait for signals from the sensor bricks that are attached to them. Controller bricks act as logic gates, such as a logical "and." By linking several sensors into a controller, the user can check if one of the sensors is activated, by using the "or" option, or if all of them are activated, by using the "and" option. By choosing the proper controller, the user can ensure that the necessary sensors are activated before the controller sends a positive pulse to the actuator brick.

Controller bricks can also be used to activate a Python script. In this case, the user sets the controller brick to "python," and chooses which script to associate with the brick. A "python" controller brick acts as an "and" brick would. The script is

only run if all of the sensor bricks send a positive pulse to the controller. When a controller is set to "python" it is usually unnecessary to attach the controller to an actuator brick.

6.1.3 Actuators

Actuator bricks wait for a positive pulse from at least one of the controllers linked to it. Like controller bricks, actuators can have any number of bricks linked to them. Unlike controllers, actuators will activate if any one of the controllers attached to it sends a positive pulse. This is beneficial if the developer needs the same action to be activated in multiple cases. Some examples of what an actuator brick can achieve include: setting the value of a property of the object the brick is associated with, moving the object a certain distance or to a certain point, or making the object invisible.

6.2 Initialization

The initialization is the set-up phase of a game. The initialization process is used to makes sure all the game objects are in place and that all the variables and functions that are needed in the rest of program are ready for use. Our initialization code is not complex. We declare three global 2-dimensional arrays with eight rows and eight columns. The first, "Board," we initialize with the starting placements of our chess pieces. Unoccupied spaces on the board are denoted by "FREE." The next two global arrays, "WCheck" and "BCheck," keep track of the spaces on the board

that are unsafe for each player's king, respectively. In addition, we run a script that constantly makes the mouse cursor visible while the game is running. Lastly, we initialize global integer variables "sel" and "turn" which keep track of how many pieces are selected and which player's turn it is, respectively. We use a simple "try" and "except" block in our initialization code to ensure that it only sets the global variables once. We do not need to set the starting positions of the individual game objects in our initialization code, because the game begins running with the pieces starting where they are placed in the viewport (see Figure 8). We have the board and pieces set up properly in the viewport before the game begins.

6.3 Movement Scripts

We achieve movement with a combination of logic bricks and Python scripts. Each piece has logic bricks attached to it that allows it to be "selected" by being left-clicked. This means that a boolean game property called "Selected," associated with each piece, is set to true if that piece is left-clicked. Another left click on the piece will set the property to false. While a piece's "Selected" property is true, if the player clicks anywhere else on the board, a Python script, specific to that type of chess piece, will execute. There is one section of code that is common to all of our movement scripts and runs in the beginning of each script. This section takes the coordinates of where the player clicked to try to move the piece, and converts the coordinates into a space on the board. This allows the rest of the code to reference that space and other

spaces in relation to it. Furthermore, the "Board" array that keeps track of the positions of the pieces is updated after each successful move.

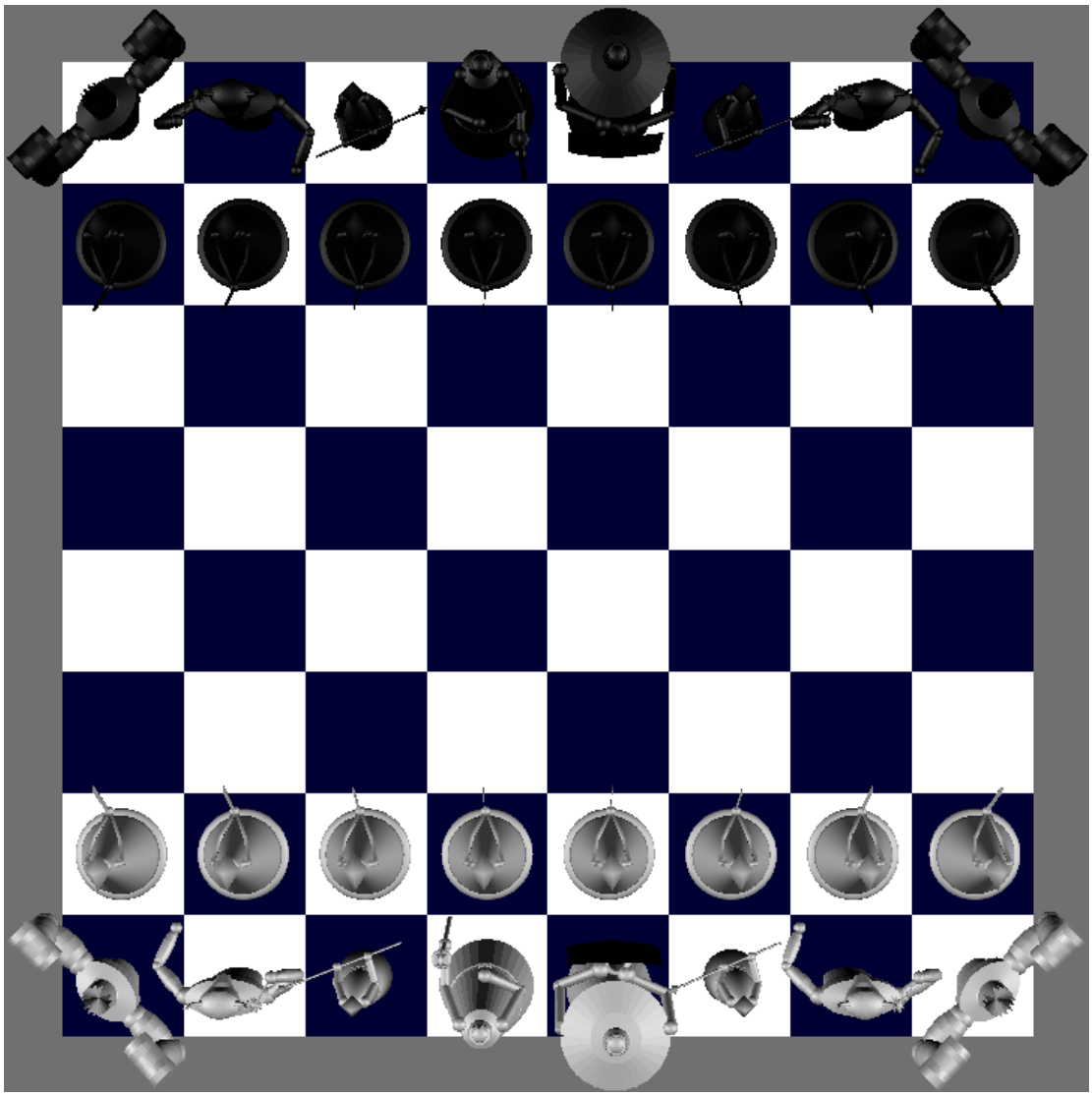

Figure 8. Initial set-up of the board.

6.3.1 Simple Movements

Movement scripts for pieces like the pawn, king, and knight require only basic checks before the movement is approved. In most instances, all that is required for

the pawn is checking if there is a piece in front of it. If so, it is not be able to move to that space. Alternatively, because a pawn can only capture a piece one space diagonally in front of it, if the player clicks diagonally in front of the pawn and there is no opponent piece there, the pawn is not able to move there. One exception in the case of the pawn is its initial move. In chess, the first time a pawn moves, it is able to move two spaces forward instead of one (see Figure 9). In this case, it is necessary to check if there is any piece on either of the two spaces. If so, this move is impossible.

The king can move one space in any direction, provided the move does not put the king into check. This condition must also be checked before any other piece moves. No piece is allowed to move to a space that would place its own king in check. The king can capture an opponent's piece if the piece is on a space that the king can move to. When trying a move a king, we need to check if the space is only one square away, and check for a piece of the same color already on that space. If the space is close enough, and there is no allied piece on it, then the movement is allowed.

The knight's movements, though unique, require the same checks as the pawn and king. The knight moves in an "L" shape. It can move two spaces horizontally or vertically, and then it can move one space perpendicular to the first two spaces. This gives the movement its "L" pattern. The knight's movement code follows many of the same procedures as the king's. First, we check if the space that is clicked is a

valid space for movement. Next, we see if there is an ally piece on the space. If there is not, and the move does place the player's king in check, then it is a legal move.

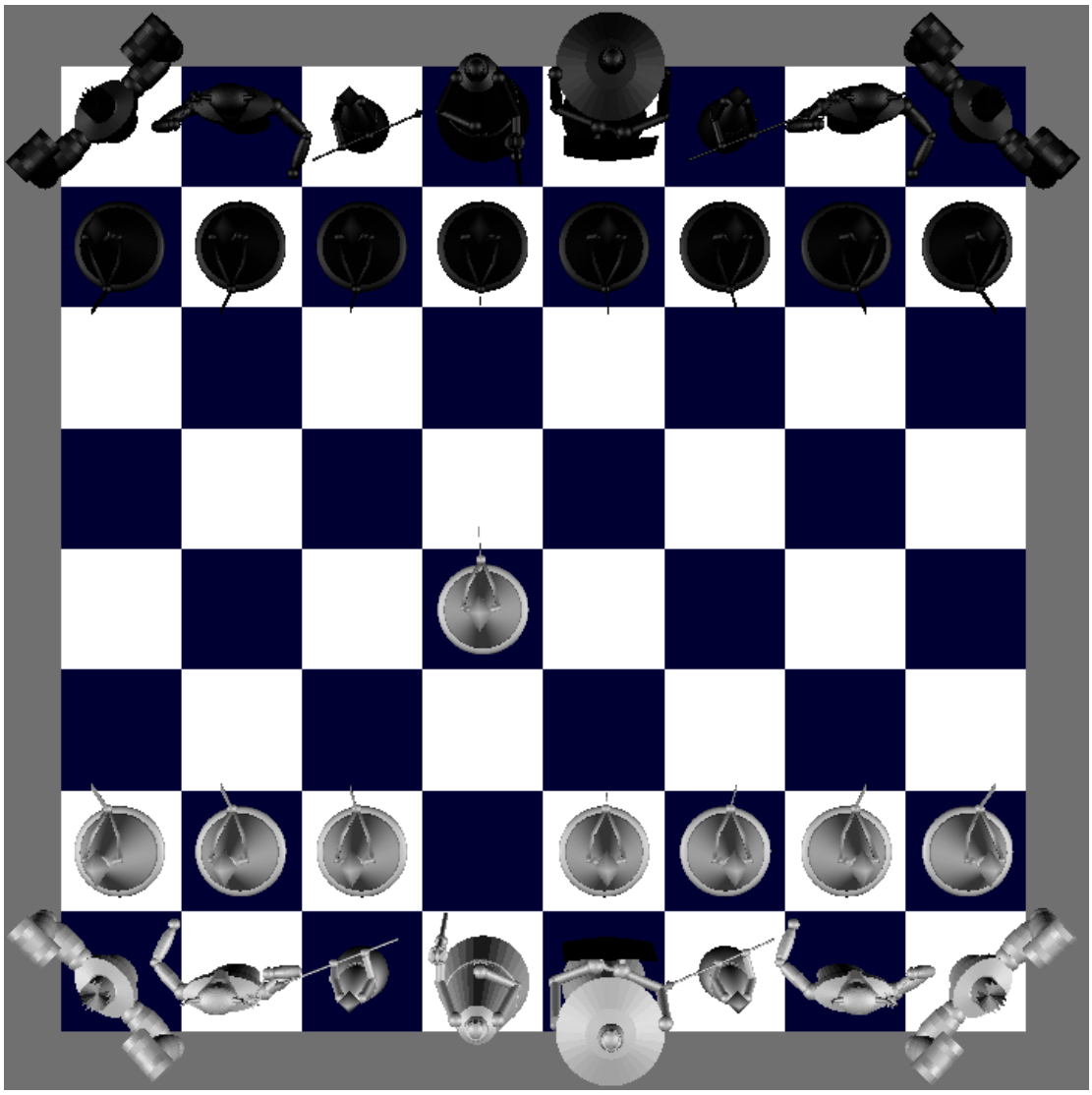

Figure 9. An example of a pawn movement. The pawn was able to move two spaces because it was the pawn's first move.

6.3.2 Complex Movements

The rook, bishop, and queen require more intricate scripts. In chess, a rook can move horizontally or vertically any number of spaces until either the player wants

it to stop, or until it hits another piece. If the rook hits an opponent's piece, the piece is captured and the rook takes the captured piece's position. If the rook collides with a piece of the same color, the rook must stop one space before the other piece. A bishop's movement can be explained much the same way, but instead of horizontal and vertical movements, the bishop moves diagonally. The queen can move diagonally, horizontally, and vertically.

There are several conditions that are checked before any of the pieces with more complex movements are allowed to move. We check if the space clicked aligns with the movements of the piece selected. For example, if a rook is selected, we need to make sure that the space clicked is in the same row or column as the rook (see Figure 10). Afterwards, we check if there is a piece of the same color as the selected piece on the space clicked. If so, the move is denied. Otherwise, we check if the space is occupied by an opponent's piece. If that is the case, or if it is unoccupied, we loop through all the spaces between the rook's current position and the position of the selected space, checking each space for any piece. If a piece is encountered, then the move is aborted. This same process is followed for the bishop and queen pieces.

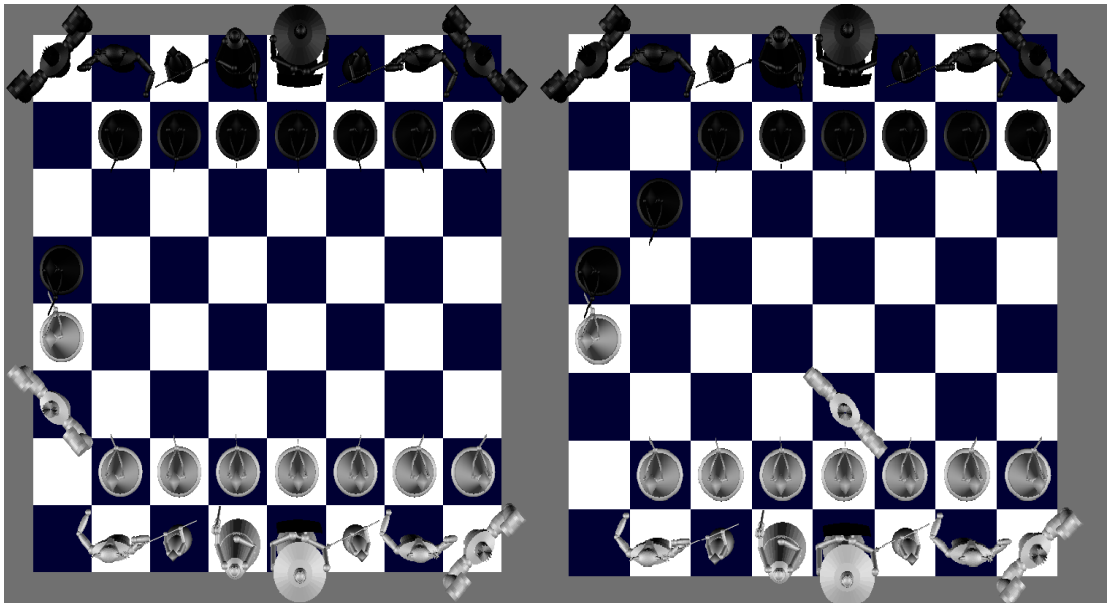

Figure 10. An example of a rook's movements. Forward movement (left) and right movement (right).

6.4 Capture Scripts

When a piece is selected and tries to move to a piece occupied by an opponent's piece, it moves there, given that all the other conditions are clear. Removing captured pieces from the board is done mostly with logic bricks and game properties. Every piece has a property "white" or "black," depending on which player's piece it is. A short Python script is run when a piece moves onto the same space as another. The script checks whose turn it is, then makes the proper piece invisible.

6.5 End Game

No piece is allowed to move to a space that places its own king in check. In order to prevent those types of movements, we add a section of code to each piece's

movement script. Before any movement is allowed, this section of code creates an empty copy of the global board array. It then checks the possible movements of each of the opponent's pieces and marks the possible moves in the copy of the board. If any of the opponent's pieces are able to move to the same space as the original player's king, the move is not allowed. When a legal movement is taken, the proper global array, either "WCheck" or "BCheck," is updated by writing over its contents with the contents of the array that was recently created. Additionally, this script will note if the move puts the other player's king in check (see Figure 11 for an example of check). If so, the other player is not able to make any move that keeps the king in check. If a player cannot get out of check, the game is over. This method of checking the legality of a move is a brute force approach, and we will be attempting to optimize it or implementing a different approach in the future.

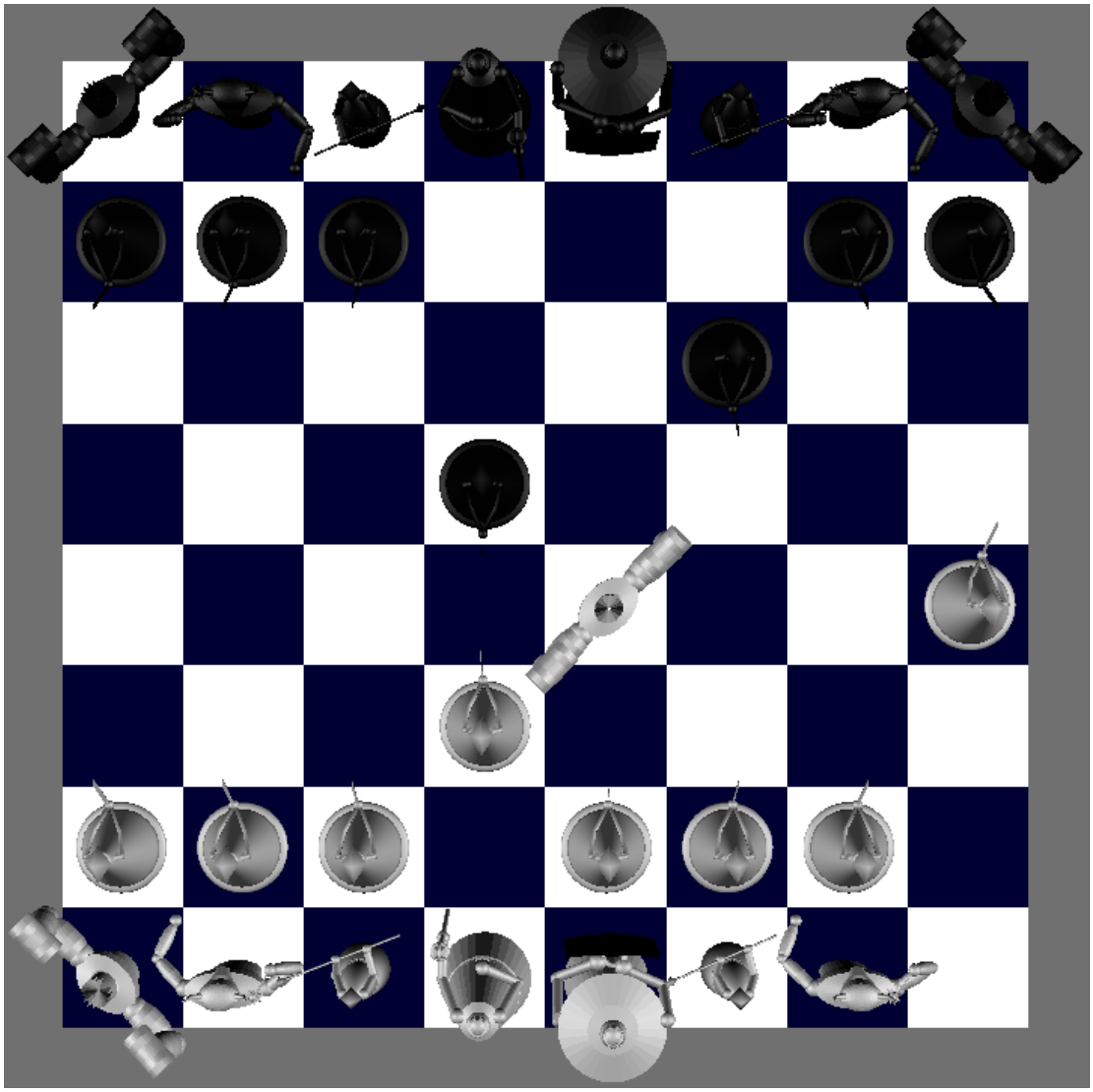

Figure 11. An example of a white rook putting the black king into check.

Chapter 7: Conclusion

Blender and 3D Studio Max are graphical modeling tools. After comparing the tools by building game objects in both programs, Blender is the obvious choice for us. It has an intuitive user interface and a short learning curve. It has a built in game engine which allows us to avoid the overhead of importing and exporting our models. Additionally, Blender is an open source tool with a large community of supporters. Once we decided to use Blender, we began building the rules for our game using Blender's logic bricks and Python scripts. Scripting with Python is easy to learn because of its simple syntax and the fact that it is a high level language. Blender is an ideal modeling tool and game development program for small teams or individual users. Its user friendliness, logic bricks, and choice of programming language make it an easy tool to learn.

Chapter 8: Future Work

We plan to optimize our end game method or change the method from a brute force implementation to a more elegant solution. We also would like to add the ability to "castle." Castling is a move in chess that allows players to move their king and rook in the same turn. The king moves two spaces to the right or left, and the corresponding rook is moved to the opposite side of the king. This special move is only allowed if the rook and king have not moved at all during the game. Furthermore, the king is not allowed to castle out of check, into check, or through any space that would place it in check. We would also like to add the option for a single player to play against an AI opponent. After adding the possibility of a single player game, we plan on adding the RPG elements mentioned in Chapter 4, namely: the turn-based battle system and piece-specific skills.

References

[1] Autodesk Area History Page. http://area.autodesk.com/maxturns20/history

[2] Banks J. (1991) Selecting simulation software. In *Proceedings of the 23rd conference on Winter simulation* (WSC '91). IEEE Computer Society, Washington, DC, USA, pp. 15-20.

[3] Blender Foundation History Page. http://www.blender.org/blenderorg/blender-foundation/history/

[4] Bleszinski, C. (2000) The Art and Science of Level Design. *GDC* pp. 107-118.

[5] Neumann, S. and P. Oberhuemer. (2009) User Evaluation of a Graphical Modeling Tool for IMS Learning Design. In *Proceedings of the 8th International Conference on Advances in Web Based Learning* (ICWL '009), Marc Spaniol, Qing Li, Ralf Klamma, and Rynson W. Lau (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 287-296.

[6] Sanner M.F. (1999) Python: A Programming Language for Software Integration and Development. *J. Mol. Graphics Mod.*, Vol 17, February. pp. 57-61.

[7] Verma, R., A. Gupta, and K. Singh (2008) Simulation Software Evaluation and Selection: A comprehensive Framework**.** *J. Automation& Systems Engineering*. 2(4). pp. 221-234.