Darien Nurse
COMS W4172 HW2

3DTin vs Tinkercad

An analysis of 3D-modling interfaces via Jakob Nielsen's *Ten Usability Heuristics*

Section 1

3DTin and Tinkercad are web applications that allow users to create, manipulate, save, and share 3D models. Tinkercad was founded in by Kai Backman and Mikko Mononen in 2011, thus bringing the first browser-based 3D design platform to the masses. In June 2013 Tinkercad became part of Autodesk. 3DTin was developed in Mumbai, India on March 2010. Over a course of three years it amassed a user base of more than one hundred thousand users. In 2013, 3DTin was acquired by Lagoa, a cloud based 3D render solution. The purpose of this paper is to breakdown, analysis, and critique 3DTin's and Tinkercad's user interface design. This evaluation will be based on how well these applications adhere to Jakob Nielsen's *Ten Usability Heuristics* and characteristics specific to modeling software like object creation, object selection, object manipulation, and camera manipulation.

When a user creates a new project on 3DTin or Tinkercad, he is presented with an empty coordinate plane in 3D space. Naturally, the user would assume this

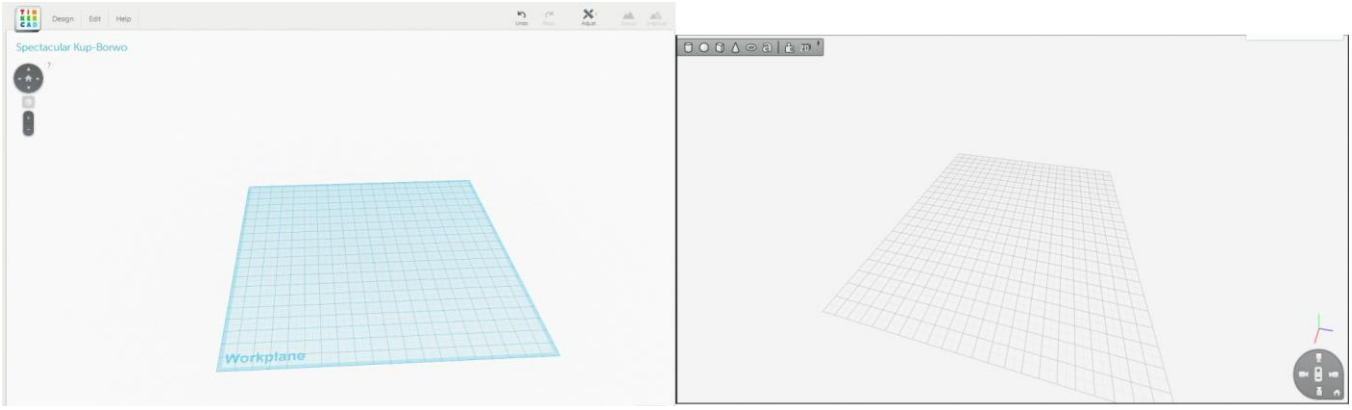


Figure 1: Empty scene in Tinkercad (left) and 3DTin (right)

is the system's default state and new object can be created.

According to Nielsen's first heuristic for interface design, the state of the system should always be visible to the user. Meaning that when a user interacts with the system, he requires some sort of audio or visual feedback indicating that the system is working or that what he has done is either correct or incorrect. For example, when a system is performing multiple calculations or experiencing network lag, it may be placed in a state that cannot handle user inputs effectively. If the system fails to convey that it cannot handle user input at the moment, the user might assume the system has crashed. Users might encounter this issue when working with Tinkercad. In Tinkercad, the undo and save commands depend heavily on the strength of the user's network connection, because these commands require Tinkercad to access its cloud servers to retrieve or store data. Tinkercad uses a rotating arrow symbol followed by a status message to indicate that the system as entered into a save/undo state (figure 2). Because these commands depend on the network connection, the lag on these commands range from a few milliseconds to several

seconds. A progress bar would better illustrate how much time remains for long stalls.

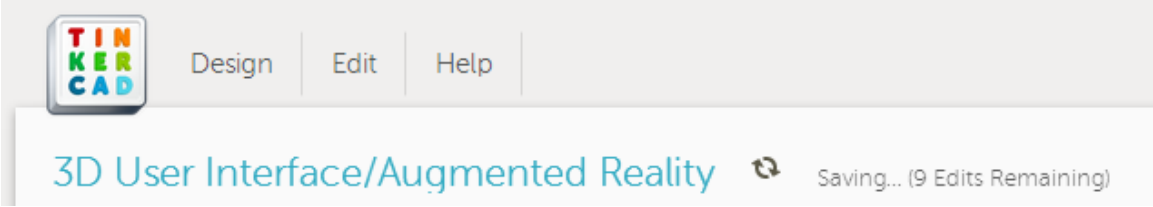


Figure 2: Tinkercad undoing 9 edits

Tinkercad also has a tool bar of icons that become enabled when the user enters the appropriate state. For example, the *group* command, which allows the user to group objects together, becomes enabled when the user selects multiple objects (figure 3).

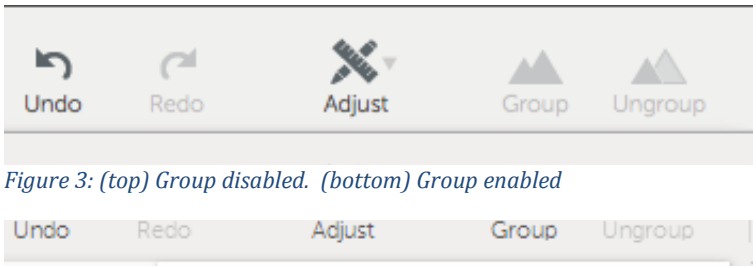


Figure 3: (top) Group disabled. (bottom) Group enabled

Like Tinkercad, 3DTin uses a toolbar with various commands that become enabled when the user enters a particular state (figure 4).

Figure 4: 3DTin tool bar. Looking at the undo and redo commands, one would assume that the redo option is disabled.

3DTin implements a significant state change when the user decides to create a new object. When the user chooses to create an object, 3DTin disables the main editor window and brings up a new window with new editing options. Tinkercad uses a subtler approach to object creation by allowing the user to drag shapes from

the sidebar into the main editor. Afterwards, a small window called the *inspector* allows the user to further edit the shape; the main editor is never disabled during this process (figure 5). Lastly, these interfaces have use separate modes for selecting and editing objects. These modes will discussed in great detail in the second section of this paper.

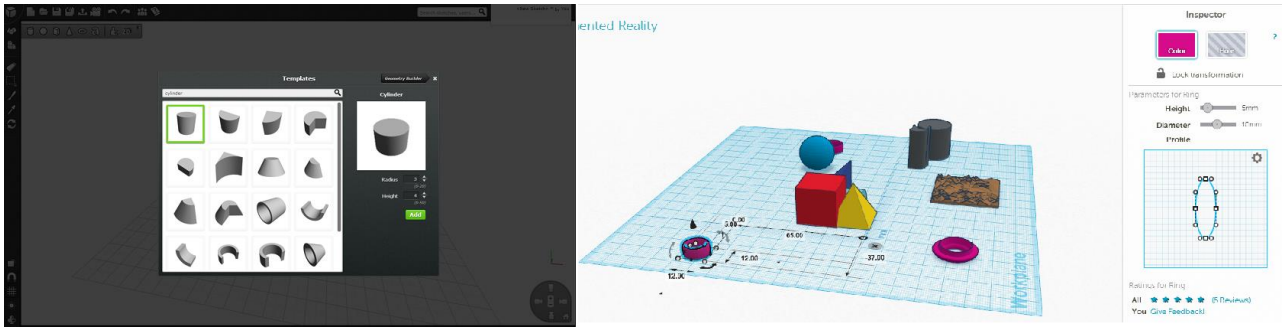


Figure 5: Object creation in 3DTin (left) and Tinkercad (right)

Nielsen's second heuristic states that there should be a match between the system and the real world, meaning that the system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Both editors use terms like, *shape*, *camera*, *save*, *undo*, which are all typically found in commercial modeling software. Overall, both editors avoid using any overtly technical language and sometimes opt for causal language instead (figure 6). That being said, both editors have slight consistencies in their vocabulary which will be discussed in greater detail at the fourth heuristic.

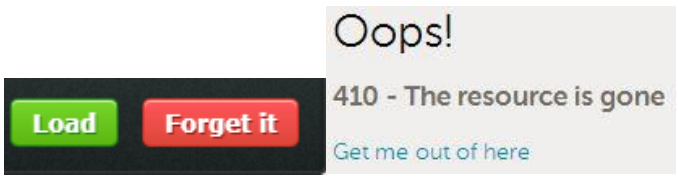


Figure 6: Using phrases like 'Forget it'(3DTin) and 'Get me out of here'(Tinkercad) convey a casual tone to users

Nielsen's third heuristic states that systems must grant user control and freedom, meaning that users should be able to easily undo/redo mistakes. Both editors support *undo* and *redo* and their associated keyboard shortcuts, ctrl+z and ctrl+y. However, the implementation of undo and redo vary slightly between 3DTin and Tinkercad due to their backend configurations. Tinkercad maintains a list of every edit made to the project in chronological order and stores this list in the cloud. The benefit to this cloud based system is that a Tinkercad user can undo any change at any time, even if he close his project and comes back to later. However, the tradeoff for this feature is that efficiency of Tinkercad's *undo*/*redo* commands depends on the strength of the user's internet connection. If the connection is poor or the user issues multiple *undo*/*redo* commands, Tinkercad may experience similar to the lag users might experience when saving in Tinkercad. Without a connection, any changes made to the project will be group together as one single edit. Therefore when the connection is reestablished, the *undo* command will undo every edit made in the connectionless state (figures 7 & 8).

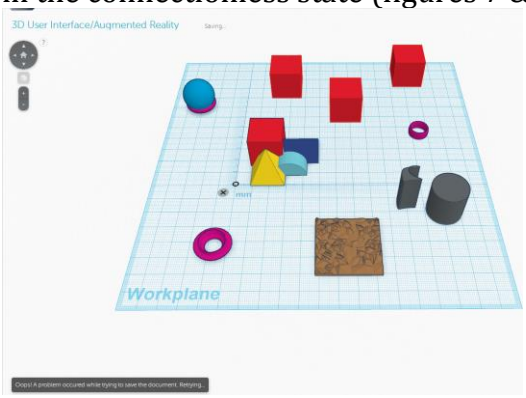


Figure 7: Tinkercad has lost its data connection. This is indicated by the grey error message in the bottom left corner. During this time, the user created the three red cubes in the upper right corner.

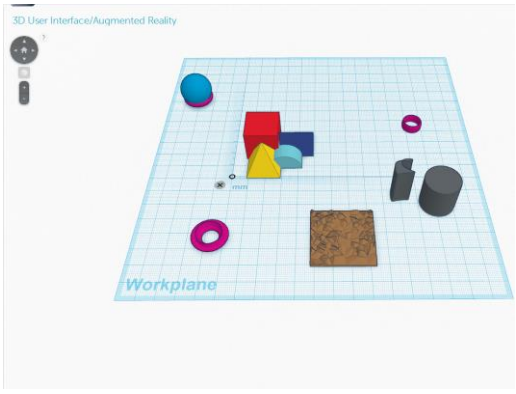
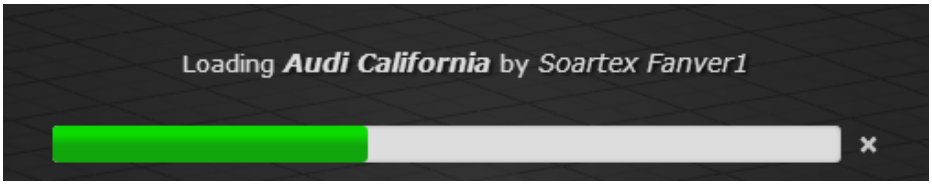


Figure 8: Tinkercad has regained its data connection. When the user clicks undo, Tinkercad undoes every change made during the connectionless state, i.e. the three red cube

Changes made in 3DTin are stored locally, therefore the system will not stall due to a poor network connection or multiple undo commands. However if the user forgets to save his changes prior to closing the browser, then all changes will be lost. To safeguard against this potential loss of data, 3DTin auto saves to the browser periodically. If the browser cookies are not cleared, then the user can recover unsaved changes (figure 9).

Figure 9: When the user reopens 3DTin, it asks him if he would like to load unsaved changes.

Another key component of the third heuristic is that the system should provide a clearly marked *Emergency Exit* out of the current task the user is performing. These exit*s* could be in the form of a cancel button or a link directing them to the beginning of the task flow. The user should not have to wait until the current time consuming task has completed to be able to exit out of it. For example, when importing a model, 3DTin allows the user to cancel the import but Tinkercad does not (figure 10).



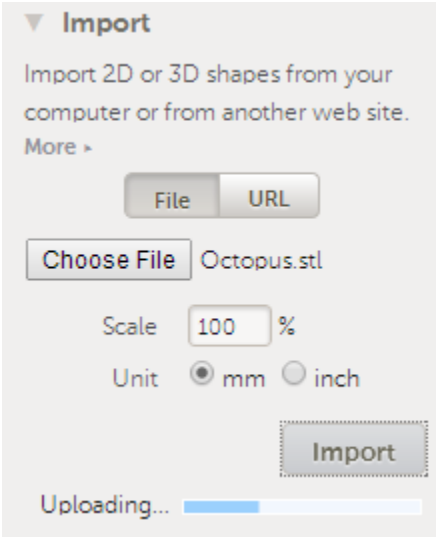


Figure 10: (top) When importing an image, 3DTin allows the user to cancel by pressing the x on the right-hand side of the progress bar. (bottom) Tinkercad does not allow the user to cancel the import

Nielsen's fourth heuristic deals with consistency and standards. Users should not have to wonder whether different words, situations, or actions mean the same thing. The system should follow platform conventions. Modeling software usually has a steep learning curve. Consistency allows users to learn commands and techniques

faster and interact with the system more efficiently.

Both editors are consistent fairly constant in how they name specific commands ad object. The only glaring exception being that 3DTin uses *erase* and *delete* interchangeably to describe the act of deleting an object (figure 11).

Figure 11: 3DTin uses delete and erase to describe deleting an object. There does not seem to be a distinction between the two

There are a surprising number of terms that differ between the editors, yet perform the same action or describe the same concept. For instance, Tinkercad uses *duplicate*, *inspect/inspector*, *mirror*, *workplane*, and *design* to describe object duplication, object editing/editor, object reflecting, the reference grid, and an individual modeling project, respectively, whereas 3DTin uses *copy*, *edit/editor, flip, grid* and *sketch*. Another interesting difference is that the save/cancel buttons are swapped around in the save/save-as window (figure 12). The swapped buttons could be a problem for users who are used to seeing save on the left and cancel on the, or vice-versa.

Figure 12: (top) In 3DTin, the save button comes before cancel, vice-versa for (bottom)Tinkercad

Lastly, Tinkercad violates platform standards by using unconventional terms. For example, Tinkercad uses the term *tinker* to describe opening and editing a project; 3dTin uses the common term *open* (figure 13). Also, when a new project is created, Tinkercad assigns the project a randomly generated name, whereas 3DTin expects the user to properly name the project upon saving for the first time (figure 14)., Tinkercad's unconventional terms indicate poor

interface design, but from a marketing perspective, this practice may cause users associate these certain concepts with Tinkercad's brand.

Figure 13: (left) 3DTin uses 'open' to describe opening and editing a project. (right) Tinkercad uses 'Tinker this'

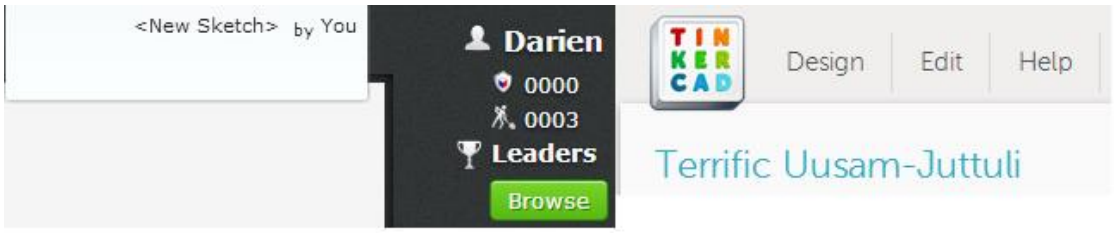


Figure 14: (left) 3DTin uses <New Sketch> to describe new/unnamed projects.(right) Tinkercad generates a random name

Nielsen's fifth heuristic deals with error prevention. In addition to providing the user with helpful error messages, the designer should try to avoid these errors in the first place. It is the designer's job to either eliminate error-prone conditions or check for them and present the user with a confirmation option before he commits to the action.

Tinkercad eliminated most of its potential user errors creating an interface that relies more on mouse interactions than keyboard input. As a result, nearly edits are limited to whatever actions Tinkercad will allow the mouse to implement. Characteristics like object length, width, height, and orientation, all of which could have been changed by a numerical value given by the user, can only be changed by clicking or dragging the appropriate dimensions of the figure. By limiting keyboard input, users cannot assign invalid values to object fields. The only problem with this method of manipulation is that it reduces the user's ability to precisely measure objects. With the aid of Tinkercad's drop down *snap grid* feature (figure 15), object dimensions can be precise up to a tenth of a unit. Tinkercad's object manipulation techniques will be discussed in created detail in second two of this paper.

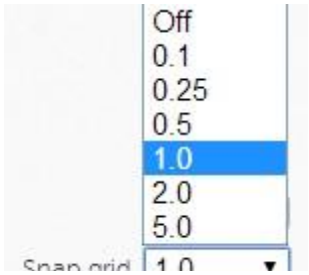


Figure 15: Tinkercad's grid snap feature allows users to scale objects to precise sizes

Unlike Tinkercad, 3DTin allows users to set objects' sizes before the objects are created. In the object creation window, each parameter of the object has a numerical value between the numerical limits in the located below the text field. If the user keyboard tries to set a parameter to a non-numerical symbol or a value that is not in between the limits, the text box will turn red and prevent the user from apply the red value to the object (figure 16). 3DTin consistently uses this method of reddening text fields throughout its interface.

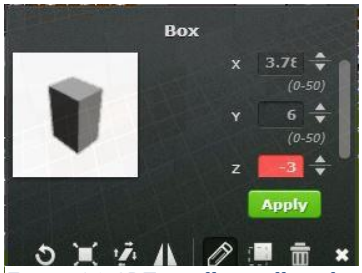


Figure 16: 3DTin will not allow the user to apply the change on the Z parameter since there is currently a negative number in the text field

For setting object color, both editors use color editing window that uses values 0-255 for red, green, and blue hex values, 0-100 for saturation and brightness/luminosity values and 0-360 for the hue (figure 17). These values can be set by the user using keyboard input, however if the user attempts to assign an invalid value to one of the values then the editor will auto-correct the value to a valid value. Both editors perform this auto-correction with notifying the user. User familiar with traditional hex-based color systems would not mind this form of auto-correction, but it could be confusing the new users.

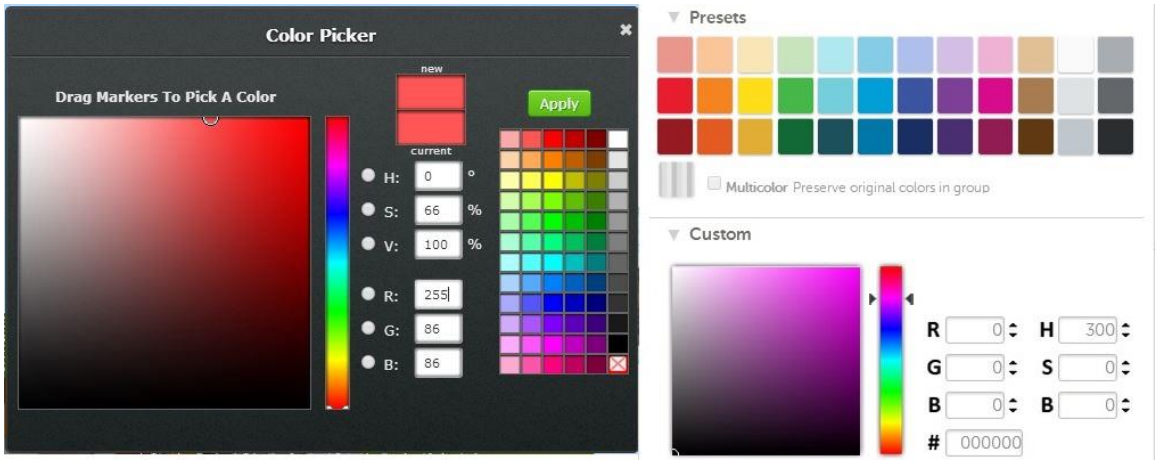


Figure 17: (left) 3dTin and (right) Tinkercad have similar color editing windows. Both uses auto-correction to prevent invalid user entries

Nielsen's sixth heuristic states that the system should emphasize user's recognition over the user's recollection. Designers should minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate. In other words, users should not have to remember sequences of

numbers, words, or long lists of items in a task. Instead, the interface should use easily recognizable pictures and symbols to communicate with the user.

Both editors frequently use symbols to denote specific commands and objects (figures 18 & 19). Tinkercad takes the use of symbols a step further by assigning each type object a color before it appears in the scene (i.e. cubes are red, cylinders are orange, cones are purple, etc.), whereas 3dTin gives all shapes the same grey default color until the user changes the color. Tinkercad's color scheme help the use distinguish shapes from one another. The organization of the interface and its overall aesthetics will be discussed in greater detail at eighth heuristic.

Figure 18: (top) 3DTin commands (bottom) Tinkercad commands

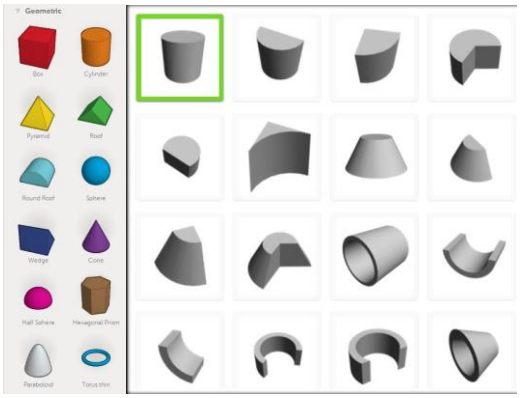


Figure 19:(left) Tinkercad shapes (right) 3DTin shapes

Notice in figure 18 how all of Tinkercad's command symbols have their name beneath whereas in 3DTin only the symbol is given. This is a weakness in 3DTin's interface because it assumes that users will be able to recognize every command by symbol alone. Symbols like the backward and forward arrows representing undo and redo will be familiar to most users, the group and ungroup symbols may not be immediately obvious. If the user does not recognize a symbol, he will have to over his mouse over the symbol until the name shows up (figure 20). This extra step will make it harder for users to adapt to the system.

Figure 20: The name of the shear command only appears when the use hovers his mouse over the symbol

Nielsen's seventh heuristic deals with flexibility and efficient of use. Designers equip their system with accelerator which can often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. In more advanced systems that are capable of learning user patterns, as novice users get acquainted with the system and perform certain tasks frequently, the system will be able to let those frequently performed tasks be accessible more efficiently. Common examples of accelerators are keyboard shortcuts, keyboard navigation, breadcrumbs, and context menus.

Since 3DTin and Tinkercad do not have multiple pages in their interfaces, there is no need for navigation based accelerators like keyboard navigation or

breadcrumbs. However, both editors have an extensive list of keyboard shortcuts to aid with object editing. Tinkercad's shortcuts can be found in Tinkercad's documentation (figure 20). 3dTin's shortcuts appear when the user hover overs an icon. For example in figure 20, the keyboard shortcut for *Shear* is 'h'. Tinkercad's shortcuts also include commands that work with the mouse and keyboard, for example alt + dragging an object can scale the object in two dimensions and shift + dragging an object can scale it in three dimensions.

Figure 21: A small portion of Tinkercad's keyboard shortcuts

Both editors allow users to login before editing (logging is required for Tinkercad). This allows users to name and organize different projects. Giving users the ability to manage their projects will interact more efficiently with the system.

Nielsen's eight heuristic deals with creating aesthetic pleasing systems. The system should not contain information which is irrelevant or rarely needed. Every extra unit of information in the system competes with the relevant units of information and diminishes their relative visibility.

Nielsen's ninth heuristic states that the system should help users recognize, diagnose, and recover from error.

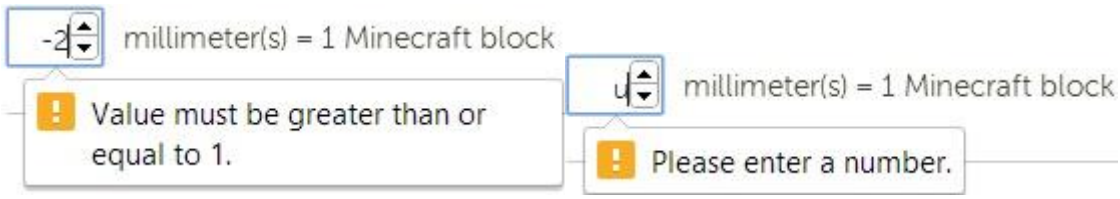


Figure 22: Some simple errors in Tinkercad caused by bad user input

Nielsen's tenth heuristic states that the system should provide the user with a wide array of help and documentation. Tinkercad's website provides several tutorial videos and documentation. 3DTin's documentation is lacking but it does have several tutorial videos that can be found at the top of the interface under a symbol that looks like a book with a question mark on the cover.

Section 2

For modeling applications, the camera position and view are arguably the most important component of editing software. Regardless of the number of features the application supports, if the user cannot navigate around the scene then the application is useless.

Tinkercad's camera seems to have no prominent issues. Tinkercad allows the user to manipulate position with only the mouse: hold the right mouse button to move the camera, hold down the scroll wheel to pan, roll the scroll wheel to zoom, and the left mouse button selects objects. The camera manipulation is often described as moving the camera around the scene to get a better view, but in Tinkercad it seems as though the camera is stationary and the environment moves to around the camera. This distinct might seem inconsequential, but it might be the reason why there is not the same camera distortion a user might find in 3DTin. Also, Tinkercad's camera moves slightly slower than the users mouse after using any form of camera manipulation. This slower movement results in smoother camera movement in comparison to the fast and jerky camera movements in 3DTin.

In 3DTin, the user can only move the camera and zoom using the mouse. To pan, the user has to hold alt then click and drag the mouse in the scene. All the mouse buttons, left, right, and center, perform the same action of selecting and object or moving the camera (or panning the camera when holding alt). This severely limits the interface because panning requires another button, but more importantly the user has to make sure he is not click an object when he tries to move the camera. In other words, click the mouse results in a different behavior depending on whether or not the cursor is over an object. 3DTin's camera's most considerable drawback is its distortion (figure 23).

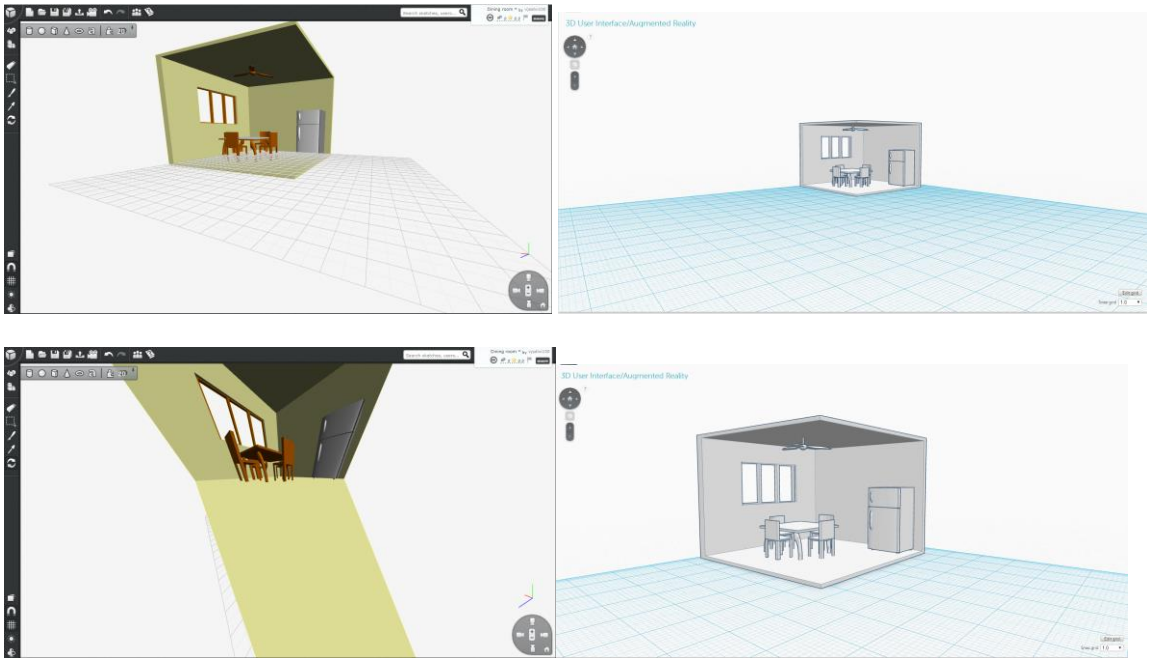


Figure 23: The top row shows the exact same dining room model in (left) 3DTin and (right) Tinkercad. The bottom row shows how 3DTin distorts models upon zooming in. Oddly, this does not happen for all models.

Object selection refers to how the user selects or deselects one or more objects or groups of objects. In Tinkercad, the user simply clicks on an object or left clicks and drags to obtain a group of objects. Once selected, the user can manipulate the object, i.e. change size color, orientation, or delete. However, 3DTin has several modes of selection; each mode changes the state of the interface and changes what happens if the user clicks on an object (figure 24). The four modes of selection (erase, select, change color, pick color, and view rotate) are on the left-hand side of 3DTin's interface. Erase deletes any object the user clicks on. Select allows the user to highlight an object or group of objects and then perform transformations like sheer, rotate, and scale. Change color changes an object's color upon clicking. Pick color takes the color of a click on object and stores it in the color pallet at the bottom

of the screen. View rotate allows the user to move the camera and can object the user clicks on. View rotate is somewhat misleading since is also the only command that can allow the user to move an object once it is placed; move might be a more appropriate command name.

Figure 24: Five modes of selection in 3DTin. Erase, Select, Change color, Pick color, and View Rotate

After camera control, object creation and manipulation are the cornerstones of good modeling software. Tinkercad and 3DTin have different methods for object creation and manipulation, each with its own pros and cons. In Tinkercad, objects are created by clicking and dragging shapes from the tool bar on the right hand side. Objects can also be created by duplicating them from previously existing objects or by using Tinkercad's shape generators. Once in the scene, objects automatically snap somewhere on the workplane. From there, objects can be scale, colored, or reoriented simply by click on them can dragging the correspond feature.

In 3DTin, objects are first selected from the shapes menu in the left hand tool bar. Once opened, the user selects the shape he would like to add to the scene, he can set parameters like width, height, length, or radius before adding it to the scene.

After clicking accept, the user can place the shape anywhere in the scene; thus objects do not snap to the grid like they do in Tinkercad.

Overall, both editors are fantastic applications for editing models. They both have a wide array of tools for creating complex designs. In comparison to Tinkercad, 3DTin offers features like symmetry, changing background color, camera animation that Tinkercad does not have. However, Tinkercad's sleek interface minimizes clutter and uses cell shading on its objects, which cause the objects contract well with the surround environment. Tinkercad also has superior camera control in comparison to 3DTin and lacks any of the distorting feature