MELON'S BUILDER

What follows is a version of the final project documentation I submitted for my undergrad builder project in 2016. Looking back on it, it has some flaws. These days I think the program is better suited for exporting 3D models for digital application. At the time 3D printers were a big buzz topic, so this stressed those heavily.

It was my first ever attempt at a 3D program of any kind and perhaps too much of this document describes me trying to figure out raycasts! However, this is some solid design and research info worth keeping. With that in mind it's presented as written in 2016. Enjoy!

INTRODUCTION

OVERVIEW

The goal of this project was to create an environment for users to build 3d objects in an intuitive way and to have those objects exportable for other uses, primarily 3D printing. The target audience for this program is people who are familiar with first person gaming, particularly first person building games. The intended use of objects created in this program is model making, set and architectural design, prototyping, toy creation and any other areas where models or objects are required.

MOTIVATION

Building, model making and designing things is something that's enjoyed by many people from a very early age. Whether that's using wooden blocks, Lego, Meccano etc. All of these are an intuitive real world experience where components are put together to form a construction. This experience is not carried over to the digital world nearly as successfully. Most programs for creating objects and constructions on computer take their approach and inspiration from the technical and architectural industries, which are ideal for their task, however they create a barrier for people who are not trained in these areas and are looking to mirror their real world creative approach, instead of a technical one. There have certainly been some exceptions, such as Minecraft. The popularity of these games suggests that there is a demand for this kind of intuitive digital creation. This project attempts to realize a step towards bridging this gap, by combining the game style of construction with a program that is intended for producing objects instead of a game.

GOALS

The goal of this project was to have a program which could allow a user to build an object in first person, that object should be as simple or complex as the user demands, the object should use standardized importable and exportable components/blocks, there should be an established way for the user to 3d print that object, use the object in another program, or share the object.

PERSONAL NOTE

Games involving construction and building, as well as real world model making, these have always been of particular interest to me. A motivation in this project is to review the connection between physical and digital objects and how they can reference and inspire each other.

ANALYSIS

CAD PROGRAMS

Here we will review some of the key notable features from existing CAD software and note any observations about it.

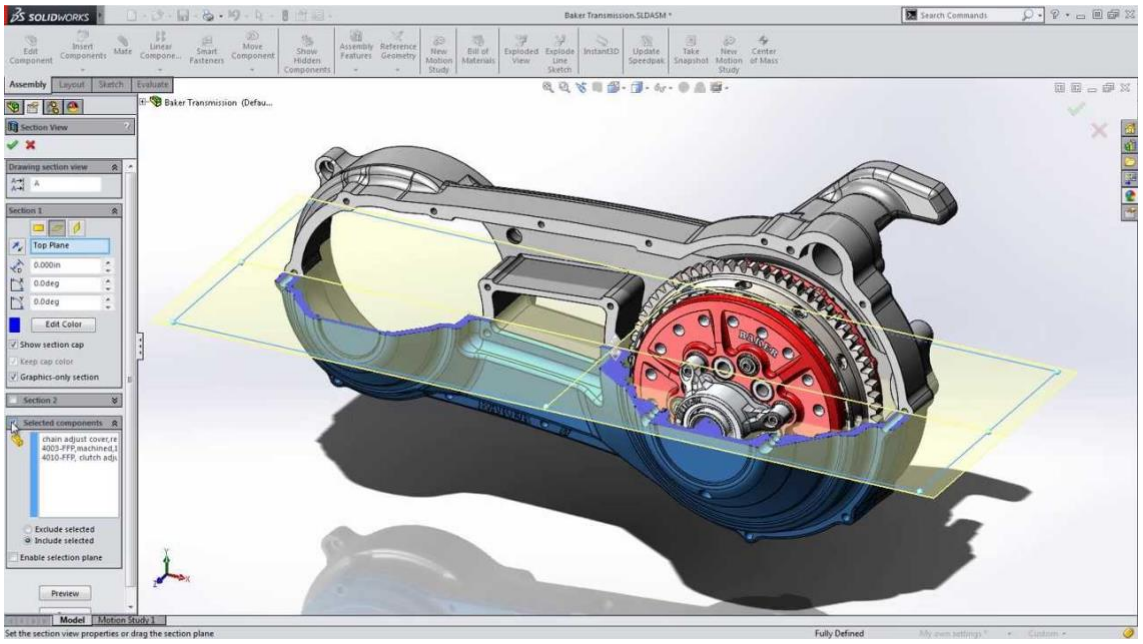


Figure 1 shows SolidWorks by Dassault Systèmes. [1]

SolidWorks is an established CAD. Its target market is the engineering and product design industry, it’s also used widely as an educational tool.

Creation in SolidWorks centers on the idea of planes, directly referencing pen and paper technical graphics. A shape can be drawn on a plane, it can then be extruded along a path, and can then be modified and adapted, such as cutting away sections, rounding edges etc.

Its learning curve is high, and entry is further restricted by the cost of the software. A potential user must invest in software purchase and training time.

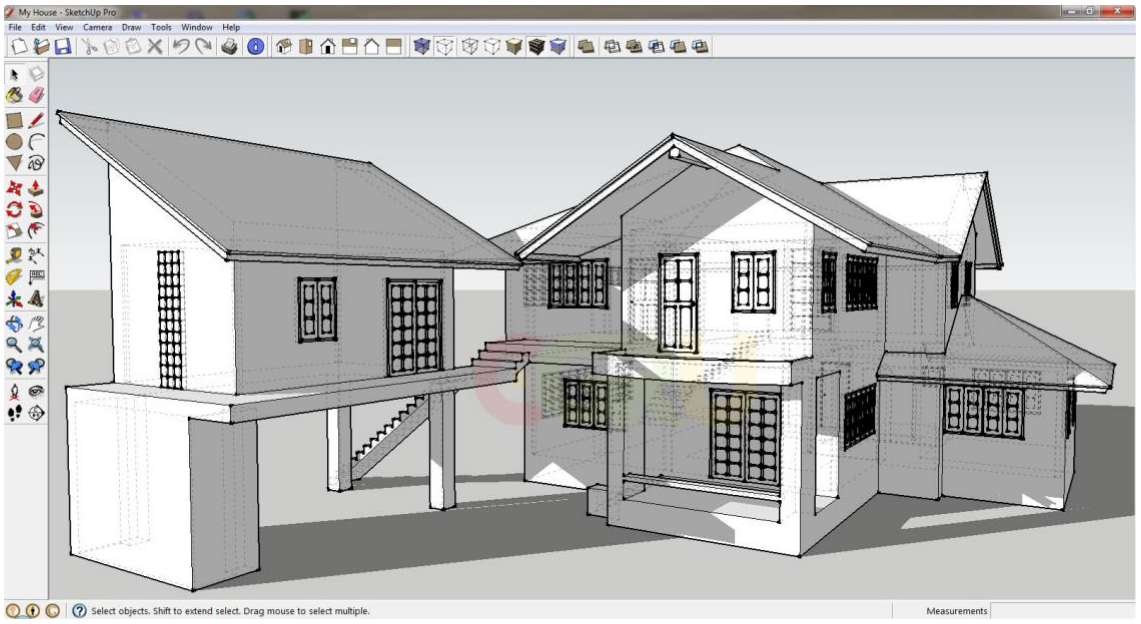

My House - SketchUp Pro
File Edit View Camera Draw Tools Window Help
Select objects. Shift to extend select. Drag mouse to select multiple.
Measurements

Figure 2 above shows SketchUp by Trimble Navigation. [2]

SketchUp is a simplified modeling program. Its target market is architectural, with the ability for some engineering and model making applications.

It also operates on the concept of planes, it's targeted at a less expert user base however. It offers a more intuitive and simplified interface than SolidWorks that references 2D drawing programs such as MS Paint in a 3D context. It does this by providing simple 2D shapes that can be drawn onto a surface that can then be extruded or cut away.

The entry level for SketchUp is much lower, requiring some brief tutorial on usage but no formal training. SketchUp in its basic FORM IS ALSO OFFERED AT NO COST.

POINTS ON CAD

- Both of these programs offer a number of camera angles.
- The camera can be located facing an axis directly as a flat surface, or the camera can freely rotate around the model.
- The user interacts with both of them by using tools to draw and modify the different components of the model.
- In both cases the user is looking at the model, but is unable to interact with it.
- Both programs are focused on geometric accuracy, offering ruled lines and measured spacing.

GAMES FEATURING BUILDING

Here are a number of games that offer the ability to build and construct models, and environments.

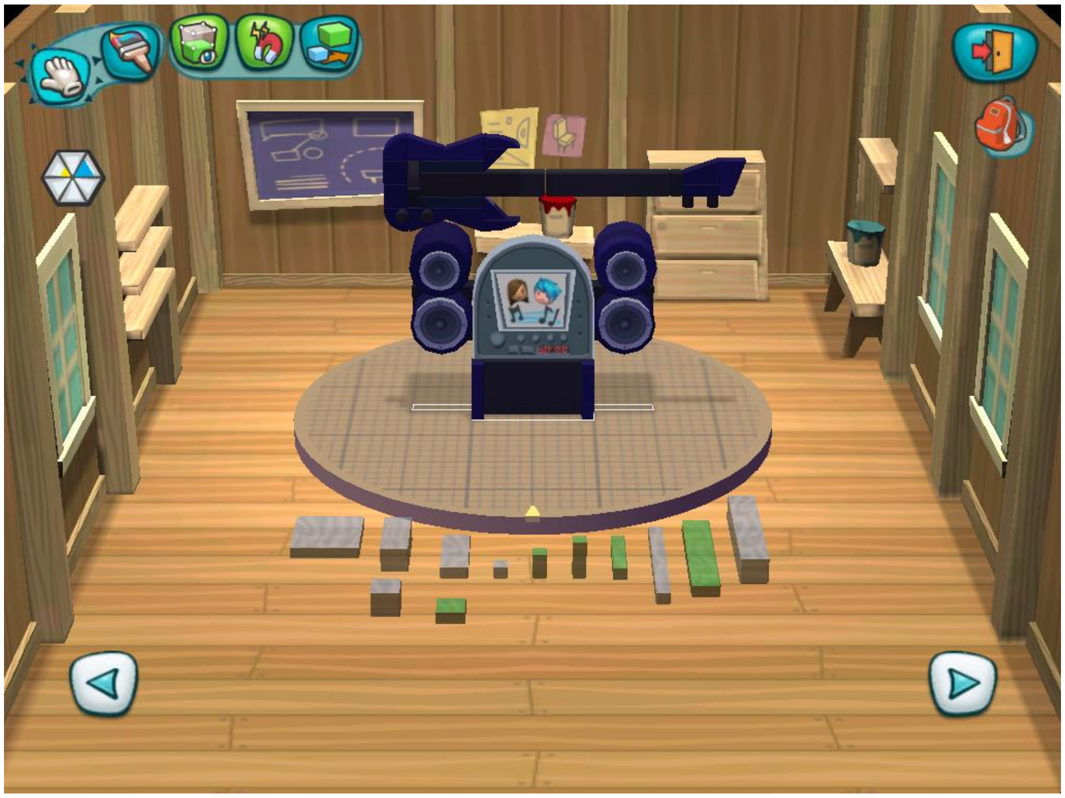


Figure 3 above shows MySims by Electronic Arts. [3]

MySims town simulation was a game. It features a grid platter with blocks of different sizes and shapes that the player uses to construct furniture.

The platter can be rotated and the camera can be zoomed in and out to get the correct viewing angle. What it offers is an in between of CAD and Lego that is very easy for a player with no experience to pick up and use.

Its limited by its standardized parts, there is no way to create or acquire a shape not provided.

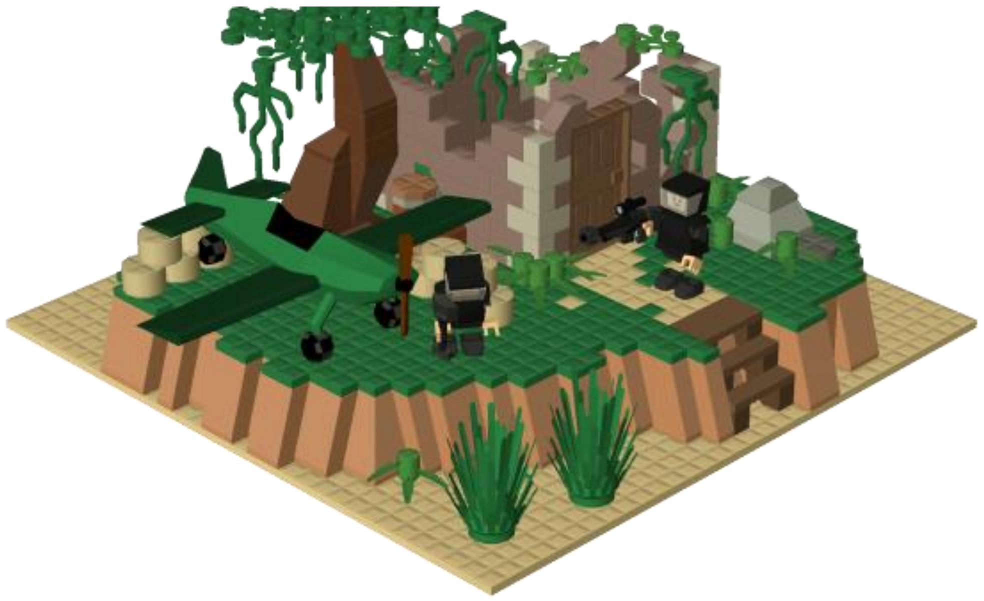


Figure 4 above shows Blockland by Eric Hartman [4]

Blockland is a first person game where the user plays a Lego-like character building with Lego-like bricks. Players are able to select blocks, place them down, adjust their position and color them. This can be expanded upon with the ability to create new blocks in an external modeling program and import them to the game.

Placement of bricks is set to a grid layout, blocks can only be rotated by intervals of 90 degrees. This is in keeping with the grid Lego style of the game.

The user can freely walk around, fly and interact with the structures they have created.

Blockland offers a high level of complexity. For a new user it’s not hard to build a simple construction, for a more experienced user it’s possible to do vastly complex builds, however there is still a limit imposed by the grid layout and the fact that the blocks cannot be modified in the game.

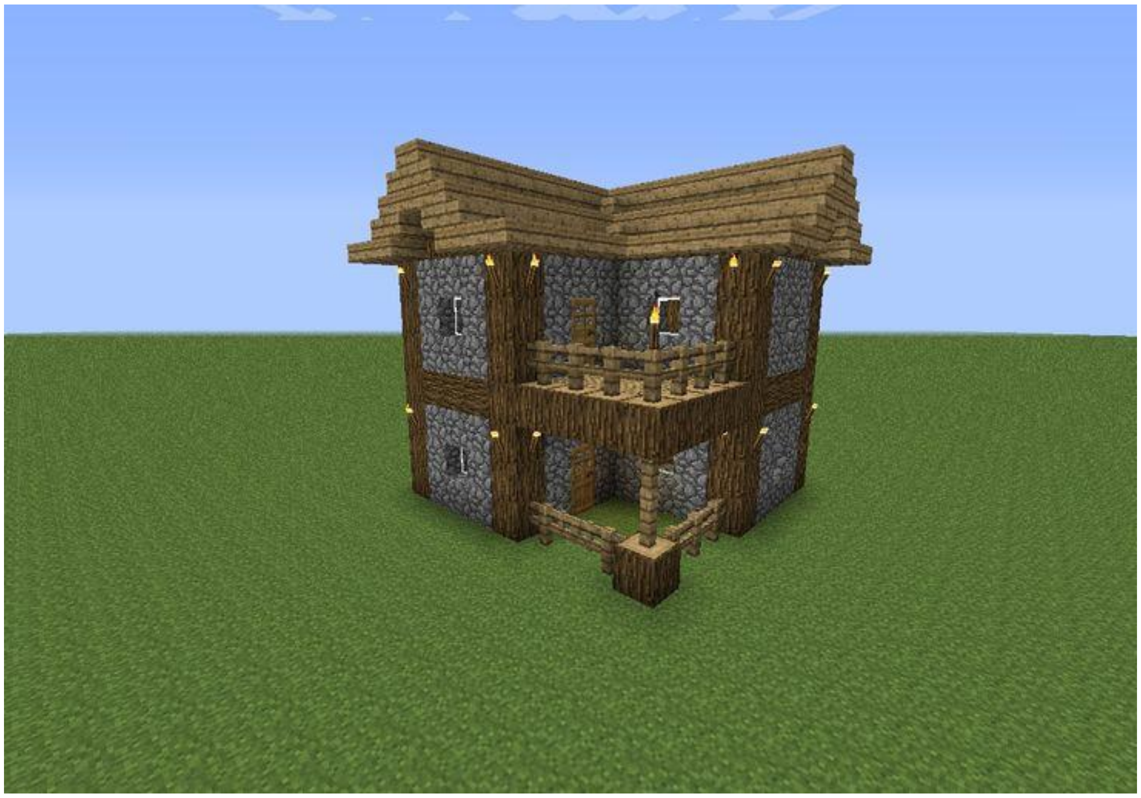


Figure 5 above shows Minecraft by Mojang. [5]

Minecraft is a first person voxel based building game. It features a world divided into a 3D grid, users can place or remove blocks off different types from this grid.

It offers a simplified approach, each block being a standard cube means that the complexity of construction is greatly limited. However this simplicity also means the entry level and learning curve is low.

Complexity can only be increased by making builds larger and treating blocks as pixels. There is no means of adjustment to placed blocks, they can only be placed or removed.

Points on game construction

- They all use a grid for construction
- They all use standardized components for construction.
- Blockland and Minecraft allow the user to interact and move around their construction.
- They all primarily use rectangular components.

REAL WORLD MODEL CONSTRUCTION

This section will briefly review the construction of models in real life.

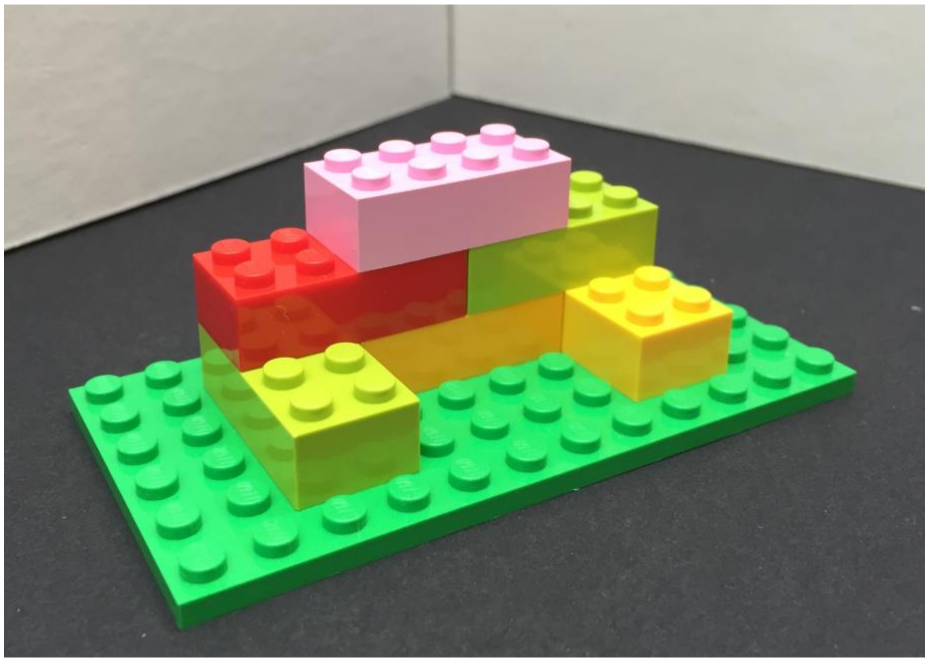


Figure 6 above shows Lego construction.

Lego blocks use standardized components, they follow a grid pattern, once positioned they offer very little room for modification. Blocks at the top can be repositioned, lower layers cannot. Lego blocks are subject to physics to an extent, however their ability to attach to each other greatly reduces the effect of physics on individual blocks.

It's not easy for the user to produce their own Lego brick, they must depend on the manufacturer for this.

Complexity can be increased by using many different standardized components and combining smaller components to produce larger ones.

Due to the grid pattern blocks can only be positioned in rotations of 90 degrees. However hinged blocks exist to work around to this limitation.

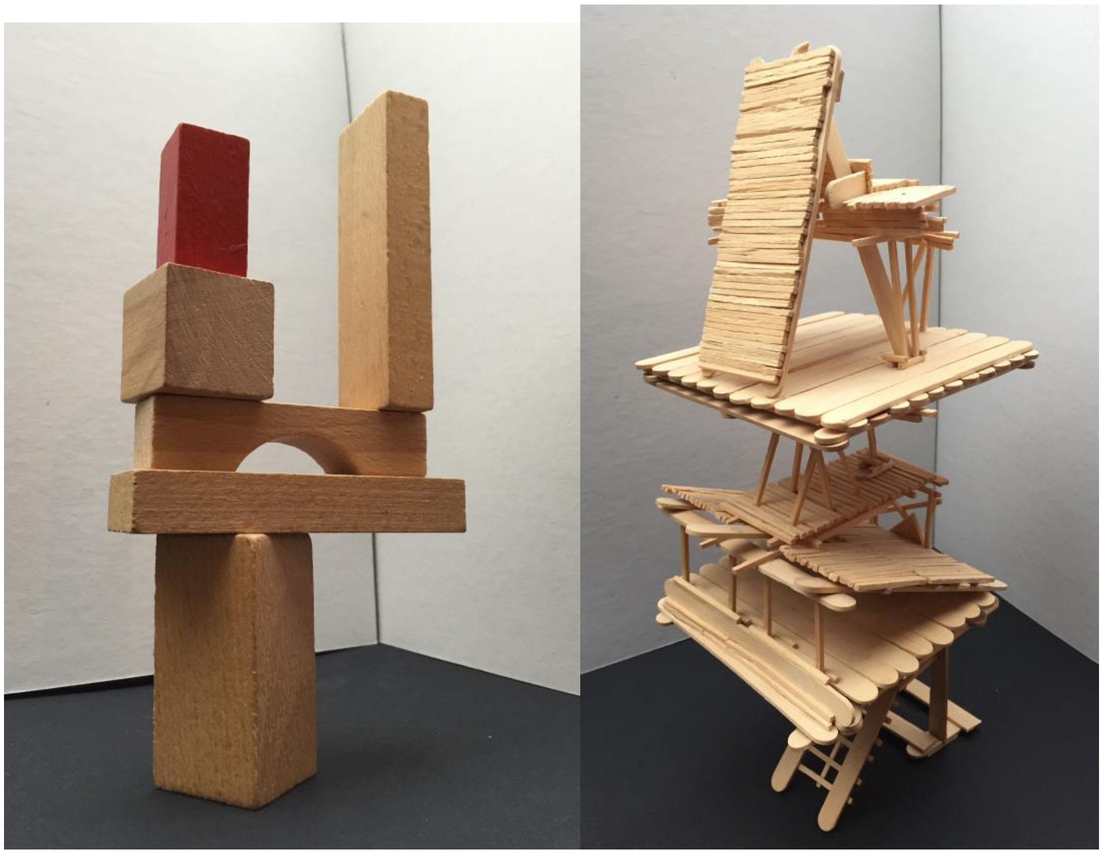


Figure 7 above shows wooden block construction on the left, and a match / lollypop stick construction on the right.

Wooden blocks come in standardized components. They are not tied to any grid allowing them to be placed in any rotation or position. They are also subject to physics, which becomes a consideration and defining feature in a wood block construction.

Complexity can be increased by using smaller or more detailed blocks. If the user can do woodwork it's possible for them to create their own custom blocks.

Due to physics, complexity is limited, once a certain scale is reached constructions may become structurally unsound.

Match / lollypop stick construction offers a finer level of detail than woodblock. It also typically involves glue, lessening the effect of physics on the components. It uses standardized components and is not tied to any sort of grid.

Components can be modified before placement by cutting sticks or joining multiple sticks together.

Points on real world construction

- Lego construction closely matches digital construction in games like Minecraft and Blockland.
- Physics is a factor that has become relevant in real world construction.
- The absence of a grid is a notable feature of real world construction, apart from Lego.
- The ability to modify components easily is a feature of both wood constructions.

3D PRINTERS

There is a wide variety of 3D printers available. Many using different technologies and materials to produce their end product.

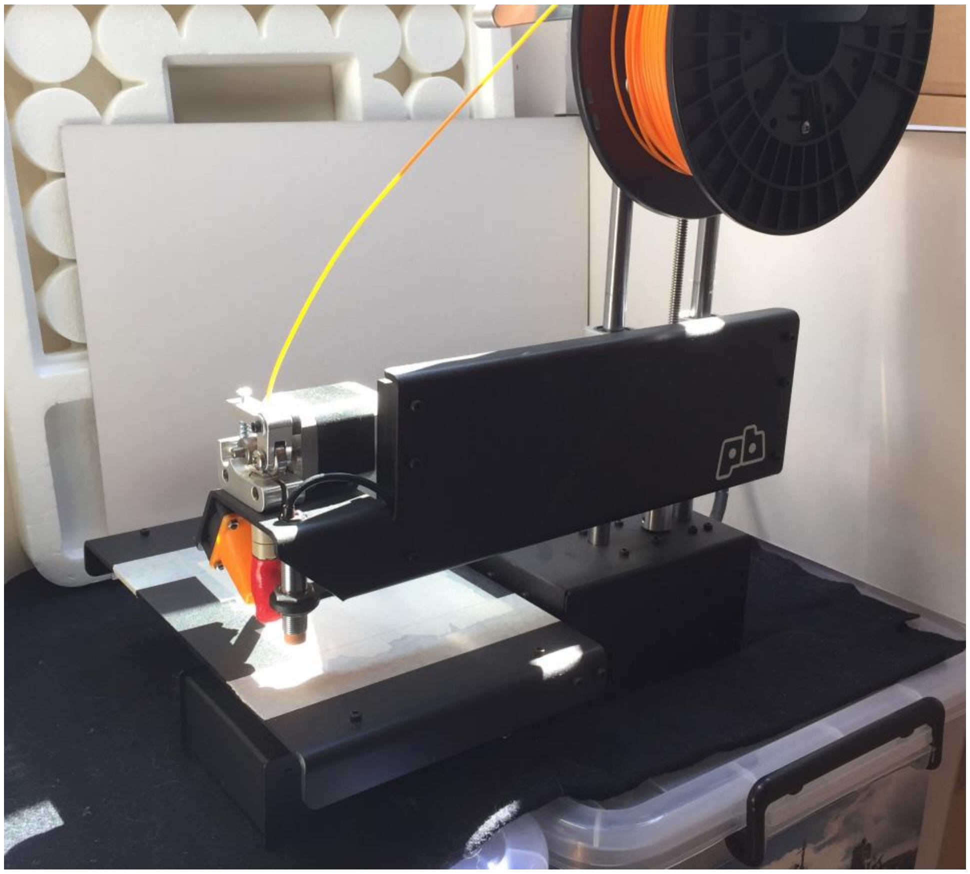


Figure 8 shows a PrinterBot Simple, this is the machine print testing for this project will be performed on.

All printers appear to be restricted to a set build volume that limits the scale they are capable of printing at. The PrinterBot prints from the bottom up, so prints are required to be structurally sound from the bottom up. Other printers may have different structural requirements.

REVIEW

There are key differences and recurring similarities between CAD, game and real world construction. Games attempting to make their interface more intuitive appear to be taking inspiration from a real world construction experience.

All the games mentioned here reference Lego closely using a grid layout. Both the games and real world construction use standardized components to simplify their process. MySims appears to take inspiration from wooden block construction, however it maintains a set grid.

None of the games offer the ability to modify and warp the shape of components in the way that CAD programs can with their extrude features. Only Blockland has the ability to import new components made in an external program.

DESIGN

EARLY DESIGN

A number of design concepts were tested during the early stages of development. The initial designs used a fusion of first person viewing with the ability to place objects using the mouse in a similar way to MySims.

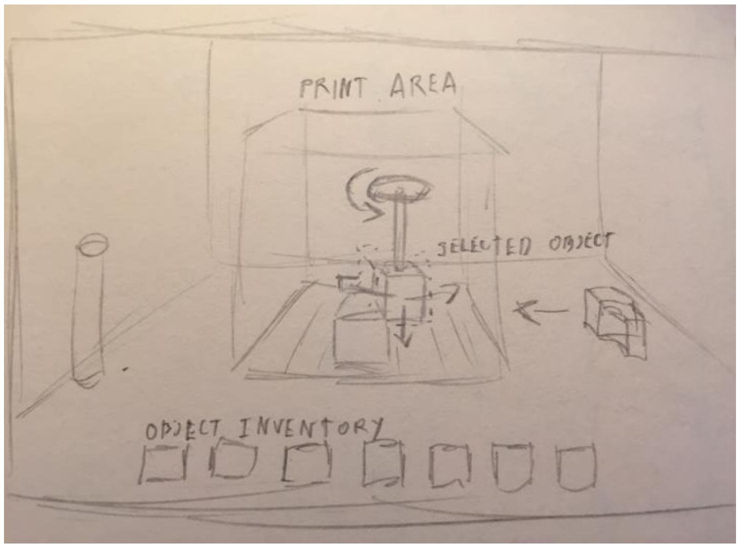


Figure 9 shows an early concept sketch of a building area.

CONCEPTS

- The user is situated in a first person view. In this they can place and remove objects, attach them to the final model, adjust their position.
- Objects can be constructed on the side, replicated and attached to the final model.
- The user should be freely able to move and interact with the model as if they were standing in the room with it.
- The program should allow for mechanical elements, where a screw or post hole can be marked and an appropriate post can be generated for later printing.

REQUIREMENTS

Primary:

- A 3D environment that users can navigate.
- The ability for users to adapt and modify their model in the environment.
- Object export to a 3D printer

Secondary:

- Multiplayer ability on the same model.
- Multiple means of modifying the model (Drawing, Object snapping etc.)
- User alerting of impossible objects to print.

FINAL DESIGN

While many of the early design goals remained, the design quickly adapted once first development started.

Firstly it became evident that merging first person camera movement and mouse placed blocks was impractical. It distracted from the first person element by requiring the user to change the action of their mouse from camera control to cursor control, and as the cursor is intended for exploring 2D space, the use of a cursor in a 3D space was unintuitive. For these reasons the project was switched to a dedicated first person environment more akin to Minecraft or Blockland.

Secondly the mechanical elements mentioned in the early concepts above where deemed to be impractically complicated for this project, the focus was moved more in the direction of architectural constructions and model making, rather than component making.

With the move to a pure first person interaction a new concept for placement was required. Using a tool that instantiates a block in the direction it’s pointed, this is the approach both Minecraft and Blockland use, and is the approach I decided to use for this project.

A master block demonstrating the currently selected block and its rotation was also included in the design at this stage.

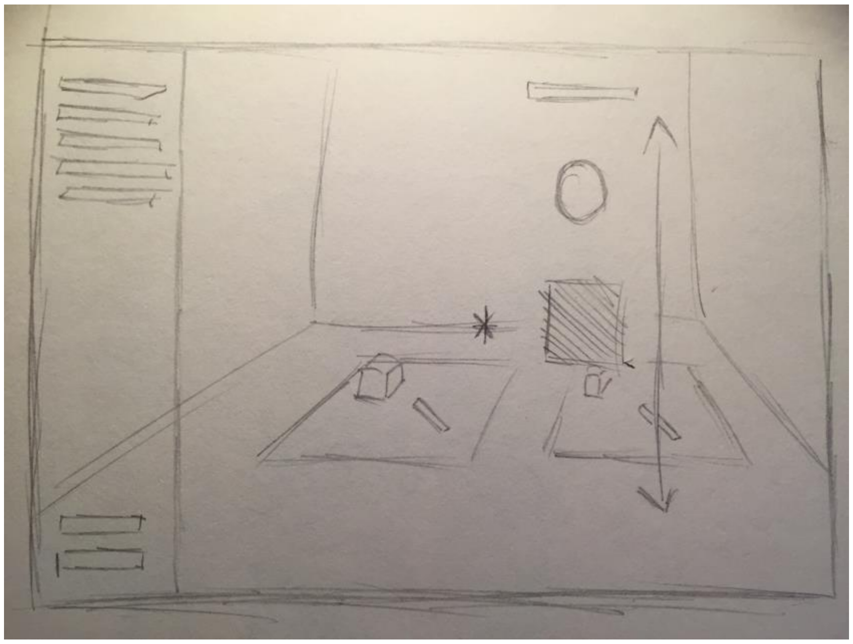


Figure 10 shows modified concepts in the final design.

It also became clear that the main feature lacking from the game approaches was the ability to place blocks without a grid, in any orientation. Another lacking feature was the ability to warp and modify blocks, not just place them. A new list of concepts and requirements was drawn up in order to represent these changes.

Concepts

- The program is first person, with the using being able to walk around and interact with their construction.
- Blocks should be able to orient in any rotation or axis.
- The user should have a way to customize their blocks.
- The program should have the ability to import new and custom blocks.

Requirements

Primary:

- The program must allow the user to place and remove blocks.
- The program must allow the user to export constructions to a format a 3d printer can read.
- The space in which the user is building should sensibly relate to a space that can be 3D printed. E.g The user can't build something too large for a printer.

Secondary:

- The program should be adaptable to allow multiplayer building.
- The user should be able to import models made outside of the program.

Considerations

In this model, it is possible for the user to build constructions that are impossible to print. Early design requirements listed checking for this as a secondary requirement, however it was dropped completely from the final requirements, as it was deemed impractical to perform this check because of the capabilities of different printers, the physical modifications users made to them, and the users intended purpose for their construction cannot be predicted or tracked.

A production program would be required to support potentially hundreds of different printer models. Each of these printers would be using a different means of printing, some using deposition modeling others may use stereo lithography and a printer may be using a particular kind of material over another. All of these effecting what is structurally possible to print. Maintaining and testing a database of variables for all printers and what they are structurally capable of becomes impractical. It's also possible that the user does not plan to print their model at all but use it as a component in another model.

TECHNOLOGIES

Based on the requirements listed above, the chosen technology to build under was Unity 5. It's a program that I already had some experience with. It offers a solid engine for creating a 3D environment, has an established community and detailed documentation. It allows for scripting to be done in C# which is a language that I was very familiar with before starting this project. It also offers support for some additional features such as multiplayer and VR support, it would be beneficial to be able to add these features to the program further along in development.

An alternative choice would have been Unreal Engine, or a Java based 3D engine such as jMonkeyEngine. Both of these would be interesting to experiment with further, but given the time constraints on this project it was a safer choice to go with tried and trusted technologies.

Early stages of design considered the need for a separate program in Java to handle the conversion of user builds from a Unity export format to the printer file format, this was found to be unnecessary as Unity and its scripting languages are capable of doing the export independently.

The 3D interface used and tested with was Cura, this is a personal preference. As the program exports to a standardized file type any printing software would be applicable.

In summary, the project engine and IDE selected is Unity 5, final implementations were developed and tested under 5.3.3f. Scripting was done in C#.

The exported file format is Obj, this was selected as a standard format, and further details on this format and the reasons for its use are discussed in the implementation.

THE TARGET USER

The intended user of this program is someone who is familiar with first person games, particularly first person building games such as Minecraft. This means that they are expected to know how to walk around a game world and be comfortable with exploring and understanding 3D environments. They should understand the idea of targeting an object and using the mouse and keys to manipulate that object in the same way as in Minecraft or Blockland.

For printing the user is expected to know what their printer is structurally capable of and build with that in mind.

USE CASE

The user enters the program and selects a printer type.

They can start building by placing, selecting and modifying blocks.

When a build is completed, ready to be exported, or the user is ready to exit the program a menu can be accessed.

This menu should have the ability to modify settings, exit the program, export the build and import new components.

Upon successful export the user should be notified.

Upon exit the build should be cleared and the program closed.

REVIEW

As this is a solo project compatibility between separately produced components of software is not a concern. For this reason the project is developed primarily under the prototype model. During the early to final design stage many changes occurred as more was learned about the development process. The goal with this design is to satisfy the requirements, while allowing the programs approach and design to adapt as needed to new discoveries.

IMPLEMENTATION

OVERVIEW

The final implementation is divided into a number of components and scripts, referenced throughout this section, those terms include the following:

The user - is the term used to refer to the actual user, or their in game body through which they are looking.

Blocks - are the virtual building blocks that the user is spawning into the world to form their creation, they are basic Unity objects, with a preconfigured set of attributes to define them in the world.

The tool - is a script attached to the user, which handles all manipulation that the user can perform of blocks in the world space, particularly the placement/spawning of blocks and movement of them.

The Master Block - is the block that the user is holding, it acts as a template for the tool to use when deciding what block to place.

The pointer/hologram - is the dot or transparent block that shows where the user is looking or what is about to be placed. It also acts as a kind of virtual arm when manipulating blocks, while the pointer decides which block is manipulated.

The GUI/Menus - form any overlay ui or menu screens that can be opened, including the main menus and the tool bar when building.

Unity specific terms that occur:

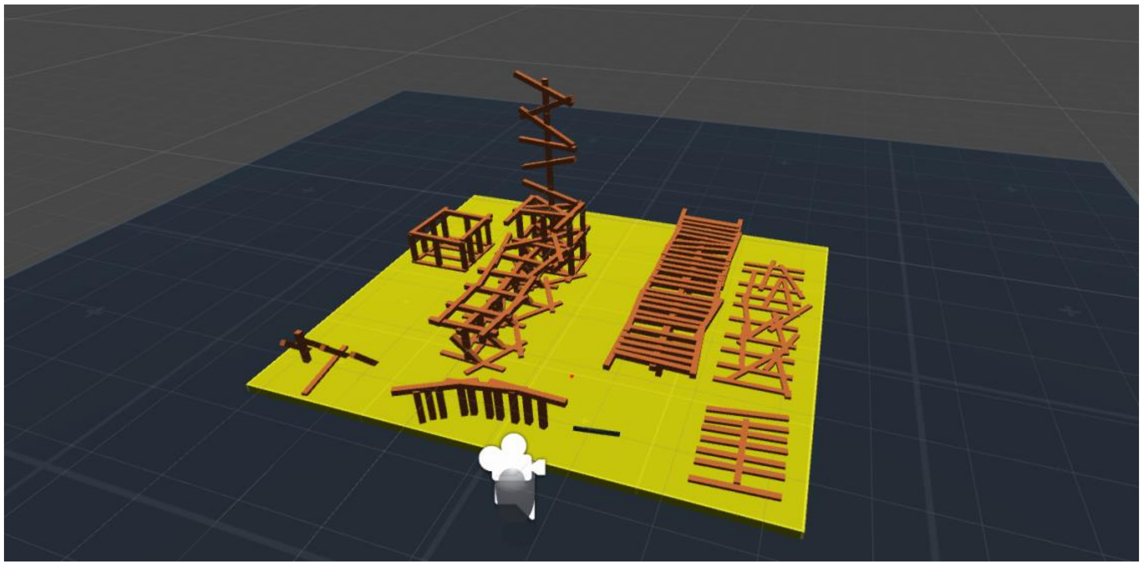
Prefabs – Preconfigured objects in the world, blocks.

RayCast – a line from a point in a direction that will continue until it is blocked by a rendered object.

Scene – The area loaded without need of a loading screen, this project only uses one.

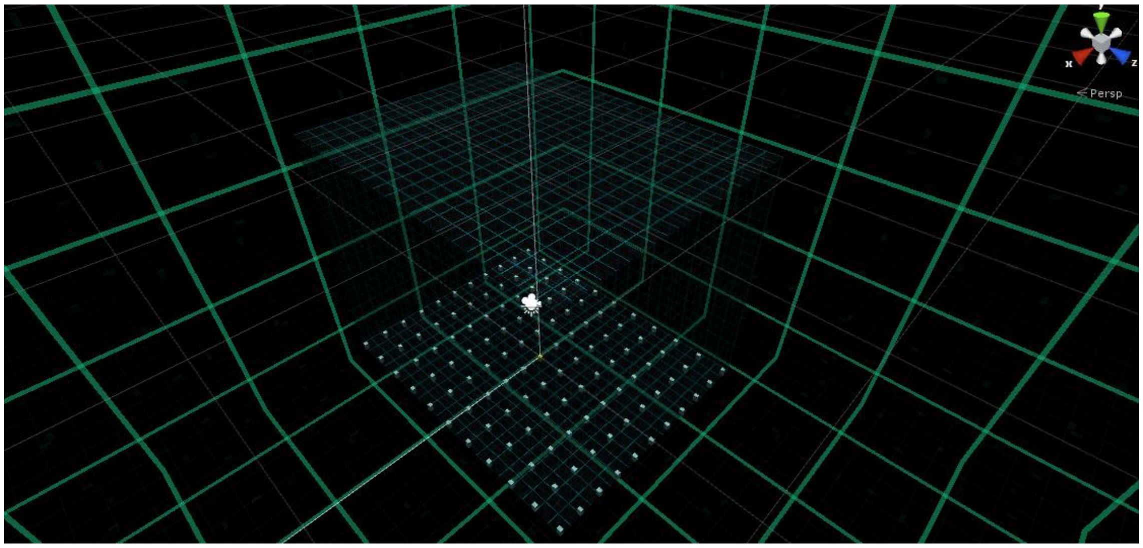
THE WORLD

The world is centered on a platter area, which represents the print platter of a 3d printer. This can be adjusted to match the model and size of the printer connected.



The above figure 11 shows an early implementation of the platter.

In the early implementations, the yellow area acted as the platter, the user could exit and enter this area. There was also a separate platter showing the object that would be exported, which was a clone of the users construction.



The above figure 12 shows the final platter room.

This was later changed to the cubed model or room shown above in the final version. The reason for this was to more accurately match the limitations in size of the printer and prevent the user from building outside the printable area. It also became evident that it was unnecessary to show the render version of the users build, since this was ultimately an exact clone of the users own construction.

The render and export will be covered in a separate section.

The room created is a standard cube with an arbitrary scale, the room management script will then apply a multiplier to this scale warping it to the dimensions of a selected printers build area.

The grids provided inside the room and the placement markers on the floor are not of any set real world scale, they are purely there as a visual aid to the user when lining up and placing blocks. It would certainly be possible to focus more on developing a grid layout that related to a physical dimension, and continued development of this project would almost certainly see that feature added. However the matching up of a real world scale to the virtual scale was beyond the scope of this project and was not vital to the primary goals outlined for it.

World Physics

It is possible for the user to enable physics on the blocks in the world, this is useful for constructions that require a natural physical look, such as a collapsed wall in a model. Physics use Unity’s inbuilt in the physics engine.

The physics feature works by looping though each existing block in the scenes individual frames, checking if the physics are enabled and enabling them if they are not. Disabling physics works in the reverse. Since all blocks are naturally physics disabled this feature runs continually. In a scene where large volumes of blocks are being loaded, such as, the user building a whole world rather than just in a small area, this approach would need to be reconsidered for performance, in this case though its effect is negligible.

Due to the limitations in Unity, objects with complex colliders for example, such as objects imported with the obj importer, which have complex poly colliders, these cannot use physics. The simple solution would be to give them a simple box or sphere collider that roughly matched their size and shape. The physics here would be imperfect to the objects appearance, so it was decided to not apply physics to these objects at all in order to avoid confusion for the user. The user would be made aware of this in the UI or documentation.

Block Clearance and Room Resizing

In order to avoid losing user settings such as imported blocks the scene never reloads when resetting or clearing.

Upon selecting clear blocks from the main menu, the room management script will loop through all existing blocks in a similar fashion to the physics enable feature. In this case it will call Unity's destroy method on the block, removing it from the scene.

Upon selecting a new printer type from the main menu, the room management script will first clear all blocks, then apply the scale multiplier to the room. The downside to this is that if the user starts building on one printer mode, then attempts to change it, their build cannot transfer over. The reason for this is the Y axis multiplier, it will cause the floor of the room to move up or down depending on the printer selected. Some attempts were made to correct this by resizing the room, then moving it to the 000 position, as the room is positioned from the center, though this cannot correct the floor position.

One solution would be to determine the movement of the floor from the size of the room and the multiplier being used on it. Then offset the room by that amount, or even adjust all blocks in the scene by that amount.

In practice and testing, this never became a large issue, ether builds did not require the full platter size of the printer, or they could be adjusted after export in the printing software. Due to the constrained time of this project this issue was not perused further, but should be readdressed in future development of the software. For a production release though it should be ensured that a warning is presented to the user before clearing blocks on a resize while a more advanced solution is not in place.

THE USER

In earlier versions the user was represented as a pill object, this gave them mass and a physical body to interact with the world and constructions.

In later versions the pill was replaced by a rectangle, to provide a more solid interaction with the primarily square nature of the world, the render was also removed as it proved to be obtrusive, leaving ONLY THE RECTANGLES COLLIDER.

MOVEMENT

Movement was originally handled by Unity’s provided character controller script. This was later tuned to allow an easy fine control, which was needed, head bob, which is found in most games was removed, walking speed was slowed and running speed was increased.

It was later decided, based on feedback from testers, that while Unity’s controller offered a solution to user control, it was oriented too much towards standard gaming in a shooter or adventure game, where the player is moving around a large world. It was replaced with a custom movement script, aided by an open source camera control script provided by Unity’s online community.

Unity’s character controller appears to operate in a kind of simulated step, where with each press of a movement key the user takes a certain number of steps.

The custom movement script focused on allowing fast but very fine movements of the user instead of natural steps. This is achieved by incrementally adjusting the players XY position directly each time a movement key is pressed.

To avoid unexpected speed changes caused by multiple speed inputs, such as the user moving forward and left at the same time, the script keeps track of the number of active inputs and divides the standard movement speed by the number of inputs.

If the shift key is held down, the standard movement speed multiplies by two in order to let the user move around the room faster.

FLIGHT

The user can also fly if the space bar is held down, this adds force to the user object in the Y axis, which simulates the feeling of a jetpack. However, it still uses the incremental XZ movement intended for ground use. This can produce a slightly awkward behavior where the flight uses physics and natural force, but XY movement uses exact changes with no force.

An attempted solution to this was to have the user switch from incremental XY position adjustments to directional force when flying. This meant that while on the ground placing blocks, the movement would be exact, and while flying it would simulate hovering with force being applied to push the user in the direction they wished to go.

This approach worked correctly when flying, and when on the ground, however the switch over became extremely jarring. The speed difference between ground movement and flight meant there was a sudden stop in forward momentum when switching into flight. The clearest solution would have been to apply the same force to the user on the ground and essentially push them around, however this would sacrifice a huge amount of precision for the user's movement, which would have defeated the purpose of the custom movement script. Ultimately it was decided that precise movement when standing was more important than a smooth flying experience, so further investigation of this issue was not perused.

Another issue with the switch from flight to ground was determining when the user was touching the ground and when flying. One solution was to measure the user's velocity, since when flying this number would constantly change, when velocity was at 0, the user must not be flying. While in theory this solution should have worked, the Unity engine appeared to continue minor adjustments to velocity while the user appeared stationery, perhaps due to the physics slide. This meant that the user permanently appeared to be flying.

The next solution to this was to send a RayCast down from the user, and measure the distance before the ray hit an object. Under a minimum tolerance it could be assumed the user was standing on an object. This situation could be manipulated however when standing on very narrow objects that the ray missed.

As this feature was dropped, no end solution was found to this issue.

THE CONSTRUCTION

Initially constructions were limited to rectangles resembling match sticks. This was chosen in order to simplify the basic functionality of construction. Using a real world object such as a match stick also gives a real world reference to the user in terms of the scale and physics of their construction. This was later expanded upon by adding the multiple block sizes and shapes for the user to select from, this was expanded upon again when the obj importer was added, and again when the ability for the user to reshape and warp blocks freely was added.

Each block in a user's construction is an independent Unity object generated from a prefab block that contains most of the standard settings for a block, such as material, physics and render size.

One interesting thing to note about this approach is that, while in this project it was not utilized, it would be possibly to apply independent scripts or logic to each block. This could allow for blocks with additional features, such as the ability to resize themselves depending on placement. This may also be a future alternative to the physics system described in the world section, rather than looping through all blocks constantly to check physics settings, blocks could independently spread messages to enable or disable physics to their neighbors.

The size and amount of blocks possible in the construction is limited by the power of the host computer, and the unity engines ability to handle large numbers of blocks. The user could notice slowdown on an extremely large build, however given the limited space of the build room it's not likely any build would reach such a large quantity of blocks.

Importing and Exporting

The import and export features use a number of provided, or modified scripts made available by the Unity wiki.

Both import and export work around the fact that meshes in Unity are simple poly maps, these consist of a number of points and so can be written out or read in from a file structure that also uses poly points.

While there are a number of file types that could be used, such as stl which is commonly used in 3d printing due to its simplicity, the obj format is used because of Unity. Both stl and obj are widely supported by 3d modeling and printing software, however due to the nature of Unity as a game platform, the vast majority of support for object exporting comes from developers who are generating and exporting objects that are intended for game use. This means the added complexity and support for things like textures in obj are more beneficial to them. While it would certainly have been possible to develop a stl export script specifically for 3d printing, there were many benefits to using the established obj approach.

These included:

Wider base of support and established pretested code provided by the Unity community for obj exporting.

Future expandability for 3d printers where different textures could be applied to define different materials in a multi material 3d printer.

Future expandability of the software's usage, it may be desirable to expand the functionality of the program to not only export for 3d printing, but also export models for other 3d applications such as game development.

Currently there is no support for textures or texture changing in the program, however there is room for it to be added without a major rework.

Export

There is a single render object with no mesh placed far away from the build room so that it is invisible to the user. When the user calls the export script by clicking export in the main menu, the script will first do some validation to ensure there is something to export, then call the render method attached to the render object.

This method will move the render object to the center of the room at 000, this is to make sure that the final render is centered to the users build. Early tests lacking this step would render correctly, however the rendered blocks would be offset from the center of the render object.

It then generates an array of all blocks and an empty list with a generic type of MeshFilter, that list can then be populated by the individual meshes of each block.

Finally it uses Unity's mesh combine feature, generating an array of combinable meshes from the mesh list, and combining them into a single mesh to be applied to the render object.

The render object is then moved back to its original position out of the user's sight.

This approach keeps the render object visibly rendered at all times and moves it out of sight when not needed. It may be equality possible to keep the render object at the center of the room at all times and make it invisible when not needed. Both approaches would produce no real different to the end user, however there is the chance that keeping the object in the center of the room could introduce the possibility for errors, such as RayCasts being interfered by it. Adjusting RayCasts effecting layers could compensate for this, however for simplicity and stability, storing the render object out of the way seems to be the best choice for this project.

Once the render method has been called, the export script hands over to the ObjExporter script.

The Unity wiki lists a number of export scripts for obj files, all of which are intended for developers to export game objects during development from Unity's internal menus. One of these scripts offered a simplified approach that was easy to adapt and use from runtime via an in game menu, however due to a unity update towards the end of this project, Unity 3.3.2 – 3.3.3, this script appeared to produce faulty exports. Another more complex script did not appear to be effected by this update fault, however it included a large number of render options and additional features such as texture exporting which made adapting it for runtime use less feasible in the time left before the project deadline. The script included in this project is formed by pulling the required methods from the working script, modifying them and using them to replace broken methods in the original script. The original script was also modified to no longer rely on unity's debug features or menus, and setup so the export could be called via method.

This obj export method can now be called by the export script, which passes the mesh details of the render object to it. The meshes are converted to an obj format by the ObjExporter and saved as a file to the desktop.

Unity does not seem to offer any kind of file explorer at runtime, it is possible to open an explorer that allows the user to select a location and filename for the export when in debug mode inside unity, however exported games cannot. This is the reason why renders are saved directly to the desktop. It's not an ideal solution and I have attempted to work around the Import feature. As it was only towards the end of this project that this issue came to light, no perfect solution could be developed. From research done, there has yet to be a full file explorer implementation developed for unity that offers the kind of experience you might expect from a native operating system file explorer.

Notes on export scale

A question that was asked a number of times during the presentation day was about the scale of exports and builds in the program, and how they relate to the real world scale of what the printer produces.

The size of beds are storied in inches, for example ("PrinterBot Simple", 6f, 6f, 6f) where 6 x 6 x 6 inches is the build volume of a PB Simple.

There is an arbitrary scaling factor of 15. The rooms Unity scale is then set as Vector3 (printer.X / roomScaleFactor, printer.Y / roomScaleFactor, printer.Z / roomScaleFactor)

The default blocks remain a standard size. The room scale factor was tuned to a point where the default blocks in relation to the room would be small enough to produce detail, but big enough to quickly allow a build be created, and big enough so that most printers would be physically able to print them.

The stick block, which was the first and is the default, is roughly scaled to the size of a matchstick, and used as the reference when deciding the size of other blocks and the room scale factor.

The grid in the room has no exact scale and is provided only as a rotation and spacing reference for the user.

Upon final export, the render objects scale is set to 14, this is again a tuned number. It was derived by building a structure that went to all 4 corners of the build room. This structure was then exported and imported to the Cura printing software, which also has a build room set to the volume of the printer. By adjusting the export scale the build could be made to match closer or further from the corners of the build room in Cura. Once the exported object appeared to be an equal scale in Cura as it was in the program that export scale factor was used for all future exports.

What this produces is a scale relationship between built objects and printed ones that is very close, but does not connect to any exact real world measurement. There are a number of reasons why this approach was followed.

1. Scale adjustments can be made at multiple points, blocks can be scaled, exports can be scaled, and scaling can be done again in the printing software, if an exact size is needed, the part can be built in the program and then that size set after export.
2. The goal of the program was to create an intuitive building program that was more focused on creative construction, with the ability to export that construction. Exact measurement was not a priority feature.

What would certainly be possible would be to attempt to change the room grid from being a simple pattern applied to the room’s floor and walls to being some sort of adjusting ruler that showed marked distances in real world length.

Another option would be to have an overlap when the user is resizing a block that could show the blocks current size in a real world measurement, this might aid a user in more size sensitive builds.

For this project though, it was very much the focus to create the builder and the ability to export, exact sizing would be an area to develop further in the future.

IMPORTING

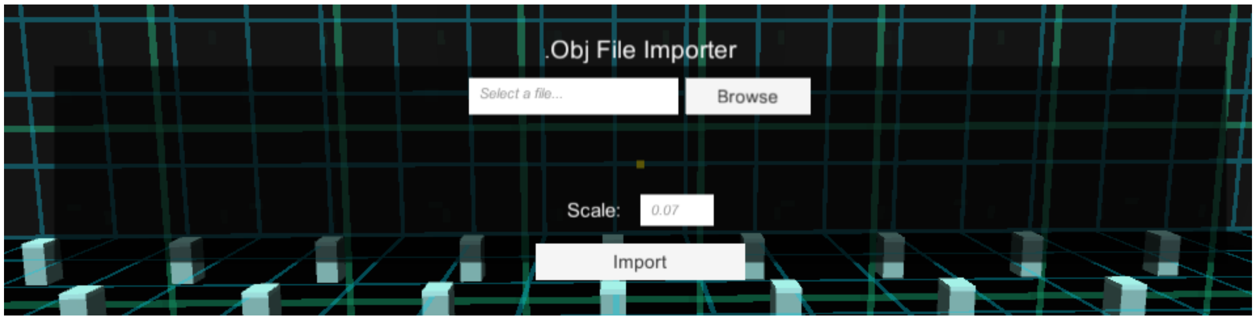


Figure 13 shows the object importing menu

Importing uses a Unity wiki obj import script, it essentially performs the task of the export script in reverse, reading in a file and converting it to a Mesh. The import script required less modification than the export script, the correct method to call already existed in the script, and the only modifications required were to remove the sections dedicated to adding menu items for the importer to the unity screen.

The import script did require a way for the user to select the file to be imported. As mentioned in the export section, Unity has no means of opening a file explorer. A rudimentary file browsing UI from the Unity wiki was included, it is then toggled on and off by the import script when needed. This browser does not match the style of the program, and in terms of usability and readability it’s less than ideal, for example, it does not highlight the selected file. The only solution appears to be creating a file browser for unity that does not have these issues, this was far beyond the scope of this project however.

Once a file path is selected, its run through the obj import script, the generated mesh is saved, a generic block is created and the mesh is applied to it. Methods are called to ensure the proper bounds are calculated and the import scale that the user selects is applied. The default scale of 0.07 is tuned to import objects that have been exported by the program at a similar size to their original in program size. This can be adjusted if objects come from other sources with different sizes.

Finally the newly imported block can be saved to the master block selector and placed down indefinably as if it were a default block.

Imported blocks are complex poly meshes, even for simple shapes. Unity has a number of render types, such as rectangle, sphere etc. These types all work correctly with physics in the engine as they are regarded as simple types. The complex poly meshes do not work with physics and have a number of other limitations and quirks throughout Unity's engine. The import process is unable to determine complex shapes and simple shapes. So a cube imported will not be regarded as a simple type cube. In most cases this is correct as imports would be complex shapes, but it's something to be noted.

There can also be a level of degradation during the import process, this is due to the obj import script and could be resolved by adjusting import resolution.

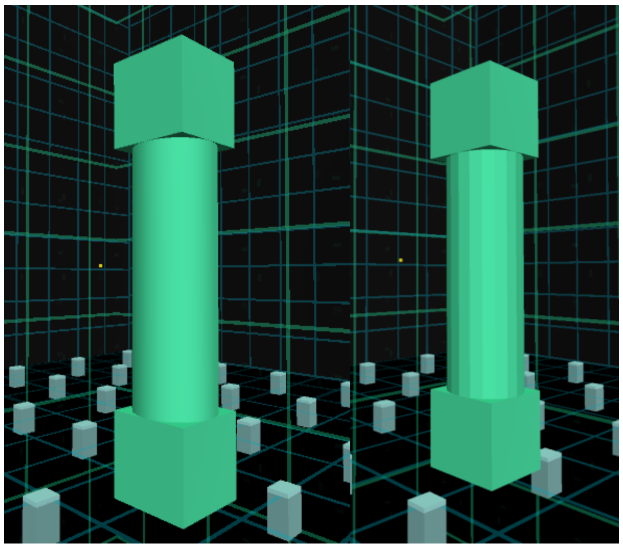


Figure 14 on the left shows a pillar built in the program, the right shows that pillar after being exported and imported again.

PLACEMENT TOOL

Initially the user held a tool that sent a RayCast spawning or clearing an object in the direction the tool was facing. This meant it was very hard to judge where the object would spawn, making fine detail and placement difficult.

Due to this the ray is now cast from the user's center of vision and the tool no longer has a physical object in the scene, it is a script attached to the player object itself. This means the tools effect will always be directly in the center of the screen making it much easier to judge the position. This was later aided by the addition of a pointer object and later again by the hologram template.

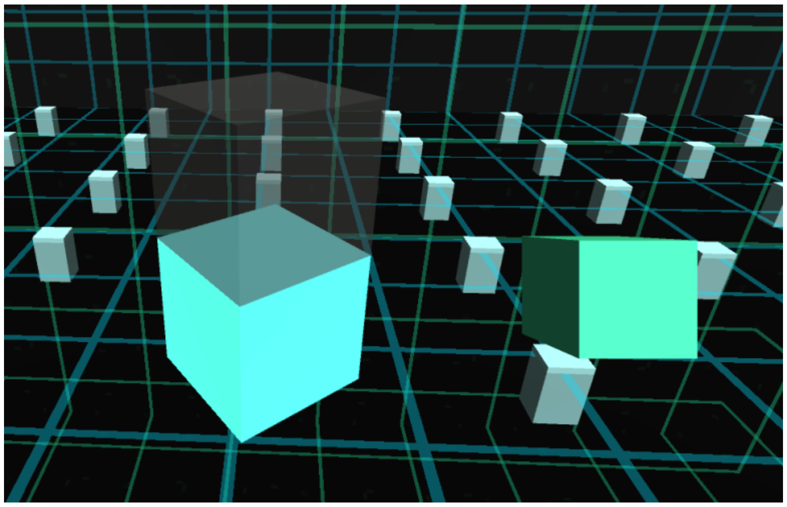


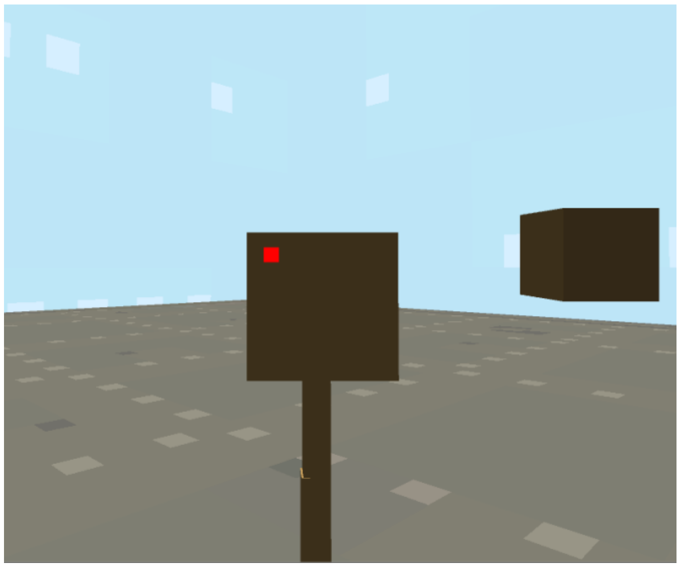
Figure 15 shows a hologram cube on top of an existing cube block on the left. With the master block to the right.

When the user places a block, the tool is responsible for determining the correct position, instantiating the new block and adjusting its attributes as needed.

It does this by first acquiring the position, which will be discussed in the Placement Section. It then uses the master block as a template, creates a clone of the master block, resizes it as the master block is half scale to fit on screen, and finally moves the cloned block to the correct position.

PLACEMENT

The early attempts at placement spawned the block at the exact position of the user's pointer. However it proved much more complex to adjust that position, it accounted for the physical mass of the objects, in order to make them adhere to each other as they would if they were real blocks of a solid material.



In the figure 16 above we want to place a block *at the user's red pointer.*

Objects in unity have a central position and rotation axis, so if a block is spawned at the user's pointer the newly spawned block will have its center axis on the surface of the existing block, causing them to overlap.

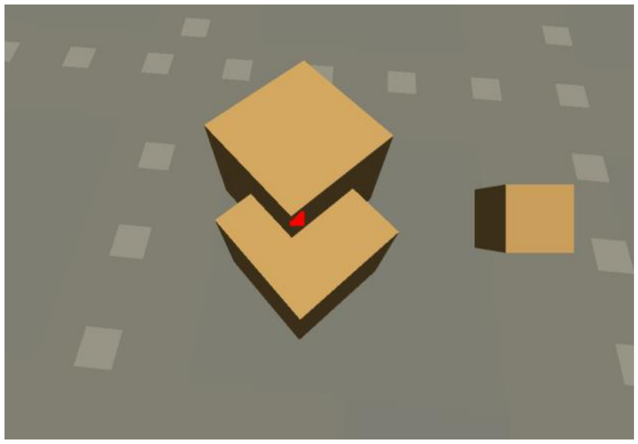
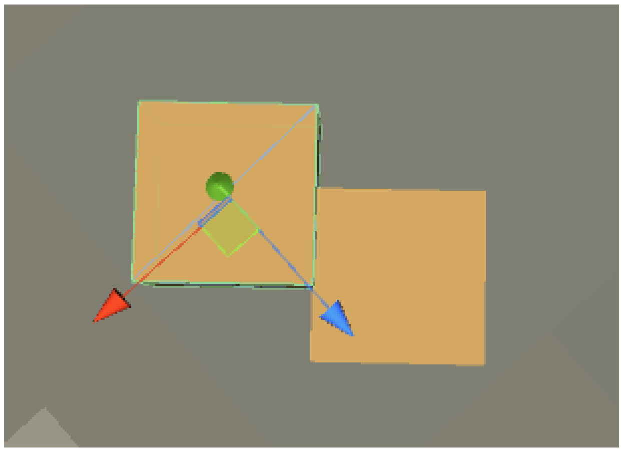


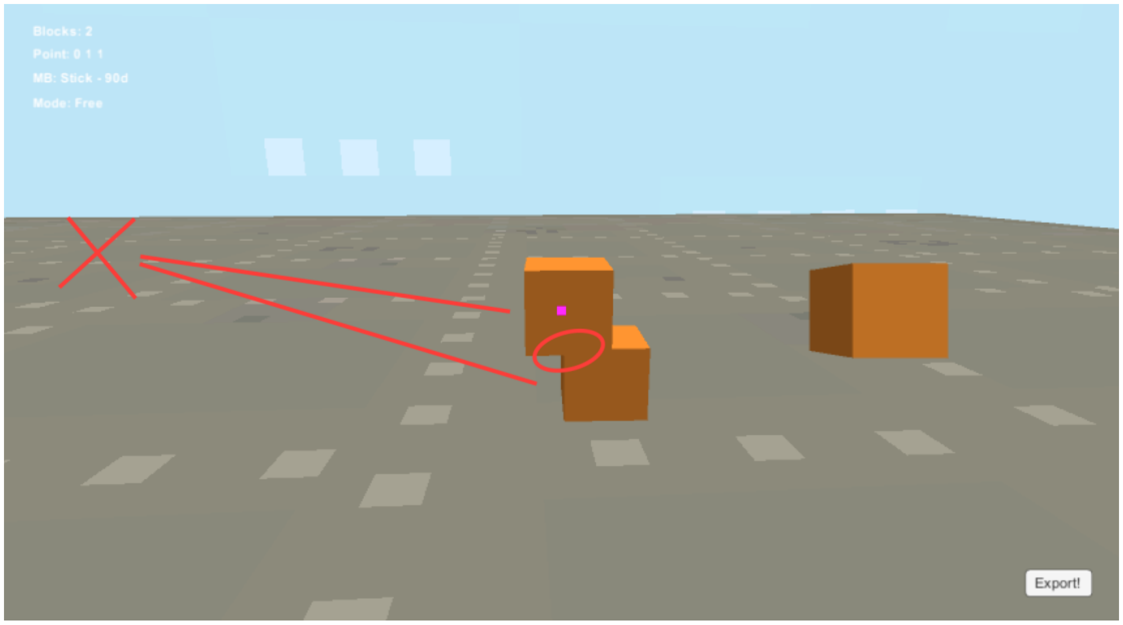
Figure 17 above shows block overlapping.

What we would like is the outer surface of the new block to sit against the surface of the existing block. The issue with this is that since we are not working on a grid plain, the objects can't simply be moved in one axis to correct this.



Above figure 18 shows the desired offset position for the new block on the left, and the blocks axis markers.

The first solution to this was to use Unity's bounding boxes, a block was spawned at the user's location, and then moved towards the pointer location until it detected a bounding box overlap, at which point it stopped and considered the block placed.

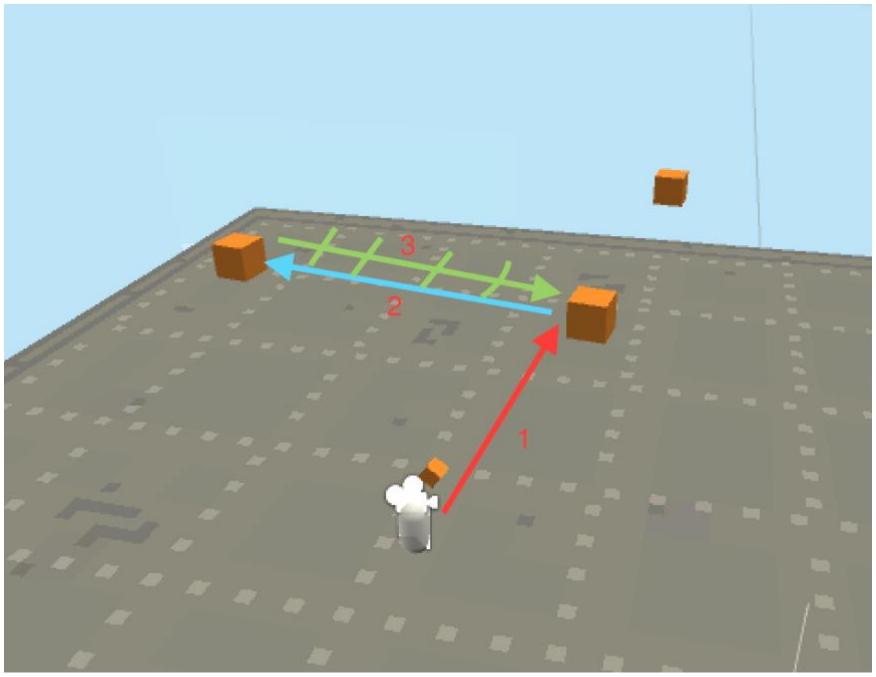


```
Blocks: 2
Point: 0 1 1
MB: Stick - 90d
Mode: Free
```

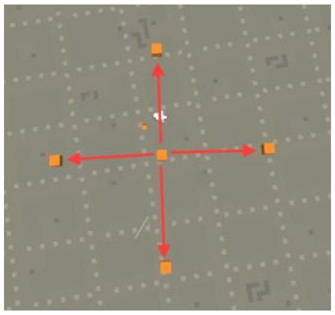
Figure 19 shows two blocks being spawned where user was at location X

The above figure 9 attempted to place a block at the center top surface of the existing block, since the block was moved from a central axis based on the user's position, its bounding box made contact with the other box before it reached the center causing this stepped appearance. Essentially, blocks would be placed within the line of sight of the user's pointer, but not exactly at the pointer. This meant it was extremely hard to judge where the block would be placed.

To resolve this the placement was split into 3 stages, the users pointer ray would hit a surface, a new ray would be sent out in the direction of that surface, and the new block would be spawned at an arbitrary distance along that ray. Finally the block would be incrementally moved towards the source of the ray that placed it, until its bounding box overlapped as before.



Above Figure 20 illustrates this three-step process of placement, below figure 21 shows blocks placed on each side of an existing block, without being moved.



The blocks placed this way appeared to be positioning correctly, vertically blocks would attach as expected, however there was a noticeable variable gap on blocks connected to the side of another block, or on blocks that have been rotated in anyway.

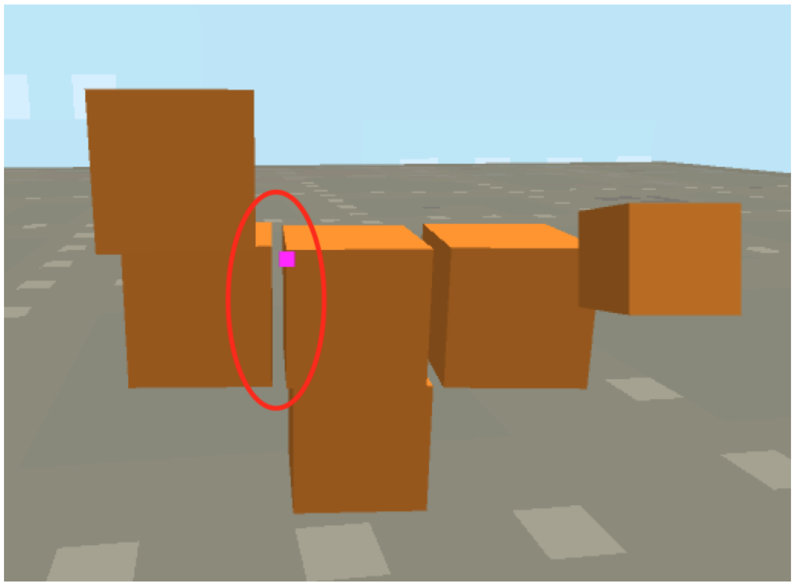


Figure 22 highlighting spacing gap.

This gap varied depending on the placemat of the block, in the above figure 12 we see the same gap on both sides of the center block. Note the vertically placed blocks have no gap and are placed correctly.

A number of tests to determine the cause of this gap where carried out. The position of the user was determined to have no effect. The angle of the block appeared to have the greatest effect. One attempted solution was to offset the time before incremental movement stopped by the rotation. This did have some results, but was found to be too inaccurate to be perused further.

The conclusion was that ether the implemented test of bounding boxes was flawed, or the bounding boxes themselves had an error. Further research on the Unity wiki and forums revealed that bounding boxes do not rotate with the objects world rotation, it resizes with it, but remains in a grid layout.

Due to the resizing of bounding boxes, if ether the old or new blocks were positioned off the 000xyz axis, then the gap would be formed.

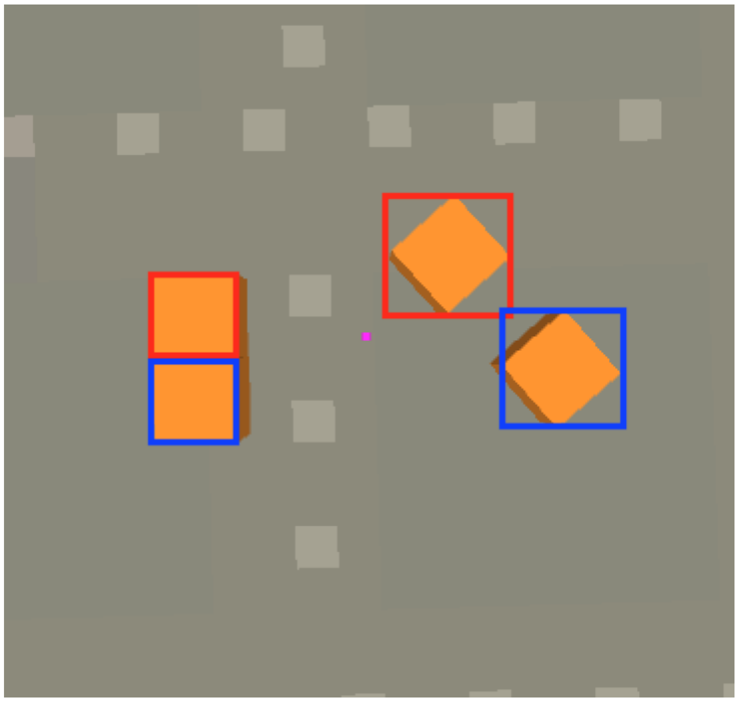


Figure 23 shows two blocks placed at a 90 degree rotation on the right and two at a 0 degree rotation on the left, with their bounding boxes marked in red and blue.

The initial attempts to solve this were to try and rotate or recalculate the bounding boxes to match the objects rotation. However it became evident that that kind of adjustment was not possible in the Unity engine. While Unity does have some support for boundary recalculation, it appears to be fixed to a set rotation.

One solution would be to attempt to convert the box bounds into poly mesh bounds. Then remap the diagonal edges of the rotated box as a poly mesh. There is no inbuilt system for doing this in Unity and while there are some 3rd party solutions, none of them guarantee an accurate poly mesh to mapping. If this were a once off action during development it could be done manually, however this project required it to be performed on almost every placement of a block.

Due to the inaccuracy of programmatically matching a poly mesh to an object, the requirement of 3rd party software, and the overall performance hit that doing such a calculation would have had on both the speed of placement, and the additional memory requirements of storing poly mesh bounds over box bounds. It was decided that another approach would be better.

Note: exact performance and accuracy analysis of programmatic bound remapping could not be tested as the software was proprietary and required an additional purchase. The decision not to use it was based on the claims of the software provider and the reviews left by users of that software on the Unity Assets Store.

THE ADAPTED APPROACH

The eventual solution to this took a number of weeks to come about, during which time development had moved to other areas of the program, it is ultimately a simple modification of the approach described above.

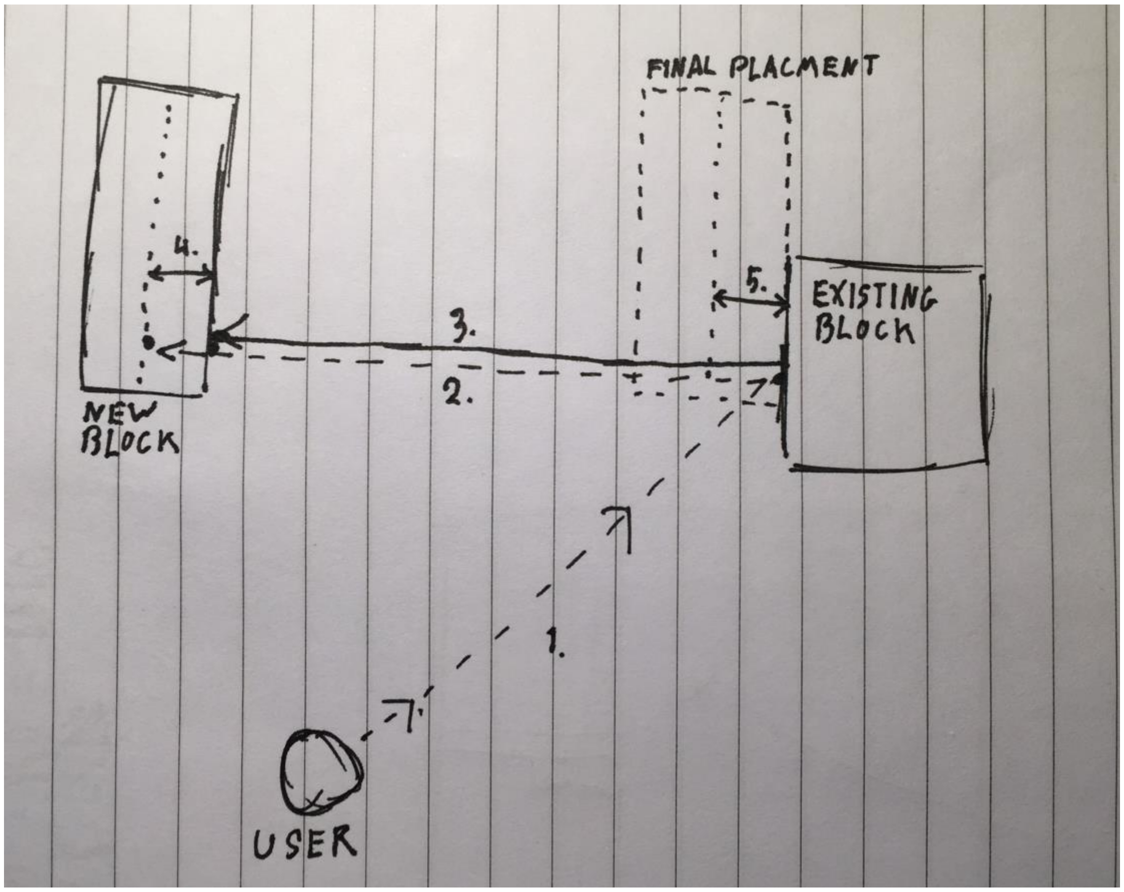


Figure 24 illustrates the adapted approach.

In step 1. The user sends a ray hitting a point on an existing block that they would like to affix the new block to.

That point then sends out a ray in direction of the normal to the surface that the point is on, marked 2. The new block is then placed at an arbitrary distance along this ray.

A second ray is then sent from the same source as 2, in this case labeled 3, after the block has been placed, it contacts the rendered surface of the new block and cannot continue.

The difference in the distance of rays 2 & 3 can then be compared to determine the thickness of the new block at the exact point that it should contact the existing block.

Finally the position of the user can be adjusted by the thickness of the block and the correct position for the bloc can be determined.

In practical application this approach is used to render the hologram that you see previewing where the block will be placed, rather than actually placing the block there.

The position of this hologram is then sent to the tool script, and when the user places a block, an animation is played moving that block to the position of the hologram.

This has a number of pros and cons. The primary con being the amount of computation required, this position computation is performed multiple times a second to ensure smooth movement of the hologram. While it would be more efficient to not display a hologram and only perform this computation once each time a block is placed, during testing on multiple computers there has been no noticeable performance impact, this however may need to be reviewed if the program were to be run on a lower powered device such as a mobile.

The primary pro is providing a very clear visual example to the user of what they are about to place in the form of the hologram. Another benefit is that different placement mechanics can be applied to the hologram, and the placement of the block will work regardless, for example the SNAP feature, which will be described later.

It's also important to note on the ray feature, that there is an amount of other steps required to ensure successful placement computation. The new block is placed at an arbitrary position along a ray, and a second ray is then sent to determine the thickness. It would be possible if unchecked that a block could be partly obstructing the new block from the existing block, resulting in a false thickness result. This is resolved by using another unity feature of layers, when placing blocks, both the new and existing blocks are moved to a unique layer, the ray is instructed to only recognize blocks in this layer, bypassing and blocks that may be obstructing placement.

Finally while it may have been possible to mathematically work out the correct positioning of the block, given the contact coordinates where it contacts (The user's cursor point) and the scaling of the block to be placed. This was not a skill that CS covered so it was not pursued.

Additional placement mechanics

By applying different placement mechanics to the hologram, that can send coordinates to the placement tool of where the final position of the new block should be, through this it’s possible to completely modify how the user forms constructions.

The primary example of how this is used in this project is the Snap feature, which follows the standard placement of a block described above, but expands on it by also centering the objects position on the existing block, and matching the rotation to the existing block. This effect is shown below.

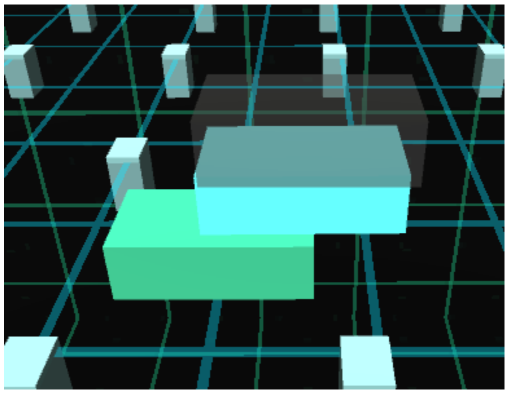


Figure 25 Snap mechanics disabled.

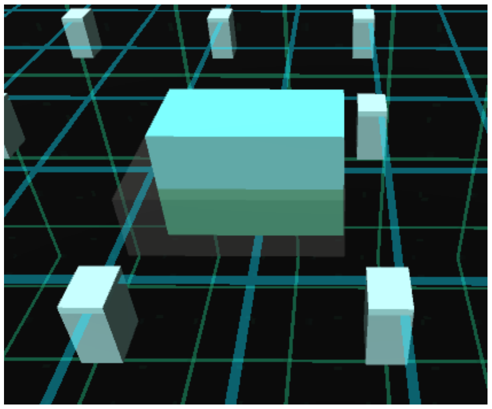


Figure 26 Snap mechanics enabled.

To calculate this, the users ray firstly contacts an existing block. That block then sends out a ray from its center in the normal direction to the contacted surface. A return ray is then sent from an arbitrary point along the blocks center ray, this contacts the block at the center of its outer surface, determining the surfaces central point. The tool script can then continue to perform the placement mechanics, using the center position determined above, rather than the users targeted point.

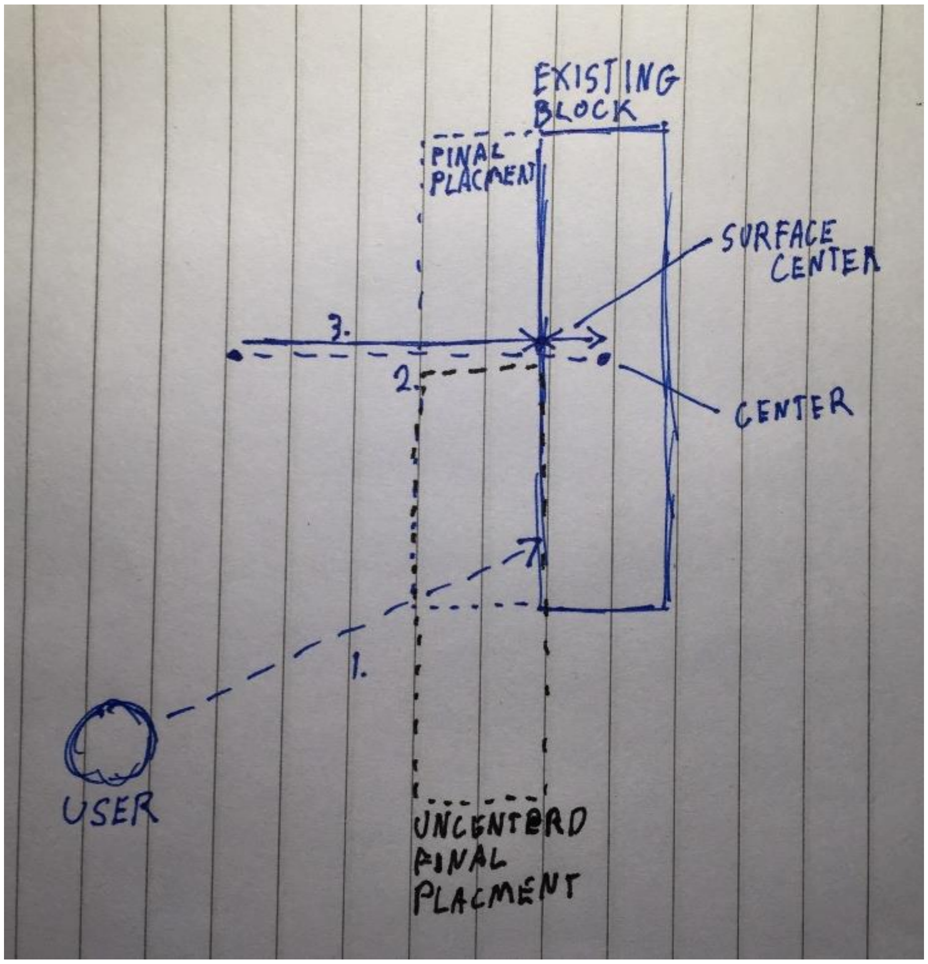


Figure 27 shows adjusted snap placement mechanic.

Features like this can further be expanded upon using other mechanics of the program. For example combining snap placement with physics enabled on blocks can create a more natural placed object look that may be useful to the user in achieving their desired goal.

FURTHER BLOCK MODIFICATIONS

As well as the initial placement features, there is a wide array adjustments that can be made using various keys to modify blocks after placement. While very simple features to implement, they greatly expand upon the range and complexity of what this program can perform.

FINE MOVEMENT AND ROTATION

For movement, the position of the block will first be acquired, that position can then be adjusted by a small amount. The MoveBlock coroutine used to place the block will then be recalled with the new position, this results in a kind of nudge effect allowing the user to fine tune block position.

For rotation, Unity’s rotate feature is called into use with a small rotation degree on the existing block.

SCALE ADJUSTMENT

It is possible for the user to reshape the scale of a block in a particular axis. This works in a similar way to the move feature taking a blocks scale and increasing or decreasing it, Unity’s lerp features was used to create a smooth resize animation. Some axis’s appeared to scale at a faster speed, the reason for this was never determined, however it was corrected by adjusting the scale amount.

CARRYING

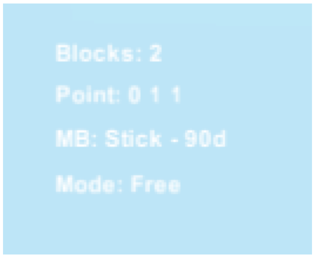
A block can be carried around by the user. This flags a block as the carried block, at each frame that blocks position will me set to the position of the users camera plus a small amount, this means the block will appear to float in front of the user, the carry key can be pressed again to set the block in place.

COPY

A block that has been modified can be copied to the user’s master block and placed down an infinite number of times. This clones the targeted block to the master block, it is cleared when the user changes the master block to another preset.

GUI

The GUI was intended to provide useful information to the user, while remaining as unobtrusive as possible, and as easy to understand as possible. Initially all information was displayed as a debug bar along the side of the screen.



Blocks: 2

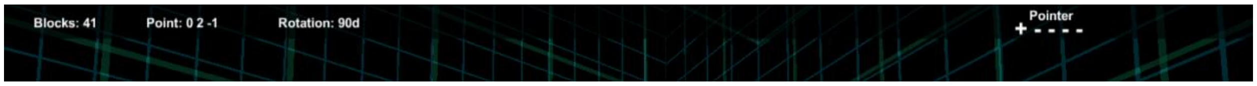
Point: 0 1 1

MB: Stick - 90d

Mode: Free

Figure 28 above shows an early UI implementation

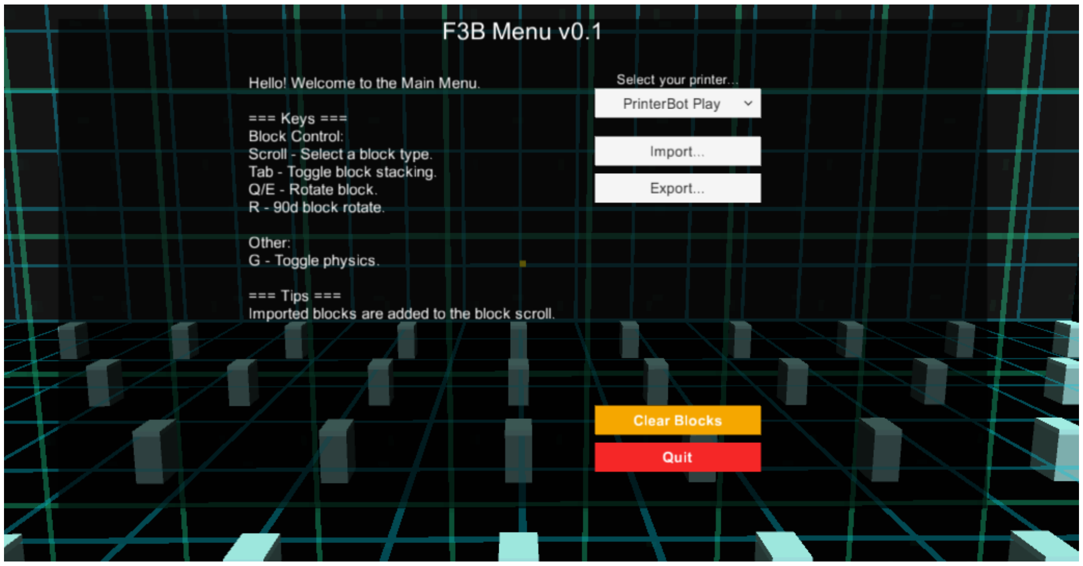
This included a counter for the number of placed blocks, a rounded position marker displaying where the users hologram/pointer was from the center of the stage, an indicator for the Master Block position and a mode indicator.



Blocks: 41 Point: 0 2 -1 Rotation: 90d

Figure 29 above shows the final UI implementation

This information was later moved to a bar along the top, with the addition of a master block selection indicator (Under the label marked Pointer) Different menu screens were also added to perform different functions.



F3B Menu v0.1

Hello! Welcome to the Main Menu.

=== Keys ===
Block Control:
Scroll - Select a block type.
Tab - Toggle block stacking.
Q/E - Rotate block.
R - 90d block rotate.

Other:
G - Toggle physics.

=== Tips ===
Imported blocks are added to the block scroll.

Select your printer...

PrinterBot Play

Figure 30 above shows the final version main menu

Initially the main menu was located on a separate scene from the build area and would load the build area once the user had selected the printer type. This was later determined to be an unnecessary step and the menu was merged into the build scene as shown above in Figure 20.

Menus are managed by keeping all menus loaded in an array, and selecting a single one that will be shown, while hiding the rest, this is refreshed at each frame. While this does hold a slight memory hit, particularly in a future event where many large menus where required, the cost is minimal. The benefit of this is that it's extremely easy to add new menus and continue to have everything managed correctly without having a situation where menus overlap.

The UI also has the ability to show a large onscreen message, this is used to alert the user to settings changes, error messages or completed tasks.

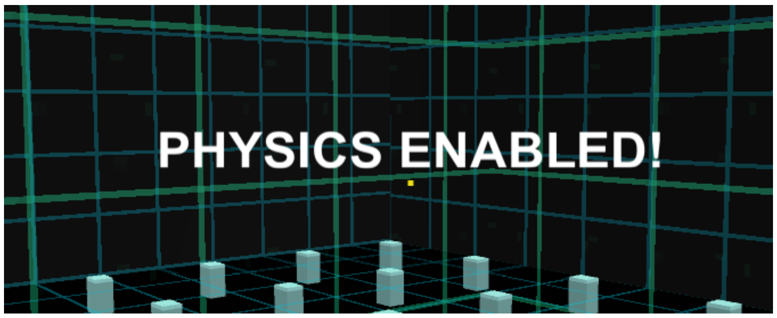


Figure 31 shows onscreen message for physics change.

This is set via a method that can be called by referencing the UI script, which in turn starts a coroutine that displays the desired text for a short amount of time, in most cases two seconds, then sets it back to an empty string.

Initially the plan was to revert it to the text it was previously displaying if one display call was made after another. This resulted in some cases where text would become stuck on screen, so each call now over rights previous ones.

EVALUATION

OVERVIEW

The program was tested at numerous stages throughout development. Each time a feature was added it was tested to find potential flaws in different scenarios. Further testing was also carried out by showing the program at various stages to 3rd parties.

Some key questions considered during these tests were:

- Is the concept and purpose of the program clear to the tester while they are using the program?
- Does the tester's character move in the way that they expect, and does it offer the level of precision and control that the tester desires when building?
- Is the tester having trouble with any particular feature of the program, do any features seem illogical or unintuitive?
- When placing and modifying blocks, does the tester feel they are able to achieve their desired goal, is the program doing what they expect to happen when they perform a placement action?
- How much of a learning curve is the tester experiencing, how long do they need to practice with the program before they feel confident using it?
- Is there any point at which multiple testers appear to be confused?

Another valuable testing opportunity was the presentation day, while small isolated tests could be done during development, presentation day offered a rapid real world test situation where many people of different backgrounds were able to try out the program. Other students were actively asked to attempt to break the project, this process did uncover some errors in the software, which was a valuable bug testing experience. This form of testing also highlighted where some changes to the interface could be made as there were notable points where different people would ask the same question or stumbled over the same feature.

NOTABLE TESTS

USER MOVEMENT

During the first stage of development, Unity's inbuilt character controller was used for user control. A large amount of personal testing and tuning went into attempting to configure it to be accurate enough to place blocks with fine detail, but move fast enough to not be annoyingly slow for the user.

While an equilibrium was reached, further 3rd party testing revealed that the solution was still not satisfactory. They noted that the usage of standard Unity movement in a nonstandard Unity game created a disjointed experience, particularly for people who are familiar with games using the character controller.

The result of this was the development of the custom movement scripts which had not originally been outlined in the design. The final movement scripts performed much more satisfactorily than Unity's character controller, they add exponentially to the quality of the program and would not have come about without 3rd party testing.

BLOCK PLACEMENT

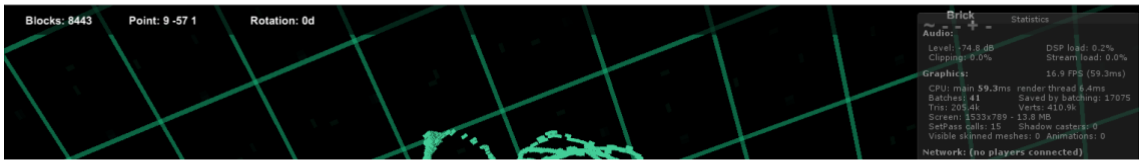
Block placement was consistently tested during its development, each time a modification to code was made that modification required testing and any occurring bugs needed to be evaluated. The placement problem section covers many of these bugs and solutions. As described in that section, the entire placement system was constructed though various stages of trial and error where each solution lead to a new bug.

There were a number of situations where infinite loops were caused by placement checks on blocks that never reached their final location.

PERFORMANCE TESTING

The program was use and stress tested on a desktop system. Stress testing involved instantiating a large number of blocks and noting the frame rate both with and without physics enabled, the test was stopped when the frame rate dipped below 20fps, at a lower frame rate than this the program becomes difficult and jarring to use.

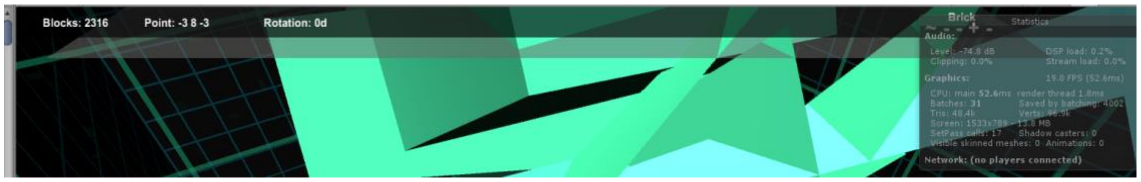
The test was performed by setting the placement tool to constantly place blocks without the need for user input.



```
Blocks: 8443     Point: 9 -57 1     Rotation: 0d

Audio:
  Level: -74.8 dB              DSP load: 0.2%
  Clipping: 0.0%               Stream load: 0.0%
Graphics:                      16.9 FPS (59.3ms)
  CPU: main 59.3ms  render thread 6.4ms
  Batches: 41          Saved by batching: 17075
  Tris: 205.4k         Verts: 410.9k
  Screen: 1533x789 - 13.8 MB
  SetPass calls: 15    Shadow casters: 0
  Visible skinned meshes: 0  Animations: 0
Network: (no players connected)
```

Figure 32 above shows stress testing details with physics disabled.



```
Blocks: 2316     Point: -3 8 -3     Rotation: 0d

Audio:
  Level: -74.8 dB              DSP load: 0.2%
  Clipping: 0.0%               Stream load: 0.0%
Graphics:                      19.0 FPS (52.6ms)
  CPU: main 52.6ms  render thread 1.8ms
  Batches: 31          Saved by batching: 4002
  Tris: 48.4k          Verts: 96.9k
  Screen: 1533x789 - 13.8 MB
  SetPass calls: 17    Shadow casters: 0
  Visible skinned meshes: 0  Animations: 0
Network: (no players connected)
```

Figure 33 above shows stress testing details with physics disabled.

Results of stress test:

	No of blocks before frame rate < 20
With Physics	*2316*
Without Physics	*8443*

During presentation day testing it was found the majority of builds ranged from between 25 – 200 blocks. This number may increase for more complex builds.

Note that these tests were performed on a system with the following configuration:

I7 5820K CPU, 16GB RAM, GeForce GTX 980ti GPU

As a standard usage test the program was run on a laptop during presentation day. No performance change was noticed during that time.

KNOWN ISSUES

There are a number of issues that came to light during the presentation day and further testing.

- If a block is copied with the pointer, the pointer will be resized as if it were a hologram however the user will be unable to place it. This should be resolved by changing the pointer to a hologram before allowing a block copy.
- Passed 8400 blocks during stress testing it appeared that lighting no longer functioned normally and the flight mechanism disabled. This could be an error in the Unity engine, it's also possible the issue is caused due to the player being

put under strain by the constantly spawning blocks used in the test, blocks spawned at a normal rate may not replicate this error.

- There have been some situations where block placement does not re-enable after the user clears all blocks from the menu.
- It's possible for the user to move blocks under the world using the movement keys, room borders are not checked by block movement, if a block is moved outside the room it's not possible for the user to remove or recover it.
- Imported objects have a reduced resolution.
- The file select menu does not appear to highlight selected files and is unable to navigate some sub directories.
- Alert text on screen can overwrite previously displayed alert text, this can cause text to flicker and not display as long as intended.

USABILITY TESTS

During development 3rd parties were frequently asked to give their opinion on the controllability and usability of the program. The goal being to highlight any points that may be unclear and to ensure they understood what they were doing in the program and how it would relate to a real world print.

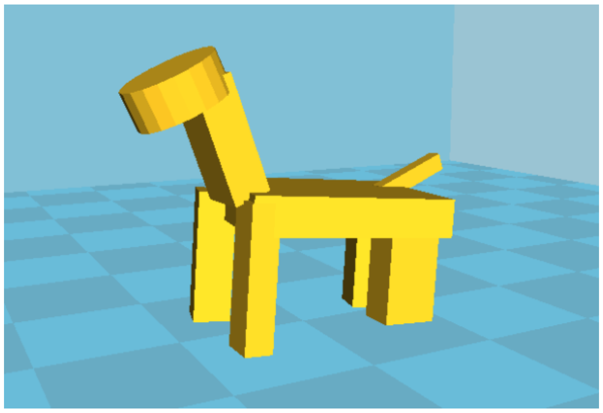


Figure 34 above is a very simple model created on the presentation day in under 3 minutes by a user who had no prior experience with the program.

The user was able to print the model shown above, however due to its vertical height, the printer provided on the presentation day would have significant trouble printing this. The program performed correctly but public knowledge of 3D printing capabilities was still limited.

PRINT TESTING

During development prints were created when new features were added to ensure that those features were producing something that could be replicated by a printer.

Circular shapes, diagonal blocks, varying thicknesses of block. It was also important to ensure that the program was joining blocks properly when printed. Even a small gap between two blocks in the program could result in confusion for the printer and printing software resulting in an unsound print.

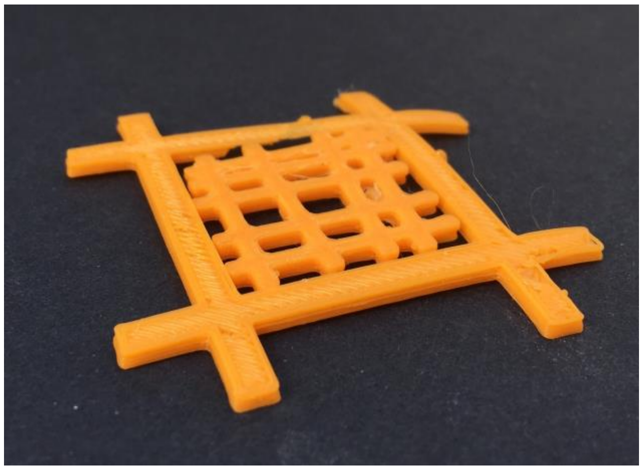


Figure 35 shows an early flat grid printout, testing connections and different scale of blocks.

REVIEW

The project completed all primary goals and its secondary goal of importing objects. The import script is not perfect, however it serves its purpose and acts as a solid test and proof of concept. The building mechanics achieve all goals, there is no set grid, there are multiple placement modes and the user has the ability to modify components. Multiplayer was never implemented, given the way the UI and settings management were implemented there would be a significant amount of reworking required to introduce multiplayer. That said, the blocks, player movement scripts and placement tool, which makes up the bulk of the project would all be convertible to a multiplayer version.

There are a number of bugs and a number of areas that would benefit from more development, however the program meets and excides initial goals.

CONCLUSIONS

OVERVIEW

The program developed is a first step towards creating an alternative approach to digital modeling. It is by no means a CAD replacement, but it does combine the intuitive construction from games with a basic set of tools and scrips that recreate as accurately as possible the experience of building with real world objects. In doing this it allows users to create constructions with little to no skill and allows those constructions to be exported and printed.

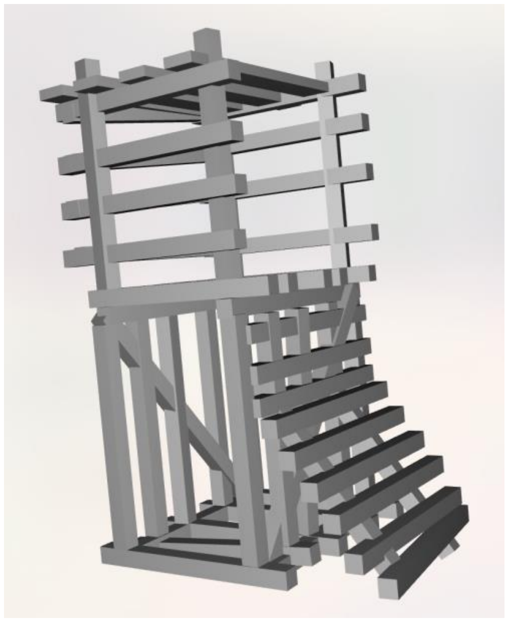


Figure 36 shows an export of a build being previewed in the Mac OSX file previewer.

FUTURE DEVELOPMENT

The placement method which does not depend on a grid, developed during this project has numerous potential applications, it could be used in further development of this or similar software, it could also be included in a future game and could even be included in a CAD suit to expand upon its existing approaches.

As further development I would suggest implementing this placement method in a number of other programs and situations. For example it does not need to be restricted to a first person program, it may function just as well in a mouse program where the user's cursor point acts at the target point from this program.

The level of placement mechanics such as the snap mechanic could also be expanded upon to perform smarter and more exact placements depending on the situation.

The use of modifiable blocks in this program is not something that featured in any of the reviewed games, it would be interesting to review why this is and possibly developed a game that focuses on the concept of warping objects around the player.

An area that would also be interesting to review would be a more detailed study of the physics used in games verses reality, and how they could be modified to produce a more realistic construction simulator. One feature that is so far not possible to simulate is how a person interacts with a wooden block when placing it in order to avoid toppling their construction.

PERSONAL VIEWS

I feel that while this program did successfully complete all of its intended goals, I would not continue further development in its current form. To me the most interesting thing to come out of this project was the placement mechanics, before any further development continued I would focus on expanding and smoothing out that feature. From the outset the importance of this feature and its ability to do something quite different from most other construction games was not at all clear. So it was fascinating to see it rise to the surface during development, that it why so much of the implementation is dedicated to talking about it.

I'm very pleased with the overall program, technically in its functionality, aesthetically in its design and practically in its usage. However I'm more excited for what can be created using the lessons learned in this project.

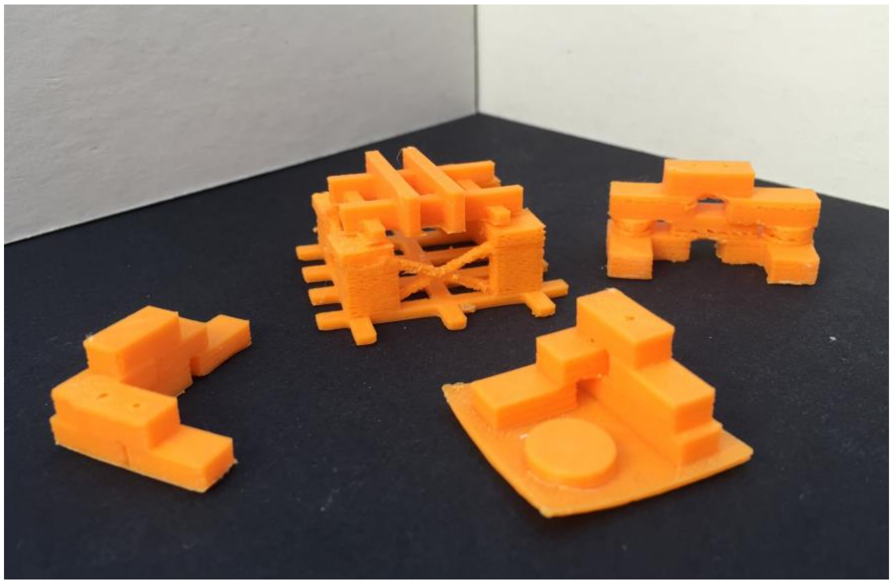
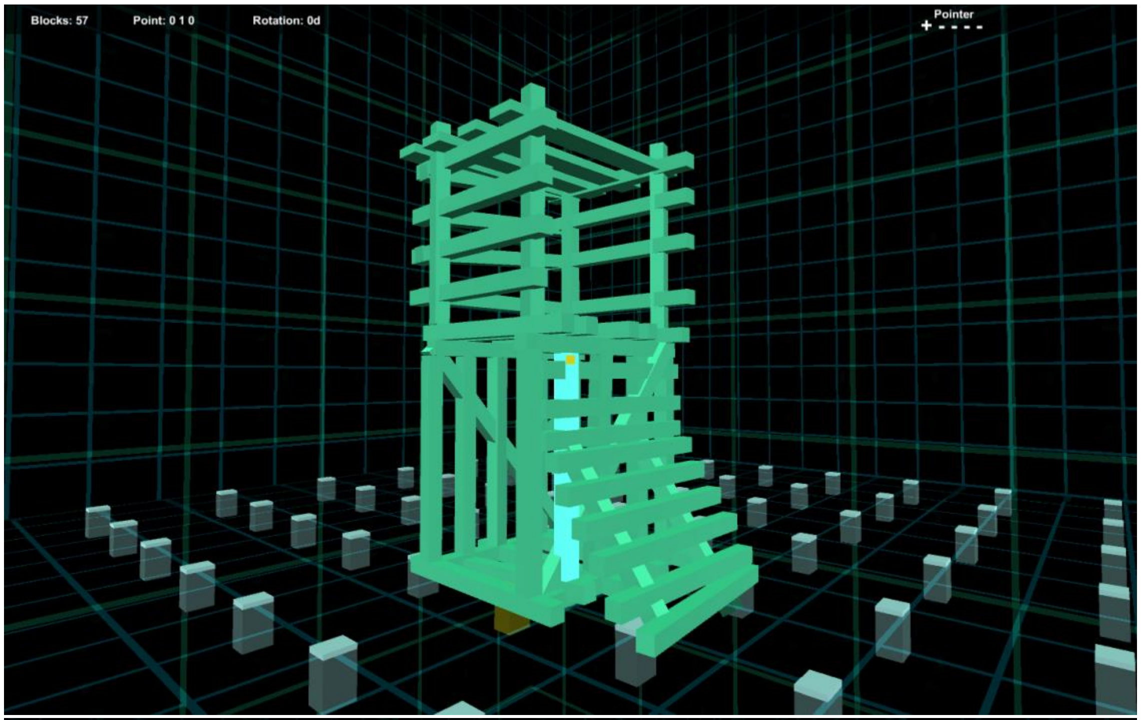
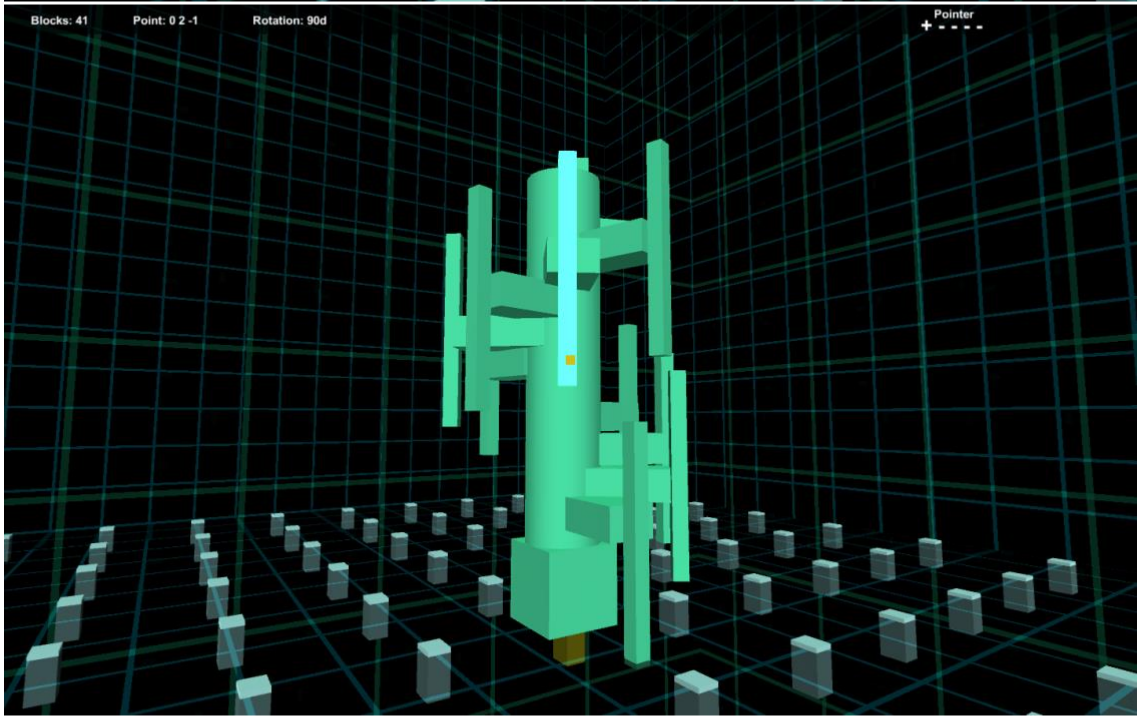


Figure 37 shows a collection of models built in the program and printed during the presentation day.





Figures 38 and 39 show the completed program in use during construction. Note figure 38 shows the construction of the export shown in figure 36.

REFERENCES

3RD PARTY IMAGES

[1] http://www.solidsmack.com/wp-content/uploads/2015/07/maxresdefault.jpg

[2] http://www.localmakers.com/wp-content/uploads/2016/01/news5a.jpg

[3] http://vignette3.wikia.nocookie.net/mysims/images/6/6a/Create-An-Object.png/revision/latest?cb=20090620210730

[4] http://images.akamai.steamusercontent.com/ugc/3298062395266434467/A7A59AE112086B31655BDCB49DBAB4021E43B81C/

[5] https://s-media-cache-ak0.pinimg.com/736x/5e/27/57/5e2757247d8136167fd42f19cecc27db.jpg

CODE SOURCES

Code sources below are not directly referenced but are used in the project.

MeshCombining - http://docs.unity3d.com/ScriptReference/Mesh.CombineMeshes.html

GUILayoutx - http://wiki.unity3d.com/index.php/ImprovedSelectionList

FileBrowser - http://wiki.unity3d.com/index.php?title=ImprovedFileBrowser

ObjImporter - http://wiki.unity3d.com/index.php?title=ObjImporter

ObjFileSelect - http://wiki.unity3d.com/index.php?title=ImprovedFileBrowser

ObjExporterScript - http://wiki.unity3d.com/index.php?title=ExportOBJ and http://wiki.unity3d.com/index.php/ObjExporter

SimpleSmoothedMouseLook - http://forum.unity3d.com/threads/a-free-simple-smooth-mouselook.73117/