Joshua Dempsey

CS-480

Capstone Design

Dr. Bruce Bolden

CS-480 Log Book: XR Assisted Farming Applications

Table of contents:

Preliminary entree:

After the initial project presentations, we have placed our bids and received our respective projects. For this capstone project, I will be on a team composed of Conrad Mearns, Damon Schafer, and Me(Joshua Dempsey). Our client will be Dr Dev Shrestha, and our focus will be on precision agriculture and VR assisted tools. We plan on doing some preliminary work beforehand before we meet with our client to discuss the scope and requirements of the project.

Initial team meeting:

In accordance with convenience and all our respective schedules, we have determined a desirable time slot and location for bi-weekly meetings moving forward, this will allow us to be able to give consistent updates to project progress and maintain a healthy schedule.

We have determined that using discord would be of ample service to us. Discord is a communication app that allows us to have solid communication access between all members of our party and share resources freely.

In addition to allowing us to share resources, we can utilize its conference calling features to schedule remote meetings if need be in addition to chatting / communicating with members of the group directly if an issue arises or new information needs to be shared easily and quickly.

We have set up a small private Discord server for this purpose and have already joined / downloaded the applications on both desktop and mobile.

We have decided to use Github for version control of our project code. This will allow us to review and verify code / files used for the project and allow us to manage files. Conrad has set up a private Github repository and the rest of the members have joined.

Follow-up team meeting:

The present members of this meeting discussed the creation of the required Team Contract document that contains the following information:

Team name and mission:

The name of our team has yet to be determined, and our mission was briefly discussed in the technical presentation at the start of the semester, these will be determined and more fleshed out once we have our initial client meeting and discuss the scope / requirements of the project.

Membership:

The team members are cemented as far as I know and will not change unless some unforeseen circumstance modifies this.

Roles and Responsibilities:

There were several roles and responsibilities discussed at this meeting about handling the project and project resources. These responsibilities include client communications, group meetings, group documentation, and version control administration. We hope to be flexible and understand with our roles and responsibilities to allow for a smoother project

Team Relations / Conflict Resolution

Due to a collective lack of experience, we want to be able to have a very open and forgiving forum for discussion and contemplation on project ideas and potential solutions. We

don’t want any members to feel discouraged and refrain from giving input to the group discussion.

Joint Work / Individual Work

The amount of collaboration and individual work requirements have yet to be decided. We will be able to better actualize this section after we meet with our client and discuss more of the specifics of the project details.

Due to member attendance, we were unable to finish the team contract this meeting and will have to write up a secondary and hopeful more concrete draft once all members have reviewed and discussed the acknowledged information.

Moving forward, we plan on discussing the project details with our client to get a firmer grasp on the requirements and what we will be doing for the rest of the duration of the project.

Damon has elected himself to reach out to Mr Shrestha to inquire about an initial project meeting where all members of the team can attend and discuss project specifics and our mission goals for the future.

Using the initial client meeting, we plan on solidifying the team contract and determine team member roles and responsibilities with more concrete information

Initial client meeting:

Project scope:

The initial version of this project was originally a unity application / extension to allow for VR simulation of operating a combine harvester. The project goals then transitioned in the next term to an already existing game framework called Farming Simulator. This semester, we will be extending the previous team's work in Farming Simulator 2 to help create a mod or extension to meet 3 primary discussed goals / requirements.

Project requirements:

Path finding optimizations

One of the major goals for the project is to develop a path finding algorithm and implementation to allow for the user to generate optimized paths for harvesting / tilling a simple field that consists of a bounded polygon. We will also need to calculate some general resources for a field such as number of turns, time needed, and fuel cost. The path finding requirement should consider several different factors for optimizations.

Time optimizations:

One of the simpler optimization requests is for an algorithm to determine a path that yields the best time. This means naturally that the path with the fewest turns

while maximizing area and minimizing backtrack. Combine harvesters are already operated at a slow speed, but frequent turns amplifies the time needed.

Fuel optimizations:

Another optimization request is for an algorithm that will determine the best path in regards to fuel usage, fuel usage has several impacting factors and requires more research to determine efficiency and average fuel rates of modern combines. Similar to time optimization, frequent turning will lower optimized efficiency.

However, height or altitude changes also greatly affect fuel usage. A combine travelling parallel upwards a slope will burn through fuel at a higher rate than a combine travelling perpendicular to the slope of the hill. More research will need to be done to determine what calculations and equations will suffice to determine fuel usage sacrifices and suboptimal angles.

Soil erosion:

Finally, we need to determine / develop an algorithm to take soil erosion rates into consideration. As a combine operates on a field, it's heavy wheels tear up the soil as it traces its path. Tracing the same path multiple times over the years ruins the soil and creates channels in the field that collect water, hurting potential crop yield and further eroding the soil. We will need to design some way to record and track the location of the combine's tracks and simulate soil displacement / erosion that compounds upon itself with repetition.

We considered using a heat map for this that increases the ‘erosion’ of the soil underneath the combine as it moves along, thus repeating and backtracking places the combine has already visited compounds the erosion. More research will need to be done to learn more about soil erosion rates and how much a combine operates in reality in terms of detrimenting soil quality.

GIS interpolation:

Another project requirement is to fix / improve the previous team’s work with GIS interpolation and map input. The previous team implemented a method of taking GPS data from an existing spreadsheet database of simple field coordinates and heights. Using the GPS data, they construct terrain in Farming Simulator’s game world and can visualize it properly in a VR 3D environment.

However, they were unable to interpolate the data, so the virtualized world does not have slopes, but rough jagged voxelized increases in height very suddenly that damage the experience and usage of a VR system. Our requirement is to interpolate the data somehow to make smoother slopes that can be used more effectively / comfortably.

In addition to interpolation, there exists a more precise set of GPS data that the previous team was unable to get working for in game visualization.

Mr. Shrestha has requested that if we have time and are able to, to improve the system to allow for inputting this newer more precise data. However, this is considered a soft requirement and will be attempted if the project schedule allows.

User documentation:

In addition to developing all project requirements, we will need to consistently provide user documentation for future teams and project goals expanding on our work.

Previous resources:

https://github.com/palouse-agriculture-in-virtual-reality/FS-19-Cook-Farm-Mod

The above link is to a github repository of the previous teams' work on the projects in it's past states. Included also is contact information that we may utilise if we have any questions that we cannot figure out ourselves about the previous teams' implementations.

Project (access, location, procedure)

This project requires us to familiarize ourselves in using and developing for VR. On campus, there exists a small engineering outreach building that has a VR console available for us to use and develop on. In order to gain access to this resource, we need to apply for and receive access cards for unlocking and entering the outreach building where the single console is located.

Mr Shrestha also expressed during the meeting that a weekly schedule for student meetings would be beneficent for us to maintain a healthy schedule and remain productive.

Team meeting:

During this meeting, we reviewed and discussed the project requirements to consolidate and verify that we understood the project goals and what we would need to implement in the coming semesters.

Meeting Location / Time change:

In adherence to Mr Shrestha's scheduling idea, we have agreed to move from a meeting schedule of once every two weeks to a more frequent meeting schedule. Due to some unforeseen circumstances, meeting locations will also be rearranged to better suit our schedules.

Contract Finalization:

All team members were present so we could finalize and electronically sign the team contract after reviewing it's information once more.

Role Delegation:

During the meeting we discussed initial role delegation. We determined that Damon would be our communications manager and would handle future client communications. Conrad elected himself to the role of group documentations which includes recording meeting minutes and attendees. I elected to administer and overview version control so we can reduce as many future conflicts once development officially starts.

Being a very small team, role delegation is somewhat fluid and additional responsibilities are more than likely to be assigned as they present themselves as the project advances.

COVID situation

For some context, the recent developments in the COVID-19 virus pandemic situation have affected how the university operates and subsequently how our team and project have functioned. The university has determined that all classes and meetings will be held online and campus will be closed for the remaining duration of the semester. In order to mirror and adhere to these new circumstances, we have discussed and looked into holding remote meetings through Discord and Zoom.

However, while we can still meet online, the current situation has made us unable to receive our access cards to the engineering outreach research building where the VR equipment is held. The current situation makes the idea of sharing and collaborating with a single VR console somewhat dangerous.

We will attempt to do some self research and work on things we are able to while the situation hopefully dies down and we are able to return to a sense of normalcy and contribute in a more literal sense next semester if the situation allows.

Personal investigation

Previous team's Github

https://github.com/palouse-agriculture-in-virtual-reality/FS-19-Cook-Farm-Mod

Official Farming simulator website:

https://www.farming-simulator.com/

I was unable to procure a link to the legacy versions of farming simulator. We will be using farming simulator 2 which is several versions behind the current product version. It is currently unknown whether we will be provided copies of Farming Simulator 2, or if we will have to purchase personal copies ourselves to familiarize ourselves with the game and modding scene.

Giants editor:

https://gdn.giants-software.com/

The main method of modding farming simulators is done through the giant's editor. The above link takes you to the main website page. However, an account is needed to download the giant's editor itself.

Modding for dummies book:

http://www.cajunwolf.com/fs17/modding/files/FarmingSimulatorModding_en.pdf

The above link contains a friendly user entry level pdf book for the introduction of modding in farming simulator using the giant's editor.

Path finding algorithms:

https://en.wikipedia.org/wiki/Pathfinding

A general purpose wikipedia article on pathfinding and some promising algorithms.

https://en.wikipedia.org/wiki/A%2a_search_algorithm

The A* algorithm is apparently the golden ticket when it comes to pathfinding algorithms and should be considered as a possible contender if further investigation deems that it meets the requirements and can be utilized to our advantage.

Combine speed on field capacity:

https://store.extension.iastate.edu/Product/Estimating-Field-Capacity-of-Farm-Machines-pdf

From the above pdf, we can determine the average operating speed of combine harvesters during field operations.

Tractor forum:

https://www.tractorbynet.com/forums/attachments/141958-tilling-speed.html

Fuel required for field operations:

https://www.extension.iastate.edu/agdm/crops/pdf/a3-27.pdf

The above pdf has some compiled resources that include average fuel required in combine field operations. We can use these averages to better construct a model for what we should expect fuel costs are.

New Semester Begins

The second capstone semester has officially started which is traditionally when capstone teams begin the majority of work in order to accomplish their project tasks. The previous semester has been particularly crazy due to the current pandemic situations around the world and have changed how we may want to approach things as a team. We will communicate with the client to inquire about how we should safely proceed with the project. We don't believe it to be wise or safe to continuously meet in person and share lab equipment if that can be avoided and hopefully we can address these concerns sooner than later.

Acquiring Hardware

Our original project specifications required us to obtain lab access to an on campus engineering outpost where we would have access to the VR hardware for use in assisting us verifying one of our project specifications. Specifically the project task of GIS interpolation. The previous team's GIS data was somewhat incomplete and when turned into a map interpretation for Farming Simulator 19 produced a very unpleasant VR experience where the created terrain was extremely jagged and caused a very poor user experience. After talking with the client, he agreed to let us take the VR equipment from the lab and keep it with us for the semester if we wished to use it in a safer environment.

Missing Files

While researching and observing the previous team githubs more extensively for a better grasp of what has been accomplished already, we discovered several references to files and mods that were not included in the github repositories itself. After asking the client about this, he informed us that he did indeed have a mysterious hard drive floating around that the previous team's had been using that may have some of the files we were looking for. After getting the client to share the hard drive with us through OneDrive we indeed found several files that were not included with the previous' teams githubs.

In particular, we discovered that the previous team was using a mod called AutoDrive, (this may be useful and should be looked into). While we found several of the files that were being referenced in the github, we did not find any files pertaining to the VorpX mod on this new hard drive or in the github repositories. We may have to purchase and acquire this ourselves in the future to allow us to actually use the VR equipment we acquired.

Proper Game Acquisition

The time has come to properly acquire the correct version of the game. After some miscommunication with the client (At the start of the project, I had believed we would be using “Farming Simulator 2”, however after some research and clarification from using the previous team’s github repository, we deduced that we should be using Farming Simulator 2019, Farming Simulator 2 doesn’t actually exist) we asked the client about getting a proper copy of the game. The client provided us a physical DVD copy of the game that came with an activation code.

However, our team quickly came to the realization that none of our members have machines that actually optical DVD drives so we attempted to use the code instead. From this, we discovered that the code had already been activated (most likely by the previous team) and was no longer usable. We decided to create a communal steam account and shared the credentials amongst ourselves and purchased a copy ourselves.

Giants Editor

Our team finally has access to the right version of the game. After verifying that it is installed and runs properly on our respective operating systems, we went about acquiring the Giants Editor. The Giant's Editor is a helpful utility that allows people to make, test, and package mods for Farming Simulator 19. After creating a GDN (Giants developer network) account, it's as straightforward as downloading and installing the software.

Note: The Giants Editor is a windows exclusive piece of software, however I personally as well as other members of my team prefer to run a linux distribution. So long as we can get Giants Editor running properly and functioning without issue, we will continue development on our respective operating systems. However, If we run into issues that cannot be circumvented due to glaring differences in our systems, we have agreed to use Windows 10 either through dual boot or virtual machine.

The only slight difference that needed to be taken into account was some path differences in where the default installation of Giants Editor looks to find your copy of Farming Simulator. After editing the settings of the software and pointing it properly towards where it needs to go, it appears to be working properly.

AutoDrive Mod

From the previous team github repository. We discovered several mentions, and screenshots to a mod called AutoDrive. This peaked my interest and I have decided to explore it more thoroughly. I believe using this mod would be very useful for the path finding / soil erosion project specifications by allowing us to automate tractor operations and test things more rigorously. I have decided to download the latest available version of the mod from the author's github and create a small write up for the mods usage, limitations, and problems I had with it.

A Quick AutoDrive Writeup

The AutoDrive mod is a farming simulator 19 mod that allows the user to create a network of interconnected nodes that enable them to create a path that can be followed to the best of any in game's abilities.

There is also an AutoDrive course editor that I have not explored but if I think it sounds useful, there might be a similar writeup for that.

Acquisition:

It can be downloaded for free from the author's github and the following link https://github.com/Stephan-S/FS19_AutoDrive

Installation:

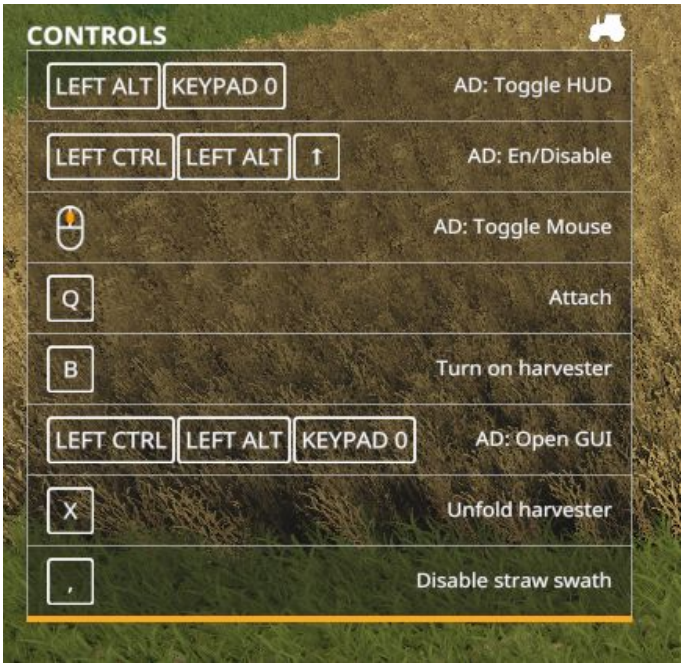
Installation of the auto drive mod (and all farming simulator mods) is pretty straightforward. You simply need to place the zipped document into Farming Simulator 19's mod directory. On Windows this is extremely straightforward and is located in the 'MyDocuments/MyGames'/FarmingSimulator/' directory.

I prefer to run linux on my system and play steam games through Proton/SteamPlay so my installation path is more obscured and my mods folder is located deep within the local steam directories but installation works identically, simply place the zip in the mods folder.

Usage:

Upon launching the game and navigating the menus to enter a game session, you will be prompted to select what properly installed mods to include in this play session, here you can activate or deactivate the AutoDrive mod.

Upon entering a vehicle, the controls menu for that vehicle is displayed by default. The shortcuts with the labels prefixed with ‘AD’ are AutoDrive specific controls

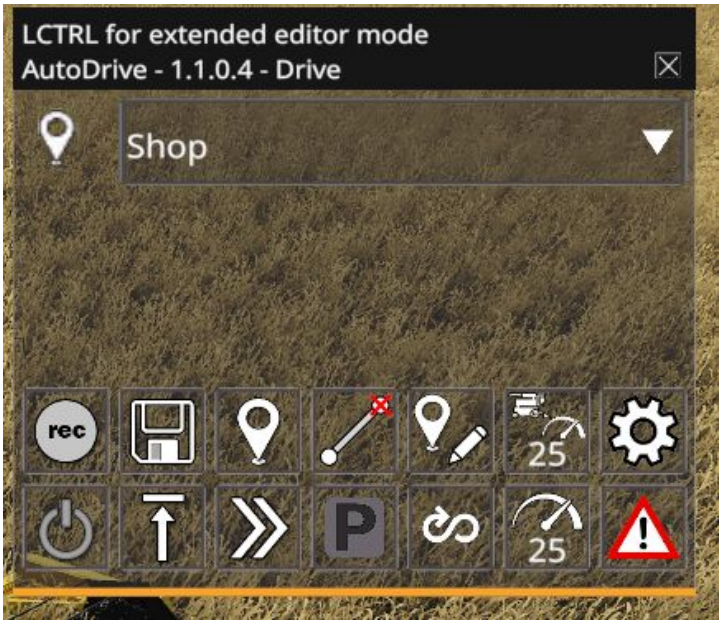


To actually create and use paths, the HUD will be needed to be activated with L-ALT + NUMPAD-0.

The HUD itself is fairly small but can be rescaled and/or set to wide mode in the AutoDrive GUI (which is a settings menu). You will also need to enable the mouse pointer on screen to be able to press the buttons using the middle mouse button.

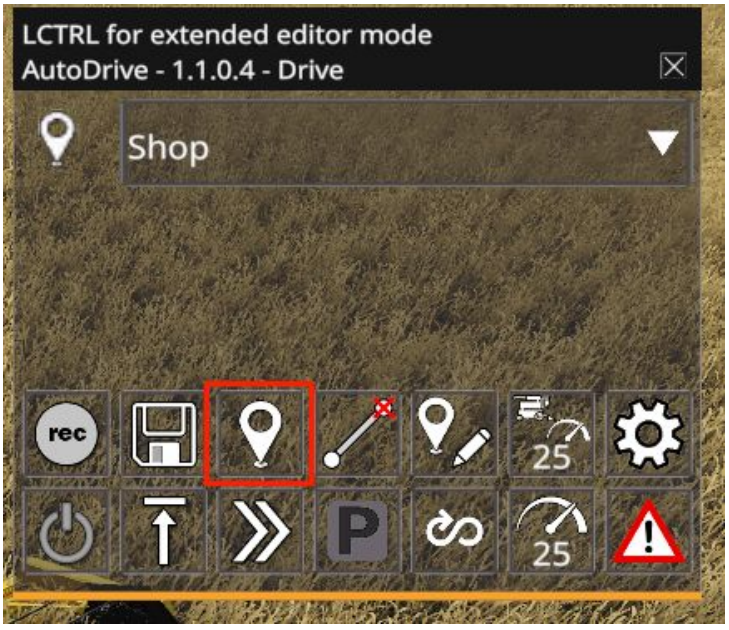
If you're playing one of the original vanilla maps that come packaged with the game, there are already a list of target locations and routes created by the mod developers, this list of routes and your created routes can be seen at the top of the panel.

To create routes you will need to press the 'Edit mode' button to expand the HUD menu. This is the bottom right triangle button with an exclamation point.



An Autodrive route requires two things to function: A set of connected points to create a circuit that can be used to track and drive along, and a Target location marker designating what route AD should be following.

To create a target, use this button and name it with the following popup.



You can place points in a couple different ways. You can place them by recording your vehicle's path as you drive through the record button (the one labelled 'rec') or manually.

To place points on the map you can press L-CTRL and left click with the mouse, while holding control. By holding L-CTRL and holding the right click you can move existing points. Deleting points can be done by holding L-CTRL + L-ALT and left clicking.

If points are not being connected properly you can connect them manually.
To connect points, while holding L-CTRL, click an existing point, it will highlight green, then click a different point.

A path is considered complete only when it connects back to itself. So make sure your path connects back to its start creating a euler circuit preferably. More complicated paths with multiple targets, branches, and various goodies are possible but my understanding of the mod is lacking.

Once a complete path and a proper target destination for your path are set, you can save your path with the save button.

To actually start the autodrive mod to drive the path, make sure the target you created is selected from the dropdown menu and press the power button. Hopefully nothing goes wrong and the tractor starts driving along the lines you've drawn.

Notes:

Autodrive's default behavior to obstacles or items in a path's way is to give up and stop dead in its tracks.

Unconnected nodes in a path will display a red 'X' above their point.

If you have messed up some part of your path or an element of the path creation process and attempt to drive on an incomplete or incompatible path, a small error message about not being able to reach the target will be displayed.

FS mod installation

As we are making a mod for farming simulator. It is important that we understand how to properly install mods and eventually our own into the game. Fortunately, the installation process is pretty straightforward and already mentioned more in depth in the installation section of the AutoDrive mod write up. But once again briefly, simple drag the zipped mod folder into the Mod directory for Farming Simulator.

Note: On Windows this is extremely easy and is located in your My Documents folder. On my system the path was much longer as it was obfuscated and trapped deep within local steam folders. My installation directory turned out to be the abhorrently long:

```
'/home/_user_/.var/app/com.valvesoftware.Steam/.local/share/Steam/steamapps/compatdata/787860/pfx/drive_c/users/steamuser/My Documents/My Games/FarmingSimulator2019/mods'
```

Our team unanimously agreed that we should create a very simple HelloWorld mod that we can properly install and verify running in the game to learn more about Farming Simulator 19 mods and LUA in general.

In particular FS19 mods have a specific structure to them and must include some XML specifications.

FS19 mods are a zip file containing their code and a file marked modDesc.xml that is used to properly install and load the mod's contents into the game:

A simple modDesc.XML file:

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<modDesc descVersion="43">

    <author>conrad</author>
    <version>1.0.0.2</version>

    <title>
        <en>TestMod</en>
    </title>

    <description>
        <en>This is a test mod.</en>
    </description>


    <!--<iconFilename>icon.png</iconFilename>-->

    <specializations>
        <specialization name="myTest" className="TestScript"
filename="test.lua" />

    </specializations>

</modDesc>
```

The XML file breakdown:

```
<author>conrad</author>
   <version>1.0.0.2</version>
```

The XML file has fields for authors and versions, these appear in game under the mod selection menu when entering a game map instance and FS19 prompts you what mods you currently have properly installed in your game to include or exclude.

```
<title>
       <en>TestMod</en>
   </title>

   <description>
       <en>This is a test mod.</en>
   </description>
```

The title and description fields allow you to name and describe the contents of your mod. These will appear in the mod selection menu.

```
<!--<iconFilename>icon.png</iconFilename>-->
```

Currently commented out, but an icon image can be used to visually differentiate your mod in the selection menu in addition to its title and version

```
<specializations>
       <specialization name="myTest" className="TestScript"
filename="test.lua" />

   </specializations>
```

The specializations field contains a necessary list of your lua files. A specialization entry is needed in this field for each file that is used by the mod in order for FS19 to properly load it’s contents.

The test.lua file:

```
TestScript = {}; -- This is a Lua table

function TestScript:cmdTest(args) -- this how how we OOP
   if args ~= nil then
       print(type(args))
       return "args are "..args;
   end;
   return "Error: no arguments specified. Usage: test args";
end;

function TestScript:cmdHello()
   print("Hello!");
   return "World!";
end;

--                FS CMD    HelpText     Func        Table
addConsoleCommand("test", "View args.", "cmdTest", TestScript);
addConsoleCommand("hello", "Print a hello world statement.", "cmdHello",
TestScript);
```

Test.lua breakdown:

```
TestScript = {} -- This is a Lua table
```

Lua features tables as one of it's most used data structures. A lua table can specify a collection of objects, functions, or entries. In this case, FS19 uses a table by its name to identify functions and execute code belonging to that table:

```
function TestScript:cmdTest(args) -- this how how we OOP
   if args ~= nil then
       print(type(args))
       return "args are "..args
   end
   return "Error: no arguments specified. Usage: test args"
end
```

A simple function definition in Lua.

Some notes:

Lua uses ':' as a specifying operator. Here it is used to denote that the function cmdTest belongs to the TestScript table.

Lua uses ~= instead of !=

Lua uses nil instead of null

The operator .. allows us to use an indeterminate number of items

```
function TestScript:cmdHello()
   print("Hello!")
   return "World!"
end
```

A very basic hello world function

```
--                FS CMD    HelpText     Func        Table
addConsoleCommand("test", "View args.", "cmdTest", TestScript)
```

```
addConsoleCommand("hello", "Print a hello world statement.", "cmdHello",
TestScript)
```

The above snippet is FS19 specific code. It allows us to take code that we have written, and so long as it is attached properly to a Lua table, can be assigned to a console command to be executed in game.

Testing the HelloWorld mod:

Accessing the developer console:

In order to execute console commands, we need to enable developer console access for our installation of Farming Simulator. To do this, we need to modify the game.xml located in the base directory for the game's installation.

The game.xml file:

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<game>
   <language>-1</language>
   <graphic>
       <display>
           <width>1600</width>
           <height>900</height>
           <gamma>1</gamma>
           <fullscreen>false</fullscreen>
           <vsync adaptive="true">true</vsync>
           <userConfirmed>true</userConfirmed>
           <previousWidth>1366</previousWidth>
           <previousHeight>768</previousHeight>
       </display>
       <scalability>
           <gpuPerformanceClass>auto</gpuPerformanceClass>
           <performanceClass>Very High</performanceClass>
       </scalability>
       <renderer>D3D_11</renderer>
   </graphic>
   <audio enable="true" volume="0.200000"/>
   <input>
       <joystick enable="true" vibration="false" deadzone="0.14"/>
       <mouse enable="true"/>
```

```
        <keyboard enable="true"/>
        <headTracking active="true" trackir="true" tobiieyex="true"/>
    </input>
    <logging>
        <file enable="true" filename="log.txt"/>
        <console enable="true"/>
    </logging>
    <development>
        <controls>false</controls>
    </development>
    <startMode>6</startMode>
</game>
```

```
 <controls>false</controls>
```

We need to change this to:

```
 <controls>true</controls>
```

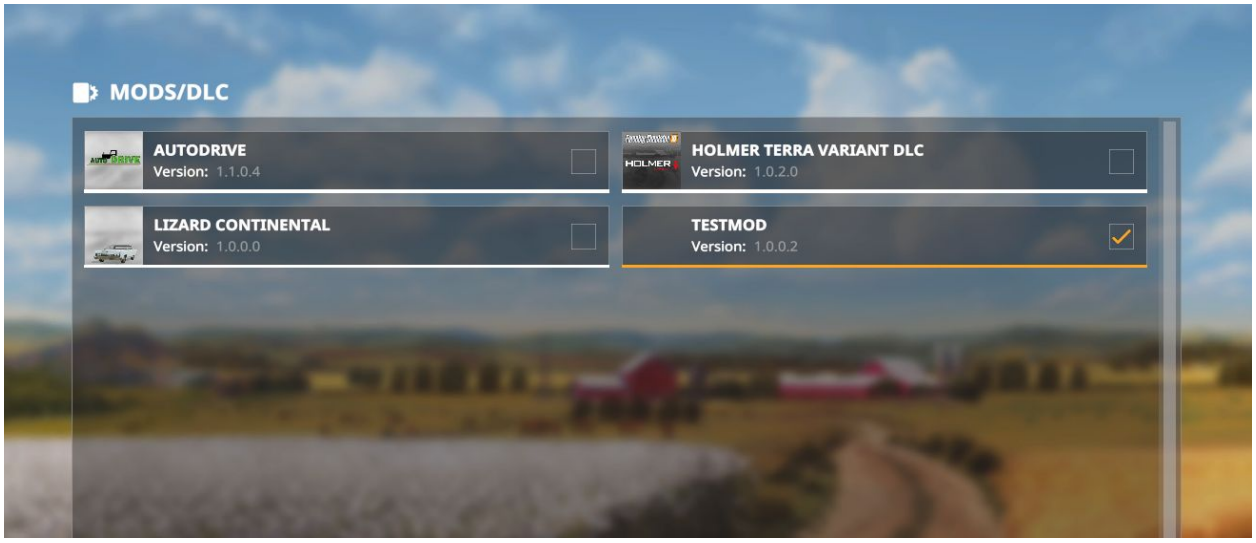
After saving the xml file, we have access to the developer console in-game, we can go about testing our simple test mod.

Testing our Simple mod:

To test the simple HelloWorld mod conrad drafted. We simply need to zip the lua file and mod description file together, and place the zipped file into the proper mod directory.

Now we simply run the game and navigate through the menus to load a map and play.

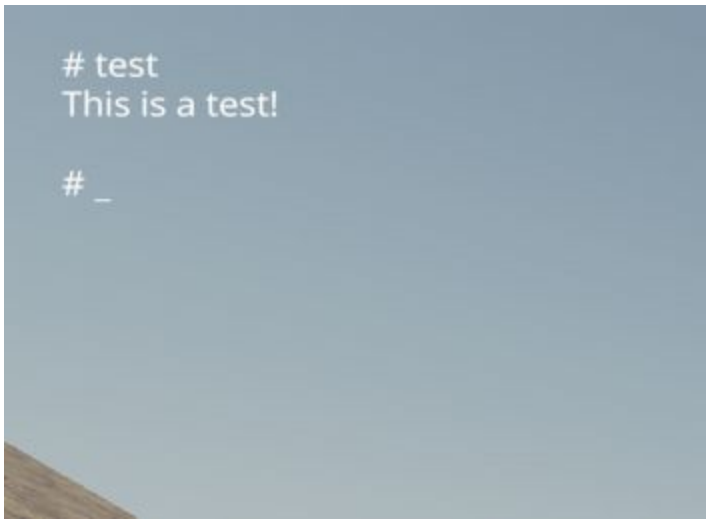
Before we can actually play, we are prompted with the mods/dlc menu



This is where we can verify that our mod description is functioning properly and our zip file is in the proper place.

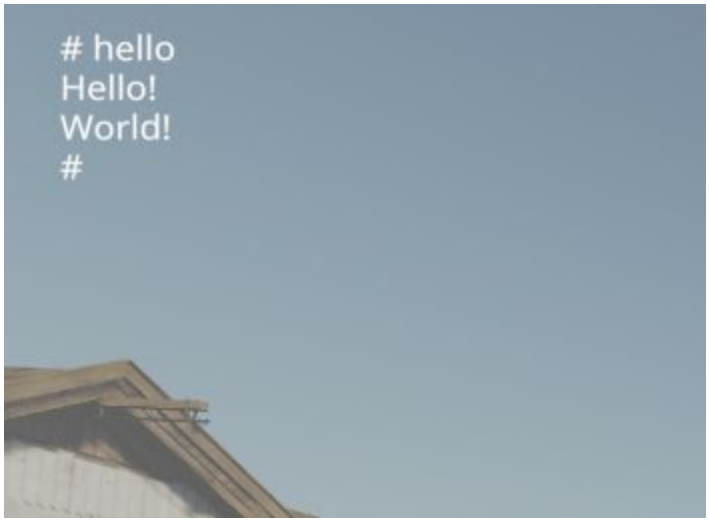
Once we are in the actual game, we can press the ‘`’ key to open the console, and press ‘`’ again to accept input.

Running the test command:



From this we can see that our test command is properly being called and executing its respective code.

Running the hello command:



From this we can see that our hello command is properly being called and executing its respective code.

Intro to Path finding:

A general high level attack plan for path finding:

1. Initially we need to determine the bounds of our ‘field’. From this we construct a set of points that denote the shape of our ‘field’.
2. Using this shape, we need to generate a set of sub points that belong to that field. Each of these points will need to be ‘visited’ in our path finding algorithm to complete a route for our path.
3. We need to denote a starting point for our field, semantically, the ending point can be the same as the starting point, to create a simple closed circuit.
4. For each point we are currently at, we need to determine the best next point to go to.
5. Once we have determined the best point through some selection algorithm (this is where we should optimize for certain things like distance, height differential, and so on), we can traverse to the next point and mark the old as visited.
6. Continue this approach until we run out of points and have a ‘path’ or route denoting what we should follow.
7. We should then use this generated path to create an autodrive compatible network of nodes to visit that we can test and show being driven.

I’ve decided to try and do as much development outside of FS19 directly as possible. The workflow of installing, booting, running, and checking code through FS19 is relatively time expensive and tedious. I think it would be smarter to develop a general Lua compatible algorithm that performs the required task without needing specific game information. Then writing smaller

wrapper functions that allow us to interact with my general code using game specific information should alleviate a lot of headaches. Conrad has done specific research into how FS19 and Lua interact and has discovered that FS19 places some heavy limitations on what we can do with Lua. FS19 specifically sandboxes and disables a lot of io functionality for security reasons.

Code Breakdown and WalkThrough

Lua doesn't directly support OOP inherently, but can be achieved through some clever usage of tables and metatables. The following file is a class.lua file I found online from a Lua OOP tutorial.

Class.lua File:

```
-- class.lua
-- Compatible with Lua 5.1 (not 5.0).
function class(base, init)
   local c = {}    -- a new class instance
   if not init and type(base) == 'function' then
      init = base
      base = nil
   elseif type(base) == 'table' then
    -- our new class is a shallow copy of the base class!
      for i,v in pairs(base) do
         c[i] = v
      end
      c._base = base
   end
   -- the class will be the metatable for all its objects,
   -- and they will look up their methods in it.
   c.__index = c
   -- expose a constructor which can be called by <classname>(<args>)
   local mt = {}
   mt.__call = function(class_tbl, ...)
   local obj = {}
   setmetatable(obj,c)
   if init then
      init(obj,...)
   else
      -- make sure that any stuff from the base class is initialized!
```

```
      if base and base.init then
      base.init(obj, ...)
      end
   end
   return obj
   end
   c.init = init
   c.is_a = function(self, klass)
      local m = getmetatable(self)
      while m do
         if m == klass then return true end
         m = m._base
      end
      return false
   end
   setmetatable(c, mt)
   return c
end
```

Point.lua file:

```
Point = {x = -1, y = -1, z = -1}

function Point:new(o, x,y,z)
   o = o or {}
   setmetatable(o, self)
   self.__index = self

   self.x = x
   self.y = y
   self.z = z

   print("Created point with: ",x, y, z)
```

```
   return o
end

function Point:print()
   print("Point info: [",self.x,", ",self.y,", ",self.z,"]")
end
```

Point.lua breakdown:

```
Point = {x = -1, y = -1, z = -1}
```

Creates a table called Point that has 3 associated fields, an x,y,z entry

Point:new function

```
function Point:new(o, x,y,z)
   o = o or {}
   setmetatable(o, self)
   self.__index = self

   self.x = x
   self.y = y
   self.z = z

   print("Created point with: ",x, y, z)
   return o
end
```

A function modeled after the functionality of the class.lua file

Point:new breakdown

```
 o = o or {}
   setmetatable(o, self)
   self.__index = self
```

Initialization shenanigans that tell Lua to create a new object.

```
   self.x = x
```

```
    self.y = y
    self.z = z
```
Sets the self fields to the arguments passed

```
    print("Created point with: ",x, y, z)
    return o
```
Returns the newly created object o

After some testing, I've found that this Point code does not actually create a new object, but instead overwrites itself and can only have one object of type point at a time. So I've written a simpler approach that works as intended.

A better Point.lua:

```
Point = {}

function CreatePoint(x,y,z)
    local object = {}
    object.x = x
    object.y = y
    object.z = z

    function object:print()
        print("[",object.x,object.y,object.z,"]")
    end

    return object
end
```

Point.lua Breakdown:

Here we make use of the keyword local, this specifies that this table object should not exist outside this scope, using local is how we create lua modules. Similarly to the previous point, we

assign the object fields to the passed arguments. Note that CreatePoint does not belong to any specific table, this is so it can be called from anywhere to create a new point. Similarly note that the print function is contained within the definition of the CreatePoint function, this allows me to create a function that belongs only to members of this type of ‘class’

PolyTest.lua and breakdown:

HelloWorld function:

```
PolyTest = {}

function PolyTest:HelloWorld()
   print("Howdy Partner")
end
```

A simple HelloWorld function for testing

Point Print function:

```
function _pp(p)
   if p == nil then return "nil point" end
  return string.format("(%s,%s,%s)",p.x,p.y,p.z)
end
```

A simple print function to make output slightly nicer when printing points

FindMins function:

```
function FindMins(Poly)
   local numVerts = #Poly
   local minX = 0
   local minZ = 0
   for i = 1, numVerts do

       if(Poly[i].x < minX) then minX = Poly[i].x end
```

```
        if(Poly[i].z < minZ) then minZ = Poly[i].z end
    end
    return minX, minZ
end
```

A function used to determine the minimum points with respect to x and z and return the smallest values.

Note:

```
    local minX = 0
    local minZ = 0
```

Should be changed to something like:

```
    local minX = Poly[1].x
    local minZ = Poly[1].z
```

There is a potential problem with assigning them to zero initially. There are certain cases that exist where there might not be changes.

FindMaxs function

```
function FindMaxs(Poly)
    local numVerts = #Poly
    local maxX = 0
    local maxZ = 0
    for i = 1, numVerts do

        if(Poly[i].x > maxX) then maxX = Poly[i].x end
        if(Poly[i].z > maxZ) then maxZ = Poly[i].z end
    end
    return maxX, maxZ
end
```

Given a set of points, cycle through them all and return the maximum x and z values. Similar to FindMins, the zero initialization should be changed

InPoly function:

```
--[[
   The following function tests whether a point is in a certain polygonal
region in respect to their x,z values.

   Note: The order of the polygon vertices is important, the order must
not be in a sensible order that a proper polygon can be drawn
   i.e.:

   polygon =
   {
       A = 1,0,1
       B = 10,0,1
       C = 10,0,10
       D = 1,0,10
   }
   would be a sensible format as the order constructs a proper polygon
   In respect to x,z
  10 B                  C
   9
   8
   7
   6
   5
   4
   3
   2
   1 A                  D
   0 1 2 3 4 5 6 7 8 9 10
   Would construct a proper polygon
  10 B ---------------> C
   9 ^                  |
   8 |                  |
   7 |                  |
   6 |                  |
```

```
   5 |                  |
   4 |                  |
   3 |                  |
   2 |                  V
   1 A <--------------- D
   0 1 2 3 4 5 6 7 8 9 10
]]
function InPoly(polygon, point)
   local result = false
   local j = #polygon
   for i = 1, #polygon do
       if (polygon[i].z < point.z and polygon[j].z >= point.z or
polygon[j].z < point.z and polygon[i].z >= point.z) then
           if (polygon[i].x + ( point.z - polygon[i].z ) / (polygon[j].z -
polygon[i].z) * (polygon[j].x - polygon[i].x) < point.x) then
               result = not result -- flip the result
               --print("Fwip")
           end
       end
       j = i
   end
   return result
end
```

CreateGrid function:

```
--[[
   Very bruteforce, make grid of points from [minX,minZ] to [maxX,maxZ]
   (y values will be approached later)
]]
function CreateGrid(minX, minZ, maxX, maxZ, delta)
   local result = {} -- Table for holding points generated

   for i = minX, maxX, delta do
       for j = minZ, maxZ, delta do
```

```
            table.insert( result, CreatePoint(i, -9, j) ) -- Create grid
point
        end
    end
    return result
end
```

Creates a grid from mins to maxes increasing by a factor of delta and return the table of points.

GridPointsInPolygon function:

```
function GridPointsInPolygon(polygon, grid)
    local result = {}
    for i = 1, #grid do
        if(InPoly(polygon, grid[i])) then table.insert(result, grid[i]) end
    end
    return result
end
```

Isolate the points belonging to our polygon from our grid, returning only the points that belong in the polygon

Test function:

```
--[[
    Function for testing other functions and object creation
]]
function PolyTest:TestStuff()

    print("TestPoint-")
    TestPoint = CreatePoint(5,5,5)
    TestPoint.print()
    print("----")
```

```
print("TestPolygon-")
TestPolygon = {
    CreatePoint(0,0,0),     --LL
    CreatePoint(0,8,10),    --UL
    CreatePoint(10,11,10),  --UR
    CreatePoint(10,5,0),    --LR
}
for i = 1, #TestPolygon do
    TestPolygon[i].print()
end
print("----")

print('Finding Polygon Mins')
print('Mins: ', FindMins(TestPolygon))

print('Finding Polygon Maxs')
print('Maxs: ', FindMaxs(TestPolygon))

print("Testing is TestPoint is in TestPolygon")
C = InPoly(TestPolygon, TestPoint)
print(C)
print("----")

local minx,minz = FindMins(TestPolygon)
local maxx, maxx = FindMaxs(TestPolygon)

Grid = CreateGrid(minx,minz, maxx, maxx, 1)
print("#Grid Points: ", #Grid)

ConsideredPoints = GridPointsInPolygon(TestPolygon, Grid)
print("#Considered Points:", #ConsideredPoints)
for i = 1, #ConsideredPoints do print(_pp(ConsideredPoints[i])) end
print("--------------------.")

Path = GeneratePathFromPoints(ConsideredPoints)

print("Printing Selected Path:")
```

```
   print("Start.")
   print("#PATH,",#Path)
   for i = 1, #Path do
       print(string.format("%s]\t%s",i, _pp(Path[i])))
   end
   print("End.")
End
```

A testing function to create some points, create a polygon, test whether a point is in a polygon, create a grid from those polygons, and generate a path from the considered points

GeneratePathFromPoints function:

```
--[[
   Function that takes a set of points to consider, and from those points
returns an ordered path to follow.
   Treats first point as starting/ending point

   Some Finding algorithms to consider:
   Simple:
       Shortest x-z distance first (easy, quick, pick a point and go to
it)
       Straight lines (minimizes turns) by placing bias on direction
selection

   Not so Simple:
       Shortest local height distance first (optimize purely against
height)
           Will require some threshold to avoid jumping across map from
local hills
   ]]
function GeneratePathFromPoints(ConsideredPoints)

   print("Generating Path from Received Points")
```

```
StartPoint = ConsideredPoints[1]
--StartPoint.print()
EndPoint = ConsideredPoints[1]
--EndPoint.print()

Path = {}
Remaining = ConsideredPoints
Add(Path, StartPoint)
Remove(Remaining, StartPoint) -- Remove From remaining

Current = StartPoint -- Set CurrentPoint to Start Point

-- . . . Find path
for i = 1, #ConsideredPoints-1 do
    print(string.format("%s]---",i))
    print("I am at:",_pp(Current))
    -- Select Point
    NextPoint = SelectPoint(Current, Remaining)

    if NextPoint == nil then
        print("Next Point is nil, breaking")
        Add(Path, EndPoint)

        print("Returning path of [",#Path,"]")
        return Path -- Return the calculated path
    end

    -- Add point to path
    Add(Path, NextPoint)
    -- Set Current point
    print("Going to:", _pp(NextPoint))
    Current = NextPoint
    print("---.")
end

Add(Path, EndPoint)
print("Returning path")
```

```
    return Path -- Return the calculated path
end
```

Create an empty table to represent our path. Add the start point to the path. Remove the start point from the remaining points. Loop through all the considered points needed while selecting our next point, adding it to the path and replacing our current point.

Remove function:

```
--[[
    This function is very bad. Dont assert indices of a LUA table as nil
especially through iteration, create a better function
]]
function Remove(t, point)
    print("\tRemoving point from table:", _pp(point))
    --print("#Table", #t)
    local _i = 1
    for i = 1, #t do
        -- print("Searching table")
        if t[i] ~= nil and t[i].x == point.x and t[i].y == point.y and
t[i].z == point.z then
            _i = i
        end
    end
    t[_i] = nil
    print("\tPoint removed.")
end
```

This function should be reworked into something more palpable.

Add function:

```
function Add(t, point)
    print("\tAdding point to path:", _pp(point))
```

```
   table.insert( t, point)
end
```
Simply prints the point and adds it to a passed table.

SelectPoint function:

```
function SelectPoint(current, remaining)
   print("Selecting a new Point")

   BestPoint = current

   -- Remove From remaining
   print("Removing current")
   Remove(Remaining, current)

   print("Selecting point with minXZ Distance.")
   BestPoint = SelectMinXZDist(current, remaining)

   print("Current Point:", _pp(current))
   print("Best Point found:", _pp(BestPoint))
   return BestPoint
end
```
Select a new point by claiming that the best point is the current, remove the current point from the list of remaining points, and acquire the best point from some selection function, in the above case SelectMinXZDist

SelectMinXZDist function:

```
function SelectMinXZDist(point, list)
   minDist = 9999999
   minPoint = nil
   for i = 1, #list do
```

```
      if list[i] == nil then goto continue end -- Found a nil entry
(possibly removed)

      _d = XZDistance(point, list[i])
      if(_d < minDist) then
          minDist = _d
          minPoint = list[i]

      end

      ::continue:: -- goto block to simulate continue for selection loop
   end

   if minPoint == nil then print("Returning nil point from
SelectMinXZDist") end
   return minPoint
end
```

Loop through all non nil entries of a table and compare their XZDistance with the minDist,

update accordingly and return the minPoint

XZDistance function:

```
--[[
   Euclidean distance formula in respect to point's x and z coords
]]
function XZDistance(a,b)
   return math.sqrt(math.pow(b.x - a.x,2) + math.pow(b.z - a.z,2))
end
```

```
--[[
   Following code is for testing bits and bobs in LUA~land outside of
Farming Simulator to verify code functions before having to make, move,
install, launch, and load mod for FS
   require("Point")
   PolyTest.TestStuff()
```

```
]]
```

The above snippet is for testing things outside of the FS19 environment using the require keyword. Require does not work in FS19 but must be used if executing directly through Lua 5.3

PathFind.lua:

The functionality of PolyTest.lua has been moved to PathFind.lua whose code will be posted momentarily. It will not have a line by line breakdown due to personal laziness and the existence of comments throughout the code. The searching function was also changed to resolve a nasty bug that had to do with nil indices in the list of remaining points that were candidates for selection. The program at some point ran into enough nil points it decided to stop the path generation and returned an incomplete list. This was changed so that instead of removing something from the list, a boolean flag tied to the corresponding point was set and used to exclude from selection after being previously chosen.

PathFind.lua Code:

```
PathFind = {}

--[[
   Simple function for printing point data in a slightly nicer looking
format
]]
function _pp(p)
   if p == nil then return "nil point" end
  return string.format("(%s,%s,%s)",p.x,p.y,p.z)
end

--[[
   Add Point to end of table.
]]
```

```
function Add(t, point)
   print("\tAdding point to path:", _pp(point))
   table.insert( t, point)
end

--[[
   Find and set a point's visited flag as true, the visited flag is used
in generating paths to exclude points for selection, as they've already
been Selected before
]]
function MarkAsVisited(t, point)

   print(string.format( "\tMarking %s as visited", _pp(point) ))
   for i = 1, #t do
       -- print("Searching table")
       if t[i] ~= nil and t[i].x == point.x and t[i].y == point.y and
t[i].z == point.z then
           t[i].visited = true
       end
   end
end

--[[
   Given a set of points to consider, all which need to be visited and
trating the first point as the start and end point, create an ordered list
of points that
   is considered a 'path' given some Selection function or optimization
]]
function GeneratePath(ConsideredPoints)
   print("Trying to make a Path from Recieved Points")
   StartPoint = ConsideredPoints[1]
   EndPoint = ConsideredPoints[1]

   Path = {}
   Remaining = ConsideredPoints
   Add(Path, StartPoint)
   Current = StartPoint -- Set CurrentPoint to Start Point
```

```
    -- . . . Find path
    for i = 1, #ConsideredPoints do
        print(string.format("Step %s]---",i))
        print("I am at: ",_pp(Current))

        -- Select Point
        NextPoint = SelectPoint(Current, Remaining)

        -- Add point to path
        Add(Path, NextPoint)

        -- Set Current point
        print("Going to:", _pp(NextPoint))
        Current = NextPoint
        print("---.")
    end

    Add(Path, EndPoint)
    --table.insert( Path, EndPoint) -- Add ending Point to Table (Also
starting point)
    print("Returning path")
    return Path -- Return the calculated path
end

--[[
    Selects point by using Euclidian distance formula in respect to point's
x and z coords
]]
function SelectPoint(current, remaining)
    print("\tSelecting a new Point")

    BestPoint = current

    -- Mark current
    MarkAsVisited(remaining, current)

    print("\tSelecting point with minXZ Distance.")
```

```
    BestPoint = SelectMinXZDist(current, remaining)

    print("\tCurrent Point:", _pp(current))
    print("\tBest Point found:", _pp(BestPoint))

    return BestPoint
end

--[[
    Iterate through a list and determine what point best minimizes travel
with respect to X and Z axis
]]
function SelectMinXZDist(point, list)
    minDist = 9999999
    minPoint = nil

    for i = 1, #list do
        --print("BOOL: ", list[i].visited)
        if list[i].visited == true then goto continue end -- Found a
visited entry (don't select)

        _d = XZDistance(point, list[i])
        if(_d < minDist) then
            minDist = _d
            minPoint = list[i]
        end

        ::continue:: -- goto block to simulate continue for selection loop
    end

    if minPoint == nil then print("Returning nil point from
SelectMinXZDist") end
    return minPoint
end

--[[
    Euclidian distance formula
```

```
]]
function XZDistance(a,b)
   return math.sqrt(math.pow(b.x - a.x,2) + math.pow(b.z - a.z,2))
end


--[[
   Given a set of points, cycle through them all and return the minimum x
and z values
]]
function FindMins(Poly)
   local numVerts = #Poly
   local minX = Poly[1].x
   local minZ = Poly[1].z
   for i = 1, numVerts do

       if(Poly[i].x < minX) then minX = Poly[i].x end
       if(Poly[i].z < minZ) then minZ = Poly[i].z end
   end
   return minX, minZ
end

--[[
   Given a set of points, cycle through them all and return the maximum x
and z values
]]
function FindMaxs(Poly)
   local numVerts = #Poly
   local maxX = 0
   local maxZ = 0
   for i = 1, numVerts do

       if(Poly[i].x > maxX) then maxX = Poly[i].x end
       if(Poly[i].z > maxZ) then maxZ = Poly[i].z end
   end
   return maxX, maxZ
end
```

```
--[[
   Very bruteforce, make grid of points from [minX,minZ] to [maxX,maxZ]
   (y values will be approached later)
]]
function CreateGrid(minX, minZ, maxX, maxZ, delta)
   local result = {} -- Table for holding points generated

   for i = minX, maxX, delta do
       for j = minZ, maxZ, delta do
           table.insert( result, CreatePoint(i, -9, j) ) -- Create grid
point
       end
   end
   return result
end

--[[
   Function determines what points given a grid lie within the bounds of
the given polygon, returns the list of grid points within the polygon
]]
function GridPointsInPolygon(polygon, grid)
   local result = {}
   for i = 1, #grid do
       if(InPoly(polygon, grid[i])) then table.insert(result, grid[i]) end
   end
   return result
end

--[[
   The following function tests whether a point is in a certain polygonal
region in respect to their x,z values.

   Note: The order of the polygon verticies is important, the order must
not be in a sensible order that a proper polygon can be drawn
   i.e.:
```

```
   polygon =
   {
       A = 1,0,1
       B = 10,0,1
       C = 10,0,10
       D = 1,0,10
   }
   would be a sensible format as the order constructs a proper polygon
   In respect to x,z
  10 B                  C
   9
   8
   7
   6
   5
   4
   3
   2
   1 A                  D
   0 1 2 3 4 5 6 7 8 9 10
   Would construct a proper polygon
  10 B ---------------> C
   9 ^                  |
   8 |                  |
   7 |                  |
   6 |                  |
   5 |                  |
   4 |                  |
   3 |                  |
   2 |                  V
   1 A <--------------- D
   0 1 2 3 4 5 6 7 8 9 10
]]
function InPoly(polygon, point)
   local result = false
   local j = #polygon
   for i = 1, #polygon do
```

```
       if (polygon[i].z < point.z and polygon[j].z >= point.z or
polygon[j].z < point.z and polygon[i].z >= point.z) then
           if (polygon[i].x + ( point.z - polygon[i].z ) / (polygon[j].z -
polygon[i].z) * (polygon[j].x - polygon[i].x) < point.x) then
               result = not result -- flip the result
               --print("Fwip")
           end
       end
       j = i
   end
   return result
end

--[[
   Slightly nicer way to print lots of path points that doesn't flood
screen as much
]]
_NL = 8 -- number of points to print before going to a new line
function PathFind:PrintPath(list)
   print("--------------------------------")
   str = "START |--\n"
   for i =1, #list do
       str = str .. string.format( "\t%s -- >\t",_pp(Path[i]))
       if i % _NL == 0 then
           print(str)
           str = "\n"
       end
   end
   str = str .. "\t--| END"
   print(str)
   print("--------------------------------")
   print(string.format("Done printing path of [%s] points.", #list))
end

PolyGon = {} -- Global PolyGon table for storing point data for path
creation
--[[
```

```
   Function that empties the PolyGon table.
]]
function PathFind:ClearPoly()
   print("Cleared any previous entries in PolyGon")
   ClearTable(PolyGon)
   for k in pairs (PolyGon) do print(PolyGon[k]) end
   print("If you saw any numbers above here, we had a problem.")
end

--[[
   Wrapper function that adds a point to the PolyGon table
]]
function PathFind:AddPoint(_x,_y,_z)
   table.insert(PolyGon, CreatePoint(_x,_y,_z))
   print("Point Added: ", _pp(PolyGon[#PolyGon]))
end

--[[
   Creates a set of grid points and returns them.
]]
DELTA_LIM = 32 --This number is the divisor used to determine the delta
size for the space between points to consider
function GetGrid()
   --Get mins and maxs
   local minx,minz = FindMins(PolyGon)
   local maxx, maxz = FindMaxs(PolyGon)

   x_diff = maxx - minx
   z_diff = maxz - minz
   min_diff = math.min(x_diff, z_diff)
   a = (min_diff) / DELTA_LIM
   delta = math.max(1, math.floor(a+0.5))

   --[[
       print("x_diff:", x_diff)
       print("z_diff:", z_diff)
       print("min_diff:", min_diff)
```

```
        print("delta:", delta)
    ]]

    --Get Grid points
    Grid = CreateGrid(minx,minz, maxx, maxz, delta)
    print("#Grid Points: ", #Grid)
    return Grid
end

--[[
    Wrapper Function for creating and returning a path using the PolGon
table
]]
function PathFind:MakePath()

    --Get Grid points
    Grid = GetGrid()

    --Get important points
    ConsideredPoints = GridPointsInPolygon(PolyGon, Grid)
    print("#Considered Points:", #ConsideredPoints)

    --Generate Path
    Path = GeneratePath(ConsideredPoints)

    --print("Printing Selected Path:") --Print was removed from wrapper
function to keep from flooding output
    --PrintPath(Path)

    print(string.format("Path Generated. [%s] points long.", #Path))
    return Path
end

--[[
    Wrapper Function used for printing the contents of the PolyGon table
]]
function PathFind:PrintPoly()
```

```
   print("Contents of PolyGon:")
   for k in pairs (PolyGon) do print(string.format( "%s] [%s]",k,
_pp(PolyGon[k]) )) end
end

--[[
   Set all locations of a given table to nil, clearing it.
]]
function ClearTable(t)
   for k in pairs (t) do t[k] = nil end
end

--[[
   A quick test function to test the functionality of each wrapper
function in a more isolated space.
]]
function PathFind:WrapperFunctionUnitTest()
   PathFind.ClearPoly()
   PathFind:AddPoint(0,20,0)
   PathFind:AddPoint(100,20,0)
   PathFind:AddPoint(100,20,30)
   PathFind:AddPoint(0,20,30)
   _P = PathFind:MakePath()
   PathFind:PrintPath(_P)
end

--[[
   Following code is for testing bits and bobs in LUA~land outside of
Farming Simulator to verify code functions before having to make, move,
install, launch, and load mod for FS
   require("Point")
   --PathFind.Test()
   PathFind.WrapperFunctionUnitTest()
]]
```