Analysis

Why does it need to use a computational approach?

First of all, I need to simulate gravity. This requires a lot of calculations every tick, way too many for a human to do it by themselves in a reasonable amount of time. Also it can become even more complex if there is a scenario where multiple wells act on the same object.

Secondly it needs to be able to share users own level designs with their friends, this needs to be done computationally because if it isn’t it would take the user a lot of time to draw and use it, whereas if it is does computationally you can share and run your friends level within seconds.

Thirdly it needs to be available to 30+ children of varying ability, where the teacher would need to print off and help them without it, they can now just show them all how it works on a projector and there will be little to no problems. This makes the teacher’s job a lot easier and frees them up to help children who really need it.

Who are the stakeholders?

I myself am a stakeholder because I have a vested interest in the quality of the program because my A level grade is determined by it.

The students are stakeholders in this because they have a vested interest in the quality of the product because their education depends on how well the program communicates the concept of gravity to them.

The physics teachers are stakeholders because they will be the ones using it to teach 30+ children at once, so every little bug in the program will be amplified thirty-fold. Therefore the program needs to be suitably robust so the teacher can spend more time helping them with questions about the physics and spend less time helping them with questions about the program.

The school is also a stakeholder because it is their children being taught and the school is judged on how well the children perform. Also if the children are enjoying the program then they will not be misbehaving, which makes it easier on the teacher and the school itself will be better off.

Research

*Gravity launch – URL:* [*sciencenetlinks.com/interactives/gravity.html*](file:///C:\Users\tindallk.SPT\AppData\Local\Temp\Temp1_Project%20foldar.zip\A-Level-Project-master\sciencenetlinks.com\interactives\gravity.html)

Gravity launch lacks the feature of designing and sharing your own levels, this means the children cannot play around in an open sandbox with gravity so they will have a more limited understanding of the concept, whereas in my program they will have access to a level editor so they can try things they are uncertain about. Also then you could share the level you have just created with your friends, which will retain their attention more.

Gravity launch also takes a different approach on some matters, such as the thrust and angle gauges. While they do increase repeatability in their launches, I don’t think it is very intuitive and it feels quite clunky to use. So I think the way I’m approaching it (drag the mouse in the direction and magnitude you want the object to go in) is far superior because it is very simple to use and you can get a grasp of it very easily.

It also takes a different approach on the instructions of how to play, in gravity launch the instructions are just one big wall of text that is not very fun to read to say the least. My instructions will be short, sweet, and how pictures to help the user along. Hopefully then the students will actually read the instructions and not just see a big wall of text and think “it’s not worth it”.

*Gravity simulator – URL:* [*testtubegames.com/gravity.html*](http://www.testtubegames.com/gravity.html)

Gravity simulator also lacks some features such as there is no objective to collect. Without this vital feature it will struggle to keep the child’s attention for enough time to teach them the concept of gravity. This is why my program will have objectives on every level.

I think gravity simulator also has some bad features too, one being that it tracks where every gravity well and planet has been for the entire session, and this very quickly clutters up the screen and makes the entire thing look very messy and confusing. I do think that you should be able to see where the planets have been but there should be a cut off, you shouldn’t be able to see where they have been for the entire session.

Gravity simulator does have some interesting features though to its credit, one such example is it uses a modified version of the click and drag feature I intend to use, where instead of dragging in the direction you want the planet to go, you drag the opposite way. I like this very much and I may do it this way as it feels a little bit like you are using a slingshot to fire it in the other direction.

Another interesting feature is that you can change the strength and position of the gravity wells after you have placed it, this seems like a perfect feature to add into my level editor so that if they misplace one well they don’t need to make the entire level again. This will result in a lot less anger from the students and therefore will increase the fun and learning capacity of my program.

*Super planet crash – URL:* [*stefanom.org/spc/#*](http://www.stefanom.org/spc/)

Super planet crash, in my opinion, lacks a few features; such as you cannot share builds with your friends meaning that the students will get bored of it much quicker, this is a very similar problem to gravity launch so I will not repeat myself on this problem.

Super planet crash also has some bad features too, one such example of this is that your planets cannot go beyond 2AU from the centre star. While I recognise that it is what makes the game challenging, I think it hinders the children’s learning by making it a more confined environment and you cannot experience some of the things gravity can do, like slingshot planets around a star extremely quickly.

Another bad feature is that if you spawn in a larger planet it will attract a smaller planet, while this would make it more accurate, I believe that it will make it too complex for the children to deal with or use properly.

One interesting part of super planet crash are the aesthetics it has, they are very simple and pleasing to look at, I also like the faded look of the tail of where the planets have been. I think I may take some design inspiration from this.

*Questionnaire – questions*

1. Do you have any trouble teaching the concept of gravity to children?
2. Are there any specific misconceptions that always crop up when teaching the topic?
3. What kind of resources do you use to teach the students about gravity?
4. What do you think of them?
5. If you don’t already, would you prefer to use software as a resource?

*Summary of the answers*

1. Being able to physically see gravity affecting the planets seems to be something the teachers want, I could perhaps do this by adding an option to see arrows of what forces are acting on an object.
2. They seem to think that gravity is the same magnitude on every planet. I could solve this by having different strengths of gravity well available.

They also seem to think that gravity is a universal down force and not a force that pulls to the centre of mass. Luckily the whole concept of my game will dispel this misconception.

They also seem to think that there is no gravity on the moon, I can dispel this misconception by talking about it in some of the explanation.

1. Teachers seem to rely on more practical demonstrations of gravity, but air resistance can get in the way of that and cause a bit of confusion with the children. They can combat this with vacuum chambers but they are very expensive and some schools cannot afford them.
2. They seem quite content with what they have at the moment, but they seem to be very open to using software though.
3. Yes, they would find it useful for demonstrating things they cannot show in class, like things that have a different gravity to Earth.

Essential features of my proposed solution

The gravity simulation is an essential feature of my proposed solution, this is because it really is the backbone of the entire concept, and without the gravity simulation you cannot really use it to teach children about gravity.

A goal is also an essential feature of the solution I am putting forward because that is what gives the game its difficulty without hindering pupil’s creativity. If I did not include a goal in the final product it may only hold a child’s attention for something around 5 to 10 minutes, where an entire lesson is 60 minutes, plus 10 minutes isn’t enough time to really familiarise yourself with the concept at that age. The goal keeps their attention and allows them to have fun.

Another essential feature is a level editor, this is a bit like the goal in that it is intended to help keep the student’s attention, although that is not its primary function. The main use of the level editor is to allow the child to play with the concept to try and deepen their understanding. This is where a lot of the potential learning would come from.

Instructions are also essential, although a lot of the children will understand how the controls work right off the bat, others may not quite understand, but this doesn’t mean they should be left out. So some instructions should be there to make sure everyone who uses the program knows exactly how to use it, they need to be short, sweet, and informative.

Limitations

One limitation is that you need Python installed to be able to run .py files, and the entire program will be written in Python so if the computer running it doesn’t have Python installed it will not work at all. The computer will not recognise the file type. It may be worth noting that Python comes pre-installed with Mac OSX and most distributions of Linux, but if you are in a Windows environment (which most schools are) you will need to install Python yourself.

Another limitation is that you need pygame installed, pygame is a Python library which I will utilise to create the GUI, so it is very essential to have this, because if you don’t the program will display anything to interact with.

Screen size and the object count are also limitations of the program, this is because if the screen becomes too big or there are too many objects flying around the place the program will become very sluggish and will not be very fun to play. It does not help that most schools around the country don’t have very good computers either so that pushes the frames per second right down. So the screen needs to be a reasonable size and the object count at all times must be kept low.

Requirements

It will require Python 3.5.0 to be able to run as intended, this is because I am writing it in python 3.5.0 and I do not know if it will be compatible with other versions. I know for a fact that it cannot be below python 3.0.0 or else it will definitely not work.

It also requires pygame version 1.9.2, this is because pygame is very temperamental with what versions of Python it will work this and I know for a fact this version works with Python 3.5.0 so for everything to run smoothly it needs to be that version.

It also requires tmtu (ToMeToUtils), it is a library that I have mostly written myself, with some help from my fellow students. It creates commonly used GUI aspects very easily, and also has some other helpful things like being able to more easily read CSV files. I will be using this library very heavily in this program so it is a definite requirement. Obviously, being written by me, it is not a default library.

Success criteria

1. There must be at least 5 in-built levels – this is because there needs to be some kind of starter to get them used to how the program works and to teach them the basics of how they can work with gravity to achieve goals.
2. It must be able to create and load up a custom level – This is because it must be able to do the actual function of level creation and sharing, like I said previously it is a big part of the program and needs to work correctly.
3. It needs to have gravity wells that affect objects around them – this is because this is the backbone of the entire game, if this part doesn’t work, the entire concept falls apart.

Naming conventions

*Scope*

Variables, functions, and classes should be kept as local as appropriate.

|  |  |
| --- | --- |
| **Concept** | **Prefix** |
| Global variables | gbl |
| Global functions | gblF |
| Global Classes | gblC |
| Local variables | [function name] |
| Local functions | [function name]F |
| Local classes | [function name]C |

I’ve given things prefixes so that it is easier to distinguish between global and local variables, this is because it is important to keep track of global variables. Also local variables shall be given a prefix of the function where they were defined, this is in an effort to help keep track of scope while nesting functions. The prefix for local variables does not include the prefix of the function, only the core name.

Exception: while using a for loop, the temporary variable does not need to conform to these rules.

*Variables*

|  |  |
| --- | --- |
| **Concept** | **Example** |
| All variables must have their type as a suffix | gbl\_counter\_int |
| All constants must have the main part of the name in all caps | gbl\_PI\_flt |
| Multi-word variable names must use camel casing | gbl\_multiWord\_str |
| Class variables and functions do not have to use the local prefix. | Self.myString\_str |

|  |  |
| --- | --- |
| **Variable suffixes** | |
| **Type** | **Suffix** |
| Integer | int |
| String | str |
| Float | flt |
| Boolean | boo |
| Array | ary |
| #d Array | #dA |
| Tuple | tup |
| Dictionary | dic |
| Class instance | ins |
| Function | fun |

*Global code*

There should be as few global lines as possible, if something is global it should be imperative that it is.

There should only be this block of code that is run globally.

if \_\_name\_\_ == ‘\_\_main\_\_’:

Main()

All other functions that need to be ran should be called in the Main() function, this is to keep as much local as possible. I want to keep the amount of global code down because you can accidentally mess with other code if you’re not careful.

Design

Breaking the problem down into smaller ones.

Gravity, gravity is a big part of my game and is required to break down into smaller, more specific problems.

* Gravity
  + Gravity well acting on player
    - Changing player vector
    - Checking if player is in range
    - Calculate appropriate vector change
  + Player being acted on by Gravity well
    - Receiving any changes from gravity wells
    - Enacting those changes on the player’s current vector

These have to be solved computationally because there are a lot of numbers going between lots of different instances every tick, way too much for a human to do in an appropriate time.

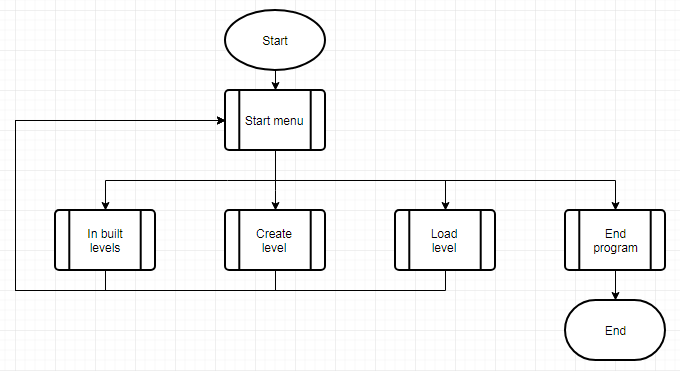
* Collisions
  + Sending out where the player is going
    - Send out the player’s Cartesian coordinates to a variable assessable to all other instances of objects.
    - Update coordinates every tick, needs to be priority one of first actions to happen each tick.
    - Coordinates should be sent out before any gravitational calculations are done on them.
  + ‘Solid’ objects detection
    - Check if the player’s coordinates are within the instance’s borders.
    - If it is, check if the instance is ‘bouncy’ or ‘deadly’, if ‘bouncy’ invert both vectors of the player, if ‘deadly’ then delete the player instance.
  + Goal detection
    - Check every tick for if the player’s coordinates are within the borders of the goal object
    - If it is, end the level and proceed to the next one, if it isn’t, do nothing.

These all have to be computationally solved because there are a lot of checks going on each tick, and a human would never have enough time in the world.

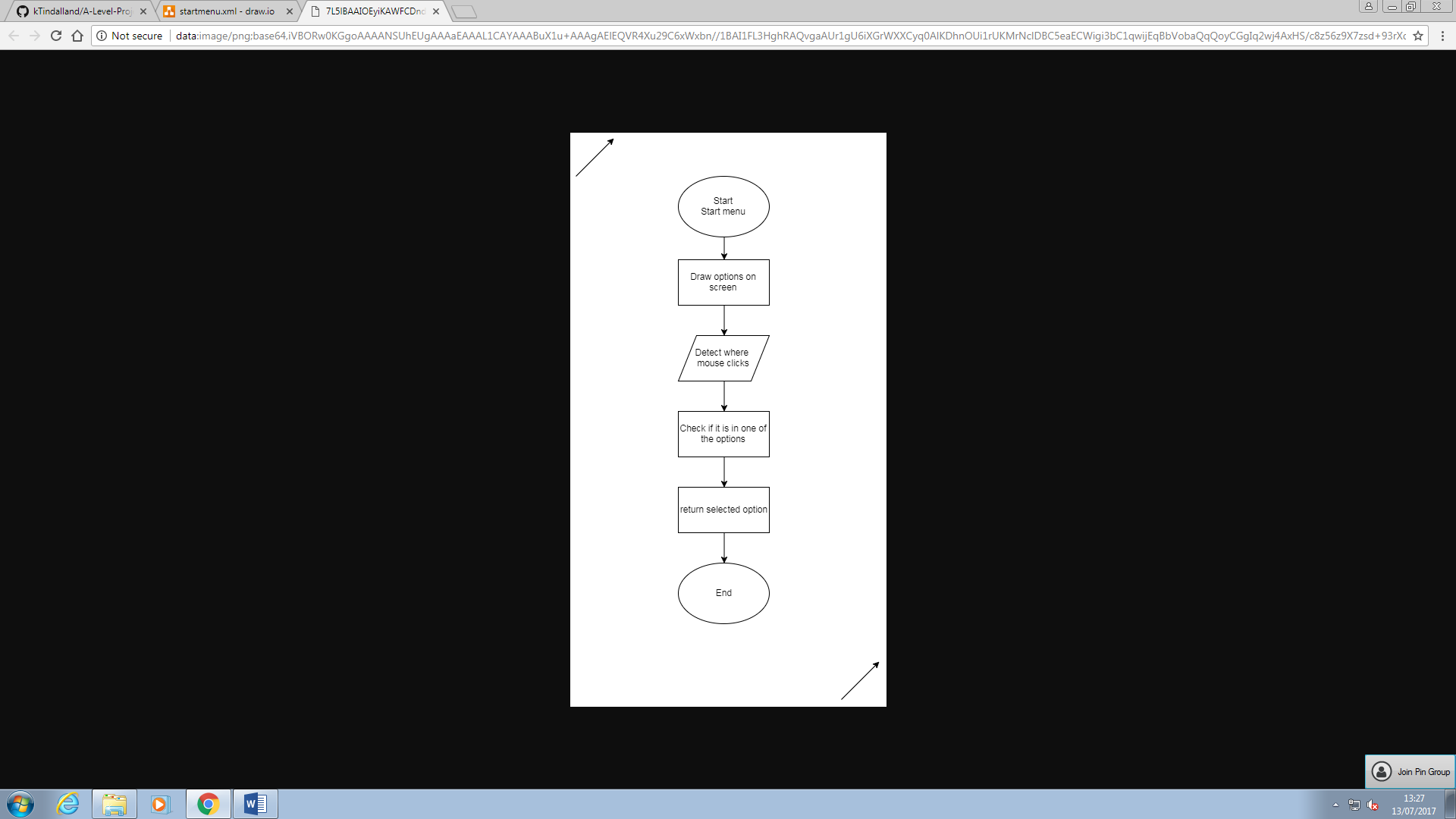
* Drawing custom maps from files
  + Reading the file
    - All custom map files will have a standard file naming scheme and content format.
    - File names will have the prefix ‘GravCustom\_’ and will have the file extension ‘.csv’
    - Read in multiple lists from the file, that will give information on where and which instances will go.
  + Go through lists and instantiate all objects with arguments from the list.
    - The list’s first element will always be the name of the class that it has the arguments for
    - [class name, [instance1Arguments],[instance2Arguments]]
    - Put all instantiated objects of the same class into one list and draw them all.

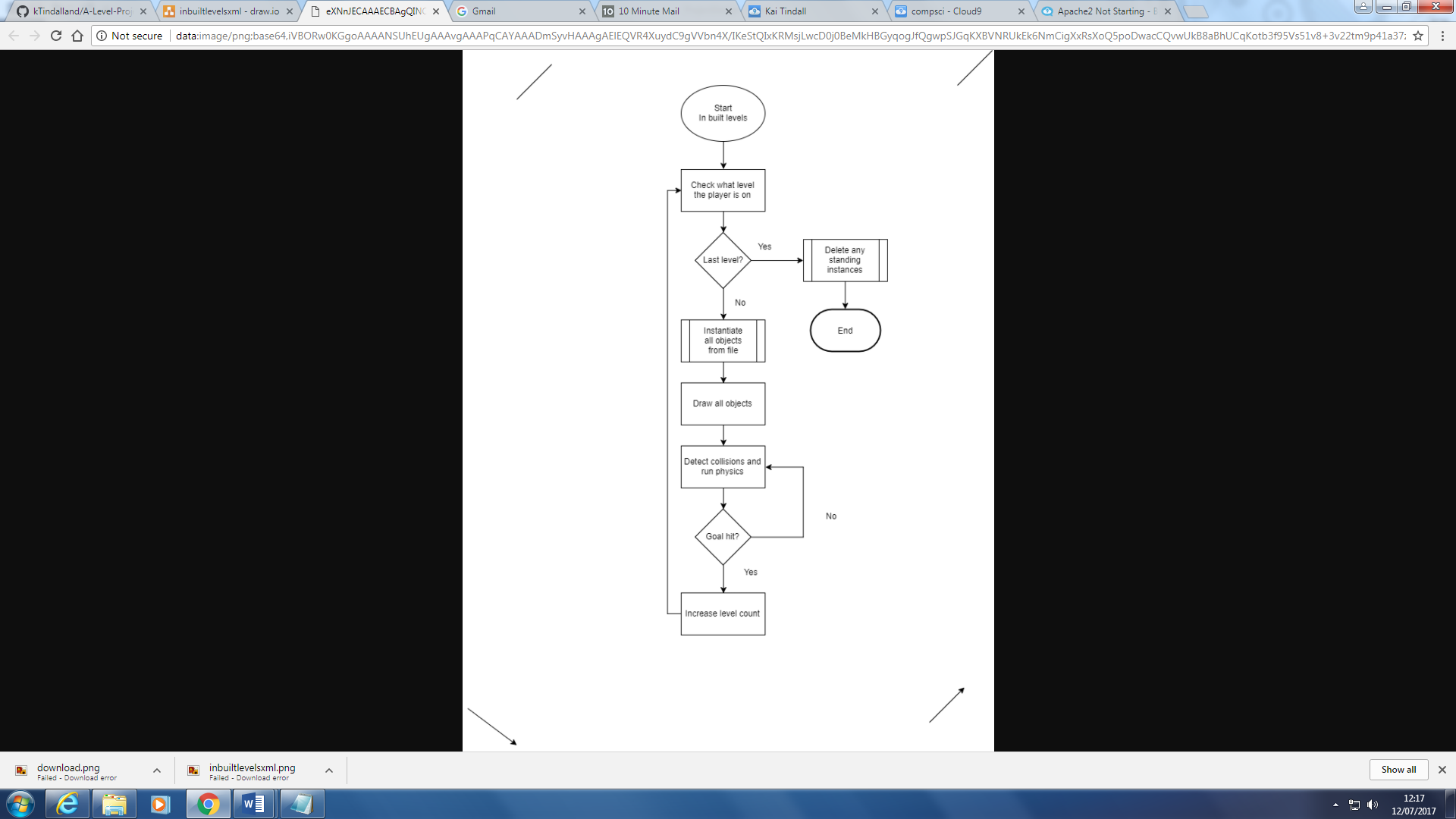
This need to be computed because there could be a lot of instances that need instantiating and then drawn and are detecting every tick.

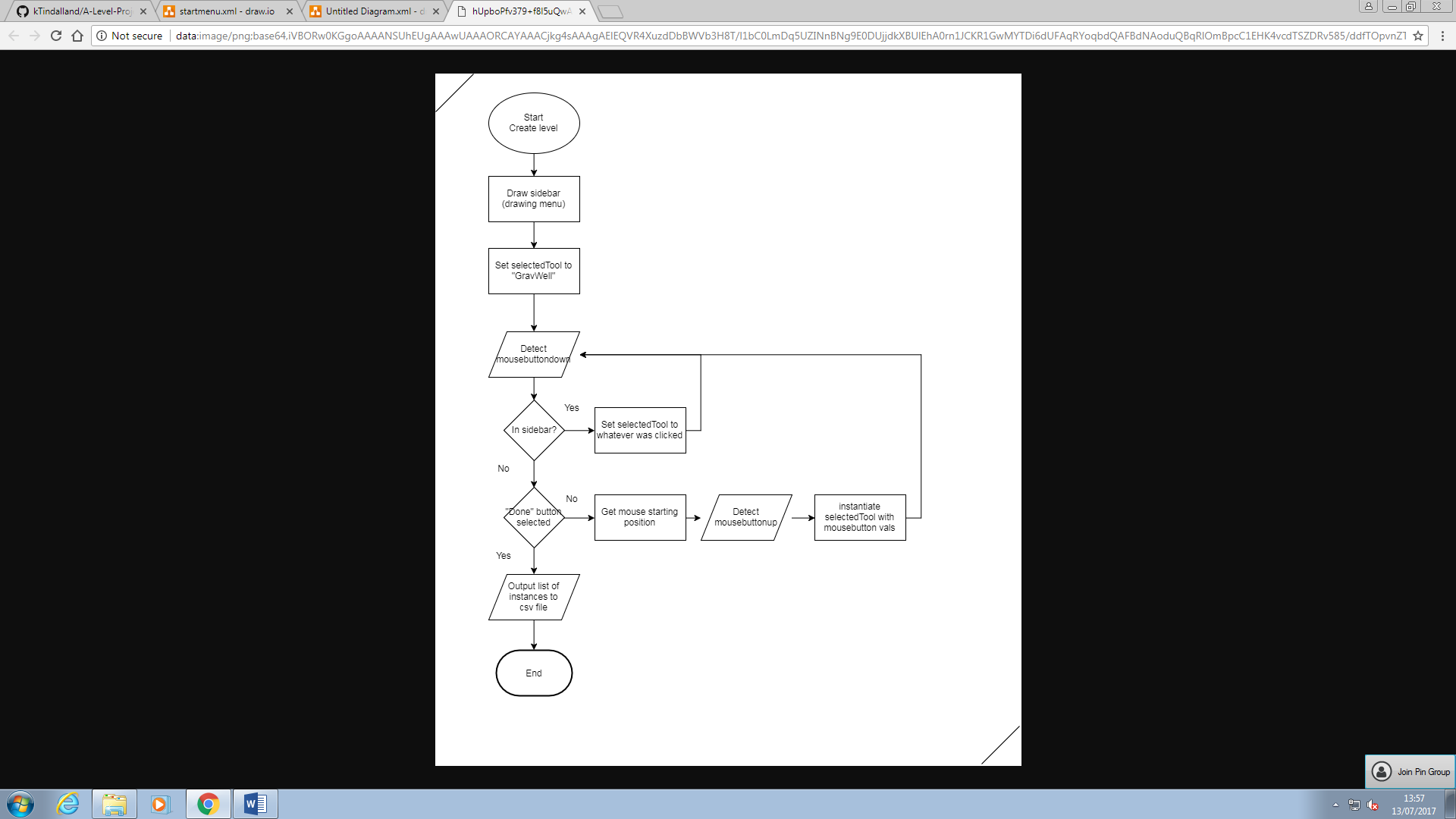
Program overview

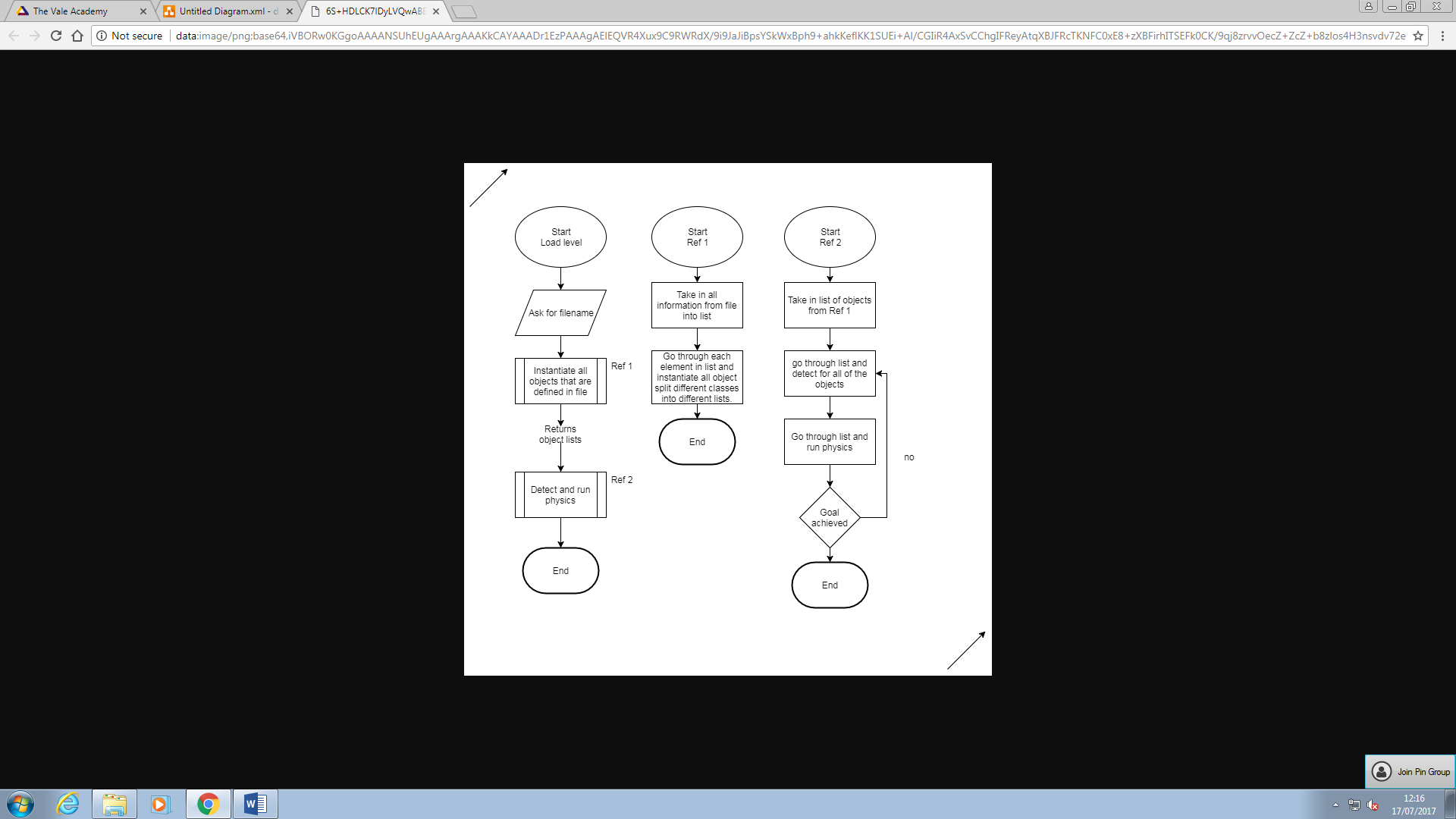


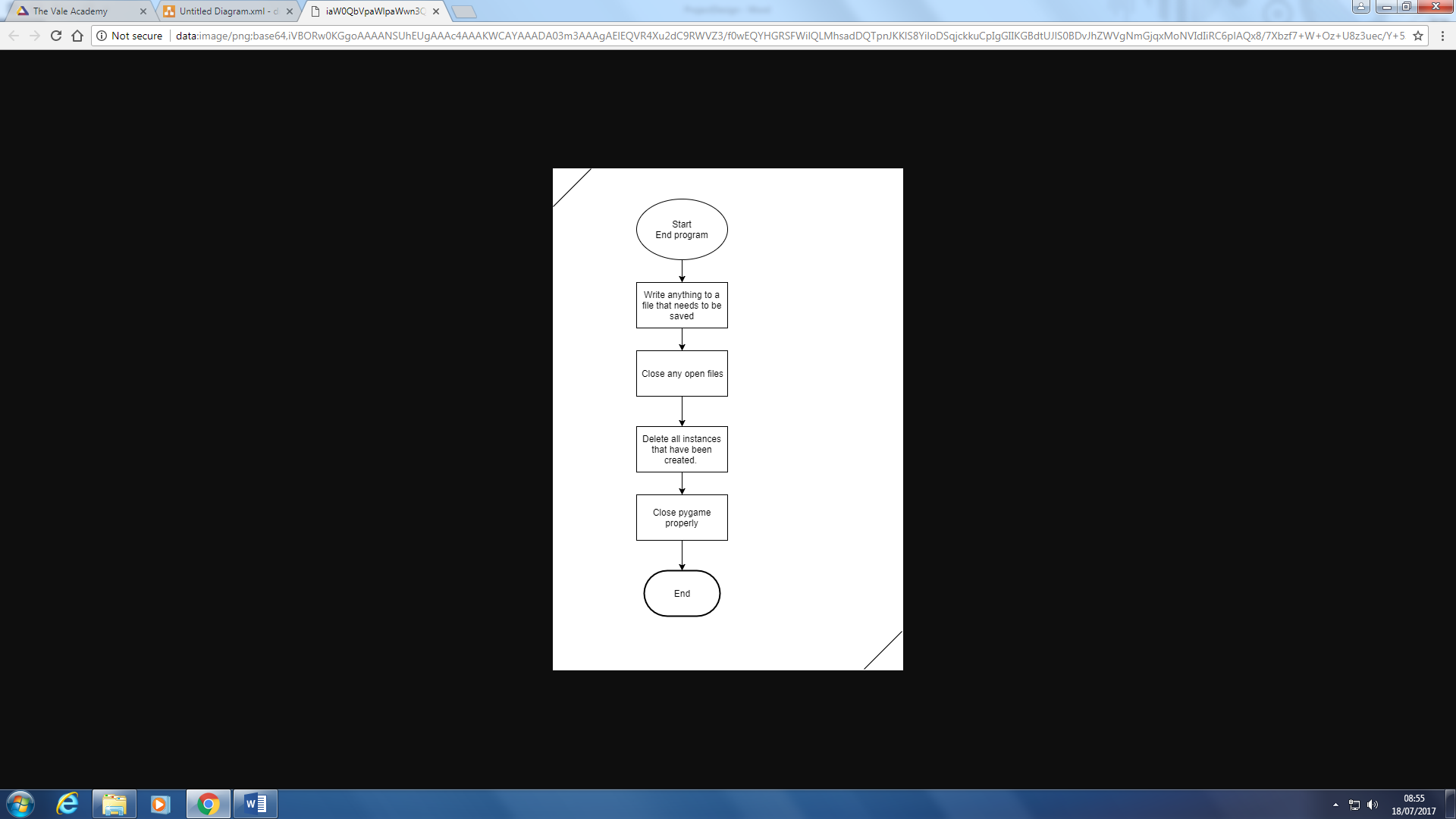
|  |  |
| --- | --- |
| Sub-process explanation | |
| Start menu | Draws menu options in boxes and detects which the user clicks. |
| In built levels | The player will go through all 5 inbuilt levels |
| Create level | Enters the player into the level creation menu |
| Load level | Takes in a filename to access and draw the map that is stored within that file. |
| End program | Close everything properly. |











Key variables

I will have some variables that are essential to the main function of the program.

*lcl\_importedInstances\_ary*

[[GravWell,[args],[args]],[Player,[args],[args]]]

This will be a 3d list that will hold all the information needed to instantiate all the objects within the level.

*gbl\_GRAV\_flt*

/\* final constant value to be determined kek \*/

This will be used with the density to determine how much the player will be pulled in.

Writing to files

I shall use an SQLite3 database to store all the instances for later retrieval. I will use a .db file to hold it. This way you can still share levels with your friends and by default you cannot open a .db file so it’s harder to break the file and corrupt the level.