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.

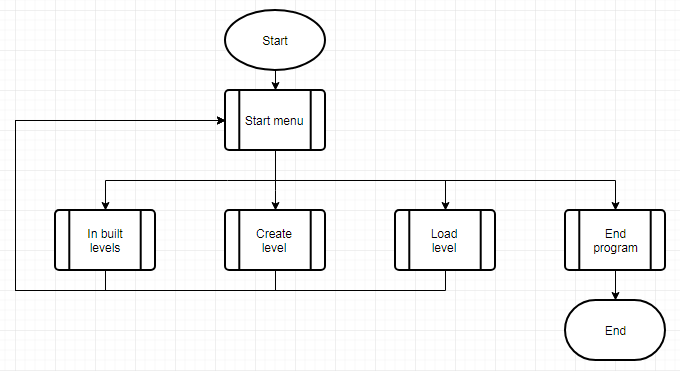
* Collisions
  + Sending out where the player is going
    - Send out the player’s Cartesian coordinates to a variable assessable to all other instances of objects.
    - Update coordinates every tick, needs to be priority one of first actions to happen each tick.
    - Coordinates should be sent out 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. |

