**AxohEngine**

Understanding the Game Engine

First of all, a game engine is software which is used to help in the creation of video games. It covers as much as possible in the background so the next game designer doesn’t have to. The goal is to streamline the creation process so it becomes easier and less time consuming. For example, your average program you may have created before this runs once and finishes. This game engine constructs a while loop which continually runs through a series of methods which do various actions like move objects or render objects. The program continually updates until a specific button is pressed which alters the while loop to end. A window object called a JFrame is also created in the engine, then a java class called Graphics2D is used to draw on to that new surface. The engine does way more than this, but this is the first goal of an engine. Loop -> window -> draw.

The other half of the classes provided, specifically the ones inside of the project folder, are project specific files. Which means they were created with the purpose of making a video game, every video game is different. They all need different variables and have different gameplay in some way. The goal is to put all of the game specific data in separated classes or in your game class and put all of the general “any game could need this” code in to the engine classes.

Maps

The goal for a map is to be able to easily create an environment that can be interacted with and traveled on. We will start with tiles which are the baseline to every map.

*Tiles*

A tile is a square of specific width and height which will be displayed in its entirety in a specific spot on the map. Basically this is a class which just contains an image from a file, usually a land texture like rock or grass. However these tiles also have option for expansion. Tiles can be animated, set as solid or slippers or breakable, tiles can be the spawn point for Mobs or points in which special actions can take place. This is called an event.

*Events*

Every tile has the option of adding an event type object to itself. This event class expands and contracts as specific needs arise as the game continues development. Basically, a tile takes this event as a variable and holds on to it, waiting, dormant. In the game engine system, all tiles which contain special characteristics like solidity or events, are constantly checked for collision with the player. When the collision takes place, the event is triggered. The event could be to give the player an item, or it could be to transfer the player position to being inside another map, truthfully the options for this are endless and the event class should be edited from here on out till your project is complete.

*Map*

Once the tiles are designed with their images and Booleans depicting their solidity or not, once the events have been designed, a map must now be constructed. Using an array of tiles, the coder can place the tiles using the variable name in a way that makes sense to their design. This array is fed in to the map class which takes all of the variable names from the array and looks at each one like a blue print. A new tile object for each spot is made based on that blueprint and saved inside of the map class. Afterwards, the coder adds all of the events and Mobs to be spawned to their respective tiles in the map. When this map is loaded by the engine, all of the data is read in and the tiles are rendered on to the screen, their attributes are retained, and when collisions occur, the actions you set take place.

*Map Database*

Simply a container class which holds all of your game specific variables, like maps, events, tiles and items. In here, items need to be created in a special order and can be taken from at any point later. The map database is the veins and arteries of your game. The order is:

Initialize Map Array -> Init. Sprite Sheets -> Init. Tiles -> Populate Tile Arrays -> Init. Maps ->Add the Maps to the Map Array -> Init. and setup Items -> Init. and setup Events -> Add Items to Events -> Add Events to Tiles in Maps -> Init. and set up Mobs -> Add Mobs to Tiles in Maps

Sprites

Sprites are broken down in to several relevant classes in order to better understand them and to add the option of only using specific pieces at times instead of all the class objects in order to save memory later.

*SpriteSheet*

Basically, this class takes a large image and cuts it up in to smaller images and stores them in to an array for use later. The goal is so we don’t have to store so many hundreds of image files with our game which takes up a lot of memory. Just specify a width and height of the sheet as well as a width and height of each smaller image and the code does the rest of the work for you.

*BaseGameEntity*

The base level of a sprite or image, this only holds location data, it’s simply a variable cache.

*ImageEntity*

Contains all of the image relevant data. This can load an image from a file and display it on screen. It can also check for collisions with other objects manually.

*Sprite*

This is the large sprite class which contains image data from the ImageEntity class, which in itself contains the location data from the BaseGameEntity class. The sprite class is much more complex because it allows for sprite sheets, a scale for rendering and several other variables relating to if the sprite was collided and what hit boxes it contains.

*AnimatedSprite*

Any object which might be animated, should start with this class. This class contains the methods needed to change an image based on a change in time. Right now, this class extends Sprite so an animated sprite can easily be accomplished, and Tile.java extends AnimatedSprite in order to give the option of having an animated tile.

*Mob*

The final class in the list of more complex objects, the idea of this class is to hold all artificial intelligence data. In here an AI can be set so a sprite moves around and changes image or speed based on other variables. This class should be updated in the future for more AI options. Right now only random and player exists. Additionally, attacks can be added to a mob for use in the AI.

Menu/Backpack

Think of the in game menu as an inventory, since it is, but also as a backpack for all of the player info like consumables or stats. In the future, the inventory in its entirety should be set to be saved at the end of a game and reloaded since this class contains all of the player’s items, current stats, level and so much more.

Resources (Images/music)

When uploading new images like sprite sheets or backgrounds, or soundtracks for use in the game, put all files in to the ‘res’ file in the git directory. This is a special spot because instead of being super specific when trying to find the file in the system for loading, you simply start with naming the file based on the ‘res’ file as a starting point. Also, you will find that some images have a very pink background that is because that specific hex color is unrendered by the system, which creates images that don’t have to be square.