AP Graphics Lib – Alec Pannunzio’s graphics library

# Overview

This will be a graphics/physics/game engine that will be able to model objects in a 2D or 3D environment. It aims to do the heavy lifting for game designers while creating either 2D or 3D games without sacrificing performance for either. It will do this by using heavy polymorphism. It will consist of a manager “Object\_draw” class that will control the Updating and painting of the objects onto the screen. The intended user of this project is anyone wanting to design a game or later anyone who wants to play the games built with the engine.

To show off some of the capabilities of the engine I plan to make a simple platforming game called “The Legend of Java”, in which players much complete a challenge to advance from room to room.

# Use case analysis

The use case analysis for a project like this is tricky. It obviously has a GUI that will be displaying all the objects in the engine but apart from that the game developers will be interacting with the library by importing different elements of it. To make this easier, I will be writing Javadoc annotations that explain key methods and classes to anyone attempting to use the engine.

Legend of Java will have a GUI displaying the player and the room that the player is in. The user will then have to use the keyboard to walk the player to the door of the room, at which the player will be confronted by some type of challenge. If I have time I hope to also have some combat-based rooms where the player will encounter an enemy they have to fight key-mashing style.

## Data design

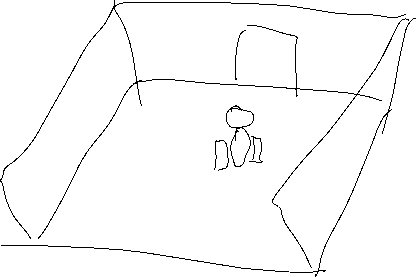
Inside my Object\_draw class I will have a LinkedList for all the objects in the engine that are tied to that Object\_draw instance. (Theoretically there can be multiple Object\_draws that control different windows. This can be used to make mini-maps or mini-games) The Object\_draw will also hold LinkedLists of Drawable, Updatable, Tangible, and Resizable type objects so that it can paint, update, check for collisions, and resize objects as needed from those lists. Objects that implement the Drawable interface must contain a reference to a Coordinate object that will store the position of the object. Objects that implement the Movable interface must have Vectors representing the speed and acceleration. Rotatable objects will have Vectors representing the rotation, angularVelocity, and angularAcceleration.

## UI Design

Similarly to the use case analysis, the engine doesn’t really have a set UI design.

Legend of Java however…

Back wall



Left

Wall floor



­­­­character right

wall

# UML

Simple version:

A close up of a map

Description automatically generated

Complicated version:

A picture containing text, map

Description automatically generated

## Key Classes and Methods:

### Object\_draw:

This class is what puts all of the elements of the engine together. It has lists of all the objects in the engine and will call their Update() and paint() methods when needed and holds an instance of Physics\_frame to paint the objects onto. It also controls the multithreading in the engine through its instance of the Object\_draw\_update\_thread object and monitors the frame rate and frameStep (how many times the objects are updated each frame) to keep the in-game time constant with real-world time while compensating for variable processing times.

## Physics\_3DPolygon

This is probably one of the most complex classes in the engine as it will have to deal with rotation and collision of points in 3D. It has a list of points that it will continuously update the position of and paint onto the screen in its paint() method. It holds an AffineRotation instance which it applies to the points each Update()

Note:

While I know that Graphics2D and Graphics3D will do Affine transformations for you, I opted to write my own versions from scratch.

## Physics\_3DTexturedPolygon

If Physics\_3DPolygon isn’t the most complicated class in the engine, this is. It will take a image file and apply it as a texture to any 3D set of points. This is housed in the setTexture() method and addPlatePoint() method. Plate points are the points that are placed on the surface of the polygon that hold RGB values from the texture. This is contrasted to FramePoints which are the points that govern the shape of the object.

# Side note:

I will NOT be using any libraries, videos, tutorials, etc. outside of the standard java libraries. ALL of this code will be written by me.

Now I know what you’re thinking: “There is no way he has enough time to do ALL of this in the time constraint”

And you would be right.

That’s why I started this project since the Covid quarantine started and have been working on it a LOT since then.

A screenshot of a cell phone

Description automatically generated