ENGR 133 LC4-05

Pole Position

Written By:

Alec Pannunzio

Team Members:

Jonathan, jdufresn@purdue.edu

Sam, graha205@purdue.edu

James, long365@purdue.edu

# Introduction:

I chose to write a pole-position game for a few reasons. Firstly, games are fun, and I have always loved wondering how video games were made and trying to make them for myself. Secondly, one of my interests in terms of a career path is to work on autonomous cars as a computer engineer. One of the main ways current autonomous cars determine what is the road and what is not is through various image processing techniques. The CPU for my pole-position game also uses image processing to drive itself autonomously.

Function Descriptions / Descriptions of Different Project Modules and Algorithms

# Graphics Engine (Pygame):

I wrote this project on the pygame graphics engine to establish a base to work on. Pygame allows me to transform images and display them to the screen in a timely manner.

#### The deformation problem:

When I first started working with pygame I found that there is a problem with pygame’s scale and rotation transformations. When an image is scaled down, it is shrunk down to fit inside a rectangular bounding box of the target scaling dimensions. This works great until the image is scaled *and* rotated *and* you want the picture to rotate about its center. The reasons that this problem originates is a bit long winded for this report but essentially what ended up happening is as the image rotated its corners stuck out from the bounding box, which meant that pygame further shrunk the image past the desired scale. What I did to solve this problem is manually calculate how far the corners will stick out and compensate by scaling the image to be that much bigger. The implementation of this is in Sprite.updateDisplayImage(self)

#### The Coordinate System problem:

Another issue I ran into with pygame is the way that it managed its coordinate system, although it stems from the computer graphics convention of the positive y direction being towards the bottom of the screen rather that the top of the screen, consistent with most math and physics axis. This means that sine and cosine will dictate that positive rotation is *clockwise* rather than counterclockwise, which is the physics and math convention. Despite this, pygame made the design decision to make positive rotation still *counterclockwise*, meaning that pygame rotation and sine and cosine calculations disagree. I fixed this issue by popping a negative in front of the rotation of my game objects before using pygame’s rotation method, but it sure was confusing before I discovered the discrepancy and got everything straightened out.

# Game Engine:

Since I was trying to program a game, I figured a good place to start would be to write a game engine. This, (surprisingly enough) is in the folder “game\_engine”. The game engine class hierarchy is as follows:

A picture containing text, black, night sky

Description automatically generated

To work with the game engine, a game developer would create a child classes of GameObject or any of its children (most likely Sprites and RelativeSprites) and an instance of the ObjectDraw class. They would then use ObjectDraw’s add() method to add the child classes of GameObject to the ObjectDraw instance. They then would create a loop at the end of their code like this:

while(not objectDraw.done):

    objectDraw.run(); # run the game engine stuff

Which will run all the game engine processes. This is implemented in PolePositionRunner.py

Anything game-specific processes that the game designer wants done will likely be put in the update method of child classes which follow the following template:

# updates the object values. is call repetively by the update loop

    def update(self):

        super(*<classname>,*self).update();

# do class-specific stuff here

#### First-person mode:

I wanted to do first-person mode right. Not by just rotating a picture around the center point and calling it a day but by actually implementing a system of cameras, where Sprites could be shown from the perspective of another Object2D. I did this with the RelativeSprite class. If a RelativeSprite does not have a camera assigned it will be shown at the exact position and rotation that its *xPosition, yPosition,* and *rotation* class members reflect (Exactly as if it was a normal Sprite instance). However, if the setCamera() method is called with another Object2D, then the RelativeSprite will be displayed onscreen from the perspective of that Object2D. This is done such that if a RelativeSprite has itself as its camera, it will be shown unrotated and at the center of the screen. The bulk of this algorithm is in RelativeSprite.update() and RelativeSprite.paint().

# Pole Position:

Pole position is essentially all the .py files outside of other folders.

polePositionRunner.py

Racetrack.py

Racecar.py

Playercar.py

Computercar.py

A picture containing dark, night sky

Description automatically generated

## Computercar Driving:

First. I passed the track image thorough my team’s edge-detection program from our team project. The Computercar then uses my raytracing function (raytracing.raytrace()) in conjunction with Racetrack.isTrackEdge() function to figure out where it is on the track. Racetrack’s isTrackEdge looks at the edge-detection output and if the point it is given is a bright pixel on the edges map it will return True. Then, the CPU algorithm compares the lengths of rays straight sideways and at 45-degree angles forward from sideways to come up with a value representing where it is on the track. This value is fed into a PD calculation as the error which determines the corrective steering output.

Additional descriptions of functions

Every user defined function has a description commented on the line above the function declaration in the code summarizing it. Additionally, every class has a description at the top of the file that class is declared in underneath the imports, and detailed descriptions of its class members and their types are listed directly below that.

User manual

Code