Object-Oriented Game Pseudocode / Notes

Player car movement:

Since the game is based around drifting mechanics, the car movement is essentially the same as an Asteroids-style space shooter, but with a car sprite on top instead of a generic spaceship:

- Multiple PVectors (acceleration, position, velocity) are required.
- When the player presses ‘up’ or ‘down’, **the game has to get the current direction the vehicle is facing** before applying the acceleration in that direction.
- Acceleration is applied to the current velocity, and the car’s position can be updated using the velocity vector each frame as well.

Processing Rotation reference ← use this to work on rotating the vehicle. Look up radians() + sin() + cos() again

TODO: add a constructor so that the main sketch file can instantiate different vehicles with new stats, sprites, etc.

Menu system:

There are 5 “screens” in the game:

- Splash screen
- Car select
- Track select
- Controls
- Gameplay

Each screen is positioned in the same place, but is only drawn when the player is actually on the screen. The player’s input depends on what screen they are currently on (i.e. vehicle controls don’t do anything when they’re in the splash screen)

Organize screens with an int currentScreen or screenNum that holds the current screen. Use this to determine which screens to draw and which player inputs to accept / ignore.

TODO: Use a switch case instead of multiple if statements for drawing each menu.

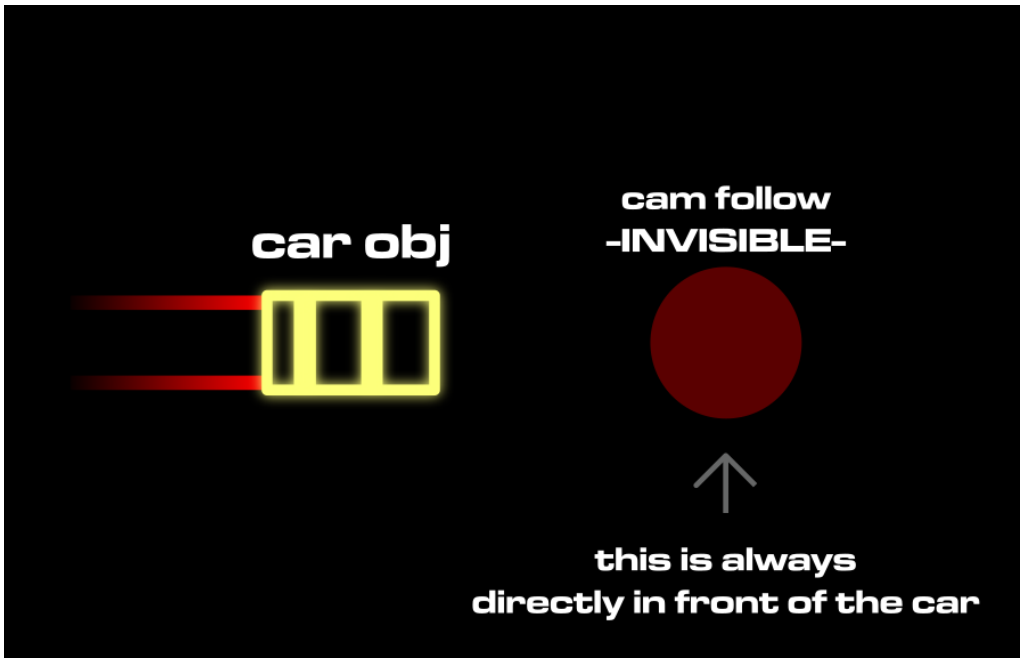
The car and track selections will be similar to this - the vehicle sprite (or model) will be spinning in the middle of the screen (if it’s a model, just use objName.rotate(number)). When the player hits left or right, the sprite / model will glide off screen (using easing) and the next vehicle in line will come up.

Smooth camera controls & movement:

The in-game camera will be updated using a combination of the camera() function along with Easing. The camera() function lets you explicitly state the position of the camera's 'eye' (i.e. where it's placed in world space), along with the center of the frame.
^^ this also allows you to animate the camera by iterating the eye position or the camera's center every frame

Rather than saying (center of frame = carPosition.x, carPosition.y), use easing to allow for smooth movement.

TODO: add a 'camera follow' shape *in front* of the vehicle at all times - right now the easing causes the car to get 'ahead' of the camera, and it could be difficult to judge upcoming sharp turns. example:



The GUI (displaying the car's speedometer, checkpoint time, etc) will be "anchored" to the camera's eased position.
^^ this has a side effect of making the GUI wobble a bit as the car takes sharp turns, fine-tune or replace it?

TODO: Since the position of the camera lens can be edited, add a 'camera' change button that changes different zoom-in sizes?
^^ would need to adjust the GUI scale as well, see if it's worth it