6-1-2012

Independent Game Development: The Making of Paze

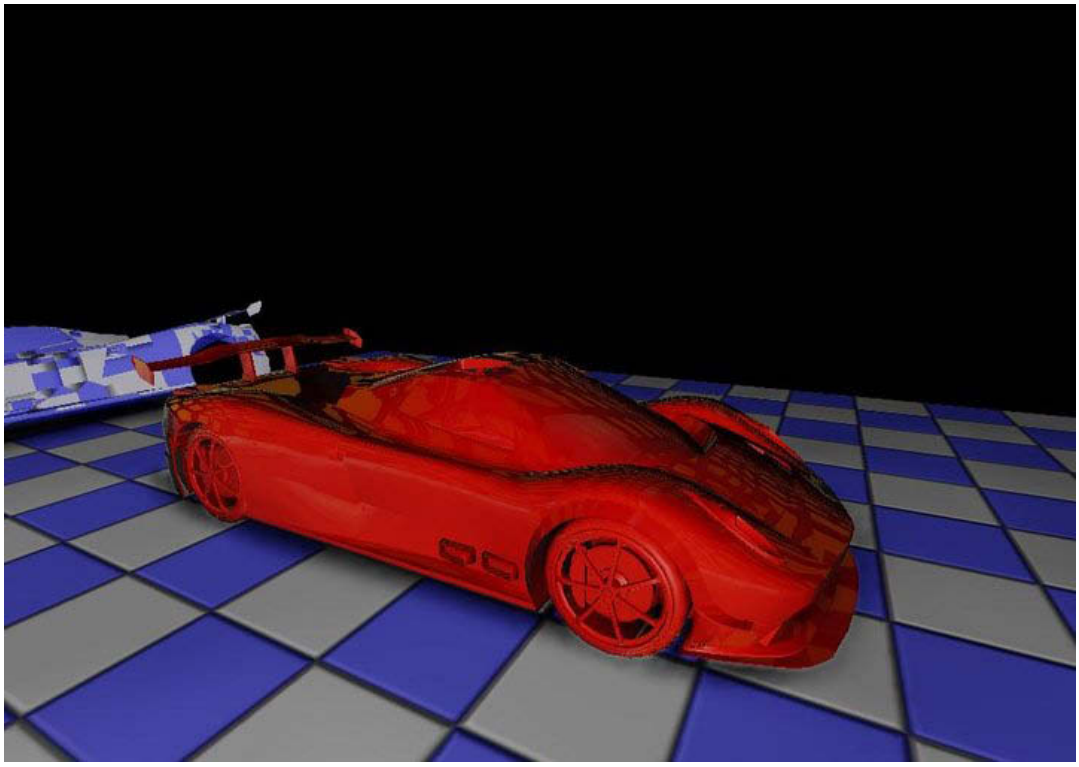
David O'Rourke
Western Oregon University

6-1-2012

Independent Game Development: The Making of Paze

David O'Rourke
Western Oregon University

Independent Game Development: The Making of Paze



David J. O'Rourke

Table of Contents:

I. Finding a Team

Why Game Development?

Since I selected Computer Science as my major, I have known where and how I wanted to apply it. I have grown up playing video games, and that childhood enjoyment quickly gave way to an appreciation for video games as an art form; thus, the opportunity to work in that field became a dream of mine. As I grew older, I came to realize that video games were going through the same sort of growing pains that film had at the turn of the 20th century, transitioning from gimmick and pure amusement to a legitimate art form, and, subsequently, really coming into its own as a form of storytelling. It seems that game development is now making that same move, and the potential for presenting a complex perspective on morality or belief analysis is exponential in video games games due to the increasingly interactive and cinematic nature of the medium.

As game development matures, so too will the themes discussed within the field, and it is my goal to be part of that sophistication movement. I want to give back to the industry that has given me so many hours of educational fun and intellectual stimulation growing up and help it along towards its establishment as an art form within mainstream society. To this end, I decided to find out what I could do to help make me a better game developer when I graduate and join a studio. When I learned various programming languages and practices, I always tried to see each in the light of their applicability to game development. Likewise, when I discussed theoretical material in my philosophy courses, I tried to imagine how I would walk a player through the concepts and present the arguments in an interactive matter. However, simply supplementing my degree wasn't going to be enough, so I decided to seek out an independent team and put theory into practice by help working collaboratively on the creation of a game.

How to Find a Team

It took a few weeks of research, but I eventually found a site where collaborative freelance work was fairly common. One could either post their team information with details about the project and

what exactly they needed to help make it happen, or one could post a notice that they were looking for work and team leaders would be in contact with them if there was a niche these freelancer workers could fill. I made a short post wherein I explained my credentials – which basically included the languages I had worked on within my Computer Science degree so far – and had five different teams contact me. I agreed to see if I could help with three of them, and I officially became a programmer on three independent game development teams: one was a large group of people that had been working on a sprite based rpg for a few years already, the other was a professional artist and programmer still in the design phase, and the last was a team focused on creating a racing/shooter multi-player game for the PC.

When I first started out as a programmer for these development groups, I tried to give as much time as possible to all three projects; however, this became an enormous challenge in itself, and I found that just trying to keep up with deadlines for the three projects took most of my time outside of class. Consequently, the time came where I had to settle for which team out of the three I wanted to focus on the most. Because one team was far more established than the others and was not hurting for programmers, and the other was in between projects at the moment, I decided to give the majority of my attention to the collaborative known as Team Orangecore and became a full time member in the spring of 2011. After a few months on the project, and as a result of the transitive nature of independent teams, I eventually climbed to the position of Lead Coder and took on the responsibility of helping new programmers get acclimated to the team, namely explaining to them all that must be done before one could actually begin contributing to the team.

What It Takes to Be a Full Time Independent Programmer

The first steps taken when joining and independent game development team – and likely any loosely regulated team with the goal of creating something – is getting used to how the team actually works. For example, the other two teams that I spent time with before coming on full-time with Orangecore utilized forums and group e-mail to communicate and used community cloud drives for

saving and transferring information. Beyond the steps where one learns to just speak with the team, one has to learn to get all of the relevant files onto their local machine,what these files actually do, and how to contribute and return those files to the group. For many people, this step of independent development easily proved to be the toughest, and it was where I lost the largest majority of people interested in independent work.

When explaining the setup phase to new members, I have likened it to the necessity of learning the basics of a culture's language and customs before one is able to actually communicate and give back to the group. Just getting the files needed onto a new users computer can take anywhere from a day to a week, depending on that particular person's familiarity with the tools and source control we use. For programmers, it was often on the shorter side; however, trying to get those files to an artist who had no idea what source control was, let alone how to operate it, took significantly longer, and it was usually several days before he was able to actively begin contributing to the team.

The first thing that someone beginning as a programmer for Team Orangecore would need to do would be to download the appropriate programs to communicate and access files for the team. We relied mostly on Skype and TortoiseSVN for this, respectively. Once those were downloaded, they would then need to download the Unreal Development Kit (UDK), with the specific version of the file we were using. UDK is the engine that Orangecore uses to implement much of the gameplay involved in Paze. Then, after those programs had been successfully installed, the new team member would have to checkout all of the codefiles from the team at the latest revision using TortoiseSVN. After that was finished, the new team member would be completely up-to-date with the rest of the team concerning raw file materials.

It was then that programmers – particularly new programmers – would have to spend some time going through the code and getting acclimated with what was actually happening in different areas and understanding the structure. It would be up to the lead coder to assist them along this process and, when they felt comfortable, to assign them a simple story, or goal,to get working. The challenging thing

as a team leader was getting somebody to completely the entire set of steps, as stepping into a behemoth amount of code and trying to understand what exactly is happening can be incredibly overwhelming. As it happened, this is where most new team members would decide the project was a bit beyond them.

II. Coding

The Car

Team OrangeCore began mostly as a group of artists that had a vision for a racing/shooter hybrid that would pit teams of two against one another and really force people to exercise great communication. When I first joined this collective, they had little to nothing in the way of code. As such, I basically started from scratch with a few art assets and had to figure out where to go on my own. The first thing I tried to get to work was to have the test vehicle they had made actually show up. Working against me in this endeavor was the fact that the UDK has not been utilized as much as other development engines yet, so we were blazing new territory in many respects; this meant that the amount of external help available to me was miniscule, if not entirely nonexistent. This reality translated into pouring over the class library for Unreal Script, the language used to communicate with the UDK. The way that I was able to do this was to first build a skeleton class for the vehicle, and then try to put the player controller directly in that vehicle as it is created using the following script:

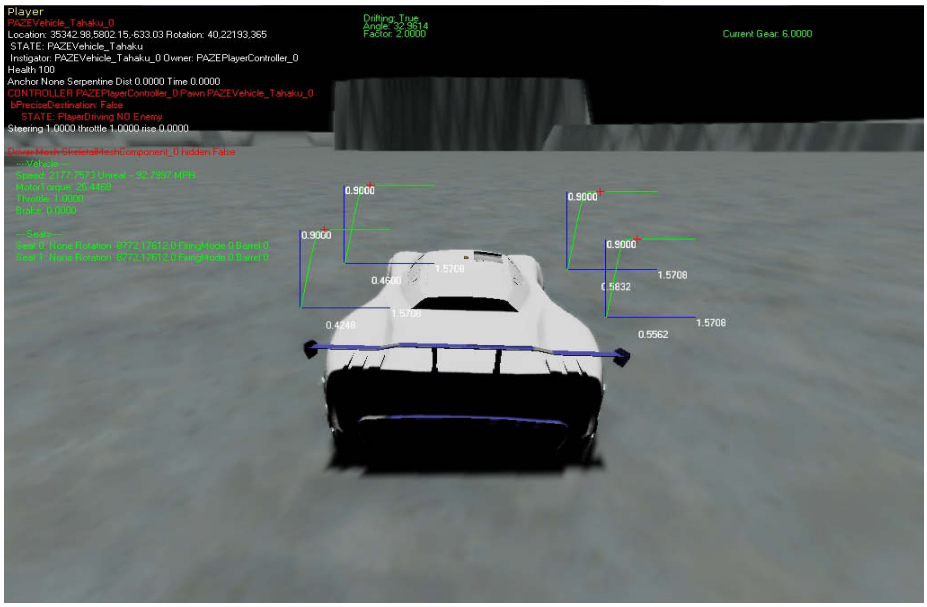
```
simulated function Possess(Pawn aPawn, bool bVehicleTransition)
{
	local PAZEVehicle PlayerVehicle;
	super.Possess(aPawn, bVehicleTransition);
	if (!bVehicleTransition)
	{
		aPawn.SetCollision(,, true);
		PlayerVehicle = Spawn(VehicleClass, , , aPawn.Location, aPawn.Rotation);
		PlayerVehicle.Mesh.WakeRigidBody();

		if (PlayerVehicle.TryToDrive(aPawn))
			`log("Driver:"@aPawn.Name@"entered the vehicle.");
		else
			`log("Error while trying to enter the vehicle!");
		aPawn.SetCollision(,, false);
	}
}
```

What is actually happening here is the class PAZEVehicle – which at the time was just a basic class that inherited the UDKVehicle with the artists prototype vehicle mesh placed in – is being called and then the pawn, or character, that the player is controlling is placed into that vehicle. What resulted was a car being spawned on the map that could actually be driven around, albeit with very primitive handling and controls. It looked something like this:

The next step was to get the car to actually handle like a vehicle in the real world would. However, I encountered some trouble because I was unsure of how to quantify differences in “feeling” between different sets of handling parameters. Therefore, I decided that I needed a good set of variables that were output to the screen – such as current angles of the tires, current speed, and the current angle of the vehicle with respect to the world – in order to objectively increase the realistic handling for the car. After building the various functions to grab those values from the UDK, I then used a DrawHud simulated function to present the information to the screen:

```
simulated function DrawHUD(HUD H)
{
	local float YL, YPos;
	YL = 15;
	YPos = 0;
	super.DrawHUD(H);

	DisplayDebug(H, YL, YPos);
	DisplayWheelsDebug(H, YL);

	H.Canvas.SetPos(800, 30);
	H.Canvas.DrawText("Current Gear:"@GetNowGear());

	H.Canvas.SetPos(400, 10);
	H.Canvas.DrawText("Drifting:"@(Drifting) ? "True" : "False");

	H.Canvas.SetPos(400, 20);
	H.Canvas.DrawText("Angle:"@mSlipAngle * RadToDeg);

	H.Canvas.SetPos(400, 30);
	H.Canvas.DrawText("Factor:"@mSlipFactor);
}
```

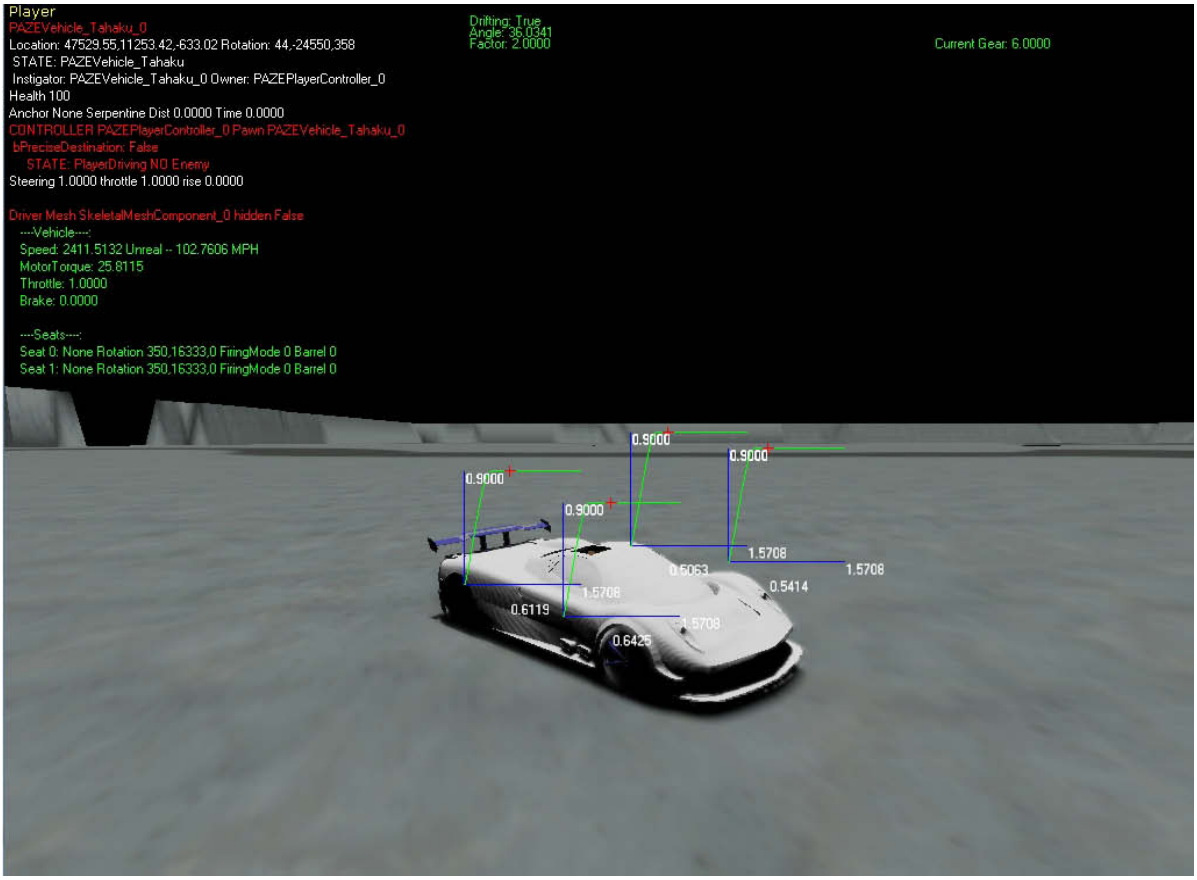
This top level function is basically calling the different variables that are resulted from the functions that get information like “SlipAngle” from the UDK engine, and then draws them to a canvas which is placed on the screen at each tick. That re-drawing of each “tick” of rendering is done by the “simulated” keyword, which means the function is not directly called, but instead occurs each time the screen is rendered. With that information presented to the screen, I then had tangible data to look at when changing the different ways the car interacted with the world, such as friction of the wheels. The end result gave me a screen that looked like this:



The last concern that I had that was vehicle centric was the camera. I wanted to make sure that various camera angles were possible, and that the user could easily switch between them. This led to the building of a fully custom camera class, which was treated much like an object would be, wherein it could be added to the scene after being built, and variables could be manipulated in it so that the camera could change modes from the simple press of a button. This code snippet shows a portion of the class that allows for simple switching between modes:

```
switch( CamType )
  {
      case 0:  // Chase Cam
          GetActorEyesViewPoint( out_CamLoc, out_CamRot );

          GetAxes(Rotation,X,Y,Z);
             out_CamLoc = Location - 400 * X;
             out_CamLoc.Z = Location.Z + 150;

             out_CamRot.Yaw = Rotation.Yaw;
             out_CamRot.Pitch = (-22.0f     *DegToRad) * RadToUnrRot;
      break;

      case 1: // Sideshot
             out_CamLoc = Location;
             out_CamLoc.Y -= 600;
             out_CamLoc.Z += 150;

      out_CamRot.Yaw = 16384;
      out_CamRot.Pitch = 0;
      out_CamRot.Roll = 0;
      break;
```

As I mentioned earlier, this snippet shows that the camera operates much like a standard object where the values can be changed externally using a simple “out_camLoc.Z” call, for example. This meant that I could have the function parsing for button presses in the vehicle class handling all the other inputs, and then simply change the values of the camera from there. With these new camera options, I could not only support different preferences of play, but also look more closely at how parts were moving on the vehicle at different turning angles and speeds, making sure that there wasn't any hidden clipping.



The Gunner

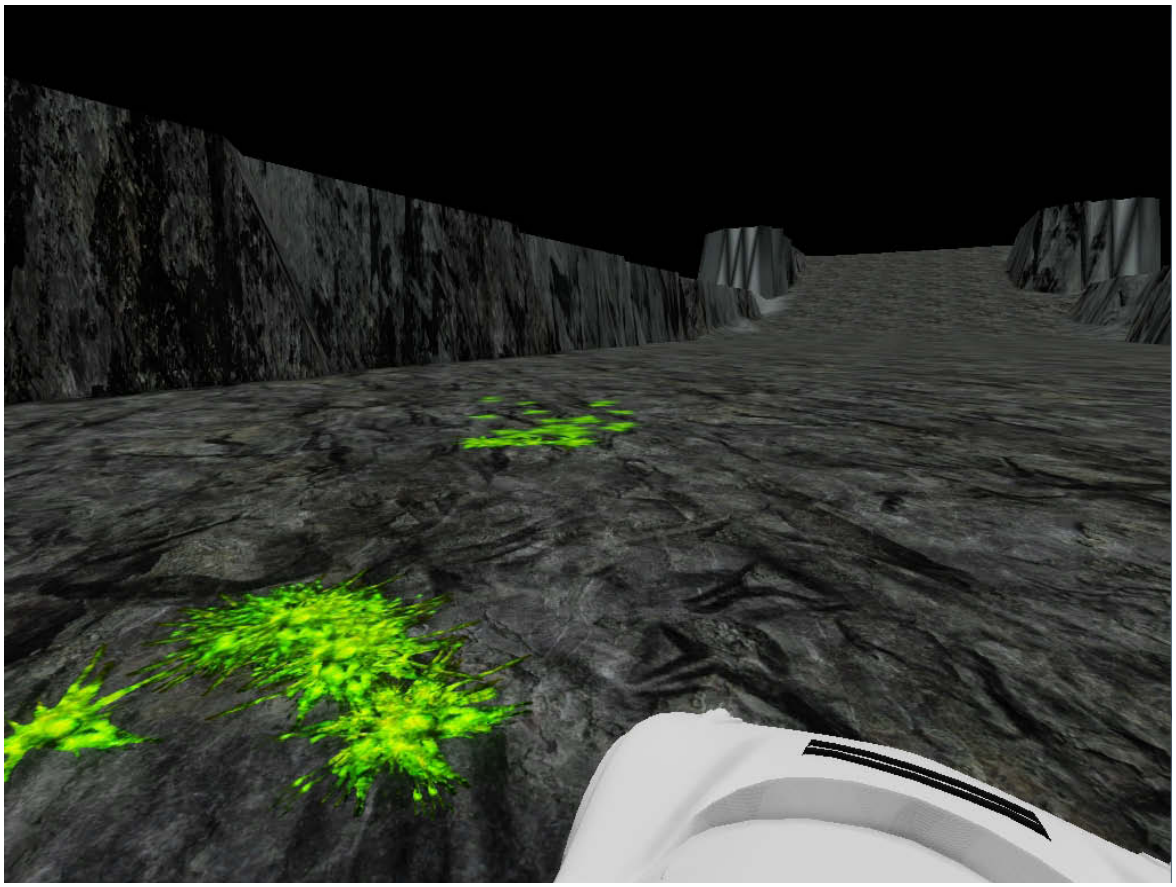
After getting the car to operate like a vehicle actually would in the real world – a task that took a considerable portion of the time that I put into the project – I began work on the gunner. This was tough, because the way that I had implemented the driver in that past did not leave room for a second user to be easily included. So what had to be done was the incorporation of seats and sockets. To expand on that a bit, the UDK has a built in class called “VehicleSeat”. It is basically an array of variable length that contains that seats for a car. What must be done, however, is to attach that array to different sockets (a 'socket' being a location in the world with respect to a bone, which is used in the rigging of a model). For the vehicle we were using, there was a main root bone that the artists built the car around, to which I added the driver socket and gunner socket, and which was then passed into the vehicle seat array to build virtual seats that players could be “placed” in. Basically, assigning a pawn to

a seat was as simple as a variable declaration of what pawn belonged in that seat. However, that was just with the virtual seats. To actually get the behavior we wanted of a driver and gunner in specific parts of the vehicle, I had to adjust the location of the sockets in the world with respect to the main bone. What all of this amounts to are two seats in the vehicle that can be moved between by the player, or ultimately, seats that allow the player being the gunner to be added to the gunner socket, and the player being the driver to be added into the driver socket. Then, variables for those seats could be easily manipulated for instant results:

```
Seats(0)=(-

              bSeatVisible=true,-
              CameraBone=("rBone"),-
              CameraBaseOffset=(X=-300.0, Z=150.0),-
              Offset=(X=-35.000000,Y=0.000000,Z=0.000000),-
              CameraTag="DriverSocket",-
              CameraBaseOffset=(X=-50.000000,Y=0.000000,Z=20.000000),-
              CameraOffset=-300.000000,-
              SeatIconPOS=(X=0.440000,Y=0.480000)-
       )-

Seats(1)={(-
              CameraBaseOffset=(X=-50.0, Z=50.0),-
              CameraEyeHeight=90,-
              GunClass=Class'Weapon',-
              GunSocket=("GunnerSocket"),-
              GunPivotPoints=("MainTurretYaw"),-
              TurretVarPrefix="Turret",-
              WeaponEffects=((SocketName="TurretFireSocket",Offset=(X=-
                     36.000000,Y=0.000000,Z=0.000000),-
              Scale3D=(X=6.500000,Y=8.000000,Z=8.000000))),-
              TurretControls=("TurretConstraintPitch","TurretConstraintYaw"),-
              CameraTag="TurretViewSocket",CameraOffset=-
                     256.000000,CameraEyeHeight=20.000000,-
              bSeatVisible=True,-
              SeatOffset=(X=-37,Y=0,Z=15),-
              MuzzleFlashLightClass=Class'UTGame.UTTurretMuzzleFlashLight',-
              ImpactFlashLightClass=Class'UTGame.UTShockMuzzleFlashLight',-
```

After moving the seat location of the gunner, I was able to make the gunner poke out of the roof of the car as the designer had intended, and through additional code I was able to give the gunner a rudimentary bio weapon that could functionally shoot. The problem that I ran into at this point, however, is one that inhibited me nearly the longest amount of time out of any problem that I faced on the project: while I was able to give the gunner a gun, and the player was able to use that gun, the

actual mesh (i.e., the visual portion of the gun) was completely missing. What resulted was a gunner positioned with an invisible weapon that could apparently summon projectiles at will; as you can imagine, this wasn't exactly what my team and I were going for.



As one can tell from the screenshot, the weapon visual is absent. What made this problem so time consuming was the nature of the vehicle implementation and the use of seats and sockets. I could, on the one hand, get the weapon to show up if I had the player spawn *outside* of the vehicle, and it would look and operate fully correctly. However, when the weapon was set as the weapon for the gunner, it would inexplicably vanish. After more time than I would like to admit, I was able to focus the problem down to one that exists within the engine itself. When a weapon is declared as a weapon for a seat, it can develop an error where it sets the mesh location to the location of the middle of the world, the effective (0, 0) spawn. As such, everything else worked fine, but the mesh was being attached to something entirely uninvolved with the vehicle or gunner. To solve this problem, I ended up having to

forget trying to root the weapon location to the socket and instead tied it to the skeleton of the pawn the player was controlling inside the actual gunner seat. This made the weapon show up, and operate a bit more in line with our goals.

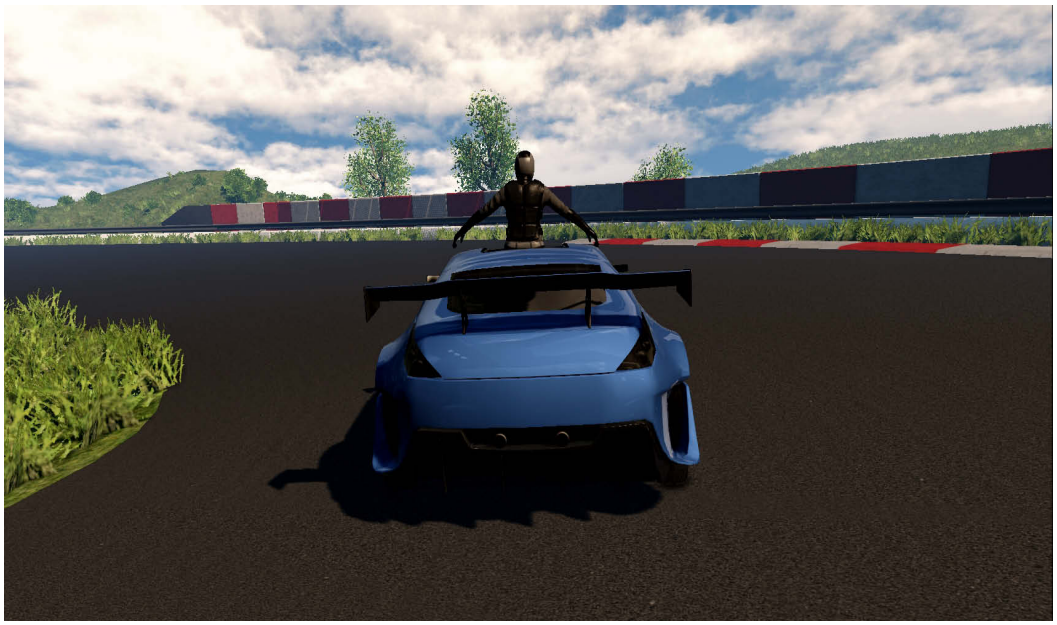
After getting the weapon in the hands of the gunner, next I had to worry about making sure the gunner was positioned properly in relation to the roof. This is critical, because while the gunner doesn't see if the model is off, the third person nature of the driver camera, and other players on the track, will be able to see if the gunner is clipping improperly through the roof or side of the car. When I was first able to actually get the gunner model to properly display in the gunner seat while playing as the driver, the mesh had heavy collision problems with the vehicle:

After a few different attempts to get the gunner model in the proper position, what finally worked was utilizing the socket system for the driver and gunner, then adjusting the socket location in respect to the main bone of the car, and finally some model adjustments to allow the gunner to fit through the hole in the roof without cutting through the roof improperly. After the gunner was put into the correct position and had a weapon, the main systems for the game were established. All that was left was the finishing touches and the replacement of placeholder models with the ones the artists have been working on.

Finishing Touches:

After getting the code in place and working for the driver and gunner, most of the remaining work was for the artists. I made sure that the default values for the vehicle and weapon were around an average of what the Team Lead wanted and then implemented an easy to read code block for the artists to alter for things like vehicle speed, acceleration, and grip. This was done so that the artists could create additional vehicles and weapons for players while not having to alter anything on the script side

except for a few lines of code and yet still obtain unique behavior for the different options. At the time of the completion of the code, the artists had one car and one track polished for release, and working on additional assets for both.



III. After Thoughts

Working With a Team:

Above and beyond what I learned about Unreal Script, and game scripting languages in general with this project, I learned a lot about what it means to be part of a development team. For independent teams in particular, communication is always a challenge. For instance, due to fluctuating schedules, Team OrangeCore had to have a scheduled meeting every Friday that all team members had to attend. We treated it much like an Agile stand up meeting. We would check the progress everyone had undergone so far, projected places they would be by the next meeting, and discussed any impediments they may have. Since everyone was learning in their respective member positions, those in lead roles had to be available whenever possible on Skype to assist; otherwise, team members would often get delayed for days at a time on a specific problem.

Source control also became an increasingly difficult thing to maintain as more members joined, due to newer members taking files and overwriting with a newer version from an older or newer UDK, and then altering the repository too much for proper rollbacks and the like. We eventually changed how source control was handled completely, and altered the file and commit status in such a way that newer members wouldn't derail the team for a few days at a time if they made a misstep. Additionally, the independent nature of the project had more than a few drawbacks, the most notable being that there was no real way to force people to keep to deadlines other than peer pressure and asking nicely, so we frequently saw deadlines come and go where artists (in particular) did not have a finished asset to commit. As a programmer, this often led to working on many different stories at once, basically waiting for a piece of art before it could be completely finished.

That's not to say that working with a team was a completely painful experience; it certainly came in handy when trying to build an opening cinematic in flash to have a programmer that had worked with flash in the past and was able to take over much of the process and teach me some things along the way. Also, having to walk new members through things multiple times really solidified concepts and project structure for me well

Impact on Career Goals

Working with Team OrangeCore has definitely solidified my want to work with a game development studio, though in a capacity a bit more strict than the independent one due to the lack of adherence to release goals that inherently exists on an indie collaboration. Overall, however, the act of being able to put time into this project and really see it through from beginning to completion has left me incredibly satisfied, and I look forward to being able to do it again in a full time position and on a larger scale. It has also instilled in me an appreciation for those in leadership positions that I previously never had, since I was charged with directing the other programmers on the team. While helping them along in their respective areas helped me learn all elements of the project, it is definitely draining to

keep all the different facets of the project clear in your head, instead of just worry about your own goals.

As far as the impact the game has had within my actual professional career – as opposed to just my intent – that remains to be seen. It unarguably does make me a stronger candidate to actually having team experience under my belt, however, even if it is only on independent development team. While it by no means guarantees me a position at a game development studio, it definitely puts me ahead of a lot of entry-level competitors. Beyond that, I started this project with the goal of becoming a stronger member a development studio someday, and without a doubt I have reached that goal. I have learned a scripting language, source control, how to lead other coders towards a goal, and most importantly, how to be part of a functioning team. I am much more prepared for my career after college due to this project, and I am confident that it has helped me toward that end more than anything else that I have done during my time at Western.