The University of Melbourne
School of Computing and Information Systems
COMP30019 Graphics and Interaction
Project 2, 2019

Kane Testa – 910748
Don't Crash

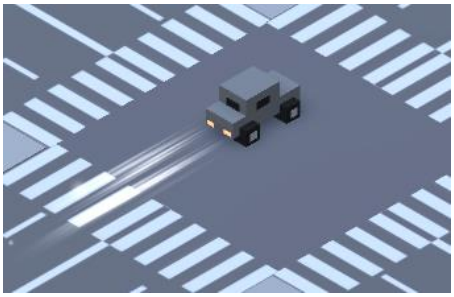
Explanation

Don't Crash is an endless game, inspired by the many popular iPhone games such as subway surfer, that consist of repeated attempts to beat your high score, with unlockables along the way. Graphics are inspired by games in the likes of Minecraft, which contain both simple, low poly graphics, and some higher detailed models. I really wanted to create a game that you can just pick up and get stuck playing for hours without realising how much time has gone by, and easy to play for any level of tech savviness.

The aim of the game is in the title, Don't Crash. All a player needs to do, is prompt vehicles in a busy intersection to speed up or stop, upon crashing, you lose the game. A player has a total score, which is the sum of all their attempts. At certain milestones in this score, you unlock new vehicles.

Controls

The controls are simple and can be found under the "Guide" section from the main menu.



"Left Click" on a car – Speed up the car

"Right Click" on a car – Slow it down

"Esc" or "P" - Pause
All UI controls are simply point and click.

User Interface:

There are 6 primary User Interfaces in the game.

Figure 1- Main Menu

This is the main menu of the game. It is the page that loads when the game starts. By pressing play, it leads you to the game. By pressing “Progress” it takes you to the progress screen (Figure 2). By pressing “Guide” it takes you to the controls screen (Figure 3). Pressing “Reset Progress” resets all your player data, being the high score and progress points fields. Your high score is viewable across all UI in the top right

Figure 2 - Progress

The progress screen maintains the main scene but pans the camera to show us the cars we have unlocked. You will find an indicator for your progress at the bottom of the screen. This will show how far you are from the next milestone. “Back” will take you back to the Main Menu.

Figure 3 - Guide

The guide is a simple informative page that explains the rules and controls if you missed them in the tutorial. “Back” will take you back to the Main Menu.

Figure 4 – Game Over

This UI indicates that a collision has occurred, and the game is over. It will show your score, along with a message regarding how well you performed. “Try Again” will restart the game, or you can “Exit to menu.”

Figure 5 – Game

The main gameplay makes use of one UI element being text. The two text elements contain the user's high score, and on top of that is the score for the current round. This will change to green if the high score is beaten.

Figure 6 – Pause Menu

The game's pause menu is simple and has two buttons. One to restart the game, the other to return to the main menu

Assets & Entities

Asset Store Items

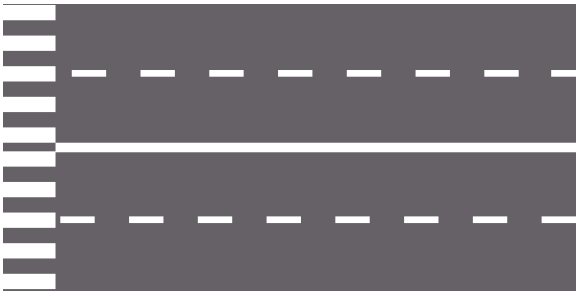
Low poly city from Viuletti - I used the prefabricated buildings from this free unity asset as it was too large of a time investment to create multiple buildings. However, due to the prefabs not having easily customisable vertex colours, I simply removed their materials and replaced them with solid colour materials I could apply my toon shader to. I also used the trees and stones from this package.

Voxel Cars Prototype – I based all my vehicles off those from this package. However, I customised them all so that I could easily change their colours at runtime, added headlights, taillights and sirens, etc.

My extension of "Voxel Cars Prototype" Prefabs contained the following.
Audio Source – Added audio source so that each vehicle could play sounds
Headlights – Added a small plane with spotlights as children, which activate when it is night
Taillights – Added a small plane with red spotlights as children, which activate when car brakes
Colour – "Body" game objects materials can be swapped so that every car's colour is randomly chosen
Wheels – Wheels were removed from front and back game objects so that I could rotate them easily when the car is moving

See the *Scripts* section for how I altered these at run time.

Texture (PNG)



I hand drew the adjacent texture (Figure 7) and then adjusted and coloured it in Photoshop so that I could have a road that I wanted, rather than a generic looking road. It is simply applied as the material for the road planes.

Figure 7 - Road Texture

Geometry Shader – Grass

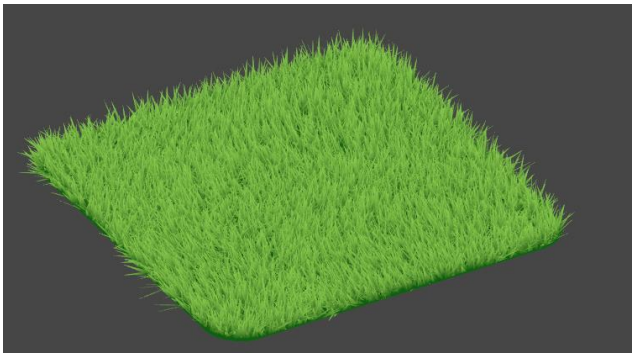


Figure 8 - Grass Geometry Shader

I combined parts of this Grass Shader with a toon shader to create the adjacent Geometry Shader (Figure 8). The reason I added the toon shader was due to the grass' inability to adjust colours based on the position of the sun in the scene. Without the grass the scenery was lacking a little bit more detail, it was almost too basic. To ensure that I wasn't creating more UV's than necessary, I carefully placed planes to minimise the amount of unviewable grass from the scene to optimise model geometry in the GPU (graphics pipeline).

See *Shaders* section below for more.

Sun

The "Sun" is just a directional light that orbits around the world, much like what was seen in Project 1. Additionally, the sun's position triggers events in the game, determining when car headlights turn on and off.

Head/Taillights

The headlights only light up when a vehicle spawns after it is night. The taillights turn on when a vehicle stops. Both lights are spotlights.

Particle Systems

The game includes three particle systems:

- Collision Particles – Attached to each vehicle was two collision particle systems, one black, one grey. On collision, these particles spawn into the air. The particles follow the laws of gravity and collide with the floor's colliders so that after multiple collisions, we can see almost a trail of where they all took place, as you would in a

real collision. The particles are small cubes and rotate over time. The speed of this rotation decreases over time and so do their size. These particles die after 10 seconds and are simulated to the world space

- Speed Particles (Streamlines)/Speed Particles (Smoke) – These are the two particle types that spawn when a vehicle is clicked to speed up. The streamlines simulate to the local space while the smoke simulates to the world space.

All particles are intentionally cartoony to fit the style of the rest of the game.

Skids

The way that I chose to represent my skids were just by spawning small planes at the x,z position of the wheels when the vehicle is braking or has collided. These planes continue to spawn from the moment the vehicle is slowing down, to the moment it has stopped. I chose not to use a particle system for the skids since I like the effect that the system I chose makes. It almost resembles ABS systems in cars and fits with the sound effects a bit better than alternatives. The length also differs depending on the deceleration of the vehicle.

Sounds

All royalty free sounds sourced from ZapSplat. The sounds were either collisions, sirens, horns or ambient sounds.

Script Rundown

AudioManager – in charge of assigning clips to the audio sources of a provided game object. These objects have usually been interacted with in some manner whether it's a collision, brake or acceleration.

CameraShake – In charge of making the camera game object shake when two vehicles collide.

CamMove – In charge of the camera transitioning between its two possible states, defined in the two child game objects in the "CameraObject" object. Created for the "Progress" scene.

Car – Controls all movement, collisions and interactions with small vehicles. Controls wheels spinning also.

CopCar – Same as above, including the siren's rotating lights.

Exclamation – Assigns exclamation marks to provided game objects. Made to let users who have no volume or are hearing impaired have a visual cue for when a car is about to recommence movement.

GameOver – Controls actions surrounding enabling, disabling and interacting with the Game Over menu.

LevelControl – Main controller. Includes spawning the vehicles, adjusting spawn rates, keeping track of scores and progress and more.

MainMenu - Controls actions surrounding enabling, disabling and interacting with the main menu.

PauseMenu- Controls actions surrounding enabling, disabling and interacting with the pause menu.

SkidManage – Manages the spawning of skid prefabs based on the position of a vehicle.
Sun – Manages the orbit of the directional light representing the sun.
Truck – Same as car.
Tutorial – Manages the tutorial. The tutorial occurs when progress is at 0. It informs the player of what the instructions are.

Other
Planes were used for the plain objects such as the concrete.

Camera Movement:
My camera is fixed within majority of the gameplay. However, during the transition from the Main menu to the Progress screen, I created two transforms with rotation that were the checkpoints for the camera to move between.

Shaders

Toon Shader. Inspired by https://docplayer.net/61098261-Introduction-to-surface-shaders-prof-aaron-lanterman-school-of-electrical-and-computer-engineering-georgia-institute-of-technology.html

The Toon shader I implemented differs from the Shader provided in the workshop due to the requirement to add a colour manually to the material. It also takes aspects from Phong shaders to add some level of reflection. The reason I didn't purely use a Phong shader is because I needed to apply the shader to a lot of prefabricated assets from the asset store which I couldn't edit the vertex colours. The shader is also faster than a normal Phong shader due to modifications to the specular reflection. This modification avoids the computation of the reflected direction of the light ray.
Normals are created out of the light and camera vectors and converted to the diffuse and specular components of the vertices.
"_MainTex" is used to set the base colour of the surface, which the diffuse tint is applied on top of it. Since this shader now didn't rely on a point light, I found that it behaved better in terms of FPS, optimising the render pipeline. As a result of this I was able to use more demanding particle systems to make the collisions better.


Figure 9 - Without Shader


Figure 10 - With Shader

***Geometry (Grass) Shader*.** Inspired by https://roystan.net/articles/grass-shader.html

The grass shader (Figure 8) was implemented so that the grass areas on the map weren't plain colours, it also makes the game feel more 3D than 2D, since the grass moves with the wind.

It works by randomly generating vertices that create triangles, each representing a blade of grass. The width and height of each blade of grass are all the same, however, their rotation differs so that it looks like they are all different sizes. These blades are randomly placed on a plane, the amount determined by the tessellation uniform.

The geometry shader itself takes a single triangle as an input and makes the grass is facing up by calculating the tangent based on the curvature of the surface. This is important since I used the shader for the plants outside one of the shops as well and this caters for the rotation of the plant object. Note that the rotation matrix calculated can make the grass have a rotation offset, making the grass more realistic. This occurs after the vertex shader in the rendering pipeline.

The wind is generated before the triangles are added to the triangle stream. I was able to customise the grass shader based on my requirements; however, the wind was quite complex so most of the code is exactly from the inspiration. UV co-ordinates are generated for each vertex. The co-ordinates were based off the grass' transform coordinates so that all grass is being blown by the same wind system. We basically just offset the position of the UB based on time and the wind distortion map to give each grass blade movement. The wind matrix contains a normalised matrix of the wind values across all coordinates on the grass planes.

I also added the Toon shader as a Sub Shader to ensure that the grass wasn't bright green all hours of the day in the game, better reflecting the time of day to the player.

Matrix Rotation Calculation Source:
https://gist.github.com/keijiro/ee439d5e7388f3aafc5296005c8c3f33
Noise function Source: http://answers.unity.com/answers/624136/view.html

Testing & Feedback:

I was able to have my game tested by 5 separate testers who each played the game for 10 minutes each. Each tested the game before and after I made changes based on their feedback.

User Breakdown:

- 2x Young Adult
- 2x 13-15
- 1x Adult

The acquaintances each undertook the game in a casual setting. It could have been a field study since each member was in a position which they would generally open their phone and play a game to kill time (lunch break). I walked the players through the UI and controls using the cooperative evaluation method, where we each asked each other questions, however, I tried to keep my questions along the following lines:

1. What do you think the objectives of the game are based on your first 1 minute?
2. How do you control the vehicles?
3. Where can you view the controls?

I liked this method of evaluation best since participants were open to criticise the game, which is what I was after.

I also did a post-game Interview to ask for some feedback. Consisting of

1. Can you see yourself playing this game and getting lost in time? Why/Why not? *To see if my game was reaching the goal I set out*
2. What are improvements you would ask for? *To find new features my audience may want to see*
3. What was the next car you were going to unlock? *To see if the audience cared about their progress or cared about their progress.*
4. Was the game tiring to play? *Want it to be a no-brainer*
5. Was the game difficult to play?

Feedback Rundown

All the players were able to successfully understand the objective and controls of the game almost instantly, which was fantastic. However, the post-game interviews posed some very valuable feedback. All the players showed interest in unlocking cars within the 10 minutes they played. Only one didn't know what car was next to be unlocked. The general consensus amongst the players believed the game to be simplistic, easy to play, hard to do well and addictive. The actionable feedback was:

1. UI scaled differently on different devices (Mac Vs Pc)

2. Police cars too fast, can sometimes rear end cars before you even notice them when the spawn rate is faster
3. Can easily get a high-ish score without having to do much
4. “How do I find my high score? I only can tell what my high score was when I beat it because it changes colour”
5. Not a wide range of sounds, sounds like the same crashes occur over and over
6. Glitch where truck tyres rotate wrong way
7. “When playing with no volume, I can’t anticipate when my cars are going to take off as I can’t hear the horn”.

And the relative actions taken were:

1. Anchored UI
2. Slowed down police cars
3. Slowed down all cars, so that spawns occur before the next car exits the screen
4. Added high score text to top right corner of screen
5. Added more sound effects for vehicles and collisions
6. Ensures prefab had consistent rotation fields on the tyres
7. Added exclamation mark over car that is < 1.5 seconds away from taking off

Each player that replayed the game after I made the changes was satisfied with the improved difficulty and other minor tweaks.

References

Majority of the code was created specifically for Don’t Crash by Kane Testa. The code that was inspired by the work from others all have their relative references throughout the report. I built on or took parts from these references, and never directly cut and paste anything.