Team Members:
Cody Jensen - 100591285 - Programmer
Nicholas Gauthier - 100754586 - Programmer
Nathan Woo - 100787454 - Programmer

Cody: Coded Update and created functionality, textures, model
Nicholas: Coded Movement & Win/lose
Nathan: Coding Shaders, Working on Explanations

The game we chose is a video game rendition of the children's game "Red Light, Green Light." The player will move in one direction along a hallway towards the exit. On random time intervals, the lights in the hallway will turn red indicating that the player must stop. If the player is caught moving while the lights are red, they will lose the game. After a brief moment, red lights will turn back to normal indicating the player can continue moving. The light is on a randomised timer.

The result of adding the last digit of each team member's student number is an odd number (15).
4 + 6 + 5 = 15

Explanation of Concepts (25%)
Don't need to implement any of these questions
Select the obstacle/enemy and explain how you can use the programmable stages of the graphics pipeline to improve the visuals of your game.

As the obstacle is a light itself, we could use the programmable graphics pipeline to transform the light from just a light being emitted, into a floating orb primitive using a programmable custom fragment shader. The fragment shader could be coded to create a "Neon" glow around the primitive, which would look better.
Nvidia goes over how to achieve it in this article.
https://developer.nvidia.com/gpugems/gpugems/part-iv-image-processing/chapter-21-real-time-glow

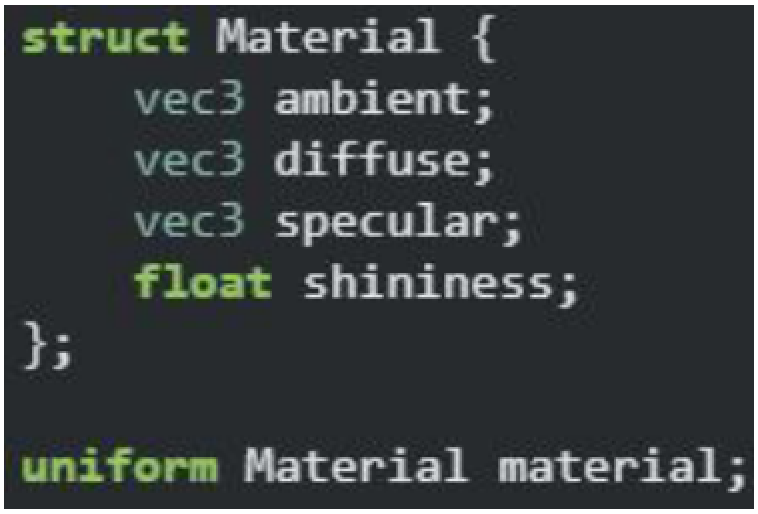
Explain how the Phong lighting model allows you to create a plastic feel for objects within the game.

Phong's lighting model generally describes light reflection off a surface as a combination of the diffuse and specular light reflection (whereas specular light is on shiny surfaces.)

The amount of specular and diffuse light (as well as how shiny the specular light is) is determined by the properties of a material which factors into the calculation of light in the Phong Lighting model.

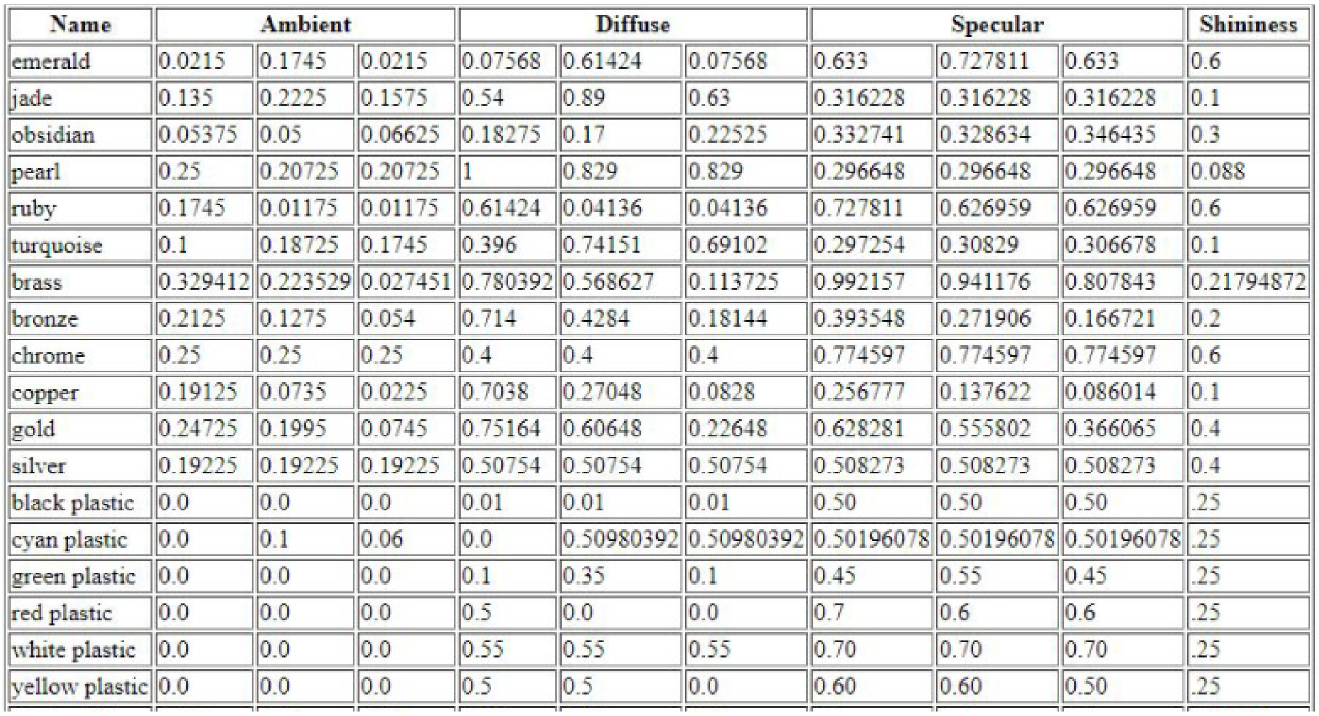
To create a plastic feel, the material can be emulated using a struct containing the lighting information stored in colour vectors (Ambient/colour, diffuse, specular, shininess) for the material uniform within the fragment shader. Plastic typically has a shininess value of 0.25.

(Lecture 3)



```
struct Material {
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
    float shininess;
};

uniform Material material;
```



Name	Ambient			Diffuse			Specular			Shininess
emerald	0.0215	0.1745	0.0215	0.07568	0.61424	0.07568	0.633	0.727811	0.633	0.6
jade	0.135	0.2225	0.1575	0.54	0.89	0.63	0.316228	0.316228	0.316228	0.1
obsidian	0.05375	0.05	0.06625	0.18275	0.17	0.22525	0.332741	0.328634	0.346435	0.3
pearl	0.25	0.20725	0.20725	1	0.829	0.829	0.296648	0.296648	0.296648	0.088
ruby	0.1745	0.01175	0.01175	0.61424	0.04136	0.04136	0.727811	0.626959	0.626959	0.6
turquoise	0.1	0.18725	0.1745	0.396	0.74151	0.69102	0.297254	0.30829	0.306678	0.1
brass	0.329412	0.223529	0.027451	0.780392	0.568627	0.113725	0.992157	0.941176	0.807843	0.21794872
bronze	0.2125	0.1275	0.054	0.714	0.4284	0.18144	0.393548	0.271906	0.166721	0.2
chrome	0.25	0.25	0.25	0.4	0.4	0.4	0.774597	0.774597	0.774597	0.6
copper	0.19125	0.0735	0.0225	0.7038	0.27048	0.0828	0.256777	0.137622	0.086014	0.1
gold	0.24725	0.1995	0.0745	0.75164	0.60648	0.22648	0.628281	0.555802	0.366065	0.4
silver	0.19225	0.19225	0.19225	0.50754	0.50754	0.50754	0.508273	0.508273	0.508273	0.4
black plastic	0.0	0.0	0.0	0.01	0.01	0.01	0.50	0.50	0.50	.25
cyan plastic	0.0	0.1	0.06	0.0	0.50980392	0.50980392	0.50196078	0.50196078	0.50196078	.25
green plastic	0.0	0.0	0.0	0.1	0.35	0.1	0.45	0.55	0.45	.25
red plastic	0.0	0.0	0.0	0.5	0.0	0.0	0.7	0.6	0.6	.25
white plastic	0.0	0.0	0.0	0.55	0.55	0.55	0.70	0.70	0.70	.25
yellow plastic	0.0	0.0	0.0	0.5	0.5	0.0	0.60	0.60	0.50	.25

Explain what approach allows you to create a dizziness feel using shaders.
To create a dizziness effect, the fragment shader can be used to offset colours to make a sort of moving chromatic aberration that simulates not being able to a single image and "seeing double." To do this, multiple colour vectors can be created, and the colour channels (RGB channels) of the pixels can be offset with a sin function and given variation and "movement" over time. The final colour is calculated by mixing the offset colours together and outputting it as the fragcolor.
Another possible way could be using a fragment shader to make the scene monochromatic and have an overdraw effect.

Explanation of Implementation (25%)
Making moving objects (e.g., main character, an enemy, or a moving obstacle) emit light under a certain condition (e.g., power up). This includes proper light behaviour when moving away or closer to objects.

The player object is set to move when the spacebar is pressed. The transform for the player object is updated while the spacebar is held down and the camera's position is updated to match the same position as the player object. If we were to add a light to our player, we would do the same thing as our camera and match the position of the light to the player's position, and then recalculate the shader with the new position.

In our project, we have lights being emitted on a randomised timer that switch the colour and the intensity of the light for a period of time to indicate a state change within the game's functionality.

Explain how you implemented the shader for this Midterm and indicate why this choice was made.
We could not implement a shader in time, however, we believe that a fragment shader that emits a glow would be beneficial towards our game.