University of Arkansas – CSCE Department
Capstone II – Final Report – Spring 2018

Traffic Control Game Update

By: Anthony Lam, Jonathan Ma, Mitchum Dillard, Perry Mills, Rodrigo Sanchez, Tanner Wilson

Abstract - Tanner Wilson

The traffic lights and other constructs of automobile-based transportation are a familiar part of everyday life for many people. However, people who have never driven on a public road may not know how traffic intersections work. Even if a non-driver understands traffic laws, without waiting on a green-light themselves, they may never ponder potential optimal designs of traffic intersections. This provides an obstacle when teaching traffic-based civil engineering principles to say, young children. One way to *show* these students how traffic intersection throughput varies with dynamics like traffic-light timing is to create graphical simulations of intersection scenarios - video games.

The University of Minnesota has previously fulfilled this task with their creations **Gridlock Buster** and **MyTrafficKontrol** - these two games give users a way to directly interact with simulated traffic intersections. Unfortunately, both games were released some time ago and are seemingly unmaintained. As a result, neither game works perfectly 'out of the box' (troubleshooting is required to use the programs). Furthermore, both pieces of software have software bugs that detract from their usability.

As a resolution, our team will create a successor update (actually a complete re-write) to both games, merging them into a single, multi-platform enabled Unity project. The final deliverable will feature corrected logic, improved graphic design, and new reporting features compared to

the two original games. In doing so, we will reduce complexity for end-users and provide an up-to-date tool for teaching basic civil engineering concepts.

1.0 **Problem** - Mitchum Dillard

There are currently two Traffic Control games made by the University of Minnesota in 2009. **Gridlock Buster** and **MyTrafficKontrol** have become outdated over the years and have become very cumbersome to use, especially for an average user. Both games are also aesthetically unappealing by common standards, which could lose the attention of K-12 students meant to play them. **MyTrafficKontrol** in particular has a large number of bugs in its code and has many issues regarding the actual meaning of its output statistics. Due to these issues, neither game is very effective in appealing to younger students and the difficulty of use discourages teachers from trying to use them in the classroom. If these games were usable and appealing they could be great teaching resources for learning the importance of traffic control and the nuances that go into making a large road network function in an efficient manner.

2.0 **Objective** - Anthony Lam

The objective of this project is to redesign both the **Gridlock Buster** and **MyTrafficKontrol** games into a more friendly and usable software. Our primary focus is to be on the **MyTrafficKontrol** game as **Gridlock Buster** received an update in 2016. We need to update its graphics to become more appealing to younger students and to offer a more engaging learning environment. It is a top-down *blocky*-style strategy game with little traffic variation. We need to maintain its original core game style while improving its functionality and understandability. It currently contains a data graphing system to show traffic backup and delays. We are also tasked to redesign this system for a more legible graph with an ability to save data sets to an easily accessible location. The other aspect that needs to be reworked is the Performance Index statistic, as currently there exists no intuitive way to understand what it means and how well your traffic system is functioning.

Following **MyTrafficKontrol,** we need to brush up **Gridlock Buster**. It is mostly functional and updated but has a few quirks that need to be worked out. We need to implement a persisting audio control function, as currently when you mute game sounds it only works for the duration of the level you are currently in (as soon as a new level starts the audio settings are reset). Another aspect requiring work is the timed traffic control system. You are meant to be able to set fixed time intervals for the intersections but this does not always operate as intended or in an intuitive manner. In previous levels you can click either the intersection or the traffic light to activate the intersection, but with the fixed time system you must click the road intersection to be

able to alter the timing. Our project must resolve the user interface issues present in the original as well as streamline any explanations and first-time user experience.

3.0 Background

3.1 Key Concepts - Mitchum Dillard, Jonathan Ma

The Unity Game Engine[6] is a modern platform for the development of a variety of games ranging from 2D to VR. Unity offers a multitude of tools and resources to build and develop expansive games. Its program build operations allow for broad development, upon designing an initial program it is possible to export the build to work with: WebGL, Microsoft, MacOS, Linux, Android, and iOS. This versatility allows for many options of distribution to ensure our game can reach as many people as possible. Unity is also tied to a paid and free Asset Store[7], where we can obtain visual graphics for our project and remove a significant chunk of development time from the graphic design portion of the project.

Since the intended audience for the initial project is students from Kindergarten to 12th grade, we need to keep in mind the level of understanding these students will have when interacting with the game. Kindergarten students will have a vastly differently outlook and experience playing this game than 12th grade students. We will need to maintain the balance of entertaining all aspects of audience while informing them respective to their current understanding of traffic control.

Traffic light timing can influence wasted time and energy for traffic users as well as preventing (or causing) traffic gridlocks. In order for students to realize and correlate these traffic light timings to their resulting cumulative wait times, a proper simulation software is necessary.

3.2 Related Work - Perry Mills, Rodrigo Sanchez

There are two traffic control games designed by the Intelligent Transportation Systems Institute at the University of Minnesota:

Gridlock Buster is an uncomplicated, appealing introduction to traffic control concepts. The original Flash release in 2009 featured simple point-and-click controls and colorful top-down graphical elements [3]. In each consecutive level, the player is introduced to a new concepts and more complex street configurations. The game was updated with stylized isometric graphics in 2016 by University of Minnesota's Center for Transportation Studies [4]. In its current state, high scores can be earned by only maximizing yellow-time, an unrealistic strategy for real traffic flow. Additionally, the audio mute state is not maintained between levels or attempts.

MyTrafficKontrol is a more advanced simulation with graphs and statistics to track traffic flow [5]. As a standalone Java game, it is out of date and difficult to run for the average user. The multi-window interface is difficult to navigate, and the metrics presented to judge the user's performance are puzzling.

4.0 Design

4.1 Requirements or Use Cases - Perry Mills

Publicly available game which fulfills use cases:

- Introduction for K-12 students to traffic control
- Increasing levels of difficulty/complexity which introduce new concepts to the user
- Advanced simulation mode to challenge user, applying all previous concepts
- Exportable graphs and statistic tracking
- Metric to compare user solutions and denote improvement

4.2 Detailed Architecture - Mitchum Dillard

- Using the Unity Game Development Engine we have begun to hybridize both games into a singular software, allowing easy access to either a simulation based traffic environment or an arcade level based environment. As they both use very similar components we feel that this is be the best approach to developing these two softwares. By using a main menu system we will be able to offer an easy and intuitive way to switch between the different modes.

- When designing the simulation software we want to improve upon the existing software by implementing multiple different vehicle types beyond cars, trucks, and buses. This will allow us to better represent actual traffic conditions and provide better simulation results.
- In the simulation we will also be redesigning the Performance Index metric as in its current state it is unclear how well the user's system is performing. On top of this we will also be adding additional features to the performance based graphs that are currently implemented as they do not show light states at any moment. To fix this we intend to include a tracking metric to the graph system to show individual intersections and which direction is red/green at each plotted point on the graph.
- As both systems use very similar traffic light systems we will be implementing them in both the arcade and simulation instances of the game. The lights and intersections will work in tandem to control the vehicle instances around them. By using Unity's built in collision detection system we have set up an invisible collision wall that will be activated by the light's current state. If the North/South direction is red then those directions will have a collision wall set in place and that will trigger the cars from those directions to stop their forward progress.
- Currently, MyTrafficKontrol has extremely outdated graphic systems and is visually simplistic, especially to K-12 students. Gridlock Buster does not suffer quite as much from this issue as it was updated somewhat recently. It order to remedy this we plan to use the Unity Asset Store to find open source, modern vehicle models to provide a more interesting visual experience. We also intend to put focus into redeveloping the user interface to be more intuitive and easier to understand at a glance. This can be done by making sure control buttons have obvious usage and by play testing our alpha version we will be able to see what systems we can use that will work properly towards this goal.[8]
- 

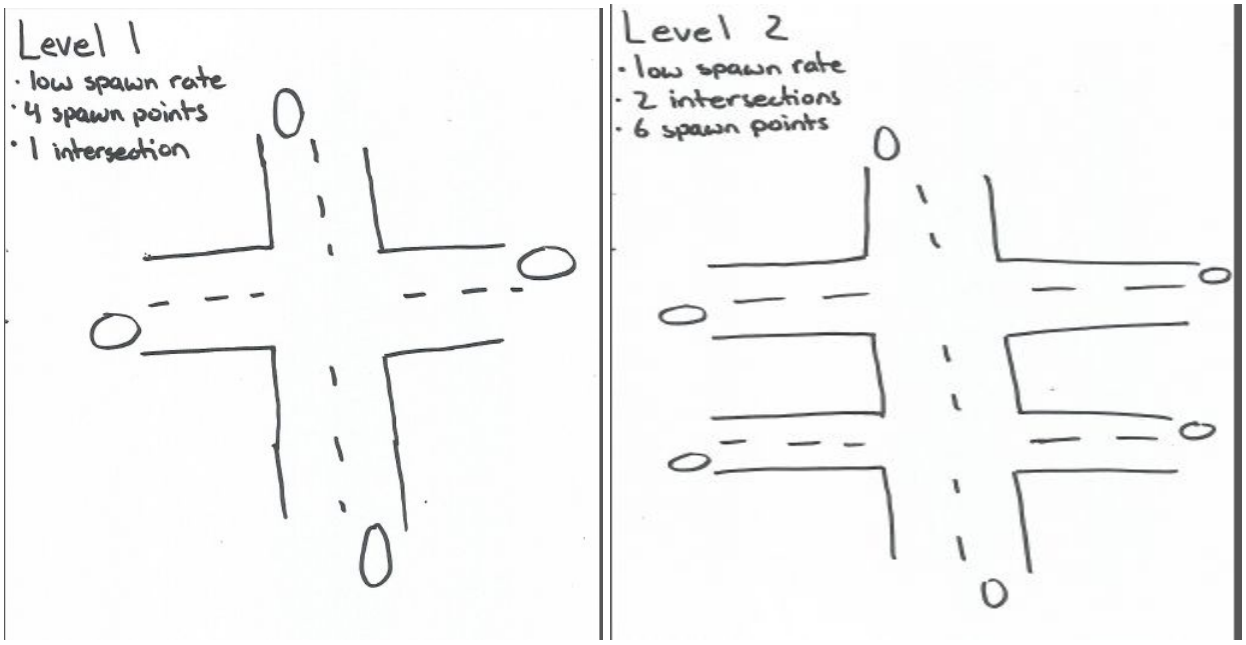
- **Level Design Layout Schematics -** Anthony Lam

  Level 1 - Low spawn rate/4 spawn points/1 intersection
  Level 2 - Low spawn rate/6 spawn points/2 intersections
  Level 3 - Low spawn rate/8 spawn points/4 intersections
  Level 4 - High spawn rate/8 spawn points/4 intersections
  Level 5 - *Extends* **Level 1** with a Controller to control traffic light delays
  Level 6 - *Extends* **Level 2** with a Controller to control traffic light delays
  Level 7 - *Extends* **Level 3** with a Controller to control traffic light delays
  Level 8 - *Extends* **Level 4** with a Controller to control traffic light delays

  Alternate Layout for Level 3/Level 4

  Original designs, but with shifted intersection.
  Spawn points will be slightly different. There will be two spawn points that will spawn cars in two different directions.

  Level 3 - Low spawn rate/10 spawn points/4 intersections
  Level 4 - High spawn rate/10 spawn points/4 intersections

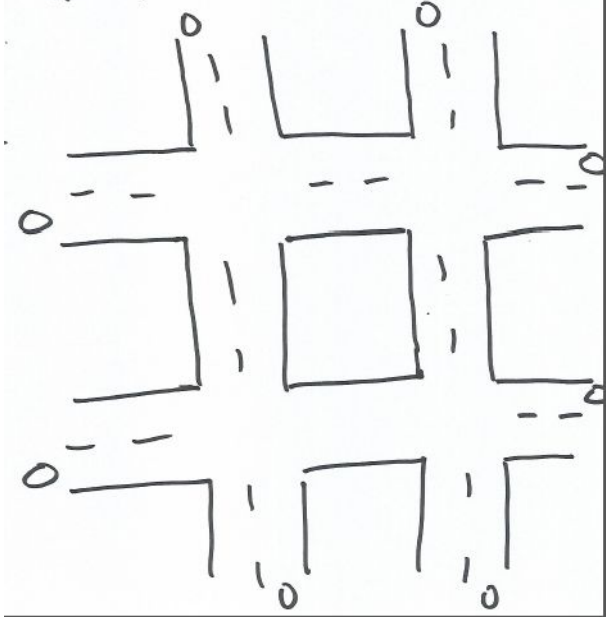


Level 3

- low spawn rate
- 4 intersections
- 8 spawn points

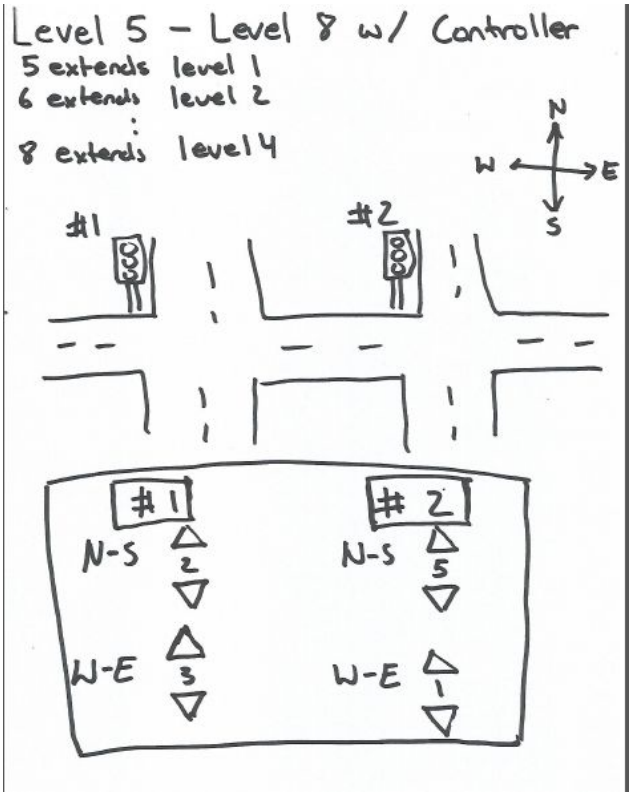
level 4

- high spawn rate
- 4 intersections
- 8 spawn points

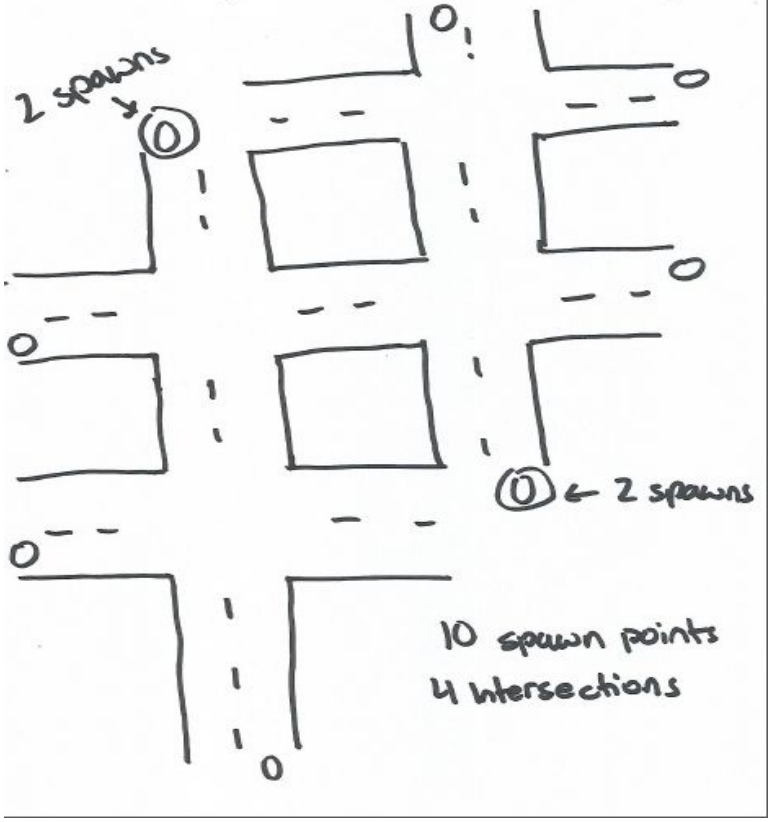


Level 5 – Level 8 w/ Controller

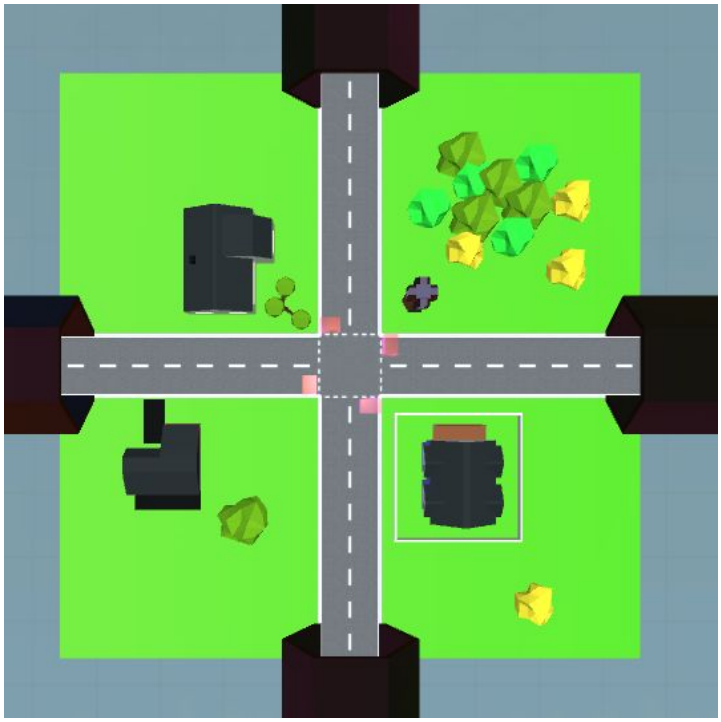
5 extends level 1
6 extends level 2
⋮
8 extends level 4

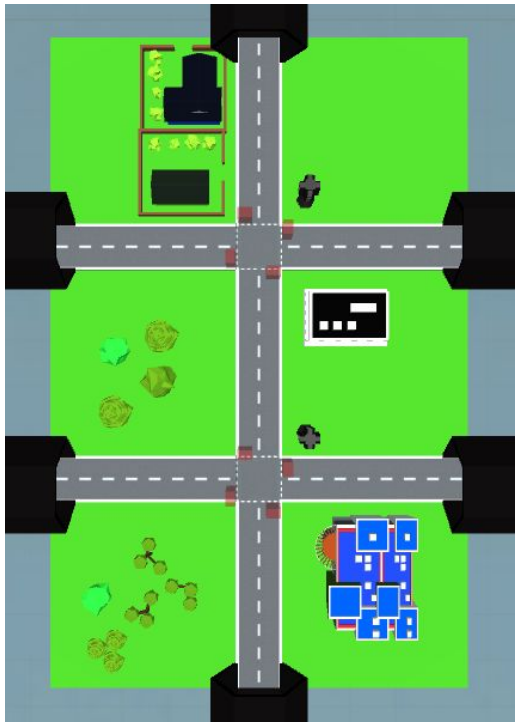


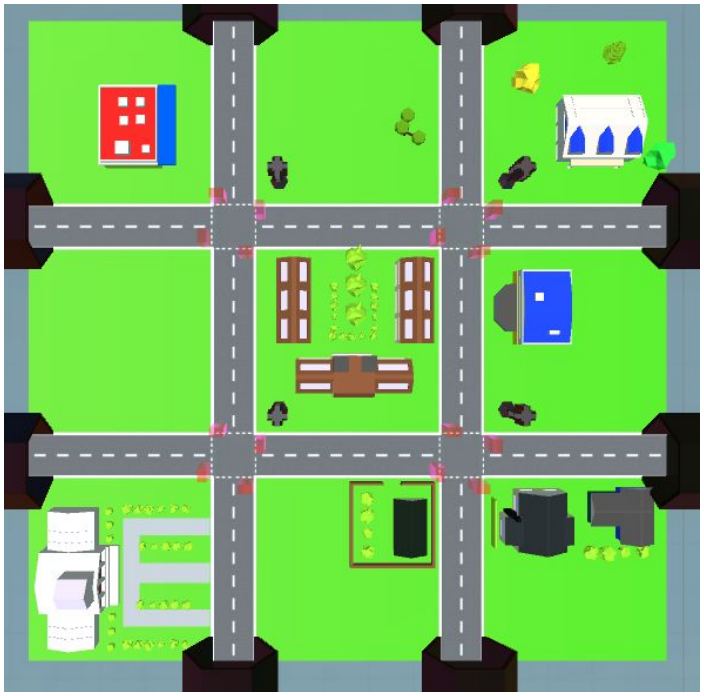
Alternate layout for Level 3 and 4



Implemented Level Screenshots:







Traffic Light - Mitchum Dillard

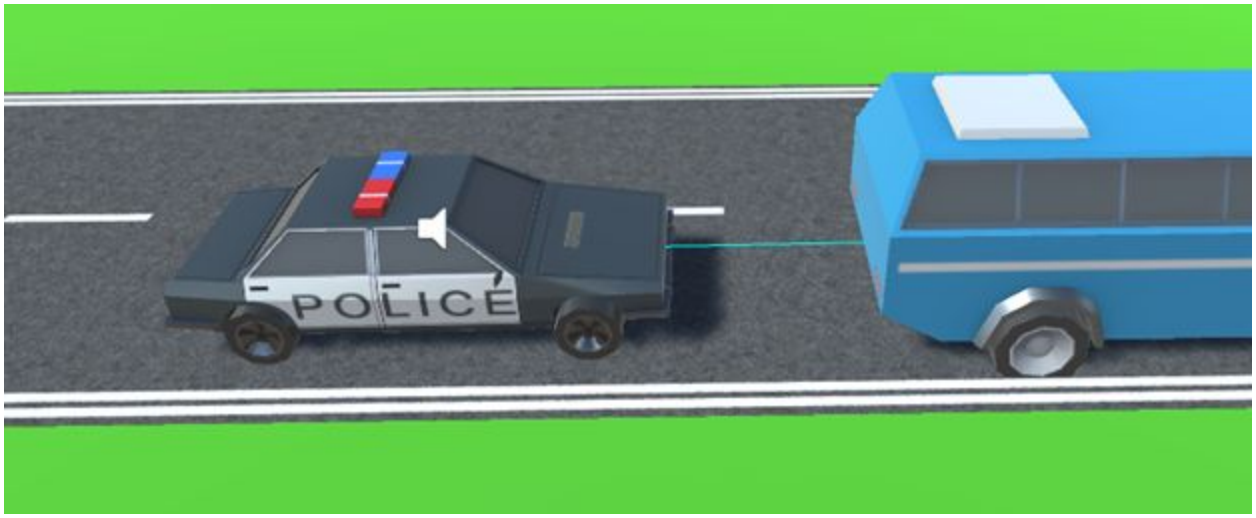
- Set up at each intersection stands a traffic light that controls the flow of traffic through its area. Set up as four radial entities the Traffic Light changes state when the user clicks on it. The Light entities are paired off to make the state change smoother, North paired with South and East paired with West.
- Each light is made up of three light objects: a Red, Green, and Yellow light set to activate under certain criteria. If the light is set to the Green state the intersection from that direction is allowed passage by removing a road blocker. Upon a click, the Green light deactivates and the Yellow light turns on. This places down the road blocker in the appropriate direction to begin restricting traffic from that direction. The Yellow light holds its state for one second before deactivating and being replaced by Red. The light then holds this state until the traffic light is clicked again. Upon a click in this state the light waits for one second before turning off and being replaced by a green light, simultaneously removing the traffic blocker.

- For each of these state changes the paired entities share states while the non-paired entities have the opposite state. In order to allow the light system to function on its own system and not be affected by the primary game clock; each light is set up so that on click it starts a Coroutine that manages its timings. By doing this we remove extra updates from the overall game update and allow the lights to function more smoothly.

```
public IEnumerator click()
{
    if (ignore) {
        yield break;
    }
    ignore = true;
    if (northGreen==true)
    {
        yield return new WaitForSeconds(yellowDelaySeconds); //stall light change when game paused

        greenLightS.GetComponent<Light>().enabled = false;  //Green -> Yellow Switch
        greenLightN.GetComponent<Light>().enabled = false;
        yellowLightS.GetComponent<Light>().enabled = true;
        yellowLightN.GetComponent<Light>().enabled = true;
        NorthWall.SetActive(true);                          //Wall blocker activation
        SouthWall.SetActive(true);

        yield return new WaitForSeconds(greenDelaySeconds); //wait for appropriate time before Y -> R
                                                            // and for R -> G switch.
        yellowLightS.GetComponent<Light>().enabled = false;
        yellowLightN.GetComponent<Light>().enabled = false; //Yellow -> Red Switch
        redLightS.GetComponent<Light>().enabled = true;
        redLightN.GetComponent<Light>().enabled = true;

        greenLightW.GetComponent<Light>().enabled = true;  //Red -> Green Switch
        greenLightE.GetComponent<Light>().enabled = true;
        redLightW.GetComponent<Light>().enabled = false;
        redLightE.GetComponent<Light>().enabled = false;
        EastWall.SetActive(false);                          //Wall Deactivation
        WestWall.SetActive(false);
        northGreen = false;                                 //check which Pair is active
        ignore = false;
    }
```

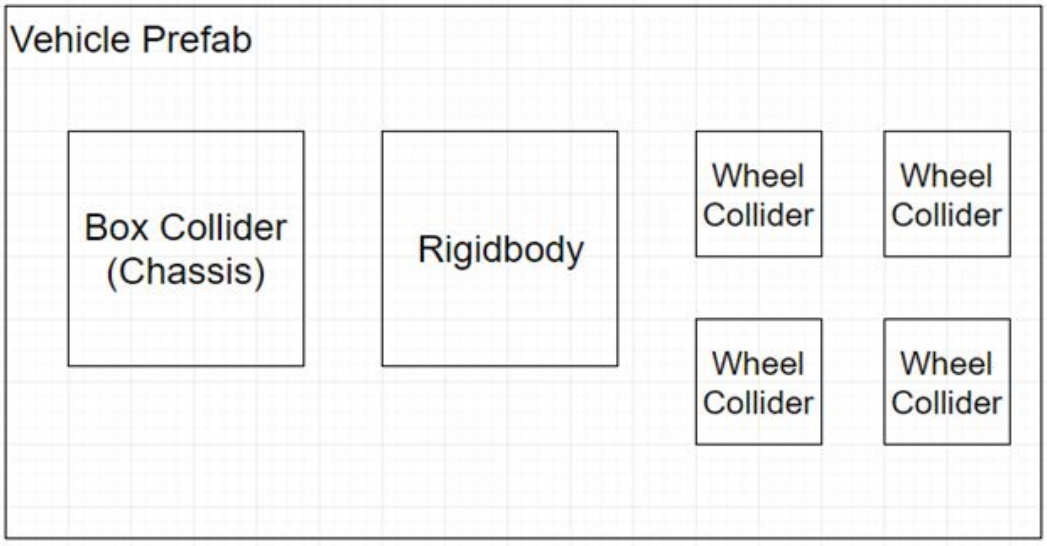
Collision Anticipation through Ray Casting – Tanner Wilson

- Unity provides a robust mechanism, colliders, to detect and handle physical interactions between objects. Each collider's size and its location (in scene coordinates) define some volume that an object occupies. Colliders were built into Unity project prefabs for each vehicle, physically approximating each vehicle's shape.
- In real-life traffic, collisions between vehicles are minimal; drivers never intend to crash into other drivers (or other objects). When in traffic, a driver anticipates their vehicle's future interactions. When a driver suspects his/her path might lead to a crash, the driver will change the velocity of the vehicle in some way to prevent it (often braking).
- Ray casting is used to implement anticipatory behavior for vehicles. A ray cast in Unity, *Physics.Raycast*, describes a line-of-travel from an origin point in a direction. A ray cast's length depends on what the ray reached – the ray will travel until it reaches a physical object that stops it (or a maximum specified length). Here, a physical object is just a Unity collider component. Each vehicle then can anticipate the need to brake by ray casting from itself towards the direction of travel. The screenshot below exemplifies this: the police car is ray casting to the vehicle in front of it; using the distance information from the ray cast, the vehicle decides to brake (*isBraking=true*).



Vehicle Braking – Tanner Wilson

- Unity provides its own physics model and related components; the Traffic Control project uses these components in vehicle prefabs. The following diagram describes the relevant physical components in a general vehicle prefab:



- For each car to brake, the *rigidbody* and all four *wheel colliders* must be controlled. The rigidbody of the vehicle holds the velocity and mass (momentum) information; the wheel colliders are specialized colliders for simulating wheel interactions on terrain. The wheel colliders provide simulated acceleration and braking through member variables *motorTorque* and a *brakeTorque* respectively.
- To determine if a vehicle should brake, **collision anticipation** is used (vehicles brake before they collide, leaving a gap). The maximum length of ray cast determines how far a vehicle *looks* ahead: the vehicle will anticipate collisions (and brake) from further away if the ray cast's maximum length is extended. The length of the ray is determined by the current velocity of the vehicle multiplied by a *stretch* factor (allowing braking distance to be fine-tuned). The ray cast length is clipped to a minimum length, *raycastLengthMin*, so that each vehicle always leaves some gap between it and the object:

```
//raycast length depends on current velocity; allows braking sooner for faster speeds.
float raycastLength = _rigidbody.velocity.magnitude * raycastBufferFactor;

//clip raycast to minimum length
if(raycastLength < raycastLengthMin){
        raycastLength = raycastLengthMin;
}
```

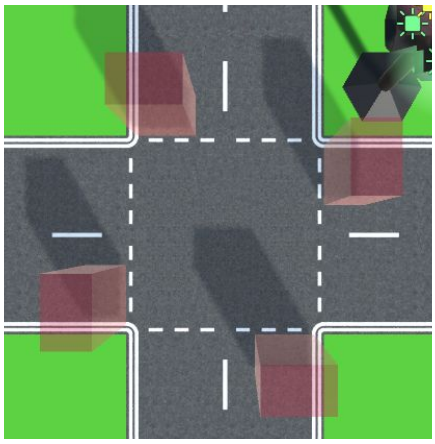
- Utilizing Unity's specialized wheel colliders does present an issue: there is no simple way to guarantee that a car brakes to a stop within a specified distance. Unity does not provide enough documentation or a method to accurately compute stopping distance for a

rigidbody with wheel colliders. Unity simulates slip effects for wheel colliders – if a car tries to stop too fast, it skids and continues moving. The wheel colliders are desirable for their accurate acceleration and steering properties, but for our application we must guarantee that vehicles can brake to a stop before colliding with intersection blockers or other vehicles.

- As a countermeasure to problems pertaining to stopping distance, if the ray cast used in collision anticipation returned a length less than *forceStopDist*, then the rigidbody's velocity is zeroed. While this can provide an unrealistic effect, it assures that vehicles behave as desired for the simulation. Fine tuning of collision anticipation or braking may remove the need for this functionality in the final build.

Traffic Intersection Blockers – Tanner Wilson

- Traffic Intersections take direct advantage of utilizing the previously described collision anticipation (since vehicles must anticipate stopping for red-light intersections the same way as stopping for another vehicle). Consider traffic behavior at an intersection: a red-light disallows traffic from entering from that direction; a green-light allows traffic to enter from that direction. Vehicles can then consider the entrance of an intersection with a red-light to be a physical obstacle: a car must stop for a red light the same way it stops physical obstacles (like cars or walls). The implementation of intersections within the Traffic Control game follows this idea: there are colliders (physical obstacles) placed at each entrance of each intersection. The standard collision anticipation and detection methods then apply to the entrances of intersections; by toggling a wall, one changes oncoming vehicle behavior from stopping (red-light) to continuing (green-light).

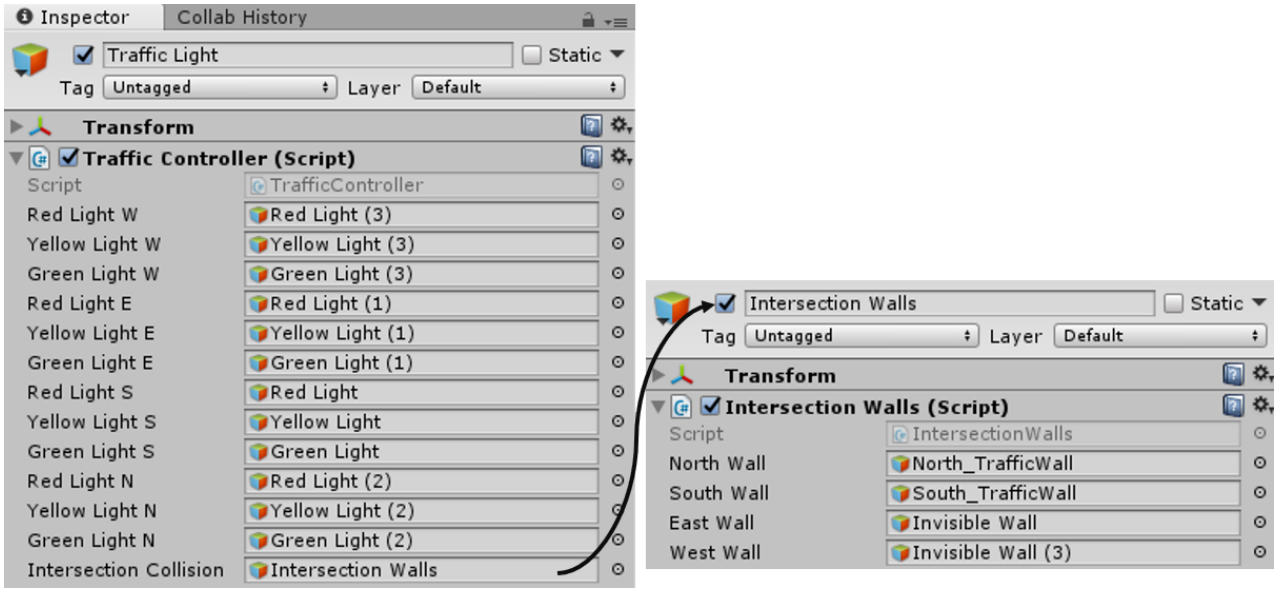


(Note: these colliders will be made 100% transparent in release builds)

- The Traffic Light object already tracks the state of each light for each direction of an intersection. Each direction, paired with its opposite, is either red, yellow, or green at any given moment. Consider how the use of intersection blockers correlates to the state of the traffic light (remember that a blocker being enabled means it stops traffic; a blocker being disabled means traffic ignores it).

- north/south is green → the north/south blockers are disabled
  east/west is red the east/west blockers are enabled
- north/south is red → the north/south blockers are enabled
  east/west is green the east/west blockers are disabled

- Unity provides a convenient analog to the above notion of enabling/disabling: when an object in a Unity scene is made inactive, almost nothing pertaining to the object is processed (including collider calculations). Thus, to enable a blocker, it is made active; to disable it, the blocker is made inactive.
- The tight cohesiveness between the traffic light object and the set of blockers allows the assembly of an inclusive prefab. When creating new maps, entire intersection system prefabs are placed in the editor, requiring no other changes before functioning. The Traffic Light object within each prefab contains references to its respective lights and a set of traffic blockers:



(the section on the **Traffic Light** object details specific scripting used on these objects)

Car AI - Jonathan Ma

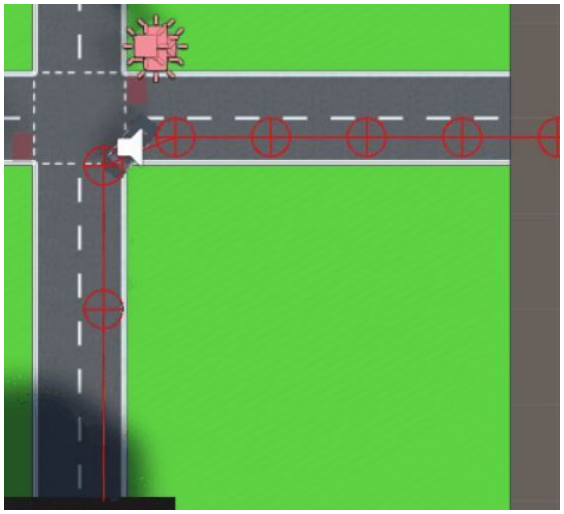
- Each vehicle consists of a body and wheels. Each wheel contains a wheel collider that performs the physics of the wheels driving. The speed that the wheel rotate is calculated by the diameter of the wheel collider multiplied by the RPM and 0.06. As the wheel colliders rotate, the car is propelled forward. A steering function calculates the angle of the car to the current unvisited node. When the car's position is less than 0.5 units from the current unvisited node, the node index is incremented to the next node.

```
private void ApplySteer()
{
    Vector3 relativeVector = transform.InverseTransformPoint(nodes[curNode].position);
    relativeVector /= relativeVector.magnitude;
    float newSteerAngle = (relativeVector.x / relativeVector.magnitude) * maxSteerAngle;

    wheelFL.steerAngle = newSteerAngle;
    wheelFR.steerAngle = newSteerAngle;
}
```

Pathing - Jonathan Ma

- A path consists of an empty gameobject that holds several empty gameobjects as children. The children acts as nodes in the path. A script takes the input of the path and runs through each child, adding each to a list. A vector is calculated of the distance from the car to the next node in the list. The paths used for this game are all in a singular line

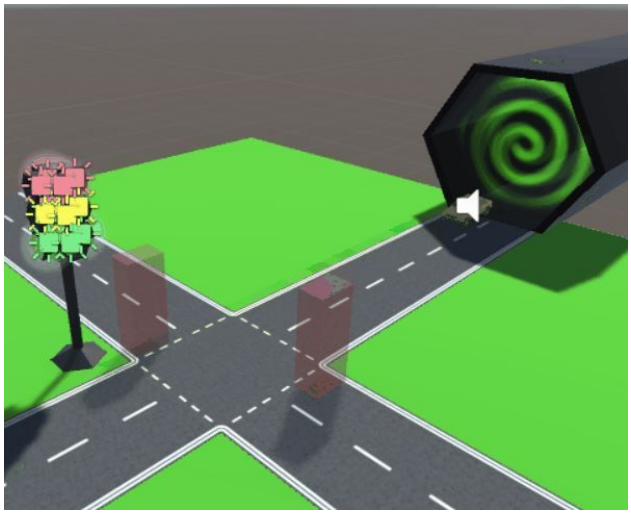
parallel to the road. If a node is translated, the car will calculate the angles and turn as much as the allocated max steering angle will allow.



Spawner - Jonathan Ma

- The spawner script will spawn a wave of objects after a specified time. Waves are set to 1 vehicle at a time and a random vehicle is chosen from the list of vehicles. The vehicle is spawned as a instantiated clone of a prefabricated game object.

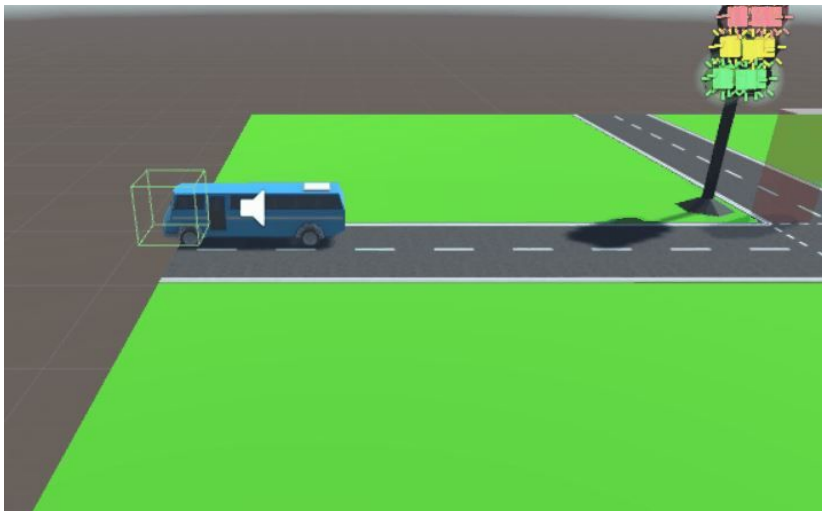
```
private void Update() {
    // Add to the timer and check if it's time for a new wave
    spawnTimer += Time.deltaTime;

    if (!spawnOnDestroy && spawnTimer >= spawnTime) {
        // Time to spawn a wave of objects!
        SpawnWave();

        // Reset the timer for the next wave
        spawnTimer = 0f;
    }
}

private void SpawnWave() {
    // First, determine how many objects we are going to spawn in this wave
    int waveCount = Random.Range(minWaveCount, maxWaveCount + 1);   // Random.Range(int, int) excludes the upper bound, so we increase by one to compensate

    for (int i = 0; i < waveCount; i++) {
        // We need to spawn one object in this iteration of the loop

        // Let's determine which objectPrefab to spawn by picking a random one out of the array
        int index = Random.Range(0, objectPrefabs.Length);
        GameObject objectPrefab = objectPrefabs[index];

        // Now we can determine a random position to spawn the object at
        float x = Random.Range(-spawnRange.x * 0.5f, spawnRange.x * 0.5f);
        float y = Random.Range(-spawnRange.y * 0.5f, spawnRange.y * 0.5f);
        float z = Random.Range(-spawnRange.z * 0.5f, spawnRange.z * 0.5f);
        Vector3 spawnPosition = transform.position + new Vector3(x, y, z);

        // With an object and a position, we can now spawn it with Instantiate()
        GameObject c = Instantiate(objectPrefab, spawnPosition, transform.rotation);
        c.SetActive(true);
    }
}
```



Despawner - Jonathan Ma

- The despawner is a simple box collider with a script to detect if a gameobject with the tag Car collides with it. If one does, it will destroy the gameobject.

```
private void OnTriggerEnter(Collider other)
{
    print("Detected collision between " + gameObject.name + " and " + other.name);

    if(other.gameObject.CompareTag("Car"))
    {
        Destroy(other.gameObject);
    }
}
```



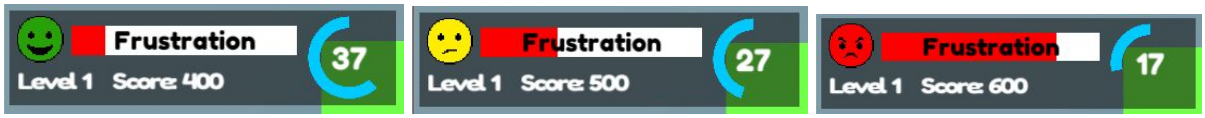
Level Information Storage – Tanner Wilson

- LevelInformation is a static class that stores general information regarding the levels in the game. It is meant to be a central location for retrieving stage configuration (such as level progression and level properties) from other scripts. LevelInformation also provides data about the currently loaded stage (level name, game mode, and similar related properties).
- The GameMode enumeration in LevelInformation describes the two possible game modes: simulation and arcade mode. Each level is assigned one of the two modes.
- arcadeMode is a dictionary object indexed by Unity scene names (a string). Lookups can be performed using the current level name to yield a levelProperties object describing the following:
    - is the traffic light control panel available?
    - is the stage simulation or arcade mode?
    - what is the next level to load
- spawnRates is similar dictionary object describing the rate of traffic creation for each tunnel in each stage (so each tunnel, in each stage may have a unique rate of creating traffic). Each level name key yields an array of of time values representing the base time (and variance) of spawn rates for each tunnel. The resulting time between vehicle spawns is therefore:

    (*intersection base spawn rate time*) ± (*level spawn time variance*)

Arcade Mode UI – Jonathan Ma and Tanner Wilson

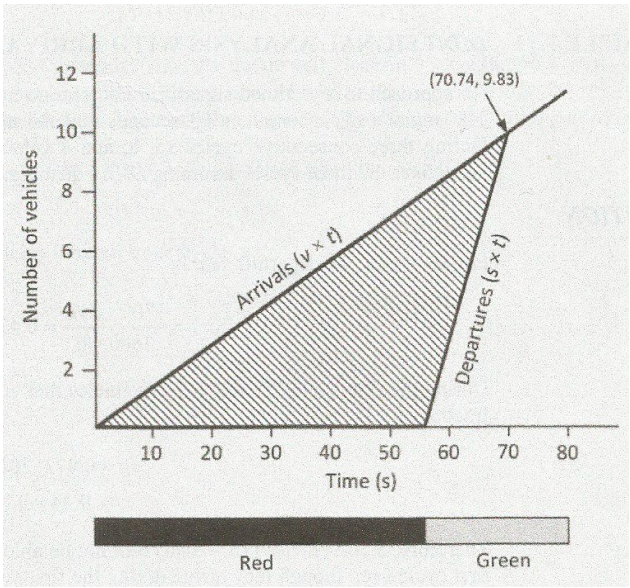
- The information displayed on arcade-mode stages (frustration, remaining stage time, score, and current level) is done so with an intuitive UI panel in the top left corner of the display area.

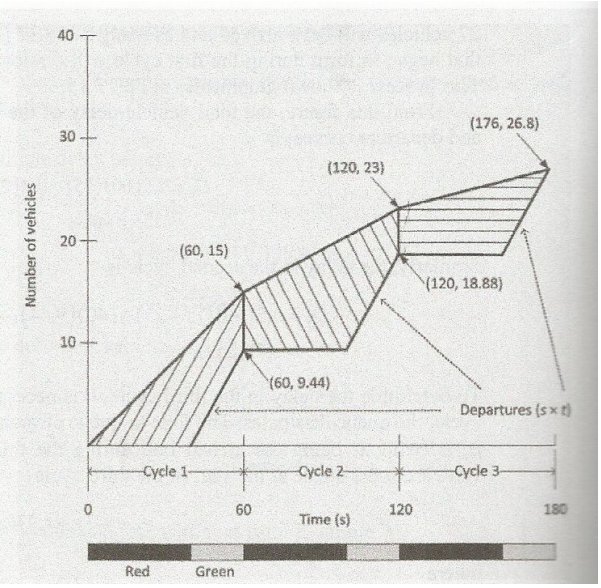


- The panel is not displayed in simulation mode.

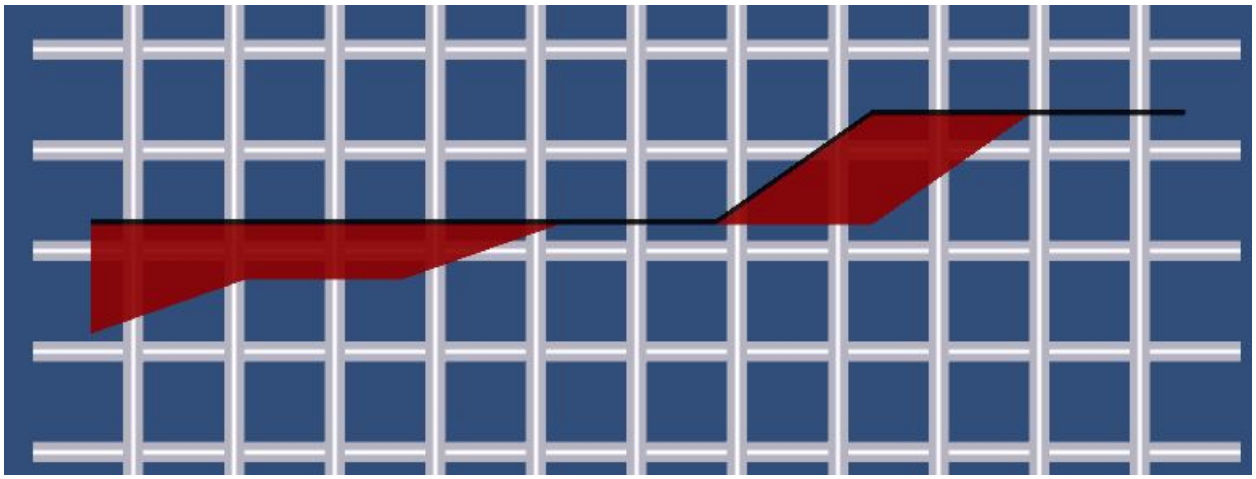
Graph System - Perry Mills

- MyTrafficKontrol featured graphs which modeled total delay and queue length over time. In addition to these two simple graphs, this software will include a more complex graph of traffic arrival/departure over time. This model better conveys the flow of traffic through an individual intersection. Less efficient intersections are modeled with more shaded area between the slopes. Textbook depictions of the model are shown below [9]:



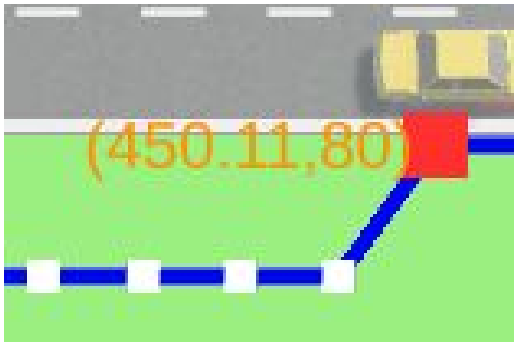


- This form of graph can be drawn in real time as the simulation runs. Below is a proof-of-concept image of the model in Unity, using the MeshChartFree [10] asset from the unity. The graph is constructed using a stacked line graph, in which the area below the first slope is transparent.



- In order to make the graph system more user friendly, it was constructed from scratch with the goal of interactivity in mind. Using a viewport, the graph is displayed as an overlay only when necessary - when minimized there is minimal draw on graphics resources. The graph system was built using Unity's 3D objects and mesh rendering feature, as shown below:

- Unity's UI components and raycasting function are used to provide mouse-over interaction, in order to display the traffic data at a given point. The user can select a vertex, which highlights the point and toggles the display of the coordinates.



- The Red/Green cycle bars which indicate the current state of an intersection are similarly drawn using meshes. The light cycle chart bars are constructed using the exact timestamps of light changes in the game, while the other data points are generated by making routine polling calls to the intersections.

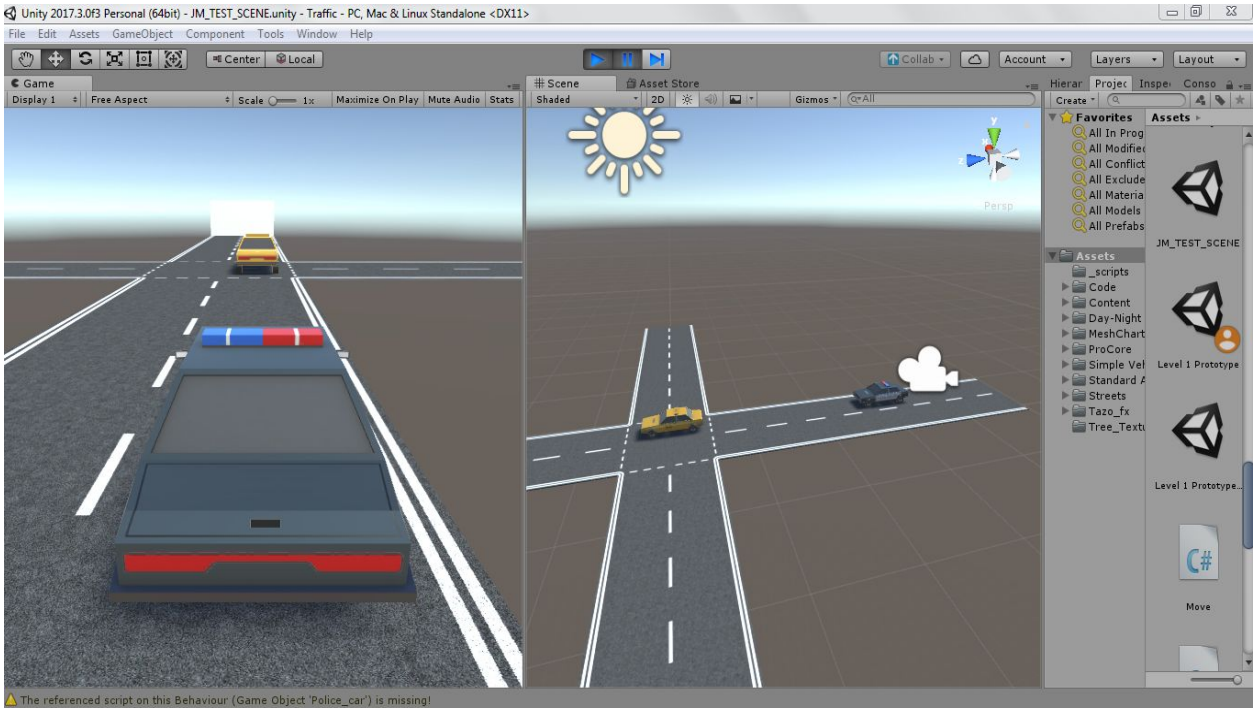
Remote Controller - Mitchum Dillard

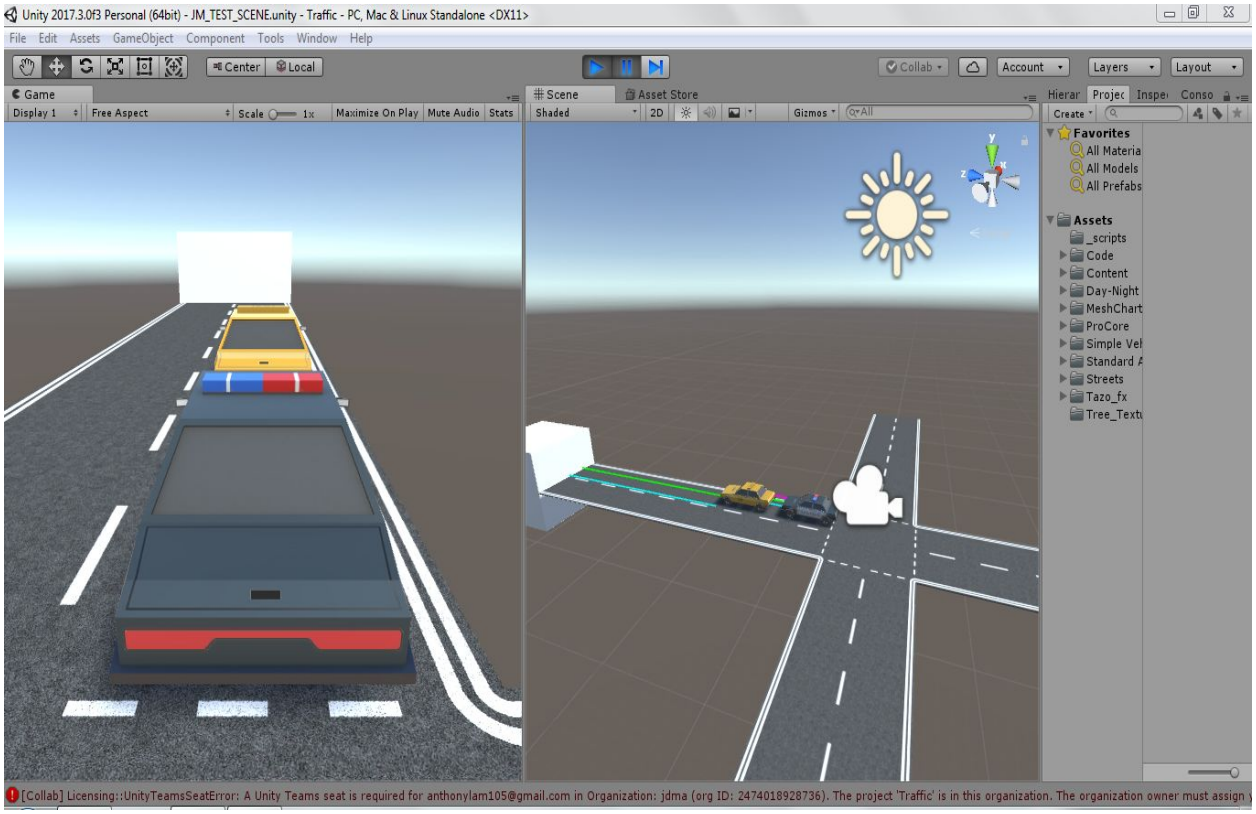
- The remote controller is a play aspect that comes into the game around level 5 and is a prominent feature in the Simulation Mode. Designed as a series of interactive panels they allow the user to set the light duration in both North-South and East-West directions individually for a chosen intersection.

- To allow for an intuitive user experience a green check mark is shown next to the chosen intersection. Coupled with this are indication arrows to show which buttons set N/S and E/W direction timers.

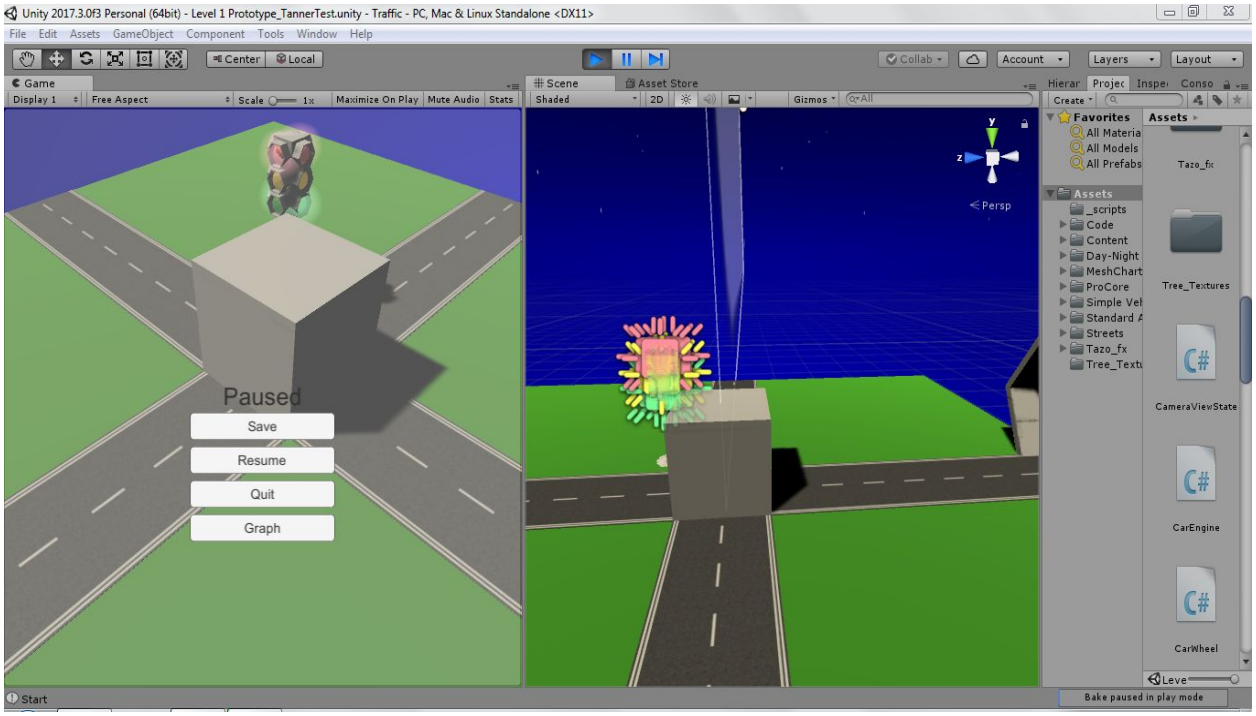
Game Screenshots (Early Versions)

- Working sensors that will detect objects in front of the cop car and cause it to stop.



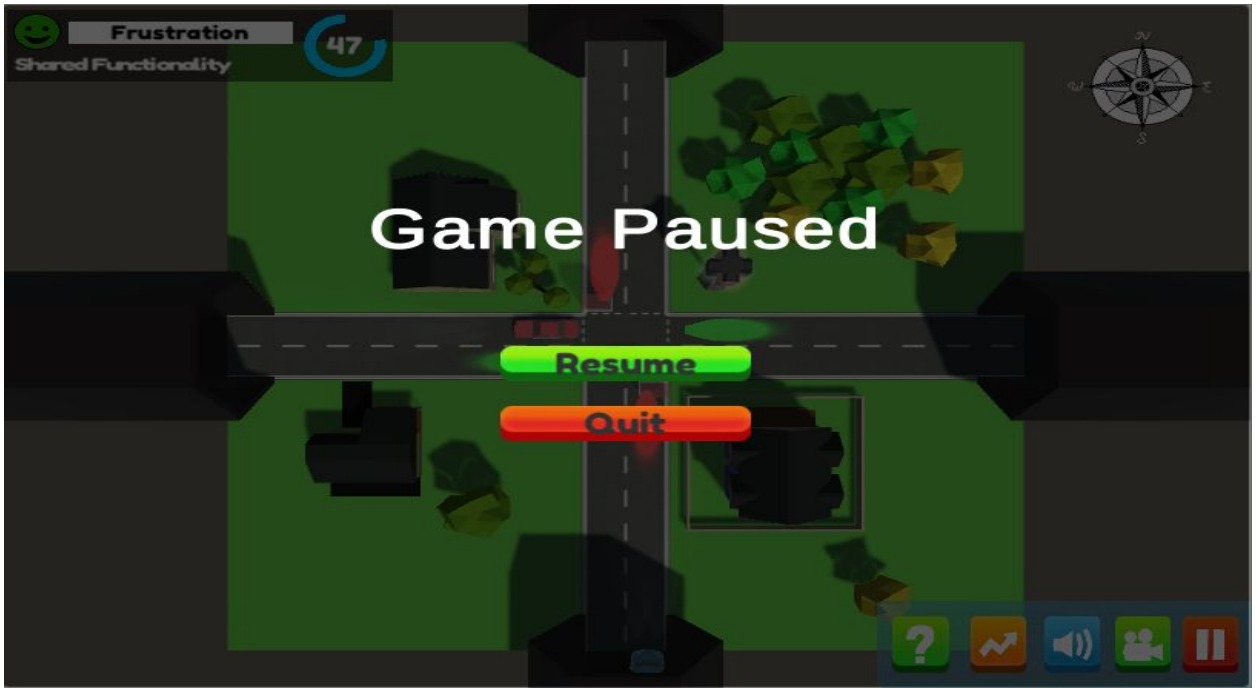


- Pause Menu that is toggle by pressing the ‘esc’ key to pause the game.

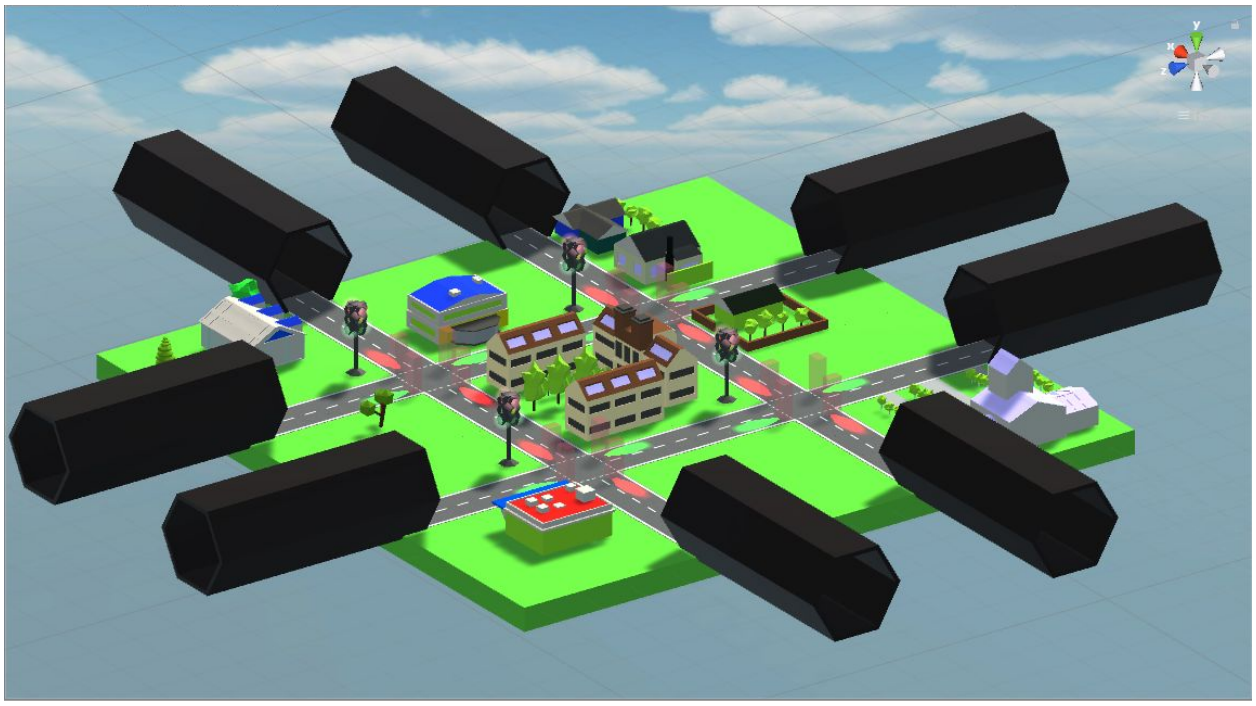


Game Screenshots (Current Version)

- New Pause Menu that is triggered by a pause button instead of “esc”.

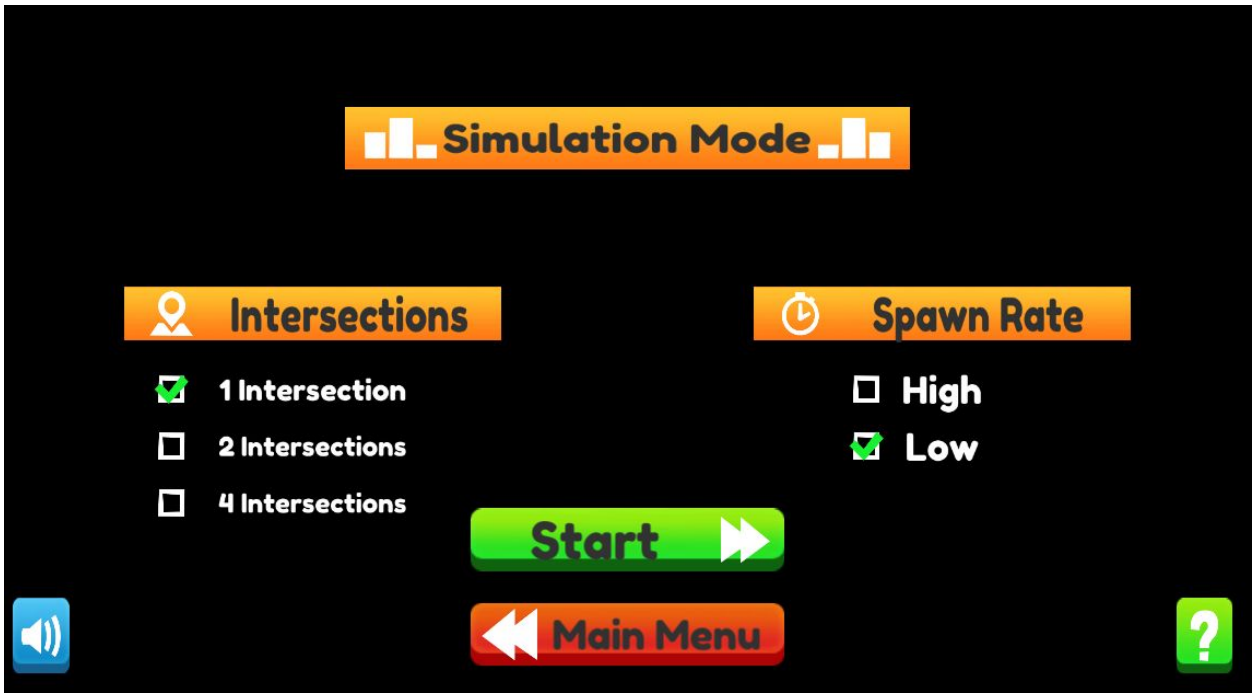


- Level 7 scene, late-development



- Arcade mode level-selection menu

- Simulation mode level-selection menu



Game Script - Rodrigo Sanchez

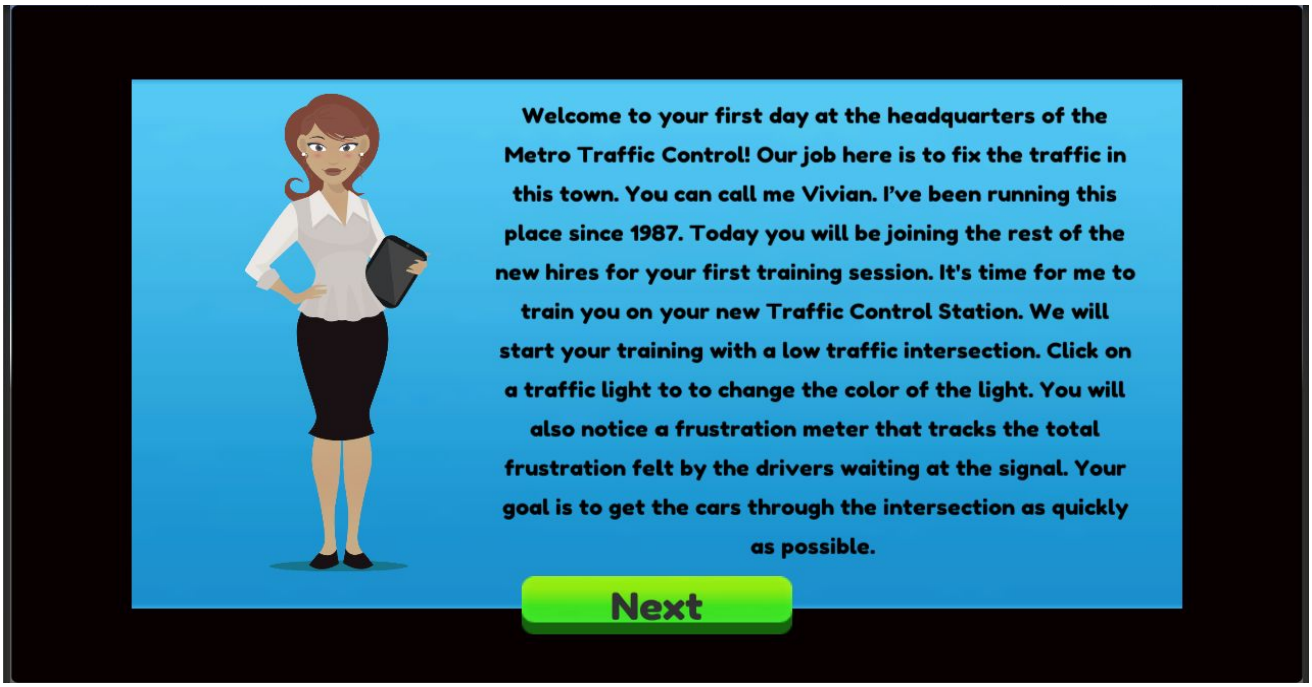
In addition to the other parts of the game that were outdated, the game script needed a major overhaul. The original script was unrealistic and did not achieve its purpose of teaching k-12 students about traffic signals. As a result the entire script needed to be rewritten with its intended audience in mind in an attempt to achieve its purpose and make it easier to understand. Time was spent going through every text slide in the game and determining whether it was effective in teaching about traffic lights. In addition time was also spent evaluating the script and its effectiveness in teaching the games mechanics in a clear and understandable way. The portions of the script that were deemed adequate were kept and the portions that were not up to par were rewritten. In addition to rewriting the script we also remade the character graphics for each of the story scenes.

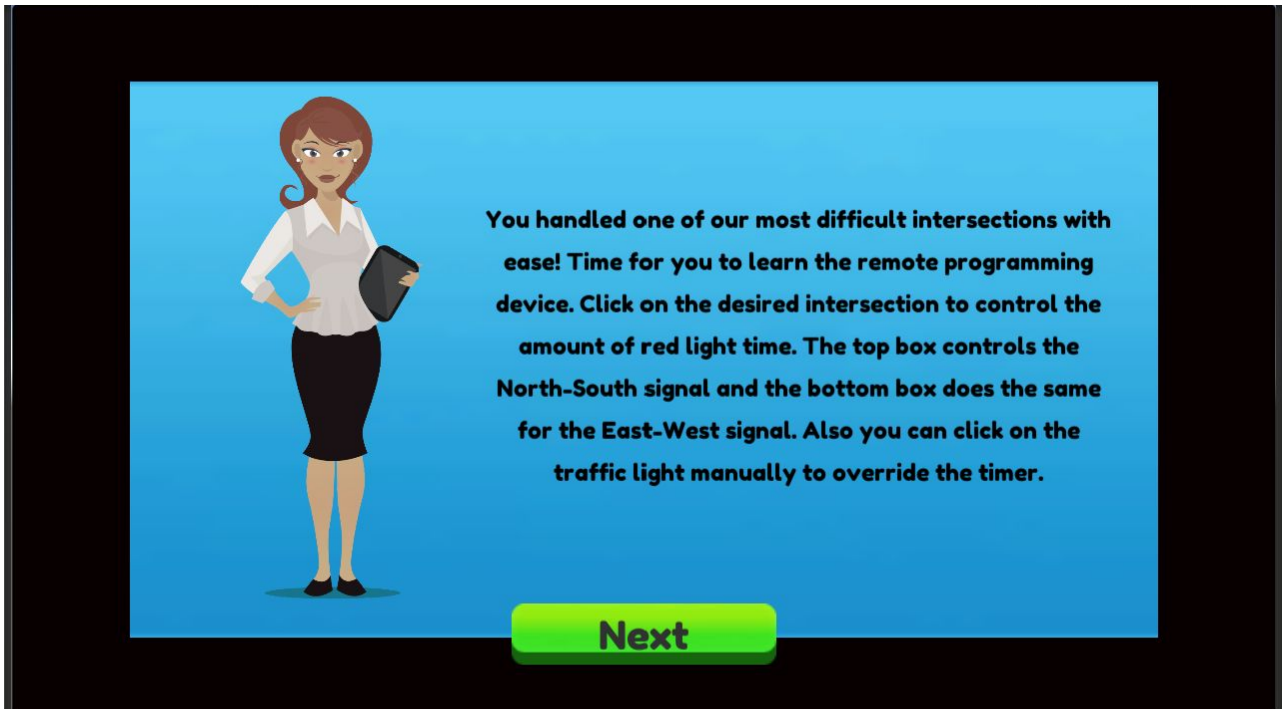
Script Samples

Intro Story: "Welcome to your first day at the headquarters of the Metro Traffic Control! Our job here is to fix the traffic in this town. You can call me Vivian. I've been running this place since 1987. Today you will be joining the rest of the new hires for your first training session. It's time for me to train you on your new Traffic Control Station. We will start your training with a low traffic intersection. Click on a traffic light to to change the color of the light. You will also notice a frustration meter that tracks the total frustration felt by the drivers waiting at the signal. Your goal is to get the cars through the intersection as quickly as possible. "

Level 1 End Story: "Great job on your first intersection! Thankfully this is one of the few low traffic intersections in this town. As i'm sure you have noticed there is an extreme traffic problem in this town. The traffic problem is mainly a result of poor traffic light efficiency. Let's continue your training in a neighborhood with a mild traffic issue."

Story Scene and Script Pictures





Lessons Learned - Anthony Lam - Rodrigo Sanchez

There were many valuable lessons that we all learned throughout the completion of Transit Trouble. The first of many was learning how to properly use the Unity game engine. The Unity game engine software offers many tools for us to create the game. Multiple members of the capstone group had never used Unity before and we had to spend time following tutorials and using Unity in order to gain an understanding of how to use it. Peer programming technique was very helpful for testing and debugging the game. Another major lesson we learned in the progress was how helpful group meetings are. Group meetings helped us get together and establish current progress and goals. In addition within these meetings large amounts of progress was made such as pushing version updates of the game quickly for deadlines. A final lesson we learned was the importance of working together with a sponsor. This was an important lesson to learn since when working on large projects there may be sponsors and other people you need to work with who have a different idea of an end product than what you initially envisioned. Adequate communication proved to be a valuable resource in our project's progression.

Potential Impact - Anthony Lam

Introducing concepts about traffic to young students from K-12. These young students will learn about traffic delays and how different timings can create different results in the flow of traffic. In Simulation mode, the students will be able to experience what gridlock is and how it happened based on the timings that they experimented with. Older students can learn how to analyze traffic systems using the built in graphing system. The interactive graph system can give students a better understanding on traffic delays. Fun environment to learn about traffic systems for little kids.

Future Work - Mitchum Dillard

Moving forward there are still steps that could be taken to improve our product. The graphing system needs a few final tweaks to ensure proper data collection and the exportation of data. On top of this the game's UI and story system need a final polish to allow it to flow smoothly without causing the user any confusion. As far as game function goes, the game works well but could use a fine toothed comb over to remove any redundant or unnecessary code. This shares two fold importance as a more optimized game will work better inside of a WebGL version. We will also continue working with Dr. Hernandez until the game finds a suitable place on the web. Finally, as the game is put under more and more stress bugs will inevitably appear and will need fixing as they are discovered.

4.3 Risks - Jonathan Ma

Risk	Risk Reduction
Converting from Java to Unity may cause potential issues, especially in a multi-platform system	Large amount of error testing and constant attention to detail during the conversion.
Licensing issues	Original projects are supposedly open source and redesigned program will be used for education purposes; will check with advisor.
Inability to implement all requested features	Estimate approximate time required for each task and confirm with sponsor about priority and optional features.
Unexpected bugs and awkward user interface	Alpha and Beta test final prototype for feedback.
Missed deadlines	Scheduled warnings, discipline in adherence to task timeline, and strong team dynamics
Advisor does not approve of prototype or features	Have open consistent contact with advisor about implemented features and demo prototype.

4.4 Tasks – Full Group Discussion

1. Initial meeting with advisor
2. Study previous games' features
3. Decide what features are necessary
4. Conceptual Design
5. Confirm with advisor of decided features to keep and features to add
6. Decide method of implementation
    - Consider advantages and issues with existing libraries and software (multi-platform support, deployment, ease of use, etc.)
7. Learn basic Unity interface
8. Learn basic Unity features

9. Build prototype game in Unity
    - Build a base level game mockup to test entity interactions and work out the best forms of sprite control.
10. Learn civil engineering background for graphs
11. Build base functionality of game
    - Anchor the base structure of the game, this will be the primary framework for all user interactions and behind the scenes game control
12. Level Design
    - Visual and difficulty design for 8-9 levels in the game. Will consist of street layouts and vehicle spawn rates
13. Build working prototype of game
    - Usable version of the game in a base sense. Lights will be changeable, and vehicles will interact appropriately. This version will only contain a minimum number of levels
14. Test prototype features with advisor
    - Present prototype to advisor and either get approval or feedback on what needs to be changed
15. Implement next levels of game
    - Add the other levels into the game and ensure they are functional
16. Test levels and features for compatibility issues
    - Bug testing and repair for the new levels
17. Implement features from advisor
    - Add the more detail oriented features requested by advisor. This would include the graphing system and improved performance index.
18. Update graphics from old games
    - Rework of the visual structure to ensure the game is pleasing to the eye, using the Unity Asset Store
19. Implement attractive user interface
    - Clean up user interface to provide a sleek and enjoyable user experience
20. Early project testing
    - Alpha and Beta testing through internal and external sources to comb for bugs left in the game
21. Export to different platforms ensuring proper functionality
    - Export to WebGL, Windows, Mac, and Linux versions.
22. Final advisor check, project delivery

4.5 Schedule – Full Group Discussion

Tasks	Dates	Complete	Assigned
Initial meeting with advisor	10/19	✔	All
Study previous games' features	10/22	✔	All
Decide what features are necessary	10/30	✔	All
Confirm with advisor of decided features to keep and features to add	10/31-11/5	✔	All
Decide method of implementation	10/31-11/5	✔	All
Learn basic Unity interface	11/5-11/18	✔	All
Learn basic Unity features	11/19-12/2	✔	All
Conceptual design	12/3-12/16	✔	All
Build prototype game in Unity	1/16-2/5	✔	Mitchum, Jonathan, Tanner
Build Road Structure / Prototype of Level	1/16-1/29	✔	Jonathan
Prototype Traffic Signals	1/16-1/29	✔	Mitchum
Prototype Traffic Spawner	1/16-1/29	✔	Tanner
Prototype point system	1/29-2/5	✔	Jonathan
Prototype audio system	1/29-2/5	✔	Mitchum
Prototype timer system	1/29-2/5	✔	Tanner
Learn civil engineering background for graphs	1/16-2/5	✔	Anthony, Perry, Rodrigo, Jonathan
-Meeting with advisor	1/22	✔	Anthony, Perry, Jonathan
-Prototype data/graph system (saving files)	1/29-2/5	✔	Anthony, Perry
-Prototype Grading System	1/29-2/5	✔	Rodrigo
Build base functionality of game	1/16-1/29	✔	Anthony, Perry, Jonathan
Level Design	2/5-3/5	✔	Anthony, Perry, Jonathan
Draft Story for Arcade Mode	2/5-2/12	✔	Mitchum, Tanner
Revise Story	2/12-2/19	✔	Jonathan, Perry, Rodrigo
Finalize Story	2/19-2/26	✔	Anthony, Rodrigo
Build working prototype of game	2/19-3/5	✔	Mitchum, Tanner, Rodrigo

Test prototype features with advisor	3/5-3/12	✔	All
Implement next level of game	3/12-3/19	✔	Anthony, Mitchum, Perry
Test level and features for compatibility issues	3/12-3/26	✔	Tanner, Jonathan, Rodrigo
Implement continuous statistic tracking	3/19-3/26	✔	Mitchum, Perry, Anthony
Build graph interactivity	3/19-4/2	✔	Perry
Implement features from advisor	3/19-4/2	✔	Mitchum, Jonathan, Anthony
Implement cooperative intersection printable	3/19-4/2	feature dropped	TBD
Update graphics from old games	3/26-4/9	✔	Tanner, Rodrigo, Perry
Implement attractive user interface	4/6-4/16	✔	Mitchum, Jonathan, Anthony
Alpha Testing	4/16-4/23	✔	All
Beta Testing	4/23-4/30	~	All, Outside Testers
Export to different platforms ensuring proper functionality	4/27-5/4	✘	All
Final advisor check	5/4-5/9	✘	All

4.6 Deliverables - Anthony Lam, Tanner Wilson

- Source Code and Build Resources: All build files and instructions necessary to build the project source into its working executable form
- Preliminary Proposal: early project definition describing initial plans (subject to change).
- Final Proposal: Finalized, precise project definition including exact goals and requirements of finished product. This report will include high-level and low-level conceptualizations and plan regarding the end-product.
- Cumulative package exceeding the features of both original games: neatly-packaged executable versions as well as web-browser enabled.
- Test and Program Design Documentation: detailed record of the algorithms, logic, product-testing, rationale behind design decisions, and instructions as to how to play the game
- Final Report: Report covering the process of our project and finalized product manual

5.0 **Key Personnel** - Group Contribution

Jonathan Ma - Ma is a Senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed a Game Design course and gained basic skills developing in Unity. He has built base functionality and structure of the game as well as implement initial functionality of vehicles, spawners, and cameras. He has also created the pixel sprites for the frustration meter and compass rose. He implemented the credits scene to animate and organized a progress list for the team.

Anthony Lam - Lam is a Senior Computer Science major in the Computer Science and Engineering Department at the University of Arkansas. He has completed Paradigms with Java experience. He has experience in game development with Apple's operating system. He worked with Java code and parse it with Swift code for Apple devices. He created and worked on the user interface for navigating through scenes and levels. Examples are the Main Menu scene, Simulation choice selector scene and Arcade Mode level selector scene.

Tanner Wilson - Wilson is a Senior Computer Science major in the Computer Science and Engineering Department at the University of Arkansas. He holds a Software Engineering internship and has accepted a full-time position with Marshalltown Co. locally. Tanner is largely responsible for collision anticipation and traffic intersection blocker functionality; he also performed various testing and debugging tasks as they arose.

Perry Mills - Mills is a Senior Computer Science major in the Computer Science and Engineering Department at the University of Arkansas. He has worked as an Information Services Intern at J.B. Hunt Transport and has developed a bioinformatics tool for use in a UAMS research lab. He helped design and implement the game which won the J.B. Hunt Hackathon in February 2017. Mills took the lead on the graph system implementation. He also helped design the scene control flow.

Rodrigo Sanchez - Sanchez is a Senior Computer Science major in the Computer Science and Engineering Department at the University of Arkansas. He has completed Programming Paradigms with Java experience. He has experience in application development for Android operating systems. He was responsible for updating the story from Gridlock Buster to be used in the final version.

Mitchum Dillard - Dillard is a Senior Computer Science major in the Computer Science and Engineering Department at the University of Arkansas. He interned at Consumer Testing Labs in 2015 as technical support. Spent 2016 summer teaching programming to children ages 8-16

using Java and Unity. He is currently employed at Helen Tyson Middle School teaching robotics and STEM classes. Project responsibilities included implementation of source code, design concept and point of contact for advisor. He took the lead on the development of the traffic light control systems and User Interface update.

Dr. Sarah Hernandez, University of Arkansas Civil Engineering Professor - Sarah Vavrik Hernandez serves the College of Engineering as Assistant Professor at the University of Arkansas in the Civil Engineering Department. Her research focuses on transport systems engineering, freight transportation data sources and needs, and intelligent transportation systems. Her teaching interests include transportation planning, transportation data analysis, and transportation engineering.

6.0 Facilities and Equipment - Rodrigo Sanchez

Equipment -
Unity Game Development Engine - Unity is a cross-platform game engine developed by Unity Technologies, primarily used to develop video games and simulations.

Unity Asset Store - A collection of assets housed by Unity, some free and others purchasable, provides easy access to tools that could take excess time or skill to produce.

Assets Used -

Simple Modular Street Kit by Jacek Jankowski (Used for Streets)
https://www.assetstore.unity3d.com/en/?stay#!/content/13811

Simple Cars Pack by MyxerMan (Used for Cars)
https://www.assetstore.unity3d.com/en/?stay#!/content/97669

Simple UI by Unruly Games (Used for Button UI)
https://www.assetstore.unity3d.com/en/?stay#!/content/103969

Lowpoly Modern City Buildings Set by karboosx (Used for Buildings)
https://www.assetstore.unity3d.com/en/?stay#!/content/64427

Lowpoly Modern City Decorations Set by karboosx (Used for Trees)
https://www.assetstore.unity3d.com/en/?stay#!/content/66070

TextMesh Pro by Unity Technologies (Used for Button Texts)

https://www.assetstore.unity3d.com/en/?stay#!/content/84126

Outside Resources
Piskel - Free online sprite editor
https://www.piskelapp.com/

Car Horn - Klaxon.wav by davidou - freesound.org
https://freesound.org/people/davidou/sounds/88465/

Civil Engineering textbooks - provided by project advisor

- Principles of Highway Engineering and Traffic Analysis by Mannerling & Washburn

Common Computer Operating Systems (used in development and/or testing)

- Windows
- MacOS, iOS
- Linux variants (Ubuntu, Android, etc.)

Web Browsers (testing and development)

- Firefox, Chrome, IE, Edge, Safari

Main Menu (Take Four) and Game Music (Steppin' out) - Purple-Planet Music Jazz - Royalty Free Music
http://www.purple-planet.com/jazz/4583971402

Credits Scene Music - Bensound
https://www.bensound.com/royalty-free-music/track/the-jazz-piano

Vector Assets
https://www.freepik.com/free-vector/businesswoman-illustration_776477.htm#term=business

7.0 References

[1] Link for a list all Games, street.umn.edu/games.html
[2] Link to the Gridlock Buster Game, http://www.its.umn.edu/GridlockBuster/
[3] Using Innovative Tools for Public Outreach,
http://www.cts.umn.edu/sites/default/files/files/sessions/17-baas.pdf
[4] Gridlock Buster Traffic Game (with more info),
http://www.cts.umn.edu/education/prospective/gridlockbuster

[5] Gridlock Buster Factsheet, http://www.its.umn.edu/Publications/factsheets/documents/gridlock.pdf
[6] Unity, https://unity3d.com/
[7] Unity Asset Store, https://www.assetstore.unity3d.com/en/
[8] Simple Cars Pack, https://www.assetstore.unity3d.com/en/#!/content/97669
[9] Mannering, F. L., & Washburn, S. S. (2013). *Principles of highway engineering and traffic analysis* (5th ed.). Hoboken, NJ: John Wiley & Sons, Inc.
[10] MeshChartFree, https://www.assetstore.unity3d.com/en/#!/content/18196