# COMP3329 Game Report
# Mini Courier

Group 24

Chan Tik Shun 3035536553

# Content

# Background

There are numerous simulation games on the market nowadays. For example, Euro Truck Simulator 2, Farming simulator 2017, Transport Fever 2, Planet Coaster, etc. These simulation games all have a specific focus and deliver an immersive experience to players who want to be a truck driver, farmer, public transportation managers and theme park owner. However, we find that there are still not many simulation games that allow player to be a delivery man. We decide to produce a simulation game that allow players to be a courier and fulfill their curiosity of the duties of couriers. We hope that players from playing our games, they will understand the responsibilities of couriers and appreciate their contribution to our society.

# General Information

Mini Courier is a simulation game. It is a minimalistic 2D single-player game. In the game, player will become a courier. They will drive a vehicle of their choice and deliver cargoes to different places.

At first glance, being a courier may seem easy and boring. It is actually quite difficult to manage all the deliveries and deliver cargoes to different places efficiently. Couriers need to maximize the cargoes delivered for each trip from the cargo distribution center so that they do not need to go to the same place for multiple times. We hope to make the game easy to learn and play, but difficult to master.

In the game, player can pick up multiple cargoes on the cargo distribution center subject to the capacity of the vehicle they chose. Then they need to plan a route and deliver cargoes they picked up earlier to the target buildings. Player will obtain a certain amount of points once a delivery is made. Multiple cargoes need to be delivered to different places at the same time within the time limit. The goal of player is to get as much point as possible before timer runs out. The game has multiple maps with different characteristics such as street grid design, bottlenecks and traffic density that make some of them more difficulty than others.

# Literature review

## Inspiration: Mini Metro

Gameplay video: https://youtu.be/fT0ETNQlE9o

Our game is inspired by the game “Mini Metro” produced by Dinosaur Polo Club. Mini Metro is a minimalistic single-player 2D simulation game. In the game, player need to connect different stations to form a line and transport as many passengers as possible before a station is full and the game ends. Mini Metro is available on many platforms such as PC, Android, iOS and PS4. We think this game is very intuitive. It is easy to learn, but difficult to master. It has very high replay value too as player can beat their own score in each map.

Our game has borrowed many ideas from Mini Metro. For example, we use similar art style, player can play different maps with its unique characteristics and player can choose upgrades periodically.

## Existing game: Fat[EX] Courier Simulator

Store page: https://store.steampowered.com/app/904850/

Fat[EX] Courier Simulator (FCS) is a similar game that is currently available on the market. In the game, player will experience the daily routine of a courier. Mini Courier is quite different from FCS.

FCS focuses more on the aggregate experience of being a courier. Player need to complete a series of actions to complete a delivery. Player need to pick up the parcel, put them in the truck manually and then drive the truck to the destination. After they arrive to their destination, player need to overcome different obstacles to deliver the parcel to the customer. For example, player need to find a way into the customer’s house since the fence to the house is locked, sometimes they may even need to deal with customers who will attack the player. FCS excels in bringing a more detailed and fun experience to player, but these experiences are not quite realistic. It is very rare for couriers to deal with these situations in real life. Also, FCS offers little customization for players to choose. Player cannot change their vehicle type. Players also cannot obtain upgrades which will aid them in their delivery.

# Game rules

## Choose map and vehicle phase

At the start of the game, players first need to choose the map they want to play on. In the increasing difficulty order, they are Tokyo, Hong Kong and New York. Players then need to choose one of three vehicles type as the vehicle they use in the game. We hope to make different variations so that player can have a difference experience in playing the game with different vehicles in different maps.

## Game loop phase

Player controls their vehicle by pressing the WASD or their respective arrow keys. Player can activate speed booster by pressing B, and use the time extender by pressing N. At any time, they can press ESC to call up the pause menu.

Cargoes will be spawned periodically. Each cargo has a time limit that the player needs to deliver to the target building before the deadline. The time limit is different for different maps. At any time, player can go back to the cargo distribution center (the black boxes on the map) to pick up more cargoes to deliver.

For each successful delivery, players can earn a certain amount of point based on the weight of the cargo, the time remaining and the number of cargoes player has delivered in a single trip from the cargo distribution center. To maximize the score a player can get, player should carry as much cargoes as possible on every delivery trip and deliver cargoes to their destination as soon as possible.

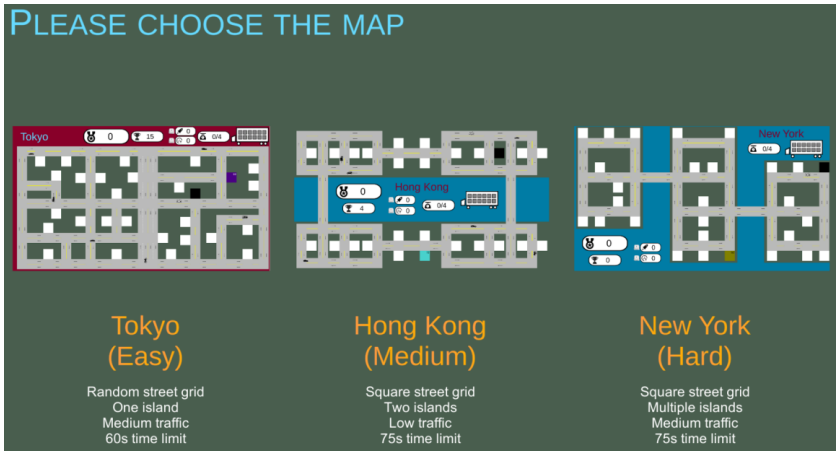
The game has a minimalistic design. Each vehicle is rendered from a simple 2D icon. Every cargo has a distinct colour. Player need to deliver cargoes to buildings with the matching colour. The player vehicle is white in colour while other NPCs are black in colour. All these make players to easily distinguish between themselves and NPCs and locate different cargoes destination quickly.
For every 20 points the player earns, they can choose an upgrade that will help them in delivery. The game will spawn cargoes with possibly larger size more frequently as player earns more points.
Cargoes are spawned in random locations across the map. Every NPC vehicle move randomly and occupy the street in the map. Players can choose and use different

upgrades whenever they want to. All these mechanics ensure that each play session is different from others and attract players to replay the game with different settings.

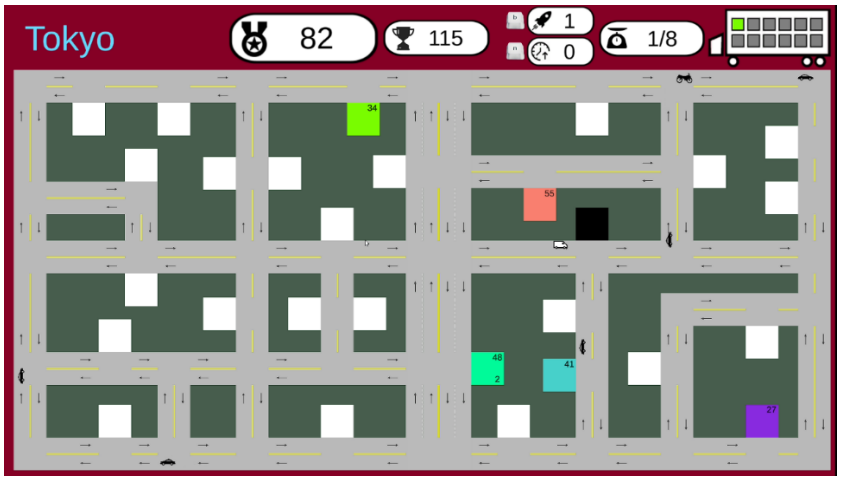
If one of the time limits of cargoes reach 0, the game will end. Player can try to compete themselves by earning more score than the personal best.

# Screenshots

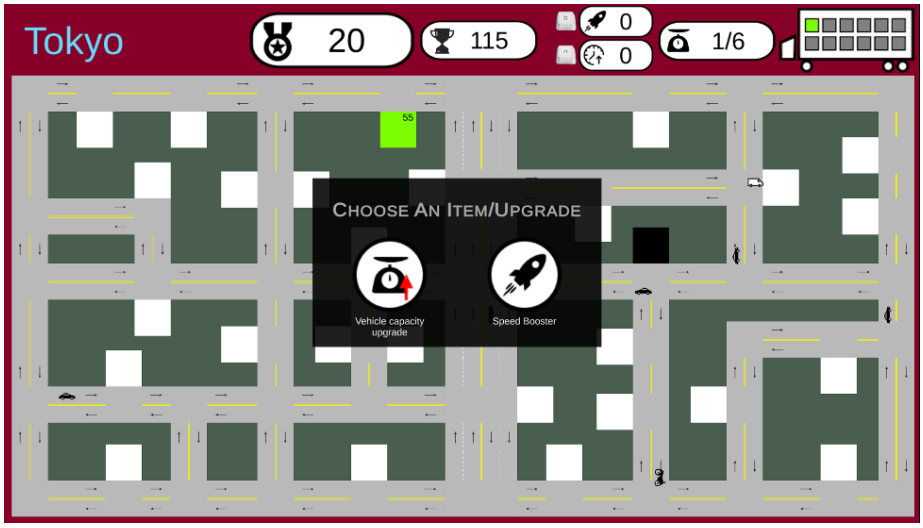
Choosing a map



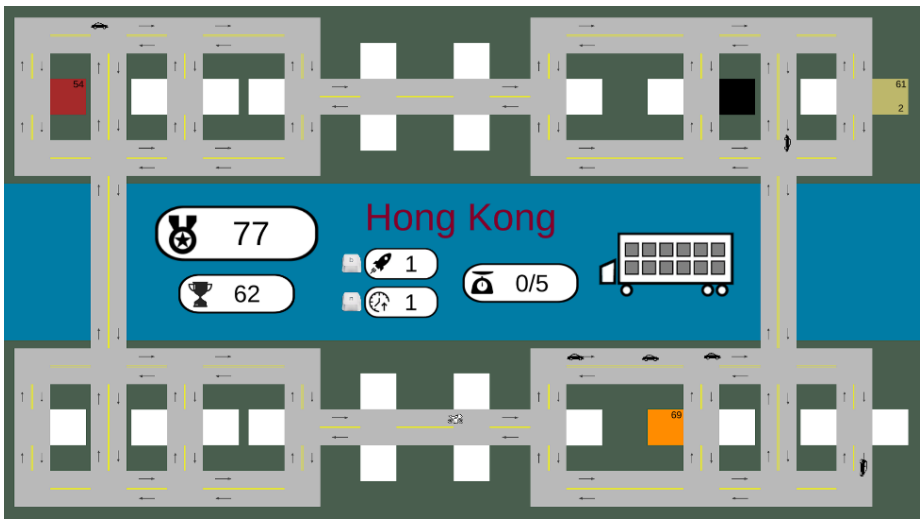

Tokyo



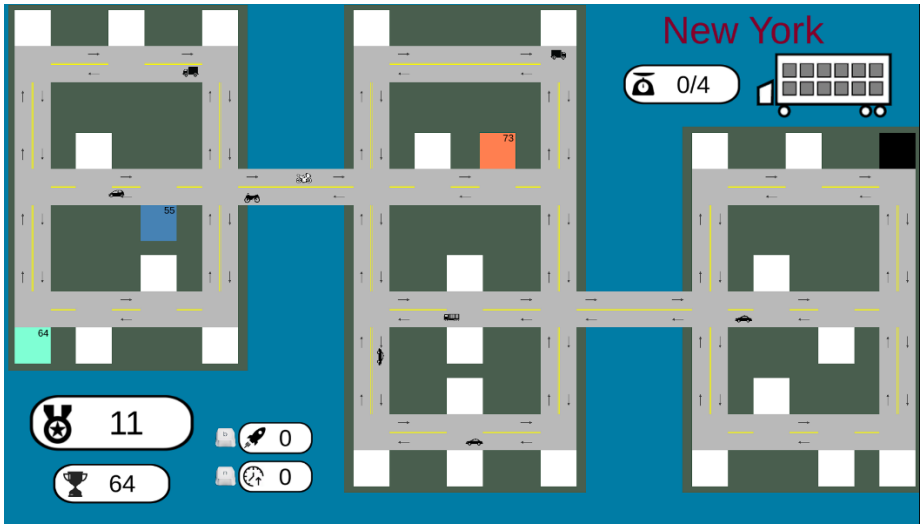
Choosing an upgrade




Hong Kong




New York



# Features

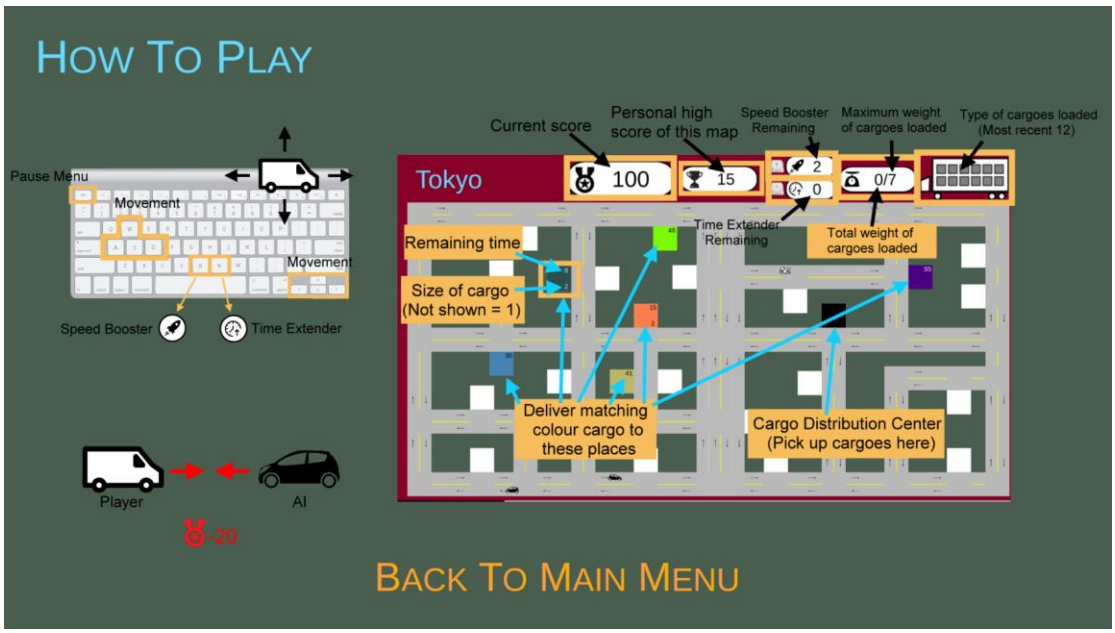
## Main menu


In the main menu, the animation on the top basically demonstrates the player what they need to do in the game. We hope to convey the following message to players in the main menu:

- The vehicle in white means the player, vehicles in black means NPCs
- The black box represents the cargo distribution center where player pick up their cargoes
- The colour box on the right represents the target building
- Player need to deliver cargoes to the building

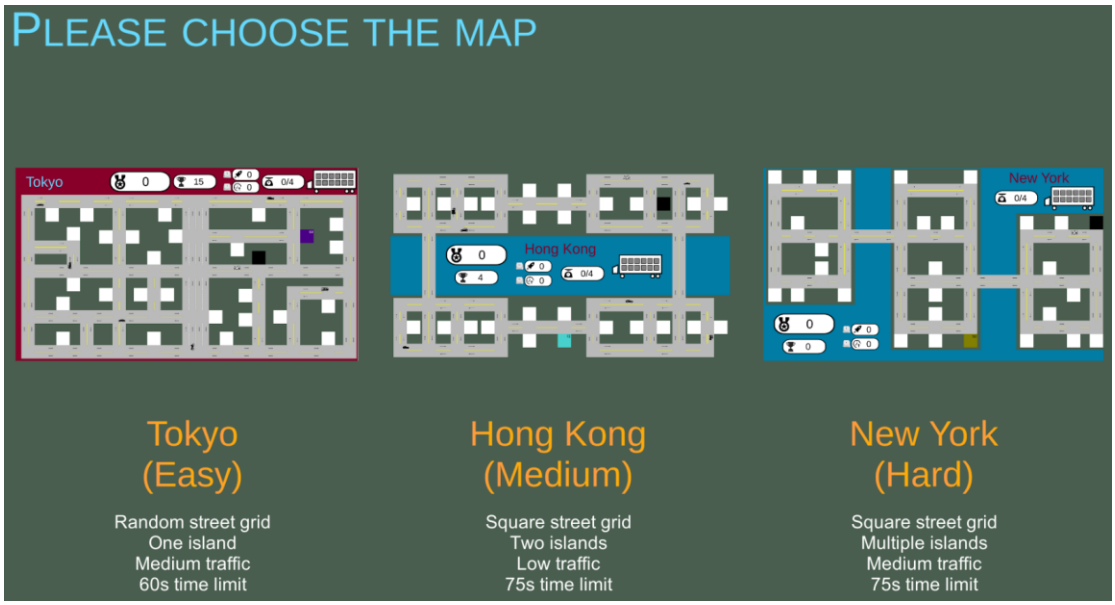
Of course, player can click on “How To Play” in the menu to learn all the mechanics of the game instead.

## How to play




We hope to point out all the important mechanics of the game in this scene and allow players to utilize all of them quickly and easily.

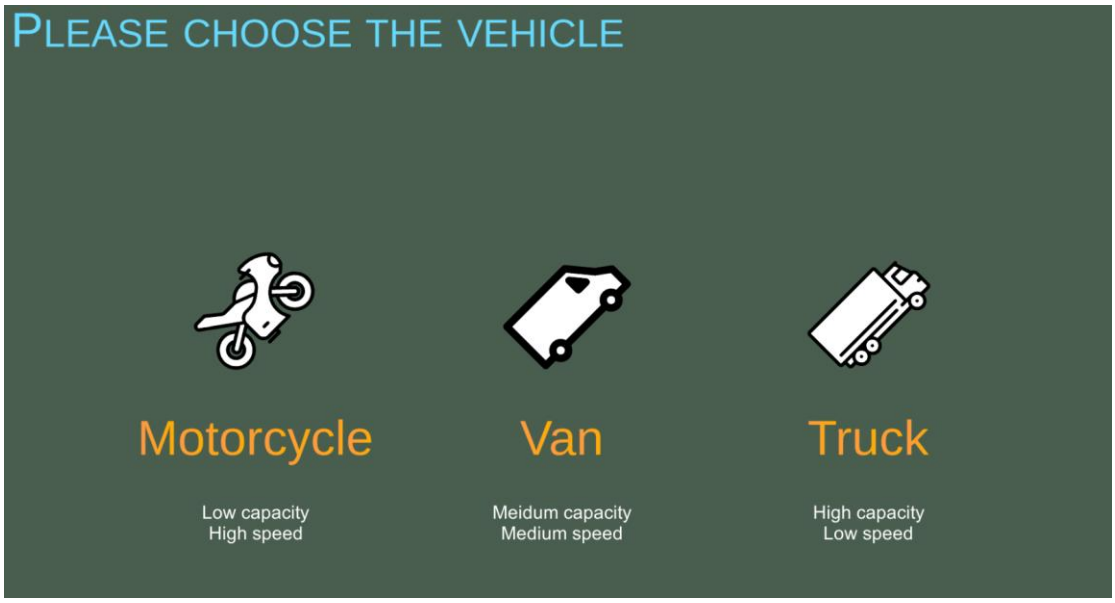
## Map choosing stage




We provide an overview of the map and a brief description of characteristics of the map to players. They should help player choose the map they want to play base on their experience of our game.

The first map of choice if Tokyo. Tokyo map is a single map design. The street grid is irregular and some streets are not as accessible as the others. Players need to careful plan their shortest path. Hong Kong map has a two islands design. They are connected with two bridges. It would take much more time to travel to other islands. New York is similar to Hong Kong but it has three islands and each of them are connected with one bridge only. New York also has the heaviest traffic.

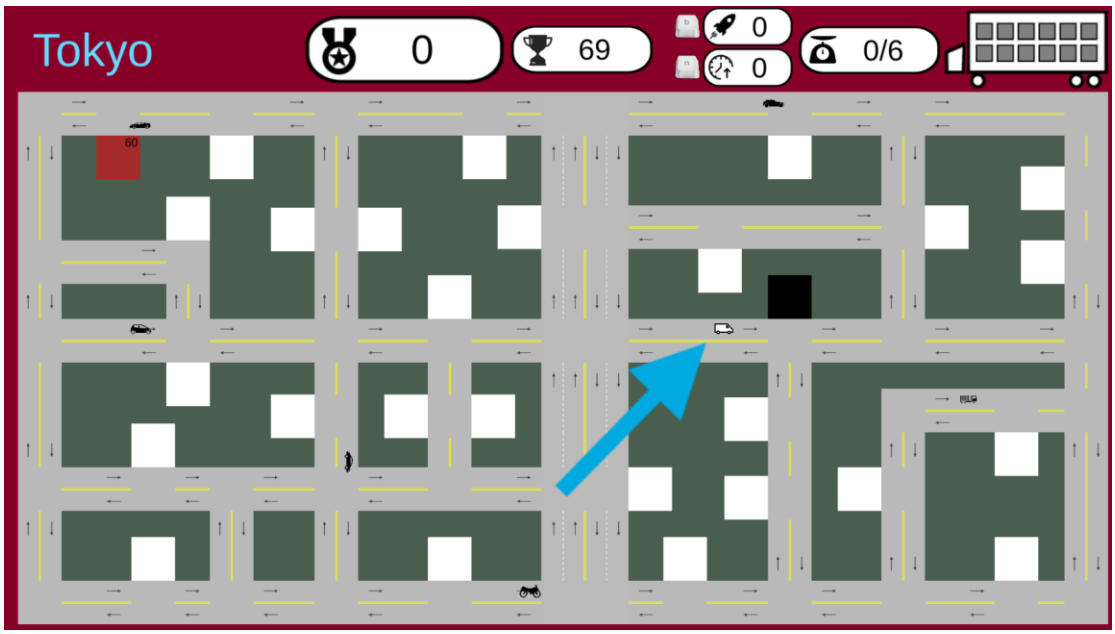
## Vehicle choosing stage




Similar to before, we provide the icon of the player vehicle and a brief description to player to help them choose the vehicle they want.

The first vehicle of choice is motorcycle, the capacity of it is the smallest, but its top speed and acceleration is the fastest. The second one is van, the capacity, top speed and acceleration is mediocre among the three. The third one is truck, the capacity is the highest, but its top speed and acceleration is the lowest.

## UI of game loop stage

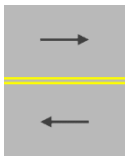



If you want to understand the layout of the UI, please refer back to “How to play” section. At the start of the game, a big blue arrow will point to the location of the player. Whenever the player presses any key, the arrow will disappear. The first cargo will be spawned immediately after player enters the map.

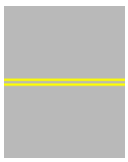
## Roads

Every road of each map is composed with the same sprite. There are only 6 sprites that the game uses. We adjusted the scaling and position of that fragment of road so that when combined with other fragments of road, it forms a complete road that player can drive on.

The following sprites are used to generate all the three maps in the game:

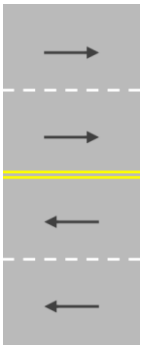


One lane road with arrows

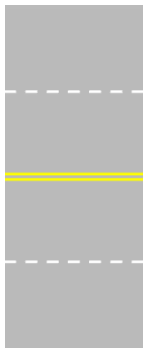


One lane road without arrows

One lane to one lane interchange



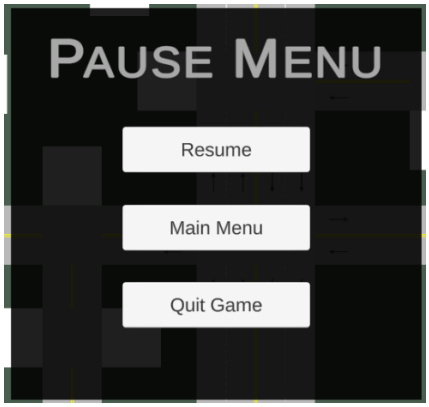
Two lanes road with arrows



Two lanes road without arrows

One lane to two lanes interchange

# Pause menu




At any point in the game, if no other menu is opened, player can open the pause menu to temporarily pause the game, return to main menu or exit the game.

# Player vehicle

## Control

Player use the WASD keys or the arrow keys on keyboard to control their vehicle. In every frame, the script will apply different value of acceleration/deceleration to the vehicle based on the vehicle they chose earlier and the key player presses. The script will also apply rotation on the model whenever necessary.

For example, if the player pressed W, the y value of velocity will increase within the top speed constraint and the rotation of the model will be changed subject to the speed of the vehicle.

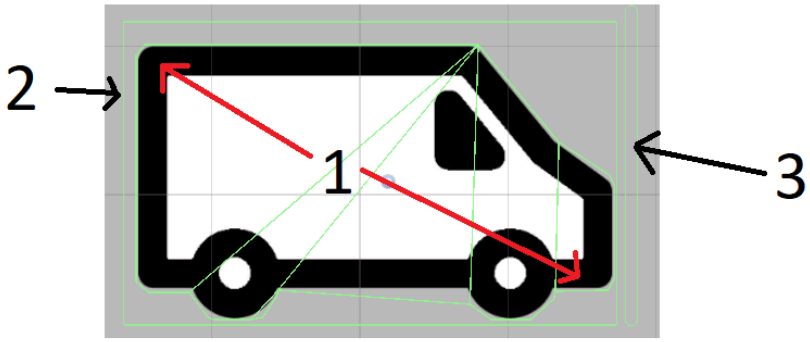
```
if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))
{
    keypressed = true;
    wPressed = true;
    if (v.y + accelAndDecelRate <= topSpeed)
        v.y += accelAndDecelRate;
    else if (v.y > topSpeed)
        v.y = topSpeed;
    if (rb.velocity.y > 0)
        transform.eulerAngles = new Vector3(0, 0, 90);
}
```

If player does not press any keys, the vehicle will decelerate naturally until it stops, which simulate the friction of the car to the road. Natural deceleration is slower than player pressing the key. It will take more time to stop the car.

The following code is used to implement natural deceleration:

```
//Natural deceleration
if (!keypressed)
{
    if (v.x > 0)
        v.x -= naturalDecelerationRate;
    else if (v.x < 0)
        v.x += naturalDecelerationRate;
    if (v.y > 0)
        v.y -= naturalDecelerationRate;
    else if (v.y < 0)
        v.y += naturalDecelerationRate;
```

## Colliders




The player vehicle consists of three colliders. The first one (labelled 1 above) is the Polygon Collider that reacts to the walls, yellow lines and each cargo destinations of the map. These colliders are used so that player cannot cross these obstacles in the map. No script uses that collider for decision making in the player's vehicle script, in other word, it is not isTrigger. The second one (labelled 2 above) is the bigger box collider that is used to detect if any AI touches the player. The third one (labelled 3 above) is the capsule collider used to detect if the player is actively ramming into an NPC.

If an AI touches with the player in other places other than the front, we give the player the benefit of the doubt and assume it is the NPC fault for this collision. We will remove the NPC so that it won't affect the position and velocity of the player and let the respawner to respawn the NPC in other places. The second collider is used in the following script on the NPC:

```
else if(name == "vehicle" && boxCol.IsTouching(obj) && obj is BoxCollider2D) //AI hit the player
{
    Destroy(gameObject); //Destroy itself to prevent it from stopping
    AI_Respawn.respawnAI();
}
```

The reason for having separate colliders for detecting player-NPC collision is because it is hard to determine who causes a collision, but if the front of the player vehicle is touching the AI, we can be sure that it's the player fault for the collision since the NPC always follow the direction of the road. The third collider has the following script:

```
else if( (name == "AI" || name == "AI(Clone)") && obj is PolygonCollider2D && capCol.IsTouching(obj)) //Player hit the AI
{
    GameManager.ReduceScore(20);
    GameObject.FindWithTag("AudioManager").GetComponent<SoundEffects_Script>().playCarCrash();
    playerCollideWithAI++;
    Destroy(obj.gameObject);
    AI_Respawn.respawnAI();
}
```

## Cargo distribution centers

There are two cargo distribution centers (CDC) in every map where player can pick up cargoes. The east CDC is unlocked at the start of the game, while the west CDC is an upgrade for player to choose once cargoes spawn more frequently. Having two CDCs reduce the number of cargoes player need to take across the map, they can take cargo closest to their current position on the map.



When a player arrives at a CDC, the cargo pickup menu (shown above) will pop up. It will list out all the cargoes the player can pick up subject to the available capacity of the player's vehicle. The number inside each button represents the size of the cargo, and the colour of the button represents the colour of the destination. When a player picks up a cargo, its cargo and capacity indicator (shown above on the right-hand side) will be update.

The underlying code for opening the cargo pickup menu is this:

```
else if(name.StartsWith("CargoDistribution_")) // Player reach cargo distribution center
{
    if( (GameManager.CDCWestUnlocked && name == "CargoDistribution_West") || name == "CargoDistribution_East" )
    {
        if (!cargo_menu_opened && (Time.time - cargo_dist_exit_time >= 0.5f)) //Not opened menu when within 0.5s of leaving the CDC
        {
            cargo_dist_exit_time = Time.time; //Record the time the player enters cargo distribution center
            GameObject canvas = GameObject.Find("Canvas");
            cargo_menu_opened = true;
            canvas.GetComponent<CargoPickupMenuScript>().OpenMenu();
        }
    }
}
```

The underlying code for rendering the cargo pickup menu is this:

```
public void OpenMenu()
{
    GameObject.FindWithTag("AudioManager").GetComponent<SoundEffects_Script>().playOpenMenu();
    if (VehicleScript.engine_sound.isPlaying)
        VehicleScript.engine_sound.Stop();

    Time.timeScale = 0f;
    pickupMenuUI.SetActive(true);
    for (int i = 1; i < GameManager.num_spawner; i++)
    {
        string target = "Cargo_" + i.ToString() + "_pickup";
        pickupMenu_buttons[i] = GameObject.Find(target);
    }

    for (int j = GameManager.num_spawner; j < 27; j++)
    {
        string target = "Cargo_" + j.ToString() + "_pickup";
        pickupMenu_buttons[j] = GameObject.Find(target);
        pickupMenu_buttons[j].GetComponent<Button>().interactable = false; //disable all buttons
        pickupMenu_buttons[j].GetComponent<Image>().color = new Color(0f, 0f, 0f); //set to white
        GameObject.Find(pickupMenu_buttons[j].name + "_text").GetComponent<TMP_Text>().SetText(""); //clear all text
    }

    PauseMenuScript.cargoPickupMenuOpened = true; //Prevent player from opening pause menu since it may clash with the cargo pickup menu
    reloadCargoPickupMenu();
}
```

And when player pressed a button to pickup a specific cargo, the following code will be executed, which will add the cargo into the list of cargoes the player picked up

```
public void ButtonPressed(Button btn_pressed)
{
    GameObject.FindWithTag("AudioManager").GetComponent<SoundEffects_Script>().playPickedUpCargo();
    string[] tmp = btn_pressed.name.Split('_');
    int cargo_num = int.Parse(tmp[1]);

    //no need check coz checked above already
    VehicleScript.cargoes_pickedup.Add(cargo_num);
    GameManager.cargoes_waiting_pickup.Remove(cargo_num);

    VehicleScript.currCargoSize += GameManager.size_of_cargoes[cargo_num];

    reloadCargoPickupMenu();
}
```

# Cargo delivery

Whenever a player collides with a building in the map. We first check whether a cargo is needed to deliver to that building first, then we checked whether the player has picked up that cargo. If all of the above is true, then we remove the cargo from the player and increase the score of the player accordingly.

The following script is used to update parameter of the player vehicle:

```
if(name.StartsWith("Cargo_")) //handle collision with cargo
{
    string[] str = name.Split('_');
    SpawnerScript spawner = obj.gameObject.GetComponent<SpawnerScript>();
    if(spawner.isActive) //If the cargo destination is active
    {
        int cargo_num = int.Parse(str[1]);
        if (cargoes_pickedup.Contains(cargo_num)) //If the cargo is picked up and in the truck
        {
            GameManager.cargoes_active.Remove(cargo_num);
            cargoes_pickedup.Remove(cargo_num);
            currCargoSize -= GameManager.size_of_cargoes[cargo_num];
            UpdateCargoPickedupMenu();
            spawner.SuccessfulDelivery();
        }

    }
}
```

The following script is used on the cargo destination to update its status

```
// Function called when the player made a successful delivery
1 個參考
public void SuccessfulDelivery()
{
    isActive = false;
    int total_score = score + (int)((time_limit - timer) / 15) + 1 + VehicleScript.currTripCargoDelivered++;
    //total score = size + (floor)(time remaining/15) + 1 + currTripCargoDelivered

    timer = 0f;
    timer_texts[cargo_number].GetComponent<TMP_Text>().SetText("");
    size_texts[cargo_number].GetComponent<TMP_Text>().SetText("");
    GetComponent<SpriteRenderer>().color = Color.white;
    GameObject.Find("GameManager").GetComponent<GameManager>().AddScore(total_score);
}
```

The following script is used to update the score, level, cargo spawn interval and open the upgrade menu if needed:

```
public void AddScore(int _score)
{
    score += _score;
    scoreMeasuredForUpgrade += _score;

    GameObject.FindWithTag("AudioManager").GetComponent<SoundEffects_Script>().playDeliveredCargo();

    while (scoreMeasuredForUpgrade >= 20)
    {
        levels += 1;
        spawnIntervalChange += (int)levels / 15 * 0.01f; //Reduce the reduce in spawn interval for larger levels
        if (spawnIntervalChange > 1)
            spawnIntervalChange = 1;
        spawnInterval *= spawnIntervalChange; //Reduce spawn interval
        GameObject.FindWithTag("Player").GetComponent<VehicleScript>().UpdateGUIText(); //Update GUI text
        GameObject.Find("Canvas").GetComponent<UpgradeMenuScript>().CallUpgradeMenu(); //Shown upgrade menu
        scoreMeasuredForUpgrade -= 20;
    }
}
```

# Upgrades

As mentioned before, the upgrade menu will be opened when the player earns every 20 points. There will be two upgrade/item choices for the player to choose. The probability is adjusted so that player will not find them too useful which may make the game too easy and affect the gameplay experience. There are four possible

upgrades for the player to choose.

We use a random number to determine the two upgrades that appear in the upgrade menu. Capacity upgrade has a 35% to appear in the first upgrade option and the speed booster has a 65% to appear in the first upgrade option. For the second option, if the current level is more than 5 (i.e. the player has earned more than or equal to 120 points), there's a 30% chance that the second upgrade is unlock the west cargo distribution center. If the first upgrade option is capacity upgrade, the second option would 100% be speed booster. Otherwise, there's a 40% chance for the second upgrade to be time extender. Otherwise, it will be a speed booster.

The two buttons for choosing upgrade/item are handled by the script of upgrade menu script. Each button has its own handler so that it can call the function to execute the script and increase the item count or unlock upgrades. Since the option of the two buttons are pre-determined by the script, whenever the player presses the button, we can read the option set by the algorithm previously and execute the correct script.

The following script is used to handle buttons:

```
0 個參考
public void Button_1_handler()
{
    GameObject.FindWithTag("AudioManager").GetComponent<SoundEffects_Script>().playClickButton();
    if (btn_1_option == "capacity")
        VehicleScript.Upgrade_VehicleCapacity();
    else
        VehicleScript.numSpeedBooster++;
    Resume();
}

0 個參考
public void Button_2_handler()
{
    GameObject.FindWithTag("AudioManager").GetComponent<SoundEffects_Script>().playClickButton();
    if (btn_2_option == "CDC")
    {
        GameManager.CDCWestUnlocked = true;
        GameObject.Find("CargoDistribution_West").GetComponent<SpriteRenderer>().color = new Color(0f, 0f, 0f); //change west CDC background to black
    }
    else if (btn_2_option == "booster")
        VehicleScript.numSpeedBooster++;
    else if (btn_2_option == "timeExtender")
        VehicleScript.numTimeExtender++;
    Resume();
}
```

## Speed booster

Speed booster is the most common item generated on the upgrade menu. When the player uses a speed booster by pressing the B key, the top speed of the vehicle will be temporarily raised by 150 for 5 seconds. It is especially useful if player is in a straight road, there are no AI on that road and they want to arrive to the destination quickly.

The following code is used to implement speed booster:

```
if (Input.GetKey(KeyCode.B) && !isInSpeedBoost && numSpeedBooster > 0) //Use SpeedBooster
{
    VehicleScript.arrow.SetActive(false);
    GameObject.FindWithTag("AudioManager").GetComponent<SoundEffects_Script>().playBooster();
    numSpeedBooster--;
    isInSpeedBoost = true;
    topSpeed += speedBoostAmount;
}
```

```
//In Speed Booster effect
if (isInSpeedBoost)
{
    speedBoosterTimer += Time.deltaTime;
    if(speedBoosterTimer >= speedBoosterTimeLimit)
    {
        topSpeed -= speedBoostAmount;
        speedBoosterTimer = 0f;
        isInSpeedBoost = false;
    }
}
```

## Time extender

Time extender is another item you can get when you earn every 20 points. When the player uses a speed booster by pressing the N key, the timer of all the cargoes to be delivered will be increased by 10 seconds.

The following code is used to implement time extender:

```
if (Input.GetKeyDown(KeyCode.N) && numTimeExtender > 0) //Use Time Extender
{
    VehicleScript.arrow.SetActive(false);
    GameObject.FindWithTag("AudioManager").GetComponent<SoundEffects_Script>().playTimeExtender();
    numTimeExtender--;
    foreach (int cargo in GameManager.cargoes_active)
        GameObject.Find("Cargo_" + cargo.ToString()).GetComponent<SpawnerScript>().IncreaseTimeLimit(timeExtenderAmount);
}
```

```
public void IncreaseTimeLimit(float amount)
{
    time_limit += amount;
}
```

## Capacity upgrade

Capacity upgrade can be obtained with a lower probability than speed booster and time extender. When a player chooses that upgrade, the total available capacity of the player vehicle will be increased by 1. Players should choose this upgrade whenever it is possible since they need to carry more cargoes for each trip as they earn more points.

The following code is used to implement capacity upgrade:

```
public void Button_1_handler()
{
    GameObject.FindWithTag("AudioManager").GetComponent<SoundEffects_Script>().playClickButton();
    if (btn_1_option == "capacity")
        VehicleScript.Upgrade_VehicleCapacity();
    else
        VehicleScript.numSpeedBooster++;
    Resume();
}
```

```
public static void Upgrade_VehicleCapacity()
{
    maxCargoSize += vehicleCapacityUpgradeAmount;
}
```

## Unlock west cargo distribution center

This upgrade is a one-time upgrade. It will probably appear when a player has reached level 6 or above, i.e. earned more or equal to 120 points. Also, it will only appear in the second upgrade choice. When a player chose this upgrade, the west cargo distribution center (CDC) will be unlocked for player to use. Players can pick up cargoes there instead of needing to back to the east CDC and save time. It is recommended that player should choose this upgrade as soon as it is available.

The following code is used to implement capacity upgrade:

```
public void Button_2_handler()
{
    GameObject.FindWithTag("AudioManager").GetComponent<SoundEffects_Script>().playClickButton();
    if (btn_2_option == "CDC")
    {
        GameManager.CDCWestUnlocked = true;
        GameObject.Find("CargoDistribution_West").GetComponent<SpriteRenderer>().color = new Color(0f, 0f, 0f); //change west CDC background to black
    }
```

# NPCs

The NPCs in the game is mainly responsible for creating obstacles to players so that they need to follow the traffic rule most of the time. They are also used to simulate a real-world city. When player crashes into an NPC straight on, he/she will lose 20 points. Each map in the game has different number of NPC depending on the difficulty of the map. If there are a lot of NPCs in the game, it is a lot harder for player to control the vehicle appropriately without crashing into them.

## Driving

NPCs in the game have different top speed. They can only drive in 4 different directions, north, east, south and west at top speed. They will reduce speed if there are NPC in front of them that have a slower speed.

The following codes are used to implement AI driving:

```
//Change the rotation of the AI
if (initialDirection == 'e')
{
    gameObject.transform.eulerAngles = new Vector3(0, 0, 0); //Rotate
    rb.velocity = new Vector2(speed, 0); //Give velocity
}
else if (initialDirection == 's')
{
    gameObject.transform.eulerAngles = new Vector3(0, 0, 270);
    rb.velocity = new Vector2(0, -speed);
}
else if (initialDirection == 'w')
{
    gameObject.transform.eulerAngles = new Vector3(0, 180, 0);
    rb.velocity = new Vector2(-speed, 0);
}
else if (initialDirection == 'n')
{
    gameObject.transform.eulerAngles = new Vector3(0, 0, 90);
    rb.velocity = new Vector2(0, speed);
}
```

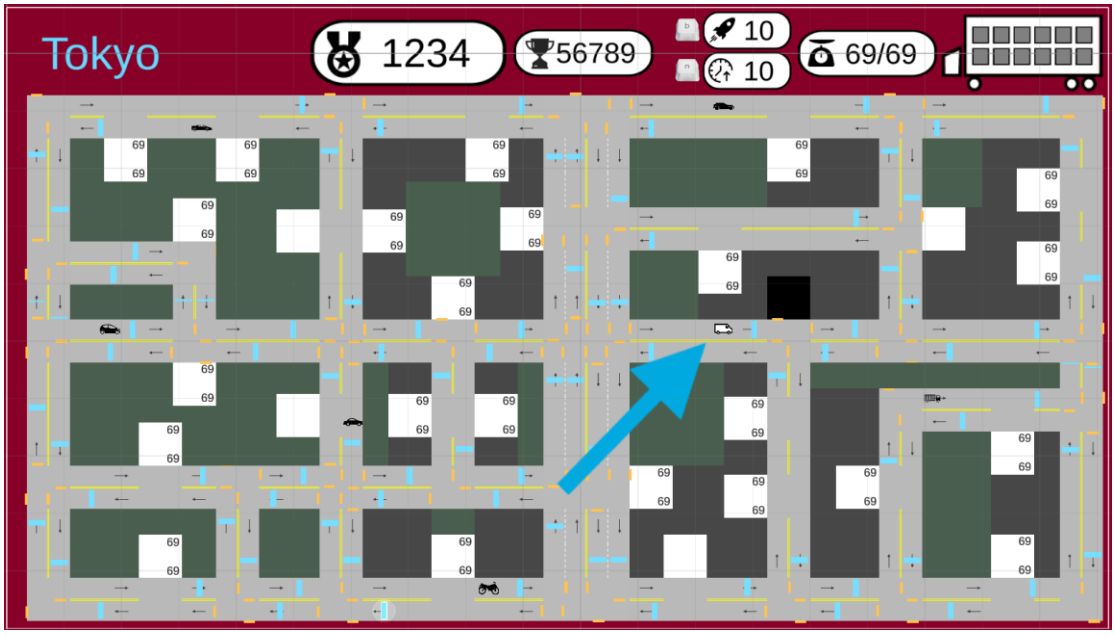
```
void Update() //If AI is in the wrong direction, update velocity back to normal
{
    if (currDirection == 'e' && rb.velocity.x < 0)
        rb.velocity = new Vector2(speed, 0);
    else if (currDirection == 's' && rb.velocity.y > 0)
        rb.velocity = new Vector2(0, -speed);
    else if (currDirection == 'w' && rb.velocity.x > 0)
        rb.velocity = new Vector2(-speed, 0);
    else if (currDirection == 'n' && rb.velocity.y < 0)
        rb.velocity = new Vector2(0, speed);
}
```

## Turning

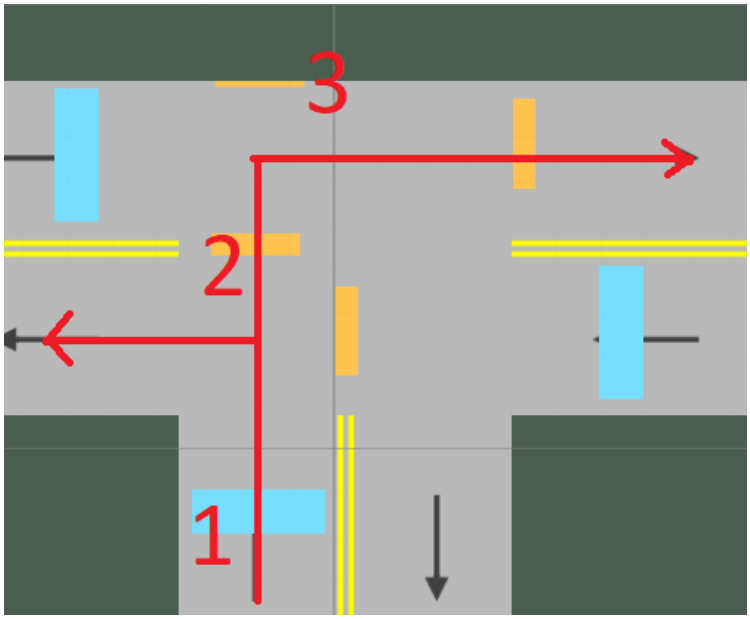
In the game, we use two different types of game object to change the direction of the NPC randomly.

When an NPC is approaching a corner, it will collide with a determine route game object (blue boxes in the picture) that determine the direction of the next turn. The script generates a random number from 1 to n, where n is the total number of possible turns of that corner. The AI will store that number in itself and continue driving.

Each corner is filled with turning points for each incoming direction. Each turning point is represented as an orange box below. When the AI collides with it, the AI will check whether it matches the direction it generated previously when it collided with the blue box and whether it matches the current direction. If yes, the AI will change its direction. If not, the AI will continue driving in its current direction.




For example, in this particular corner shown below. Assume that the AI drives to this corner from the south. At first, it will collide with the determine route game object (the blue box). Since there are only two routes possible in this corner, the AI will generate an integer between 1 and 2 and continue driving. When it collides with the orange box 2, it will first check whether its previously generated number match with the number labelled in that box (which is 1 in that box) and its target original direction (which is s, south). If yes, then it will turn to west. If not, it will keep on driving until it collides with the orange box 3. If the value in that box (2 in this case) matches with the value of the AI and its original direction, it will turn the AI to east.



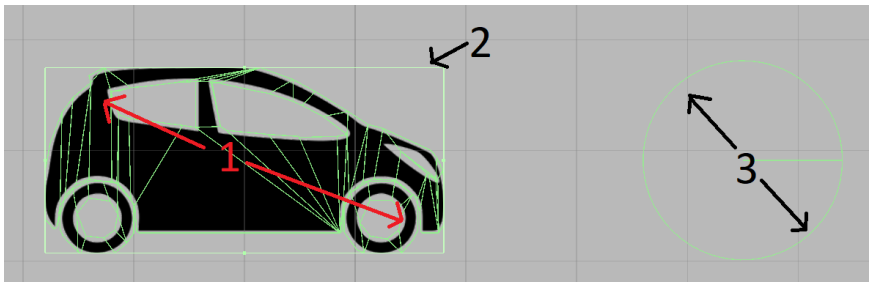
Every determine route game object (blue boxes) in the map has the following script:

```
public void OnTriggerEnter2D(Collider2D obj)
{
    string name = obj.gameObject.name;
    if(name == "AI" || name == "AI(Clone)")
    {
        AI_Script ai_script = obj.gameObject.GetComponent<AI_Script>();
        ai_script.orlginSide = side;
        ai_script.nextTurn = ran.Next(1, numberOfPossibleRoute + 1); //Generate a random number to determine the AI turning point
    }
}
```

Every turning point game object (orange boxes) in the map has the following script:

```
public void OnTriggerEnter2D(Collider2D obj)
{
    string name = obj.gameObject.name;
    PolygonCollider2D polyCol = obj.GetComponent<PolygonCollider2D>();

    AI_Script ai_script = obj.gameObject.GetComponent<AI_Script>();

    //AI touches matching the correct turning point
    if ( (name == "AI" || name == "AI(Clone)") && turnNumber == ai_script.nextTurn && boxColl.IsTouching(polyCol) && ai_script.currDirection == carOriginalDirection && ai_script.orginSide == originSide)
    {
        Rigidbody2D rb = obj.gameObject.GetComponent<Rigidbody2D>();

        // Rotate the AI and change its speed according to their turn direction
        if (turnDirection == 'e')
        {
            obj.gameObject.transform.eulerAngles = new Vector3(0, 0, 0);
            rb.velocity = new Vector2(ai_script.maxSpeed, 0);
        }
        else if (turnDirection == 's')
        {
            obj.gameObject.transform.eulerAngles = new Vector3(0, 0, 270);
            rb.velocity = new Vector2(0, -ai_script.maxSpeed);
        }
        else if (turnDirection == 'w')
        {
            obj.gameObject.transform.eulerAngles = new Vector3(0, 180, 0);
            rb.velocity = new Vector2(-ai_script.maxSpeed, 0);
        }
        else if (turnDirection == 'n')
        {
            obj.gameObject.transform.eulerAngles = new Vector3(0, 0, 90);
            rb.velocity = new Vector2(0, ai_script.maxSpeed);
        }

        ai_script.currDirection = turnDirection; //Update AI direction
    }
}
```

## Colliders




Similar to the player vehicle, each AI in the game has 3 colliders.

The first collider is polygon collider. It is used to prevent the AI from clipping through the wall and double yellow line of the map. It is not used in the AI script, but it is used to determine whether the player collides with the AI.

The second one is a box collider. It is used to determine whether the AI is colliding with another AI.

The third one is a circle collider. It is used to determine whether the AI with a higher top speed is approaching to another AI with a lower top speed.

The following collision detection code make use of the circle collider to change its speed and uses the box collider to determine whether it collides with another AI or the player.

```
0 個參考
public void OnTriggerEnter2D(Collider2D obj)
{
    string name = obj.gameObject.name;
    if (name == "AI" || name == "AI(Clone)") //AI collide with AI
    {
        if (cirCol.IsTouching(obj)) //AI is close to another AI
        {
            changeSpeed(obj.gameObject.GetComponent<AI_Script>().speed); //AI reduce speed to not crash into the front AI
        }
        else if (boxCol.IsTouching(obj) && obj is BoxCollider2D) //AI physically touches another AI
        {
            Destroy(obj.gameObject); //Destroy the AI to prevent it from stopping
            AI_Respawn.respawnAI();
        }
    }

    else if(name == "vehicle" && boxCol.IsTouching(obj) && obj is BoxCollider2D) //AI hit the player
    {
        Destroy(gameObject); //Destroy itself to prevent it from stopping
        AI_Respawn.respawnAI();
    }

}
0 個參考
public void OnTriggerExit2D(Collider2D obj)
{
    string name = obj.gameObject.name;
    if (name == "AI" || name == "AI(Clone)") //AI is no longer touching another AI
    {
        changeSpeed(obj.gameObject.GetComponent<AI_Script>().maxSpeed); //Resume back to its normal speed
    }
}
```

## Respawner

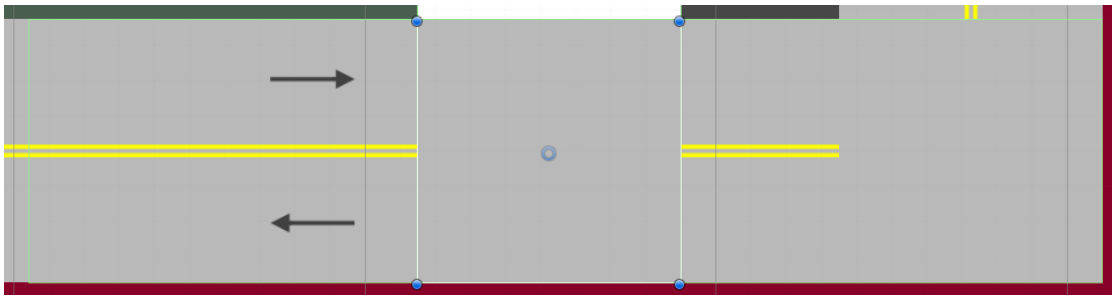
There are one respawner in each map of the game. Their main duty is to instantiate a new AI instance whenever needed given the nearby road is clear of traffic.

Whenever the player collides with an AI, or an AI collides with another AI, it will invoke the function `AI_Respawn.respawnAI()` so that the number of AI to be respawned will increase by 1.

```
else if( (name == "AI" || name == "AI(Clone)") && obj is PolygonCollider2D && capCol.IsTouching(obj)) //Player hit the AI
{
    GameManager.ReduceScore(20);
    GameObject.FindWithTag("AudioManager").GetComponent<SoundEffects_Script>().playCarCrash();
    playerCollideWithAI++;
    Destroy(obj.gameObject);
    AI_Respawn.respawnAI();
}
```

```
public static void respawnAI()
{
    numAIToBeSpawned++;
}
```

The respawner script is attached to a tile in each map. It has two box colliders to scan whether an AI or the player is close to the spawn point. If yes, it will temporarily stop respawning AI until no game object collides with the colliders.



The following script is used to temporarily stop and resume respawning AI

```
public void OnTriggerStay2D(Collider2D obj)
{
    //Debug.Log("Collision!");
    string name = obj.gameObject.name;
    if (name == "AI" || name == "vehicle" || name == "AI(Clone)")
        canRespawn = false;
}

0 個參考
public void OnTriggerExit2D(Collider2D obj)
{
    //Debug.Log("No more collision!");
    string name = obj.gameObject.name;
    if (name == "AI" || name == "vehicle" || name == "AI(Clone)")
        canRespawn = true;
}
```

The spawn point is set in the script. If there are AI to be respawned. It will spawn an instance of the AI from the prefab every 2 seconds given that it is safe to respawn.

```
void Update()
{
    if (numAIToBeSpawned > 0)
    {
        cooldownTimer += Time.deltaTime;
        if (canRespawn && cooldownTimer >= 2f)
        {
            //Debug.Log("AI Respawned!");
            Instantiate(AI_Prefab_car, respawnLocation, Quaternion.identity);
            numAIToBeSpawned--;
            cooldownTimer = 0f;
        }
    }
}
```

# Difficulties encountered

## Unfamiliar with Unity

I think most of the difficulties during development arise from the fact that I am not familiar with Unity. This is the first time I use Unity in a game project. The tutorials of COMP3329 is very limited and I can only learn the very basic of Unity. The game uses a lot of features that is not taught on tutorials of COMP3329. I need to spend extra time to self-learnt most of the features of Unity from YouTube, the Unity forum and Github. For example, I need to self-learn how to use scenes to better manage my game how to switch scenes and passes data between different scenes. Another example is UI, the tutorials from COMP3329 does not teach us how to properly write UI. I have to self-learn how the script handles when a button is pressed. Fortunately, I have allocated a lot of time to self-learn and become more familiar with Unity. In the end, I have learnt a lot of useful techniques and features on Unity.

## Game objects rotation

There are numerous ways to rotate a game object. You can use different functions to rotate an object and at first, I am quite confused on the feature of each functions. Sometimes I failed to rotate a game object when the player drives in a specific direction. Sometimes other bug arises, such as the object is flipped upside-down when I tried to use another function to rotate a game object. Through a lot of trial-and-error, I have found out the most suitable function for me to rotate a game object properly.

## NPC script

It is a hassle to properly implement AI driving. I have assumed that the player should always give way to the AI vehicle first when they reach an intersection. If I were to spend more time in implementing a full-blown traffic light system, it would take a lot of time and effort. Instead I implement the simple AI driving and turning mechanism to spend time on testing and bug fixing.

Since the AIs are very dumb in the game. Whenever a collision between player and AI happens, the AI will get stucked in the wall and not able to move. In the end, I decide to just destroy the AI and respawn a new one to save time.

# Limitations

## Determining who is responsible for the collision

Whenever a player hits the AI, sometimes it will not deduct point from the player. It is because I used box colliders to determine whether the player does not follow the traffic rule and drive on the wrong side of the road intentionally. Sometimes it is possible that the player intentionally stops at an intersection to block the road and stop AI from entering. In this case, it is the responsibility of the player to cause the collision, but since the player does not hit the AI head-on, the system will not deduct points from the player.

To implement a proper system determining who is responsible for the collision depends on different situation. It would be very time-consuming to include every situation into the game. As a result, I have given player the benefit of the doubt so that if the player does not crash with the AI head-on, it is assumed that the AI is responsible for the crash and the game will not deduct point form the player.

## More complex models

The game mainly composes of simple shapes which represents cargo destinations in the game. We also use icons to represent AI vehicles and the player. In the future, we hope to include more complex models so that the game becomes more realistic.

# Credits

Senior Testers

Sam So

Ken Lui

Testers

Kenny Tsoi

Donald Leung

Icons Provider

flaticon.com

Sound Effects Provider

freesound.org