Driver Game

Fachhochschule Erfurt

BA5 WS-2020-2021

Alejandro Restrepo Klinge

Benito Grauel

Ahmad Abo Louha

Spielkonzept

DriverGame ist ein Open-World und Rennspiel, in dem der Spieler ein Auto steuert. Der Spieler kann als Taxifahrer betrachtet. Das Ziel des Spieles ist eine Reihe von vordefinierte Missionen zu erfüllen, wobei jede Mission ein anderes Ziel hat.

In jedes Mission soll der Spieler von einem Ort zu einem anderen Ort in der Stadt fahren. Manche Missionen erfordern aber auch, dass der Spieler zwischen der Startpunkt und Endpunkt auch andere Punkte erreichen soll.

Die Passagiere müssen in einer bestimmten Zeit zum Endpunkt transportiert werden. Der Spieler muss aber darauf aufpassen, dass in manche Missionen Fremden und Hindernisse geben können, die verhindern, dass der Spieler zum Endpunkt rechtzeitig ankommt.

Das Auto hat aber auch ein “Healthbar”, welches nach jeder Kollision reduziert wird. Wird das “Healthbar” leer, so verliert der Spieler das Spiel.

Das Spielgefühl nähert sich dem Motto, “Easy to learn, hard to master”. In dem Spiel sind auch Geheimnisse versteckt, welche der Spieler erkunden soll. Da die Missionen auch eine Wertung haben, kann der Spieler immer wieder die gleiche Missionen spielen, um bessere Ergebnisse zu bekommen.

Eine sehr große Inspirationsquelle ist das Spiel “Crazy-Taxi”. Andere Spielideen sind in Open-World spiele sowie Grand-Theft-Auto inspiriert.

Es eignet sich für alle Spielertypen, die nach Spaß und Adventure beim Spielen suchen.

Auch Conqueror kommen auf ihre Kosten, wenn es darum geht die beste Zeit aus einer Mission herauszuholen oder in jeder Mission 3 Sterne zu verdienen.

Installationshinweise & Systemanforderungen

Das Spiel muss nicht installiert werden, es reicht den DriverGame zip Datei zu entpacken und die "*DriverGame.exe*" Datei auszuführen

Die Spielgröße beträgt 150 MB aber andere mind. Anforderungen konnten wir nicht bestimmen, da es keine Möglichkeit gibt, das Spiel auf verschiedene Systemen und Plattformen zu testen.
Das Spiel ist auf folgenden Rechner ausgeführt ohne probleme beim Laden oder Laufen

	Rechner 1	Rechner 2	Rechner 3
Betriebssystem	Windows 10 Education	Windows 10 Home	Windows 10 Home
Prozessor	Intel Core i5-3230M 2.60GHz	Intel Core i5-8250U 1.8GHz	Intel Core i3-6100U 2.30GHz
RAM	8 GB	16 GB	8 GB
Systemtyp	64-Bit	64-Bit	64-Bit
Grafikkarte	AMD Radeon HD 4GB	NVIDIA GeForce MX150	NVIDIA Geforce 940 MX

Bedienungsanleitung

Taste	**Funktion**
Up-Arrow, 'W'-Taste	Beschleunigen
Down-Arrow, 'S'-Taste	Bremsen
Left und Right Arrows bzw. 'A' und 'D-Tasten'	Lenken
'E'-Taste	Hupe
'Q'-Taste	Lustige Hupe
Leertaste	Handbremse
"ESC"-Taste	Pause

Spielmechanik & Game loop

Vorbereitung

DriverGame ist ein Einzelspieler, wo der Spieler ein Auto kontrolliert und in einer Open-World Stadt fährt, die Stadt hat Hoch- und Einfamilienhäuser dazu kommen Straßen, die verschiedene Größen haben. (enge und breite Straßen sowie eine Autobahn mit einer Brücke).

Der Spieler kontrolliert das Auto durch die vier Pfeiltasten sowie WASD-Tasten. Der Spieler kann auch eine Handbremse dadurch aktivieren, dass die Leertaste gedrückt wird. Außerdem kann er mit den Tasten “e” und “q” hupen.

Wenn das Auto mit einem Haus, einem Zaun oder einem anderen Auto zusammenstieß, wird das “Healthbar” anhand der aktuellen geschwindigkeit des Autos kleiner, bis es leer wird dazu wird die Geschwindigkeit auf 0 gesetzt, sodass die Zeit indirekte weise weniger wird, die für diese Mission gebraucht ist.

Die Feinde können auch Schaden anrichten. Je nach der Art der Feinde, werfen sie einen Stein gegen das Auto, Schlagen sie das Auto, oder versuchen sie das Auto zu beißen.

Ist das “Healthbar” leer, wird das Spiel beendet und der Spieler verliert. Der Spieler kann aber auch während des Spiels durch einer Tankstelle fahren und das “Healthbar” wieder auf 100% setzen.

Die Beschädigung wird nach diese Tabelle berechnet:

Aktuelle Geschwindigkeit	Abzug Health-Punkte
0-5	2%
5-10	4%
10-15	6%
15-20	8%
20-25	10%
>25	12%
Feindestein	3%
Feinde- Schlag oder-biss	4%

Das Auto könnte auch mit andere Objekte kollidieren, die keine Health-Abzug ergeben, sondern führt das zum Verschwinden dieser Objekte wie Laternen und Bäume.

Spielverlauf

Das Spiel fängt so an, dass der Spieler beliebig durch der Stadt fahren kann, ohne eine bestimmte Mission gestartet zu haben (Mission-Off-Modus). Wenn er zu einem Mission-Start ankommt, kann er mit Drücken der Taste “e” die Mission starten. Für jede Mission gibt es eine kleine Geschichte, die den Spieler sag, worum in dem Mission geht.

Nach erfolgreiche Beendigung einer Mission bekommt der Spieler eine Bewertung des Missions (Sternen anhand der gebrauchten Zeit, um die Mission zu erledigen) und kann wieder frei in der Stadt fahren. Danach kann der Spieler entscheiden, ob er zum nächsten Mission-Start fährt oder einfach die Stadt erkunden will.

In bestimmten Zeitpunkte wird das Spiel gespeichert. Einmal wenn das Spiel anfängt und nach Beendigung eines Missions. da wird die Position des Autos, Anzahl gesammelter Münzen, eine Liste der beendeten Missionen und den aktuellen Healthbar-Stand gespeichert. Die Logik für Speicherung bzw. Laden von Zustände ist in dem Script “ProccessController” zu finden.

Während des Spiels kann das Spiel mit “esc” pausiert werden. Von der Pause-Menü kann man die letzte gespeicherte Zustand laden, zurück zum Hauptmenü oder zur Einstellung gehen, wo die Möglichkeit zu finden ist, die Lautstärke, die Qualität (Low - Medium - High) sowie die Fullscreen-on/of anpassen kann.

Während das Spiel kann der Spieler Münzen sammeln, um ein Geheimnis zu erkunden.

Ein Spieler verliert, wenn entweder eine Mission nicht in der vorgegebene Zeit abgeschloßen wurde oder das Healthbar leer ist. In diesen Fälle kann der Spieler auch die letzte gespeicherte Zustand laden.

Spielende

Das Spiel kommt zur Ende wenn alle Missionen durchgespielt worden sind.

Coins-System

Im Spiel kann der Spieler Münzen sammeln, die in der Stadt verteilt sind. Beim Sammeln wird dem Spieler gezeigt, wie viele hat er schon gesammelt und wie viele noch zu finden sind.

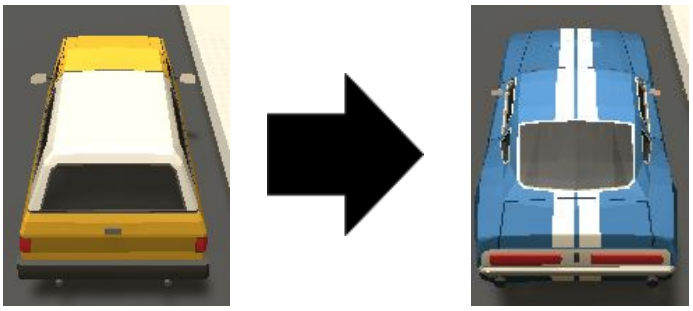
Beim startet dem Spiel wird in Process Controller eine Liste aller Münzen generiert (Die Münzen Game-Objects sollen in einem bestimmten Ordner zu finden sein). Die Anzeige, die zeigt wie viele Münzen in der Stadt zu finden sind, wird automatisch geupdated:

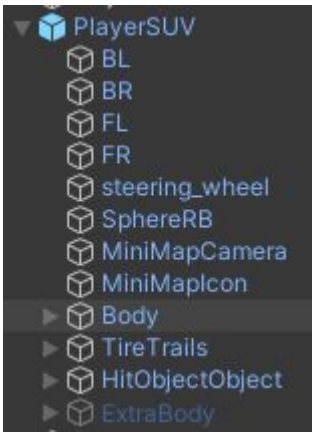

```
foreach (Transform child in coinsFolder)
{
    allCoins.Add(child.gameObject);
}

textAllCoins.text = "/" + allCoins.Count.ToString();
```

Für die aktuelle Version vom Spiel sind 20 Münzen zu finden, später wenn es noch mehr Münzen in der Stadt verteilt werden und und die richtige Ordner kopiert, wird die Zahl automatisch angepasst.

Hat einen Spieler aller Münzen gesammelt dann:

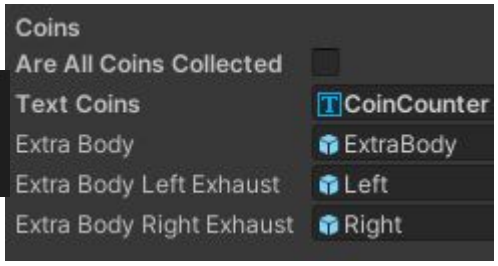
- Das Auto wird mit einem anderen Auto ausgetauscht, das Kontrollieren des Autos bleibt aber wie es ist.





Hierfür hat das Auto ein “Body” (Links) und ein “ExtraBody” (Rechts) und beim sammeln aller Münzen wird es das body ausgetauscht. Zur Kontrolle hat das Auto die Bool-Variable “AreAllCoinsCollected”

```
//////////////////////Coins-Collection//////////////////////////////
if(GetCollectedCoins().Count == proccessController.allCoins.Count)
{
    areAllCoinsCollected = true;
}
```



- Mit hilfe der vorher benannte Variable wird eine neue Mission freigeschaltet, die nach erfolgreicher Beendigung aller Missionen gespielt wird. (Logik in MissionController.cs zu finden)

```
if(carController.areAllCoinsCollected)
{
    secretMissionActivated = true;
}
```

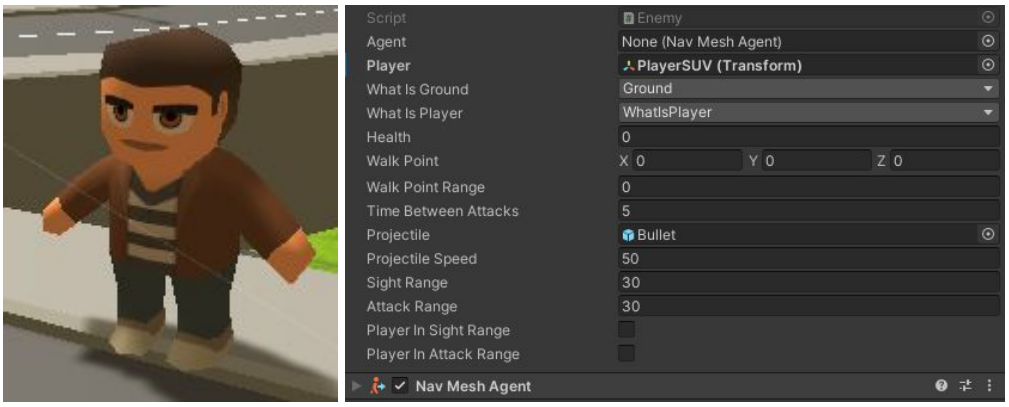
```
//Only if all missions are played and the secret mission is active
else
{
    this.allRegularMissionsDone = true;

    //Start the secret Mission
    StartSecretMission();
```

Ziel von dem ganzen war dem Spieler einer Motivation geben, aller Münzen zu sammeln.

Feinde-System

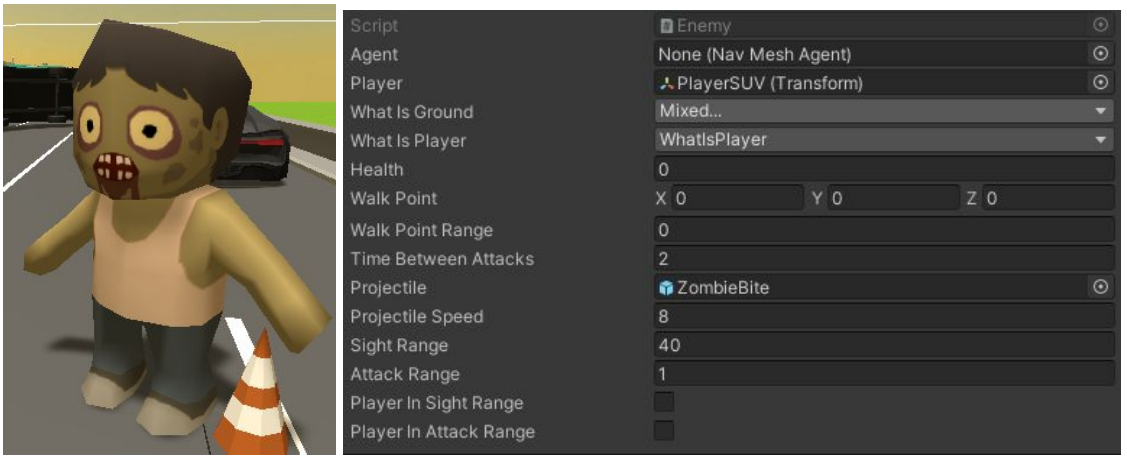
Feinde dienen dazu, die Schwierigkeit der Missionen zu erhöhen. In DriverGame sind drei Arten von Feinde zu finden:

- Statische Feinde: Feinde, die sich nicht bewegen aber werfen Steine zum Auto, wenn es in ihren “Attack Range” ist.



- Dynamische Feinde: Feinde, die in richtung des Autos laufen und es mit einem *“Baseballschläger”* schlagen.

- Zombies: Eine andere Art der dynamische Feinde, sie laufen zum Auto und beißen es. Sind bis jetzt nur in dem Secret Mission zu finden.



Jeder Gegner besitzt einen NavMeshAgent, der das Mesh der Map lädt. Somit gibt es Flächen für die Gegner wo sie drauf laufen können (Ground) und Flächen wo sie nicht durch laufen können (Häuser, etc.)
Weiterhin besitzt der Gegner ein Enemy Skript.

Mithilfe des Agents kann man nun herausfinden, wo sich der Gegner und wo sich der Player befindet.

Die Variablen playerInSighRange und playerInAttackRange geben an, ob ein Spieler in Sichtweite ist (er wird verfolgt) und/oder ob er in der Reichweite ist zum Attackieren.
Diese Reichweite kann man in Unity für jeden Enemy anpassen.

```
//Check for sight and attack range
playerInSightRange = Physics.CheckSphere(transform.position, sightRange, whatIsPlayer);
playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, whatIsPlayer);
```

```
if (playerInSightRange && !playerInAttackRange) ChasePlayer();
if (playerInAttackRange && playerInSightRange) AttackPlayer();
```

Wenn er nur in Reichweite ist soll er den Spieler verfolgen. Hierbei kann man ganz einfach durch den Agent die Position des Players suchen und sagen, dass der Gegner mit dem Skript sich zu dem Punkt Bewegen soll.

```
private void ChasePlayer()
{
    agent.SetDestination(player.position);
}
```

Wenn der Spieler in Reichweite und in Sichtweite ist soll der Gegner angreifen.
Der Gegner soll stehen bleiben und den Player angucken (um ihn auch zu treffen)

```
private void AttackPlayer()
{
    //Make sure enemy doesn't move
    agent.SetDestination(transform.position);
    transform.LookAt(player);
```

Danach wird ein Rigidbody erstellt, welches das Geschoss ist. Dieses wird nach 2 Sekunden wieder destroyed. Der Rigidbody kriegt noch den Tag ’Bullet’ damit er Schaden am Auto machen kann.

```
Rigidbody rb = Instantiate(projectile, transform.position, Quaternion.identity).GetComponent<Rigidbody>();
rb.AddForce(transform.forward * projectileSpeed, ForceMode.Impulse);
rb.AddForce(transform.up * 8f, ForceMode.Impulse);
///End of attack code

Destroy(rb.gameObject, 2.0f);
```

Der Gegner der das Geschoss wirft hat als Rigidbody eine Sphere gekriegt mit einem Collider und einem Material.

Der Gegner der schlägt hat auch eine Sphere gekriegt, nur ist diese durchsichtig, womit man denkt, dass er nichts wirft, sondern schlägt (auch weil er direkt bis vor den Gegner läuft)
Man kann des weiteren eine Zeit zwischen den Attacken einstellen (mithilfe eines Invoke).

Bemerkung: Es gibt die Attribute Walkpoints und Health beim Gegner. Jedoch hat das im Endeffekt nicht ganz so funktioniert wie wir das wollten und haben es bei dem laufen und attackieren gelassen.

Die Position der Feinde und in welcher Missionen aktiviert werden, wird in Absatz Missionen genauer betrachtet.

Screens

In dem Spiel gibt es verschiedene Screens, die entweder interaktiv oder statisch sind.

Interaktive Screens: ermöglichen dem Spieler einer oder mehrere Aktionen durchzuführen:

- Main-Menu: Game Start, Options oder Quit Game.
- Pause-Menü: letzter Zustand laden, Options, Hauptmenü, Quit oder Resume.
- Options-Menu: Lautstärke, Quality und fullscreen anpassen.
- TimerFailed- / NoHealth- Screen: letzter Zustand laden.

Statische Screen: Informationen zeigen:

- Spiel Start Screen: erzählt dem Spieler eine kleine Geschichte und zeigt ihm, wie er das Auto kontrolliert.
- Mission Start Menu: ergänzt die Geschichte und sagt dem Spieler den Sinn dieser Mission, sowie kleiner Hinweis für den Spieler. Wird solang angezeigt, wie der Spieler am Startpunkt eines Mission bleibt.
- Mission End Screen: zeigt die Zeit, die für die Mission gebraucht wurde sowie die Bewertung der Mission.

Levels und deren Assets

DriverGame hat bis jetzt drei (+eine) verschiedene Missionen.

Jede Mission hat ihre eigene Verlauf und besitzt ihre eigene Assets, sei es hinderniss, rampen, Feinde und/oder PowerUps.

Wie es erwähnt wurde, kann der Spieler am Anfang überall fahren (Free-Roam), ohne irgendeine Mission gestartet zu haben. Es wird aber mit einem “Waypoint” gezeigt, wo die erste Mission starten werden kann.

Wenn der Spieler zu Mission-Startpunkt ankommt, wird ein Screen eingeschaltet, welche eine kurze Geschichte hat und eine Einleitung, wie der Spieler die Mission starten kann. Sobald der Spieler die Mission mit “e” gestartet hat, werden die Mission-Assets eingeschaltet, die Waypoint wird auf das nächste Ziel zeigen und die Countdown wird mit der vordefinierten Zeit gestartet.

Wenn der Spieler innerhalb der Zeit zum Endpunkt ankommt, wird die Startpunkt der nächsten Mission angezeigt. Die ganze Logik erfolgt in dem Script “MissionController.cs” und wird im Teil Interne-Assets näher betrachtet.

Missions

Mission 1

Mission 1 ist die kürzeste Mission in DriverGame, dient vor allem, das Spielkonzept zu verstehen.
Der Spieler bekommt die Aufgabe, eine Frau ins Krankenhaus zu fahren.
Die Mission beinhaltet weder Feinde noch extra hindernisse. Zu finden ist nur ein Zeit-PowerUp, welches die verbleibende Zeit um 5 Sekunden erhöht.

Mission 2

Mission 2 hat eine Liste von geordnete Checkpoints, durch welche der Spieler in einer bestimmten Reihenfolge fahren muss, nur die nächste Checkpoint wird mit dem Waypoint angezeigt.
Diese Mission hat 2 Arten von Feinde, sowie verschiedene Hindernisse und Power Ups (Zeit und Geschwindigkeit), die zwischen die Checkpoints platziert wurden, deswegen ist die Schwierigkeit der Mission mittel.
Als geschichte wird den Spieler gebeten, den Passagier nach Hause zu transportieren, es wurde gefordert, den Autobahn zu nehmen und erwähnt, dass der Passagier Feinde hat.

Mission 3

Mission 3 hat eine Liste von Checkpoints, welche in eine beliebiger Reihenfolge vom Spieler gesammelt werden können. Nachdem der Spieler dieser Mission gestartet hat, wird ein Waypoint pro Checkpoint angeschaltet, sodass der Spieler genau weißt wo alle die Objektive zu finden sind. Sammelt der Spieler ein Checkpoint, so wird sowohl der

Checkpoint als auch die dazu zeigende Waypoint deaktiviert. Sammelt der Spieler der letzte Checkpoint in der Liste, so wird die Mission beendet.
Ähnlich wie Mission 2, hat die Mission 3 Feinde, Hindernisse, Power-Ups und props, welche am Missionsanfang eingeschaltet werden.
Die Geschichte diese Mission geht darum, dass der im Mission 2 transportierte Passagier eine Anzahl von Packete um die Stadt verteilen soll.

Mission 4

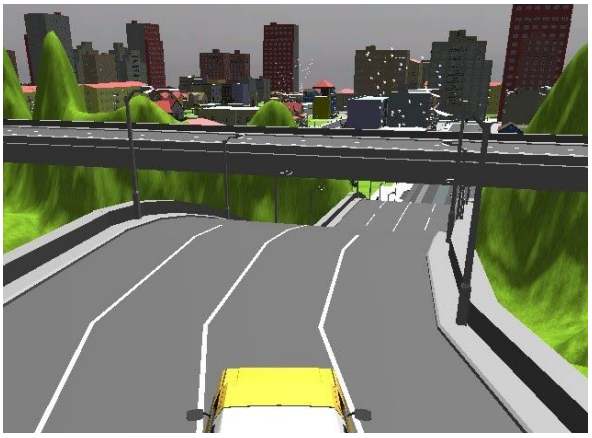
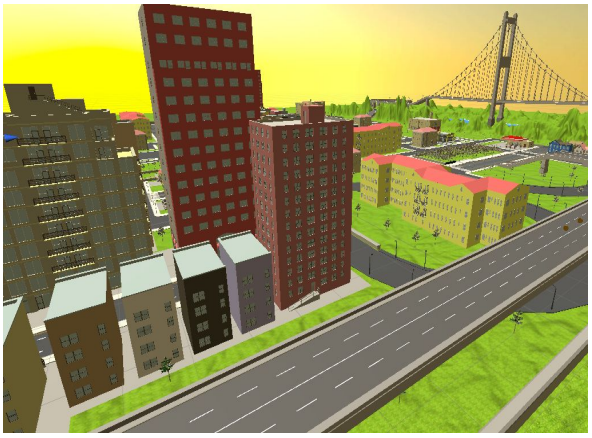
Mission 4 oder Secret Mission ist nur spielbar, wenn der Spieler alle Münzen gesammelt hat und die vorherige drei Missionen erfolgreich abgeschlossen hat. Vom aufbau her, ist diese Mission ähnlich der Mission 2 (Reihenfolge von Checkpoints).
In dieser Mission sind zum ersten Mal die Zombie-Feinde zu finden. Power-Ups und Hindernisse sind hier auch zu finden.
In dieser Mission geht es darum, dass ein Virus die Stadt angesteckt hat und alle die Bewohner Zombies geworden sind. Der Spieler muss die Stadt fliehen.

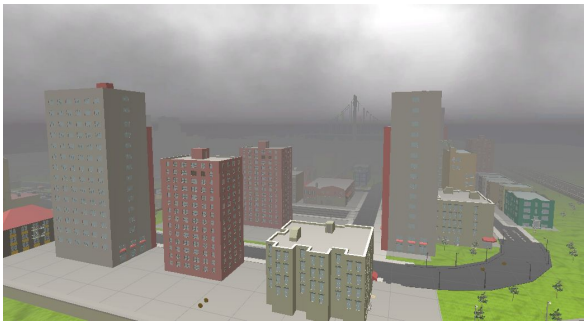
Zusammenfassung der Missionen

#	name	Feinde	Beschreibung	Story
1	Easy Mission	nein	Spieler fährt vom Startpunkt zum Endpunkt	Der Fahrer soll einen Passagier zum Krankenhaus fahren.
2	Medium Mission	Ja	Spieler fährt durch verschiedene Checkpoints in bestimmter Reihenfolge	Der Fahrer holt den Passagier ab und soll ihn nach Hause durch die Autobahn fahren. In der Autobahn sind Feinde zu finden, die verhindern, dass der Fahrer seine Mission abschließt.
3	Hard Mission	Ja	Spieler hat verschiedene Checkpoints, eine bestimmte Reihenfolge gibt es aber nicht	Der Fahrer soll eine bestimmte Anzahl an Pakete in der Stadt verteilen. Dabei gibt es Feinde.
(4)	Secret Mission	Ja	Ähnlich Mission 2. Mit der Unterschied an Musik und Design	Ein Virus hat die Stadt angegriffen. Alle Bewohner sind Zombies geworden, nun muss der Fahrer von der Stadt fliehen.

Design & Look

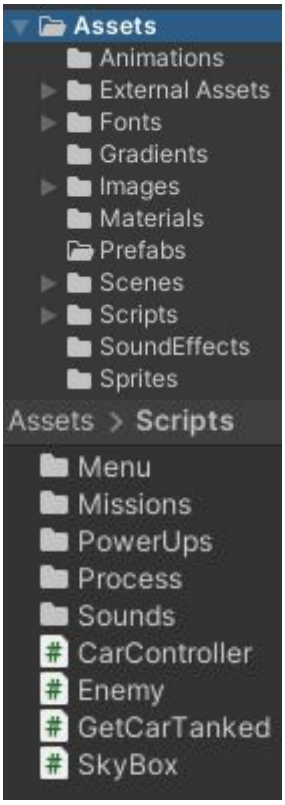
Für das Design der Stadt wurden im prinzip das “POLYGON City Pack” und “BrockenVector” von Unity Store genutzt. Welche die Gebäude, Straßen, Zäune, Lampen, Straßenschilder, Autobahnen, Brücke, Rampen und Kurven liefern.
Es sind verschiedene Arten von Gebäude vorhanden. Hoch- oder Einfamilienhäuser.
Da die “POLYGON City Pack” nur Straßen ohne Kurven hat, wurde für ein schönes Fahrerlebnis dazu “BrockenVector” benutzt, die Kurven, Autobahnen sowie Brücken hat.
Außerdem wurde für die Autos das “Stylized Vehicles Pack” importiert, die verschiedene Autos (SUV, Bus und Sport Auto) zur Verfügung stellt.
Für jede Mission gibt eine andere Skybox (Von Package “Day-Night Skyboxes”) mit verschiedene Lichteinstellungen, sodass jede Mission ein eigenes Gefühl hat. Es fängt früh morgens mit der “Easy-Mission” bis mitternacht mit der “Secret Mission”.


Assets

Assets Struktur



Es wurde versucht, eine eindeutige Struktur so halten und eine Namensgebung so zu wählen, dass es immer Eindeutig wird, welcher Element wohin gehört.
Im folgenden Absatz werden die “External Assets” erläutert.

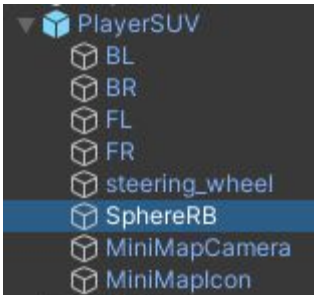
Das war besonders wichtig für das Scripts-Unterordner.
Die drei Wichtige Skripte in unserem Projekt sind:

- Car Controller
- Mission Controller
- Process Controller

Car Controller

Zuständig für sowohl die Bewegung vom Auto, als auch für die Kollisionen, die Neigung, Räder Bewegung und die Particle Effects vom Auto.

Bewegung



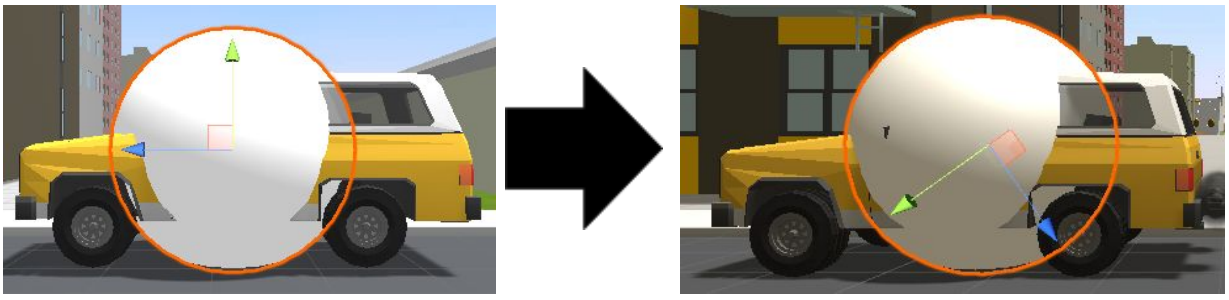
Die Beschleunigung und Bremsen des Autos sind mit einem Kugel verbunden. Rechts als “SphereRB” dargestellt.
Bewegt sich die Kugel nach “vorne”, bewegt sich somit das Auto auch.
Anhand mehrere Aspekten wird es in jedem Frame die Variable “CurrentVelocity” neu berechnet.

```
//////////////////////Speed Control//////////////////////

//Set the right current Velocity each frame depending on the user input
if (Input.GetAxis("Vertical") > 0)...
else if (Input.GetAxis("Vertical") < 0)...
//Return to zero speed if not a vertical input found
else...

//ensure the velocity never goes out of the backFinal/final boundaries
currentVelocity = Mathf.Clamp(currentVelocity, backFinalVelocity, finalVelocity);
```

Die berechnete Current Velocity wird auf der Kügel als Beschleunigungs-Kraft eingesetzt. (Current Velocity kann auch negativ werden).

```
0 references
private void FixedUpdate()
{
    //Move the sphere forward and backward
    if (Mathf.Abs(currentVelocity) > 0)
    {
        if(handbrake)
        {
            //Apply force to the Sphere
            sphereRigidBody.AddForce(transform.forward * currentVelocity * 0.9f, ForceMode.Acceleration);
        }
        else
        {
            //Apply force to the Sphere
            sphereRigidBody.AddForce(transform.forward * currentVelocity, ForceMode.Acceleration);
        }
    }
}
```

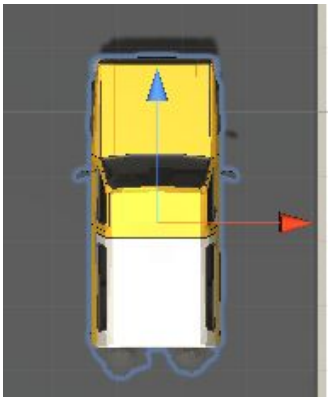
Somit wird der Kugel sich um ihre X-Achse gedreht und das Auto folgt die Position der Kugel.

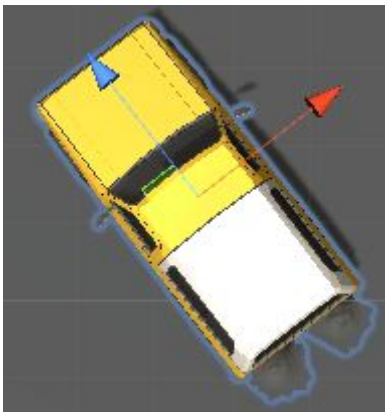


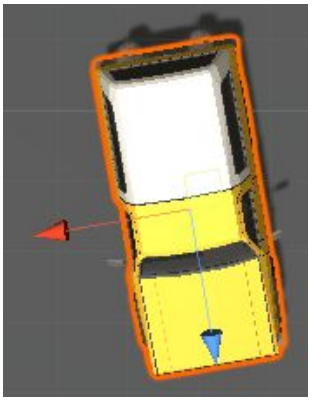
Die Lenkung des Auto wird anhand der Input der Nutzer berechnet und auf nur das Auto angewendet. (Es wird das Y-Achse der Auto rotiert)

```
//////////////////////Turning Control//////////////////////

//Lateral movement with input
turnInput = Input.GetAxis("Horizontal");

//Defines how the turning works when going forwards
if (currentVelocity > 0f)
{
    if (handbrake)
    {
        transform.rotation = Quaternion.Euler(  transform.rotation.eulerAngles +
                                                new Vector3(0f, turnInput * turnStrength *
                                                handbrakeTurnMultiplier * Time.deltaTime *
                                                currentVelocity, 0f));

    }
    else
    {
        transform.rotation = Quaternion.Euler(  transform.rotation.eulerAngles +
                                                new Vector3(0f, turnInput * turnStrength *
                                                Time.deltaTime * currentVelocity, 0f));
    }
}

//Defines how the turning works when going backwards
if (currentVelocity < -5f)
{
    transform.rotation = Quaternion.Euler(  transform.rotation.eulerAngles +
                                            new Vector3(0f, turnInput * backTurnStrength *
                                            Time.deltaTime * Input.GetAxis("Vertical"), 0f));
}
```

Zur Darstellung der Lenkung:







Das Auto besitzt dazu eine Handbremse, die ermöglicht einen kürzeren Bremsweg und vereinfacht die Lenkung vom Auto. Die Handbremse wird aktiviert solange der Leertaste gedrückt wird.

Kollisionen

Das Auto selbst besitzt ein Collider Box. Mithilfe der Methode OnTriggerEnter und Setzung von Tags in der Elemente der Stadt kann genau beschrieben werden, wie das Auto mit verschiedene Elemente interagiert.

```
private void OnTriggerEnter(Collider other)
{
    // SpeedBoost
    if (other.tag == "SpeedBoost")...

    // TimeBoost
    else if (other.tag == "TimeBoost")...

    // Gravity Boost
    else if (other.tag == "GravityBoost")...

    //If the other object should dissapear on contact
    else if (other.gameObject.tag == "Obstacle_Dissapears")...

    //Needed for the passangers
    else if (other.gameObject.tag == "Person_Dissapears")...

    //Damages the car on contact
    else if(other.gameObject.tag == "Player_Damage")...

    //Damages the car on contact
    else if (other.gameObject.tag == "Bullet")...

    //Damages the car on contact
    else if (other.gameObject.tag == "BatHit")...

    //For the Handling of coins and sounf effect
    else if (other.gameObject.tag == "Coins")...
}
```

Neigung

Sodass der Spieler ein schöneres Gefühl von Geschwindigkeit und Lenkung bekommt, wurden zwei Sorten von Neigung am Auto implementiert.

- Acceleration-Tilt: Wird anhand von der Vertikalen Input (Up und Down) des Nutzer gesteuert. Bei keinen Input der Nutzer kehrt der Neigung wieder auf null zurück.



- Turning-Tilt: Wird anhand von der Horizantalen Input (Links und Rechts) des Nutzer gesteuert. Hat eine Wirkung ab einer Bestimmten Geschwindigkeit (Variable: tiltStartAtSpeed) Bei keinen Input der Nutzer kehrt der Neigung wieder auf null zurück.



Es ist auch möglich, dass beide Neigungen gleichzeitig auf das Auto eine Wirkung haben.

Räder Bewegung

Wenn der nutzer einem horizontalen Bewegung macht, so drehen sich die Räder wie im “echten-Leben” auch.

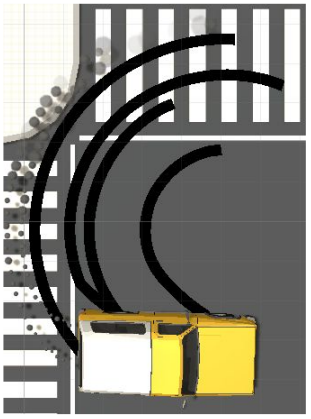

Particle Effects

Das Auto hat verschiedene Particle Effects. Manche werden durch Kollisionen getriggert (Obstacle_Dissapear, Player_Damage) andere sowie den Abgasrauch oder die hinterlassene Räder Spüren werden mit Variablen der Auto gesteuert.

- Per Kollision getriggerte Particle Effects: Anhand der Tags von den verschieden elementen werden die Particle Effects getriggert.

- Tire Tracks: Werden unter bestimmten Bedingungen eingeschaltet:

```
//////////////////////Trails Manager//////////////////////

//where should be the trails generated
if ((currentVelocity <= finalVelocity * 0.5f && Input.GetAxis("Vertical") > 0f)
    || (Mathf.Abs(currentLateralTilt) > lateralMaxTilt * 0.9f)
    || (Input.GetAxis("Vertical") < 0f)
    || handbrake)
{
    SetEmitionOnAllWheels(true);
}
else if(currentVelocity > finalVelocity * 0.5f || Input.GetAxis("Vertical") <= 0f)
{
    SetEmitionOnAllWheels(false);
}
```



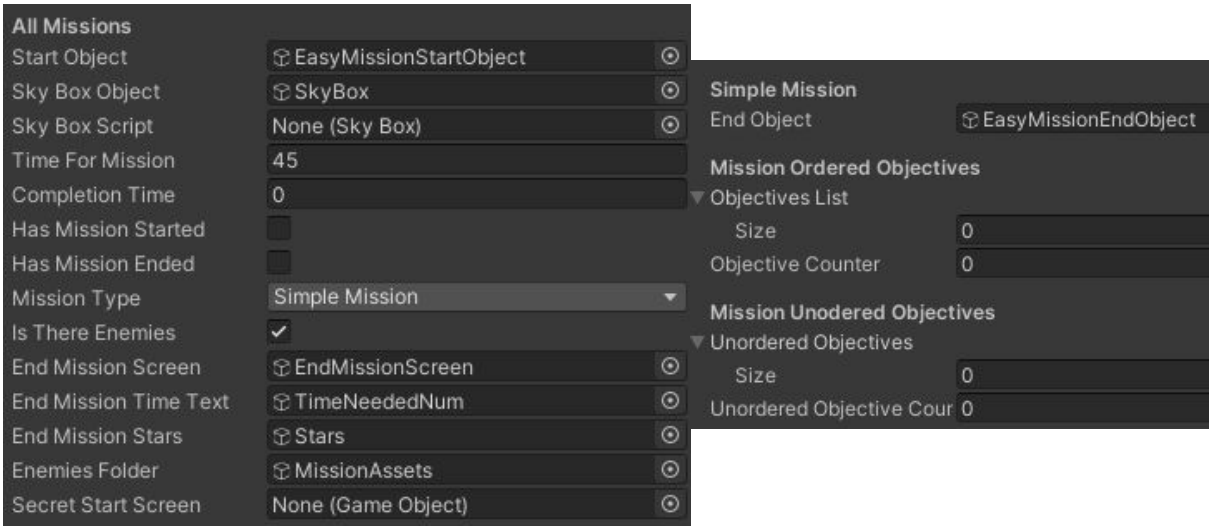
- Abgas Rauch oder Feuer: Das Auto produziert immer Rauch. Wenn das Speed-Power-Up benutzt wird, wird der Rauch zum Feuer.

Mission Controller

Kümmert sich um den Ablauf des Spieles. Besitz eine Liste von “SingleMissions”, welche zu verschiedene Kategorien gehören können. Jede Art von Mission hat einen bestimmten Ablauf, welcher in der “MissionType” festgestellt wurde.

```
//Enum for the definition of the mission type
6 references
public enum MissionType
{
    simpleMission,                          //From point A to point B
    missionArrayOfObjectivesOrdered,        //From point A to point B through a list of points
    missionArrayOfObjectivesDissordered
}
```

Dazu hat jeder Mission verschiedene Einstellungen, welche im Unity gesteuert werden können. Es gibt typspezifische Einstellungen und allgemeine Einstellungen.



In der Szene sieht in etwa jeder Mission wie folgt aus: (Nicht alle Missionen sehen genauso aus, das ist Abhängig von Mission Typ)

Medium Mission hat den Single Mission script. Das ist was in den beiden obigen Bilder zu sehen ist.

Jeder Mission hat ein Start Object mit dem Skript “StartMissionTrigger.cs”, indem die Mission angefangen werden kann.

Medium Mission ist von Typ “Mission Ordered Objectives”, deswegen gibt es ein Ordner von Objectives, die in der Liste der Medium Mission “Ordered Objectives” einzutragen sind.

Jede Mission hat ein Start Screen, welche angezeigt wird, solange der Spieler sich im Start Object befindet.

Jede Mission hat ein Ordner von Assets, welche eingeschaltet werden, sobald die Mission gestartet wurde und wieder ausgeschaltet, wenn die Mission zu Ende kommt.

Jede Mission hat ein Skybox mit dem Skript “Skybox.cs” welche die Lichteinstellungen der Szene während der Mission regelt.

An der Seite der Mission Controller wird das ganze mit einem Switch-Case gesteuert, wenn eine Mission gestartet wird. Siehe Bool “Has Mission Started” im Single Mission.

```
//Define start procedure for each mission type
switch (currentMission.missionType)
{
    //Mission is to go from point A to point B
    case SingleMission.MissionType.simpleMission:
    //Go through a list of objects in a given order
    case SingleMission.MissionType.missionArrayOfObjectivesOrdered:
    //Go through a list of objects in any order (Hit enemies for example)
    case SingleMission.MissionType.missionArrayOfObjectivesDissordered:
```

Analog zu “Has Mission Started” gibt es die Variable “Has Mission ended”. Diese Variable besagt, ob eine Mission erfolgreich beendet wurde. Wenn diese Variable auf “true” gesetzt ist, wird der Mission Controller erstmal die gebrauchte zeit im Feld “Completion Time” eintragen und dann entweder die nächste Mission vorbereiten oder das EndGameScreen einschalten. Am besten in dem MissionController reingucken und anhand der Kommentare sich orientieren.

Process Controller

Zuständig für das speichern von Zustände und das Laden von gespeicherte Zustände und das Aktivierung und steuern der noch zu sammelne Münzen.

Save Game

```
//Function to save the current progress
2 references
public void SaveGame(Vector3 position,
                     Quaternion rotation,
                     List<GameObject> collectedCoins,
                     List<SingleMission> doneMissions,
                     float currentHealth)
{
    Debug.Log("Saved Game");

    this.autoPosition = position;
    this.autoRotation = rotation;
    this.collectedCoins = collectedCoins;
    this.doneMissions = doneMissions;
    this.currentHealth = currentHealth;
}
```

Diese Funktion kümmert sich alle relevante Variable zu speichern. Die Funktion wird mehrmals während das Spiel abgerufen. Einmal wenn das Spiel gestartet wird und dann jedes mal, dass der Spieler eine Mission erfolgreich abschließt.

Load Game

```
//Function to load the last saved game
1 reference
public void LoadGame()
{
    Debug.Log("Loaded Last Saved Game");

    playerSphere.transform.position = this.autoPosition;

    player.transform.rotation = this.autoRotation;
    carController.ResetAllMovementValues();

    carController.SetCollectedCoins(this.collectedCoins);
    carController.SetCollectedCoinsCounter(this.collectedCoins.Count);

    carController.healthBar.setHealthBarValue(this.currentHealth);

    missionController.doneMissions = this.doneMissions;

    ActiveNotCollectedCoins();

    if(missionController.currentMission.hasMissionStarted)...
}
```

Diese Funktion lädt der gespeicherte Zustand (Welche mit SaveGame gespeichert wurde).

Anbei muss geachtet werden, ob eine Mission zum Zeitpunkt der Abruf läuft, wenn das der Fall ist, soll der Fortschritt der Mission zurückgesetzt werden.

Diese Funktion ist von mehrere Stelle abrufbar. Von der Pause-Menu ist es immer verfügbar. Auch wenn der Spieler eine Mission nicht rechtzeitig schafft oder keine Lebenspunkte mehr hat, kommt diesem Funktion zum Einsatz mithilfe eines Buttons.

Activate Not Collected Coins

```
2 references
public void ActiveNotCollectedCoins()
{
    for (int i = 0; i < allCoins.Count; i++)
    {
        allCoins[i].SetActive(true);
    }

    for (int i = 0; i < allCoins.Count; i++)
    {
        for (int j = 0; j < collectedCoins.Count; j++)
        {
            if(allCoins[i] == collectedCoins[j])
            {
                allCoins[i].SetActive(false);
            }
        }
    }
}
```

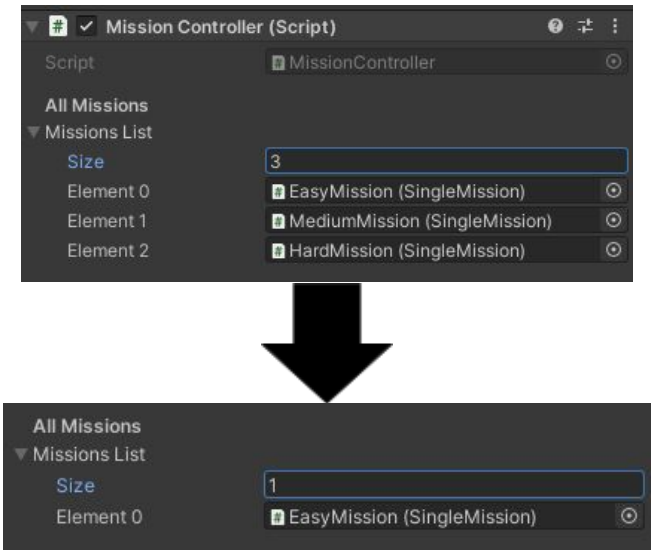
Dieser Mission geht durch die Liste aller Münzen und aktiviert nur die, die der Spieler noch nicht gesammelt hat.
Diese Funktion ist für den Fall gebraucht, dass der Spieler Münzen gesammelt hat, der Zustand nicht gespeichert wurde und er auf Load Game geht. Dann sollen die gesammelte Münzen wieder aktiviert werden und von der Liste des gesammelte Münzen entfernt.

Test Empfehlungen

Will man eine bestimmte Mission im Unity Testen oder die Secret Mission spielen ohne zuvor alle Münzen gesammelt zu haben, gibt es die Möglichkeit folgendes zu machen:

Fall 1: Ich will nur eine bestimmte Mission Spielen

1) In Mission Controller folgender Liste auf eins setzen:



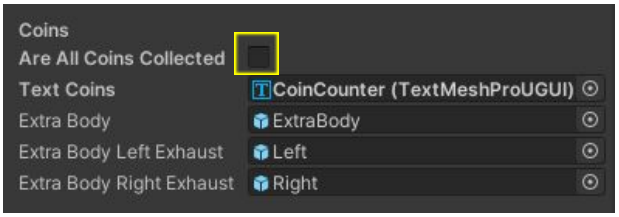
2) Dann per Drag and Drop eine Single Mission an der Stelle der Element 0 reinziehen. (Eine von folgender Drei, nicht möglich mit der Secret Mission!)

Nächstes Mal, dass das Spiel anfängt wird in diesem Fall nur das Medium Mission vorkommen.

Fall 2: Ich will die Secret Mission spielen

Falls man die Secret Mission spielen will, aber davor nicht alle andere Missions nicht durchspielen will und/oder die Münzen nicht einsammeln will. Folgendes machen.

1) Schritte 1) und 2) vom Fall 1 (Oben) machen. (Wir wollen nur eine Mission und am besten das “Easy Mission”, sodass wir schnell und einfach zur Secret Mission kommen können.
2) Bevor oder während des Spielt folgendes Checkbox im Car Controller (Im PlayerSUV zu finden) einchecken:
3) Nach Beendigung der Mission, wird das Secret Mission anfangen.



Externe Assets

Für manche Funktionen wurden externe Bibliotheken benutzt.

Package	**Beschreibung**
POLYGON City Pack	Assets für die City (Häuser, Straßen…).
Stylized Vehicles Pack Free	Verschiedene Autos einschließlich das Hauptauto.
ToonyTinyPeople	Menschliche Figuren (Enemies)
BrokenVector	Straßen Assets (Brücken, Kurven und Autobahn)
Low Poly Water	See
EffectTexturesAndPrefabs	Particle Effects
Day-Night Skyboxes	Materials für die Skybox
Ground textures pack	Materialien für die Boden (Gras)
GasPumpMiniMap	Tankstelle Minimap-Symbole
Clock Obj.	PowerUp Zeiterhöhen
NavMesh Components	Feinde Mesh
Universal Accessories	Power Ups und Auto Teile
Cron_audio_8_bit_loops_free	Hintergrund Musik
Electric Sfx	Sound effects
TextMesh Pro	Fonts und UI design
Low Poly Survival Pack	Gegenstände für Camp
Tents	Zelte für Camp
Cinemachine	Hauptkamera des Spiels

Manche Audio Clips und Hintergrundmusik wurden von folgender Seite heruntergeladen: https://www.soundsnap.com/

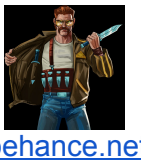
Manche Icons kommen von: https://game-icons.net/

Mehrere Youtube Tutorial wurden mitgemacht.

- Einige von diesem Channel:
- https://www.youtube.com/c/Brackeys
- Waypoints: https://www.youtube.com/watch?v=mT3ggeU82f0
- Enemies: https://www.youtube.com/watch?v=UjkSFoLxesw
- Power-Ups: https://www.youtube.com/watch?v=fDXtMIL2ahU

- Coins: https://www.youtube.com/watch?v=vvzXXkJx3BY&ab_channel=Lotosos
- Health-Bar: https://www.youtube.com/watch?v=BLfNP4Sc_iA&t=1075s&ab_channel=Brackeys
- Minimap: https://www.youtube.com/watch?v=IccrJXiCQxA&t=619s&ab_channel=Unity
- Auto Steuerung: https://www.youtube.com/watch?v=cqATTzJmFDY&ab_channel=gamesplusjames

Bilder:

SecretMission-Char: 
https://mir-s3-cdn-cf.behance.net/project_modules/max_1200/9a322113225565.56271a32c222e.png

Andere Bilder: https://www.istockphoto.com/de

Projektmanagement

Die Entwicklung war nah an einer agilen vorgehensweise. Das ganze Team hat (fast) jeder Woche sich gesammelt. In diesen Meetings wurden Themen über den Stand des Spiels diskutiert, sowie Aufgaben verteilt, die bis nächstes Meeting erledigt sein sollten.

Die Arbeitsaufteilung war so, dass jedes Mitglied an alle Themen mitgemacht hat.

Wir hatten einen Plan mit Meilensteinen, den wir zwar manchmal anpassen mussten, welcher aber an sich gut geklappt hat. Wir haben uns dabei eher aufwendige Teile auf verschiedene Wochen gelegt und nebenbei nicht so schwere Sachen gemacht. Zum Beispiel gab es eine Woche indem wir uns eher mit dem MissionController und den Missionen an sich beschäftigt haben und eine Woche danach haben wir uns die Enemies vorgenommen (beide Teile haben einen hohen Aufwand). Nebenbei haben wir uns dann mit dem Design und z.B. der Gestaltung der PowerUps oder der Gestaltung der Partikel auseinandergesetzt.

Tester Review

Tester #1:
Ich hätte mich auch gewünscht, dass die Schwierigkeit des Spiels ausgewählt werden kann. Im allgemein fande ich, dass die Idee und Grafik des Spiels sehr schön ist. Die Anleitung für das Spiel ist gut, was für als Spielanfänger sehr sinnvoll ist. Ich würde gerne die Möglichkeit haben, das Auto so konfigurieren, wie ich es gerne haben würde (Geschwindigkeit, Wie das Auto Lenkt, die Farbe des Autos uzw.)

Tester #2:
Das Spiel fühlt sich gut an und sieht auch cool aus. Was man ändern könnte ist der Wegpunkt der zur nächsten Mission zeigt. Wenn er außerhalb des Sichtbereiches ist, weiß man nicht wo man lang fahren muss. Man könnte ihn vielleicht an den Bildschirmrand immer machen, wenn er außerhalb wäre.

Tester #3:
Sehr gute Idee mit dem Spiel! Die Level sind cool und machen Spaß. Ich würde mir nur ein bisschen mehr zum Konfigurieren wünschen, wie z.B. ein neues Auto mit anderer Geschwindigkeit und vielleicht auch eine größere Stadt.

Noch To-Do & Ideen für Weiterentwicklung

Autos Raucheffekt dynamisch mit der Geschwindigkeit so zu ändern, dass mehr Rauch kommt wenn das Auto schneller Fährt oder aufsteigt.

Es ist auch gewünscht zusätliche Missionen mit verschiedene Komplexitäten zu implementieren sowie die Menschliche Figuren ins Spiel zu bringen und zu animieren.

Außerdem wäre es besser, wenn manche Elemente des Spiels künstliche Intelligenz hätten. Zum Beispiel:

- Enemies mit mehr Freiheit haben in der Stadt sich zu bewegen
- Leben in die Stadt zu bringen (Stadtbewohner und Autos, die auf die Straßen Fahren usw.)

Es war auch geplant, dass wenn der Spieler nicht in eine Mission sich befindet, andere Beschäftigungen in der Stadt zur Verfügung stehen:

- Achievements sammeln: So schnell fahren wie möglich durch ein checkpoint, eine bestimmte Zeit sich in der Luft zu befinden uzw.
- Optionale Missions

Man könnte die Stadt noch erweitern und mehr Gebiete sowie Stadtteile und Parks hinzufügen. Der Map sind keine Grenzen gesetzt, wenn es um die Größe geht.

Das CarController könnte mehr entwickelt werden.

Sonstige

Die Inspiration für das Projekt war das “Crazy Taxi” Spiel, welches wir früher gespielt haben und uns viel Spaß beim Spielen gebracht hat.

Wir haben uns überlegt, wie wir ähnliche Spiel implementieren können und die erste Vorstellung des Spiels war: