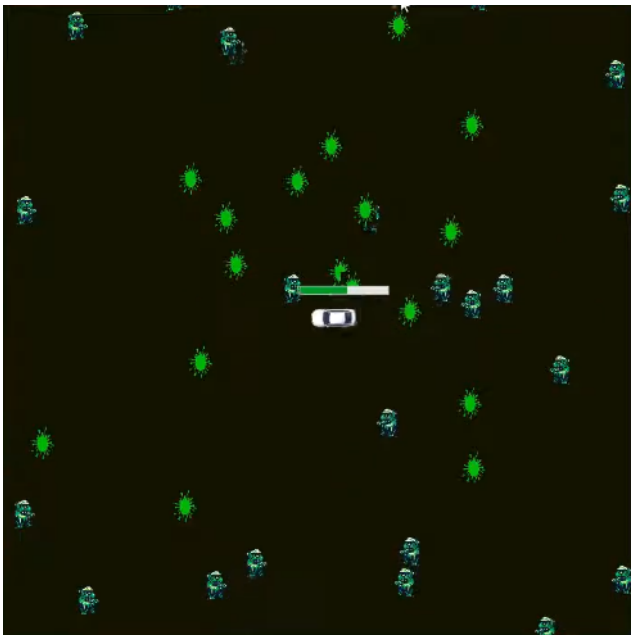
Desenvolupament d’un videojoc

Idea i concepte

Per a la programació del joc utilitzarem el motor Unity, ja que personalment tinc bastanta experiència amb l'entorn. Per a la idea del joc vull basar-me en un petit projecte personal que vaig començar fa anys a Visual Studio.

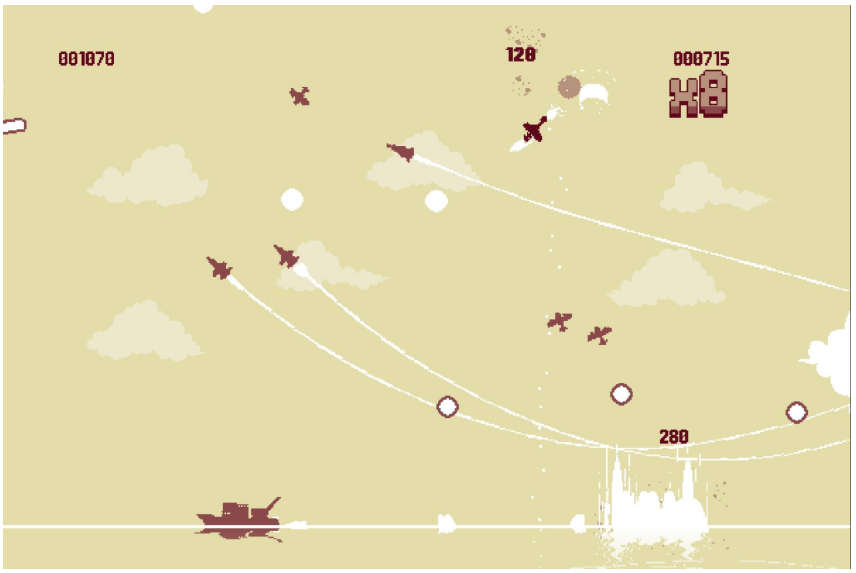


https://www.youtube.com/watch?v=v1Af82WKZXc

El concepte del joc és un arcade de conducció en vista 2D i basat en físiques. El objectiu serà eliminar conduint als diferents enemics per aconseguir la major puntuació possible. El potencial jugador serà d'un perfil molt casual, sense fer falta una gran experiència jugant, però alhora que es pugui marcar la diferència i resulti entretingut per a jugadors més habituals. Per aquest motiu, el esquema de control serà el més senzill possible, tan sols 2 accions i un sol verb, girar, esquerra i dreta, i que el cotxe acceleri automàticament.

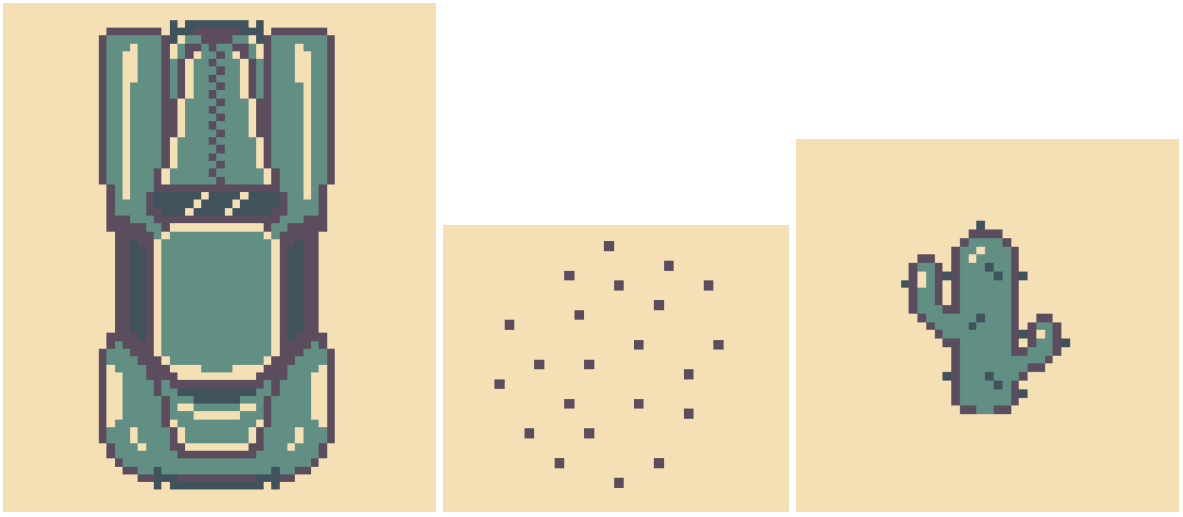
Disseny

El primer és trobar un estil artístic que resulti senzill de dibuixar, minimalista i alhora diferent. Per això em vaig inspirar en un videojoc que complia certes característiques similars al meu concepte, Luftrausers.



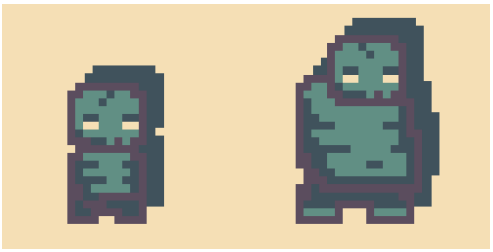
En aquest cas, es tracta d'un joc en vista vertical, i el meu concepte serà de vista "top-down", pero l'estil de conservar una sola paleta de colors durant tot el joc em va agradar, així que vaig utilitzar una paleta de colors de la pàgina Adobe Color.
En el nostre cas, vaig escollir 4 colors tan sols:

I amb els colors vaig dissenyar el personatge a controlar (un cotxe tipus HotRod anys 50) i un parell de dibuixos per fer l'escenari, inspirant-me en Assets gratuïts de la pàgina Itch.io.



Per acabar de donar-li un estil personal al meu projecte, aplicaré un efecte d'ombra darrere dels objectes, duplicant el sprite y passant-lo a una textura d'un sol color.

També dibuixaré els principals enemics que haurà de derrotar el jugador amb les seves animacions.



Programació

Moviment del cotxe

Un cop tinc els dibuixos per començar, programaré les físiques i moviment del cotxe, que estarà molt basat en el control de costat o drift.

Els programes d'Unity utilitzen dues funcions que venen al crear-los, la funció Start();
Que és molt similar al Main() d'altres programes, executa el contingut al iniciar-se, i sol utilitzarse per carregar variables i components d'Unity.
L'altre funció es Update();
Aquesta segona funciona similar a un comptador de Visual Studio, i executa el codi cada fotograma que es mostra en pantalla, aquí és on trobem gran part del codi en jocs amb bases molt reactives o d'acció, on és important actuar ràpid i tenir informació en temps real.
A la funció Update programaré els registres e inputs dels controls, petits ajustaments d'aquests mateixos i animacions com el gir de les rodes.

```
void Update() {
    //steeringInput = Input.GetAxis("Horizontal");
    bool LEFT = Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow) || ButtonLeft.ispressed;
    bool RIGHT = Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow) || ButtonRight.ispressed;
    if(LEFT){
        steeringInput -= Time.deltaTime * (wheelSteeringFactor + wheelAdaptativeSteeringFactor*Mathf.Abs((steeringInput + 1)/2));
        LeftEffect.GetComponent<ParticleSystem>().startLifetime = smokeTime;
    }else{
        LeftEffect.GetComponent<ParticleSystem>().startLifetime = 0f;
    }
    if(RIGHT){
        steeringInput += Time.deltaTime * (wheelSteeringFactor + wheelAdaptativeSteeringFactor*Mathf.Abs((steeringInput - 1)/2));
        RightEffect.GetComponent<ParticleSystem>().startLifetime = smokeTime;
    }else{
        RightEffect.GetComponent<ParticleSystem>().startLifetime = 0f;
    }
    if(RIGHT && LEFT){
        actualMaxSpeed = Mathf.Lerp(actualMaxSpeed, MinMaxSpeed.y, Time.deltaTime);
        if(accelerationFactor == defaultAccelerationFactor){
            this.transform.localScale = new Vector3(0.9f, 1.1f, 1);
        }
        accelerationFactor = defaultAccelerationFactor * boostAcceleration;
    }else{
        actualMaxSpeed = Mathf.Lerp(actualMaxSpeed, MinMaxSpeed.x, Time.deltaTime*3);
        accelerationFactor = defaultAccelerationFactor;
    }
    if(!RIGHT && !LEFT){
        steeringInput -= Time.deltaTime * Mathf.Sign(steeringInput) * resetWheelSteeringFactor;
    }
    steeringInput = Mathf.Clamp(steeringInput, -1, 1);
    this.transform.localScale = Vector3.Lerp(transform.localScale, Vector3.one, Time.deltaTime * 2f);


    if(pauseMenu.paused && Mathf.Abs(steeringInput) >= 1){
        pauseMenu.paused = false;
        pauseMenu.unPauseFirstPause();
    }

    Vector3 wheelRotation = new Vector3(0,0,steeringInput * -30);
    tyreLeft.transform.localEulerAngles = wheelRotation;
    tyreRight.transform.localEulerAngles = wheelRotation;

    if(transform.position.magnitude > escenario.escenarioRadius + 5)
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
```

I a la funció FixedUpdate programaré totes les físiques, ja que requereixen d'un control més precís alhora de reproduir un moviment més creïble.

Aquí implementaré el vector força que mou el cotxe, el límit de velocitat i un parell de correccions a les físiques com una funció que mata el vector normal del cotxe per corregir el sobreviratge o una funció que aplica més gir de volant si el cotxe es troba de costat, per a que el control respongui més fàcilment.

```
if(pauseMenu.paused)
    return;
//Create a force for the engine
float speedIncrement = (actualMaxSpeed - carRigidbody2D.velocity.magnitude)/actualMaxSpeed;
Vector2 engineForceVector = transform.up * accelerationFactor * Time.fixedDeltaTime * (1+speedIncrement);
carRigidbody2D.AddForce(engineForceVector, ForceMode2D.Force);

//Limit max speed
if (carRigidbody2D.velocity.magnitude > actualMaxSpeed) {
    carRigidbody2D.velocity = Vector2.ClampMagnitude(carRigidbody2D.velocity, actualMaxSpeed);
}

//Kill Orthogonal Velocity
Vector2 forwardVelocity = transform.up * Vector2.Dot(carRigidbody2D.velocity, transform.up);
Vector2 rightVelocity = transform.right * Vector2.Dot(carRigidbody2D.velocity, transform.right);
//Debug.Log(rightVelocity.magnitude/maxSpeed);
carRigidbody2D.velocity = forwardVelocity + rightVelocity * ((1-driftFactor) + driftFactor*(rightVelocity.magnitude/actualMaxSpeed));

rotationAngle -= steeringInput * steeringSpeed * Time.fixedDeltaTime;
carRigidbody2D.MoveRotation(rotationAngle);
```

Generació del escenari

Ja que es tracta d'un joc amb un fort component arcade, el que pot provocar que el joc es noti repetitiu, vaig pensar en un mètode per generar un escenari de forma aleatòria i procedural utilitzant un famós algoritme anomenat soroll de Perlin.

Aquest algoritme, que és molt utilitzat a videojocs de generació procedural com Minecraft per exemple, genera de forma infinita uns valor disposats en una graella que es formen de manera aleatòria però no el suficient per resultar caòtic, el resultat és una espècie de "mapa" on podem observar segons la profunditat del color fent un símil amb el terreny, diferents planicies, muntanyes i cordilleres. Això es just el que jo vull, i amb aquest mètode, generar als pics els cactus i a les planícies camins o bancs de sorra al nostre joc.

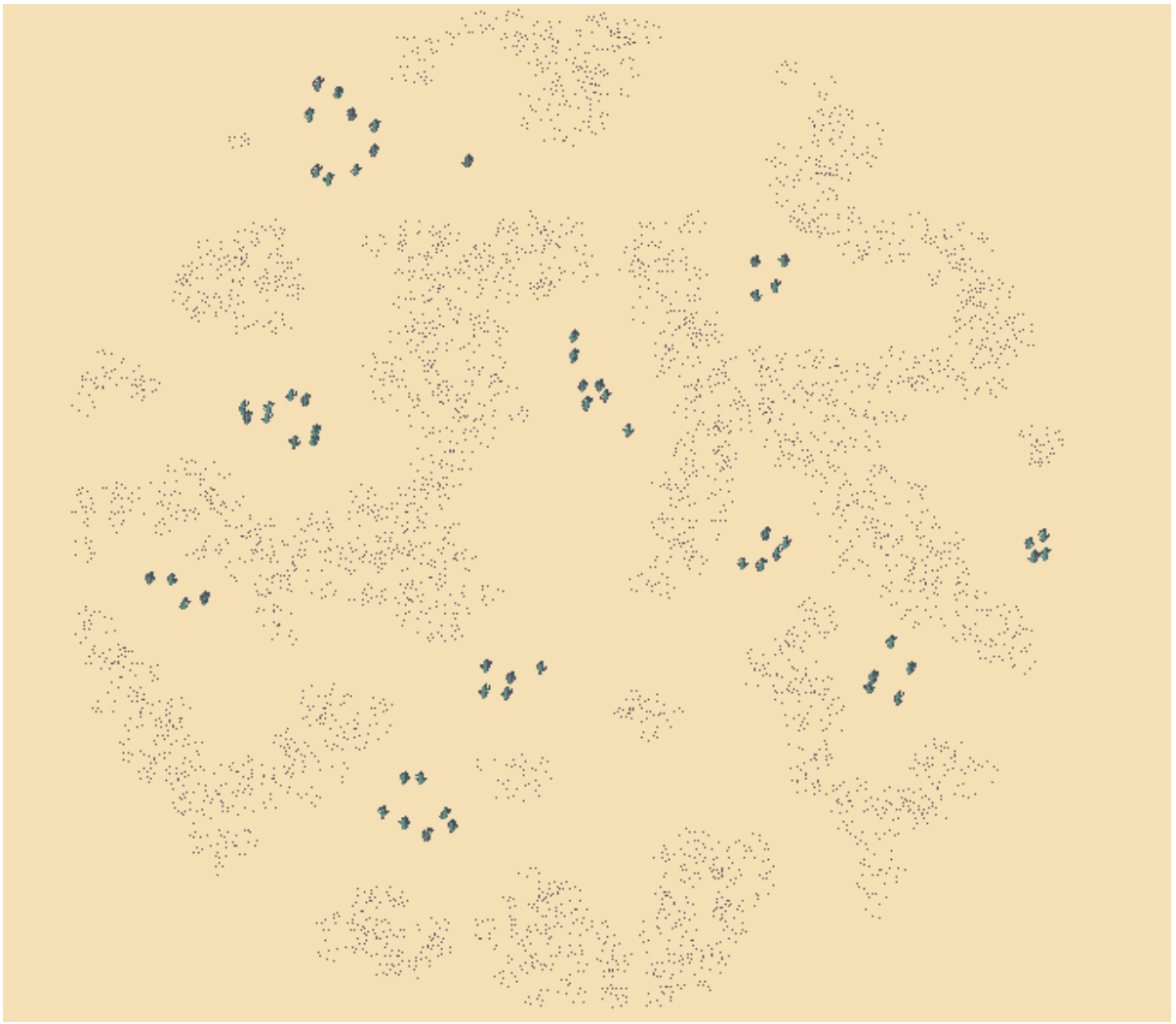
A més li dono forma de cercle i limito el moviment del jugador si intenta sortir de l'escenari generat. També afegeixo un colisionador als cactus perquè es transformin en un element jugable.

```
void Start(){
    escenarioSizes.x = escenarioRadius*2f;
    escenarioSizes.y = escenarioRadius*2f;

    perlinOffSet = new Vector2(Random.Range(0,10000), Random.Range(0,10000));

    for(float x = 0f; x < escenarioSizes.x; x += escenarioSpacing){
        for (float y = 0f; y < escenarioSizes.y; y += escenarioSpacing){
            Vector2 calcCirc = new Vector2(x-escenarioRadius, y-escenarioRadius);
            if(calcCirc.magnitude <= escenarioRadius && calcCirc.magnitude > 2){
                float sample = Mathf.PerlinNoise((x + perlinOffSet.x) * perlinScale, (y + perlinOffSet.y) * perlinScale);
                if(sample > 0.75f && sample < 0.85f){ //Probabilidad de cactus
                    GameObject scenarioObj;
                    Vector2 rndPos = Random.insideUnitCircle.normalized * escenarioSpacing*0.3f; //Random de la posicion
                    scenarioObj = Instantiate(cacti, new Vector2(x + escenarioPosition.x + rndPos.x, y + escenarioPosition.y + rndPos.y), Quaternion.identity);
                    scenarioObj.transform.parent = this.transform;
                }else if(sample < 0.415f){ //Probabilidad de camino
                    GameObject scenarioObj;
                    Vector2 rndPos = Random.insideUnitCircle.normalized * escenarioSpacing*0.3f; //Random de la posicion
                    scenarioObj = Instantiate(path, new Vector2(x + escenarioPosition.x + rndPos.x, y + escenarioPosition.y + rndPos.y), Quaternion.identity);
                    scenarioObj.transform.parent = this.transform;
                    scenarioObj.transform.eulerAngles = new Vector3(0,0, Random.rotation.eulerAngles.z);
                }
            }
        }
    }
}
```

Després d'ajustar uns quants paràmetres el resultat és prou bó:
Es formen petits boscos de cactus i planicies de manera aleatòria i no es nota massa artificial.



Gameplay, combat i enemics

Aquesta és la part que més temps pot consumir del projecte, ja que trobar el balanç adequat en un joc basat en gran part en el combat i aconseguir la màxima puntuació pot ser difícil.

El primer de tot va ser colocar un punt al cul del cotxe per identificar la velocitat angular a la que es mou, i així poguer saber quan es troba derrapant per saber si pot atropellar els enemics. Igual d'important era fer un sistema de detecció de danys coherent; si el cotxe impacta sobre un enemic amb el costat, el dany serà nul, pero si golpeja de morros, el dany que rebrà dependrà de la força del impacte. Això ho calculo gràcies al vector velocitat del enemic amb el vector velocitat del cotxe i buscant el producte del vector combinat per saber com de proper és a un impacte directament al front.

I un cop tenia el combat queda programar el comportament dels enemics, ja que si aquests es mouen de manera massa aleatòria, el jugador no podrà preveure els seus moviments i el resultarà frustrant, per l'altre banda, si és massa predecible, no trobarà dificultat i s'avorrirà.

Per això he programat un sistema similar a PacMan, on existeixen 4 tipus de comportaments diferents per cada enemic:

```
void FixedUpdate()
{
    if(!pauseMenu.paused){
        Vector2 v2 = car.transform.position - this.transform.position;
        float carDist = Vector2.Distance(transform.position, car.transform.position);

        if(comportamiento == 1){ // NORMAL
            if(carDist > 2f){
                rb2d.AddForce(v2.normalized * moveForce);
            }else{
                rb2d.AddForce(v2.normalized * moveForce/5f);
            }
        }else if(comportamiento == 2){ // DIAGONAL
            rb2d.AddForce(v2.normalized * moveForce);
            v2 = (car.transform.position + car.transform.right*10*(defaultMaxSpeed > originalMaxSpeed ? 1 : -1)) - this.transform.position;
            rb2d.AddForce(v2.normalized * moveForce);
        }else if(comportamiento == 3){ // FLANQUEADOR
            if(carDist > 5f){
                flankVec = v2;
            }
            rb2d.AddForce(flankVec.normalized * moveForce);
        }else if(comportamiento == 4){ // COBARDE
            if(carDist > 3f){
                rb2d.AddForce(v2.normalized * moveForce / 2f);
            }else{
                rb2d.AddForce(v2.normalized * -moveForce / 7f);
            }
        }

        maxSpeed = defaultMaxSpeed + carDist * 0.125f;
        if (rb2d.velocity.magnitude > maxSpeed) {
            rb2d.velocity = Vector2.ClampMagnitude(rb2d.velocity, maxSpeed);
        }
    }
```

El sistema escolleix aleatòriament 1 dels 4 tipus de moviment:

- El primer es dirigeix directament cap al jugador, aplica pressió i força aquest a moure's contínuament.
- El segon és similar al primer, però és més àgil i esquiva els moviments del jugador i busca atacar el front del cotxe, aquest és més intel·ligent i força al jugador a improvisar i no confiar-se.

- El tercer proporciona varietat, ja que procura mantenir la distància amb el jugador i allunyar-se i obliga al jugador a canviar de rols i perseguir els enemics de tant en quant.
- I l'últim és el més diferent de tots, promou estar atent i realitzar reaccions ràpides per part del jugador, ja que agafa carrerilla desde una punta de l'escenari a l'altre i el creua dirigint-se al jugador a tota velocitat.

Alhora, el segon tipus d'enemic, dels dos el més gran, crearà una prioritat en el jugador per acabar abans amb ell. Això ho aconsegueixo fent que per cada segon que passa viu, aquest enemic deixarà diversos núvols tòxics que faran malbé el motor del cotxe.

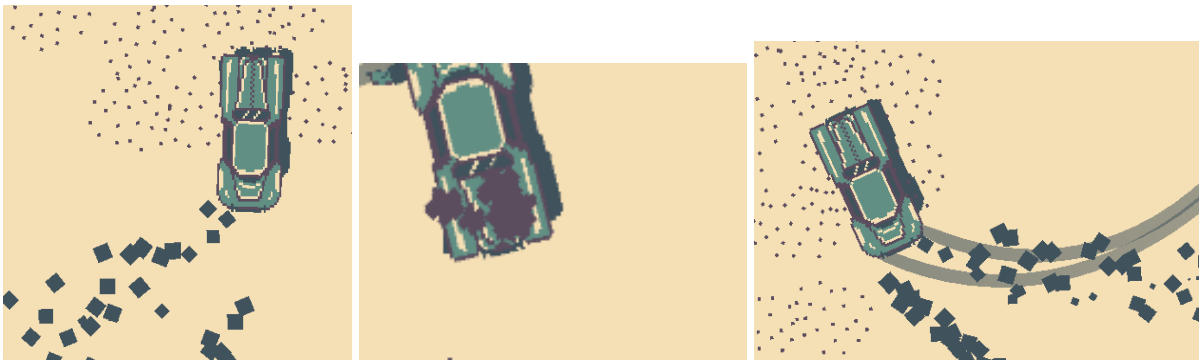
També per donar un objectiu, he implementat un sistema de puntuació, combo i highscore. Per donar-li un petit gir, la puntuació funciona de manera inversa, és a dir, si marca **130** per exemple, aquesta serà la nostra millor puntuació i anirà restant per cada punt que guanyem fins arribar a zero, on registrarà que ara es tracta del nostre millor intent i la puntuació començarà a marcar de nou (**+131+**).

En quant al sistema de combo, aquest sol utilitzar-se a la majoria de jocs com una recompensa per jugar de la manera intencionada que té el dissenyador, en el meu cas, acabar amb els enemics mantenint el cotxe derrapant de costat. Així que cada enemic que derrotes sumarà 1 al multiplicador fins un màxim de x10.
Aquest multiplicador baixarà molt ràpidament, però si mantenim el cotxe de costat, baixarà molt poc a poc.

Pulit

La importància de les partícules

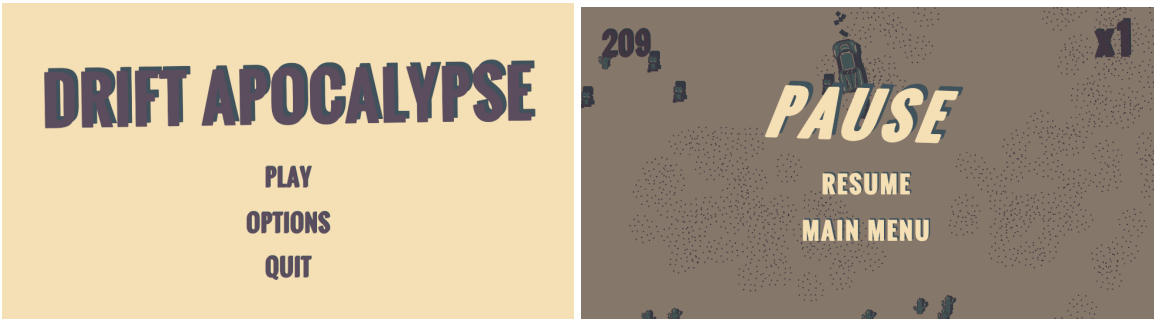
Ja que Unity porta un motor de partícules per generar efectes i és molt senzill d''utilitzar, aquesta resulta una eïna molt bona per fer el joc més vistós. Aquestes partícules les farem servir per indicar al jugador diferents estats en els que es troba:



La primera ens serveix per saber quina tecla està prement el jugador.
La segona ens indicaria la vida, segons el fum que surt del cotxe.
I la tercera deixa unes marques al terra per indicar-nos quan ens trobem derrapant.

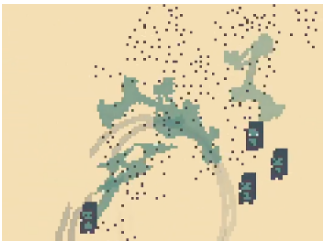
També les farem servir per indicar moments clau en el que el jugador necessita rebre informació de manera immediata, com per exemple, alhora de derrotar un enemic, rebre un cop o modificar un element del escenari com els cactus.

Menús i interfície



Per dissenyar els menus i HUD vaig fer servir el Canvas de Unity amb el sistema de escenes i vaig buscar un estil minimalista. També vaig afegir un parell d'opcions per modificar la càmara per afegir més personalització en cas de que l'usuari pateixi mareig o no es trobi còmode.

Efectes i sò



Per complementar el joc, afegeixo unes taques que duren uns segons
 al acabar amb un enemic, que són direccionals depenent la força i l'angle amb el que realitzem l'acció.

I per acabar de arrodonir-lo, he afegit sò a totes les accions del joc gràcies a la pàgina Freesound.org. Com les eliminacions dels enemics, el boost, al destruir un cactus, accelerar, derrapar, realitzar un combo etc.

Enllaços i documentació

Enllaç al repositori de GitHub:
https://github.com/towernow/DriftOverZombies/tree/STABLE

Enllaç al video penjat al meu canal de YouTube:
https://www.youtube.com/watch?v=7DSwk783CHY

Enllaç a la descàrrega de la APK:
https://www.dropbox.com/s/4tcujylw72vipoa/DriftApocalypse.apk?dl=0
Enllaç a la descàrrega de l'aplicació d'escriptori:
https://www.dropbox.com/sh/p8h66ftznkm4opw/AABSJB-56LctygHFCHBSBVOQa?dl=0