Videojoc de plataformes 2D: When a ninja meets zombies

Pep Pallarés Reverté
Grau d'enginyeria informàtica
Videojocs educatius

Jordi Duch Gavaldà
Helio Tejedor Navarro
Joan Arnedo Moreno

16/06/2017

FITXA DEL TREBALL FINAL

Títol del treball:	*Videojoc de plataformes 2D: When a ninja meets zombies*
Nom del autor:	*Pep Pallarés Reverté*
Nom del consultor/a:	*Jordi Duch Gavaldà, Helio Tejedor Navarro*
Nom del PRA:	*Joan Arnedo Moreno*
Data d'entrega (mm/aaaa):	06/2017
Titulació:	*Grau d'enginyeria informàtica*
Àrea del Treball Final:	*Videojocs*
Idioma del treball:	*Català*
Paraules clau	*Videojoc, plataformes, windows*

Resum del treball (màxim 250 paraules): *Amb la finalitat, context de aplicació, metodologia, resultats i conclusions del treball.*

Actualment ens trobem amb una industria del món del videojoc que sembla que hagi oblidat un dels gèneres que la va fer gran i que gairebé ningú s'atreveix a crear-ne: els plataformes 2D.

Es per això que he decidit crear-ne un jo mateix i, així, poder recuperar aquesta part de la infància i poder evadir per un moment aquesta industria governada per altres gèneres.

Per a fer-ho he utilitzat el motor Unity. Un motor molt complet i senzill d'utilitzar amb el que he creat el joc des de zero, sense utilitzar scripts de tercers, per a així aconseguir un desenvolupament més personal i facilitar-me les possibles modificacions o adhesions del codi. Si que he tingut que recórrer a altres creadors per als *sprites* i els sons.

El videojoc disposa de cinc nivells jugables amb una dificultat creixent, on en l'últim d'aquests el jugador s'ha d'enfrontar amb un *boss* final. A més, el joc disposa de menú de pausa, de menú de final de nivell, i d'un menú d'inici on el jugador pot consultar els crèdits, llegir les instruccions o obrir un selector de nivells en el que els nivells seran accessibles a partir del moment que el jugador hi arribi.

El joc final potser no és dels mes atractius visualment ni presenta una jugabilitat molt complexa però, en general, a les persones que l'hem jugat ens ha suposat un repte, sense arribar a ser excessivament difícil, ens ha divertit i ens ha recordat aquells jocs als que jugàvem de menuts.

Abstract (in English, 250 words or less):

Nowadays, the videogames industry seems has been forgotten one of the genres that made it big and almost nobody dares to create one: the platforms 2D games.

For this reason i decided to create one by my own and, so, can retrieve this part of our childhood and be able to avoid by a moment this reign of high budgets games.

To do that I've used the Unity engine. A complete and easy to use engine with which I've created the game from zero, without using third party scripts, to achieve a more personal development which one makes me easier modify or add things to the code.

The videogame has five levels each one with a higher difficulty than the previous where, on the last level, the Player has to fight with a final boss. Moreover, the videogame has a pause menu, a level-end menu, and a start menu where the Player can checks the credits, the instructions and go to the level selector in which the levels will be available when the Player reach them in the videogame. .

This maybe isn't one of the most atractive videogames and don't has a too complex gameplay but, generally, people who played it found a not too difficult challenge, enjoyed it and remembered the videogames that we played when we was childrens.

Índex

1- Introducció

La indústria del videojoc cada cop creix més, atrau a més públic, genera més diners i, per conseqüència, s'allunya dels seus orígens.

Per una banda, gràcies als telèfons intel·ligents, el jugar a videojocs ha deixat de ser una cosa tabú per a moltíssima gent. Tot i que ho considero un gran avanç, aquesta gent sol demanar jocs casuals, sense una gran dificultat i que no tinguin cap tipus de corba d'aprenentatge. És a dir, són jocs que a la gent que porta anys jugant no solen aportar-los gran cosa. La majoria de jocs de l'empresa King en són un exemple.

Per una altra, hi ha un gran nombre de consumidors que que semblen valorar més característiques com ara si el joc te un gràfics punters, si el fotogrames per segon als que funciona són suficientment elevats o si te multijugador. En definitiva, són persones que no tenen en compte si el joc realment proporciona una experiència gratificant com a jugador. L'exemple més clar dels darrers anys ha sigut el "The order: 1886", joc tècnicament impressionant però amb una jugabilitat que no aconseguia en cap moment atraure al jugador i amb una duració ridícula que no feia altre que aguditzar el sentiment de decepció.

Amb tot això, ens estem allunyant d'aquells jocs que buscaven aportar reptes als jugadors, fer que aquest es divertissin, proporcionar hores de joc sense que el jugador necessites comprar-ne de nous. Jocs que realment generaven ganes de tornar a jugar-los, de superar-los costes el que costes i que, molts del que van tenir la sort de jugar-los, encara busquen ara formes de tornar a jugar-los com els emuladors o demanant *remakes* a les empreses.

Poques són les empreses, i normalment són estudis independents, que s'atreveixen a crear nous jocs que recordin a aquelles joies. Entre aquestes, un dels generes que personalment més m'agradava i més trobo a faltar és el plataformes 2D. Jocs com els Super Mario Bros, Rayman, els diferents Megaman, alguns dels Castlevania amb 2D...

I es per això, per aquesta nostàlgia, per aquestes ganes de recuperar una mica aquest gènere oblidat que he decidit crear un videojoc de plataformes en 2D que en jugar-lo m'aporti una mica de tot allò que molts jugadors trobem a faltar.

1.1- Descripció del joc

El joc, destinat a ordinadors amb el sistema operatiu Windows, és del gènere plataformes 2D, per a un jugador i amb temàtica fantàstica, en el que s'ha de fer avançar un guerrer *ninja* a través de diferents nivells mentre lluita contra hordes de *zombies* amb l'únic objectiu d'arribar al final del nivell.

Aquest, tot i ser un plataformes, també forma part del gènere acció degut a la possibilitat d'eliminar els diferents enemics situats al llarg dels nivells, a més de requerir destresa i temps de reacció ràpids per a poder arribar al final d'aquests.

La jugabilitat base del joc és similar a la dels Castlevania en 2D, on s'ha de fer avançar el personatge pels diferents nivells, els quals tenen una progressió lineal, mentre s'utilitzen els diferents atacs que aquest té per a derrotar els enemics. Tot i això, aquest joc està orientat per a jugadors més casuals, per tant, la dificultat és notablement menor a la dels jocs de la saga esmentada.

1. Imatge d'un dels Castlevania en 2D

El jugador, utilitzant el teclat, pot indicar al personatge que faci qualsevol de les següents accions:

- Moviment a dreta o esquerra
- Salt simple
- Salt doble
- Atacar

D'aquesta manera, el jugador ha de fer-lo avançar per les diferents plataformes que conformen els nivells que s'estenen a dreta, esquerra i per la part superior de la pantalla, esquivar els diferents obstacles inclosos en aquests, i poder d'eliminar els diferents enemics amb que es trobi. L'eliminació dels enemics, tret del enemic final del joc, no és necessària per a finalitzar els nivells.

El joc informa al jugador en tot moment de les vides restants fins que el nivell finalitzi. Si el jugador mor se li restarà una vida i apareixerà en l'últim *checkpoint* pel que hagi passat. Quan el jugador es queda sense vides s'indica un *game over* a partir del qual el jugador te les opcions de repetir el nivell des del principi o tornar al menú d'inici.

1.2 - Conceptualització

El joc transcorre en un món imaginari on un guerrer *ninja* es perd i entra a un cementiri ple de *zombies* que decideixen que menjar-se'l és una bona idea. El *ninja* haurà de creuar tot els cementiri per aconseguir escapar-ne.

L'ambientació del nivells serà obscura, tant pel que fa als *sprites* com a la música, i tota l'acció transcorrerà de nit, per a aconseguir una millor immersió del jugador.

El *ninja* és un hàbil guerrer que utilitzarà les seves habilitats i gran destresa per saltar d'una plataforma a una altra mentre va acabant amb tots els enemics que sigui necessari.

Els *zombies* són cadàvers que han tornat a la vida amb un únic desig, alimentar-se.

2. Concep art inicial creat amb sprites seleccionats per a realitzar el joc

Aquests interactuaran entre si de la següent forma:

- Si el *ninja* toca un *zombie* amb l'espasa, el *zombie* morirà.
- Si el ninja cau a sobre d'un *zombie*, el zombie morirà.
- Si el ninja s'apropa molt al *zombie* sense donar-se cap dels casos anteriors, el *zombie* atacarà i restarà una vida al *ninja.*
- Tant el *ninja* com el *zombie* moriran si cauen per sota del límit del mapa.

1.3 - Desenvolupament

Per a dur a terme un desenvolupament correcte del videojoc he tingut en compte l'elecció del motor sobre el que desenvolupar-lo i he repartit la feina a fer en diversos objectius distribuïts temporalment tenint en compte el temps disponible i les diferents entregues parcials.

1.3.1 - Motor

El motor utilitzat per a desenvolupar el joc és el Unity 3D, un dels més utilitzats i el que es pot considerar l'estàndard avui en dia.

El motor ofereix un ampli i complet apartat per a l'elaboració de jocs en 2D juntament amb una interfície d'usuari força senzilla. Alguns dels punts destacables d'aquest són:

- Facilitat d'importar *sprites* i de tallar-les en cas que aquestes siguin múltiples.
- Facilitat d'animar els *sprites* i de tractar les transicions entre animacions.
- Veure en tot moment els components de l'objecte seleccionat, a més de poder-los editar, eliminar o afegir-ne de nous sense complicacions.
- Poder canviar en tot moment la resolució de la càmera.
- Capacitat d'implementar el joc en moltes plataformes diferents.
- Integració de les variables públiques de les *scripts* adjunts als objectes en la interfície d'usuari.
- Importació immediata dels canvis fets en les *scripts* dintre del Monodevelop al Unity.

1.3.2 - Objectius

El objectius a assolir tenint en compte el compliment d'allò que es demana en les entregues parcials del treball de final de grau són:

Preparar *sprites*: incloure en el projecte els diferents *sprites* que es poden utilitzar en el projecte, generar les diferents animacions dels personatges i crear diferents objectes (*prefabs*) reutilitzables al llarg dels nivells com ara plataformes de diferents longituds.

Implementar mecàniques: generar les scripts que controlen els moviments del personatge i la interacció d'aquet amb la resta d'objectes.

Implementar les transicions entre animacions: especificar quan s'han de realitzar les transicions d'una animació a una altra del conjunt d'animacions que conformen el personatge i els enemics.

Implementar la interfície d'usuari: implementar un menú principal, un menú de fi de nivell i un marcador que mostri les vides del personatge mentre està dintre del nivell.

Generar un nivell complet: generar el primer nivell complet amb enemics, punts de control i un final.

Comprovar i corregir: comprovar que el funcionament del joc i corregir els possibles errors.

Generar més nivells: generar nous nivells consecutius als que el jugador podrà accedir un cop faci arribar el personatge al final del nivell anterior.

Proves finals: comprovar que el joc complet funciona en tot moment correctament i corregir els possibles errors que es trobin.

Durant tots els objectius s'han fet comprovacions i revisions tant de les coses ja implementades com de les noves, però igualment he dedicat 2 objectius centrats solament en buscar aquest errors per tal d'assegurar la qualitat de la feina feta.

Tenint en compte les següents dates d'entrega:

- Disseny – 12 de març de 2017
- Versió parcial – 23 d'abril de 2017
- Versió jugable – 20 de maig de 2017
- Versió final – 16 de juny de 2017

La distribució temporal dels objectius ha sigut la següent:

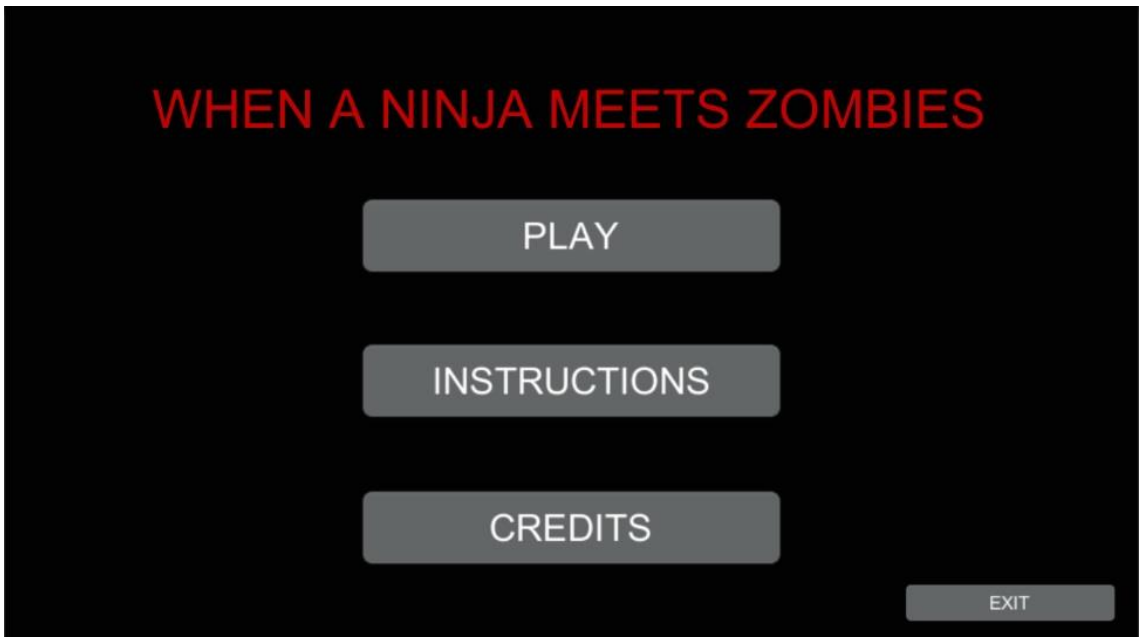
	Març				Abril				Maig				Juny			
Setmana	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Sprites																
Mecàniques																
Transicions anim.																
Interfície d'usuari																
Nivell complet																
Comprovar i corregir																
Més nivells																
Proves finals																

3. Taula amb la planificació temporal dels objectius

2- Disseny

En els següents apartats mostraré i explicaré els diferents que he utilitzat per a desenvolupar el videojoc.

2.1- Menús



4. Menú d'inici

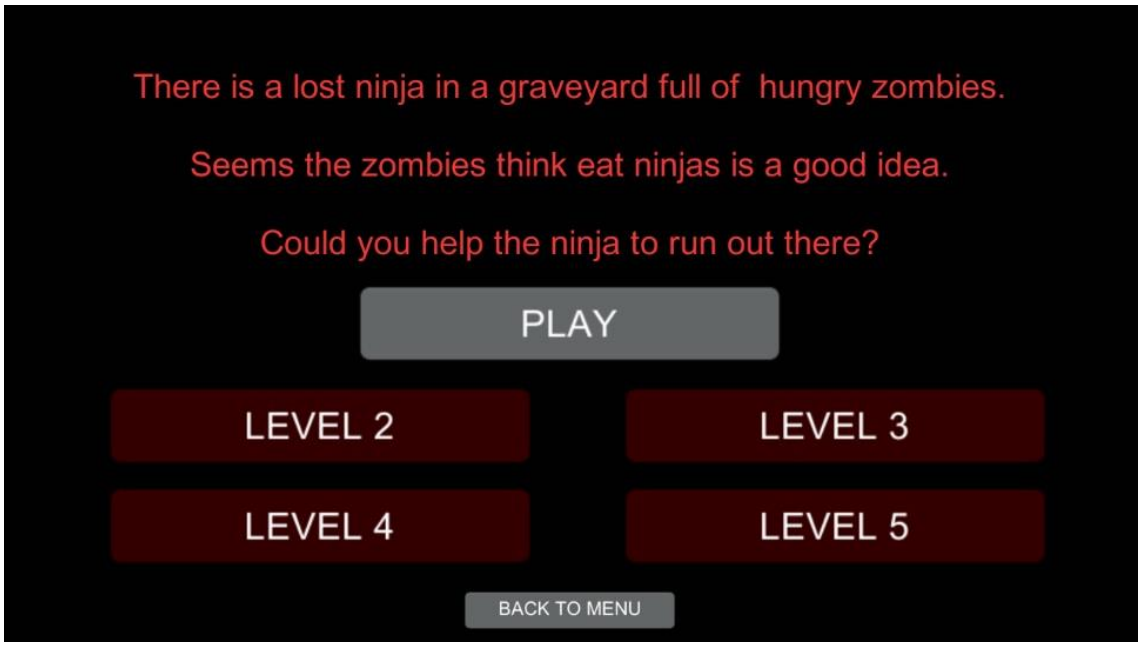
Quan s'inicia el joc el primer que veu el jugador és el menú d'inici dintre del que hi ha 4 botons:

- **PLAY**: porta al menú de seleccionar nivell.
- **INSTRUCTIONS**: mostra les instruccions.
- **CREDITS**: mostra els crèdits.
- **EXIT**: tanca el videojoc.

En el menú de seleccionar nivells es mostren els següents botons:

- **PLAY**: inicia el nivell 1.
- **LEVEL 2**: inicia el nivell 2.
- **LEVEL 3**: inicia el nivell 3.
- **LEVEL 4**: inicia el nivell 4.
- **LEVEL 5**: inicia el nivell 5.
- **BACK TO MENU**: torna al menú d'inici.

Els botons que es mostren en roig es deu a que, fins que el jugador no aconsegueix arribar a aquell nivell, els botons es mantenen inactius.



5. Selector de nivells

En el panel de crèdits es mostra el meu nom com a creador i les atribucions pertinents als creadors de la música i els *sprites* utilitzats. El botó **BACK TO MENU** té el mateix comportament que en el selector de nivells.

6. Crèdits

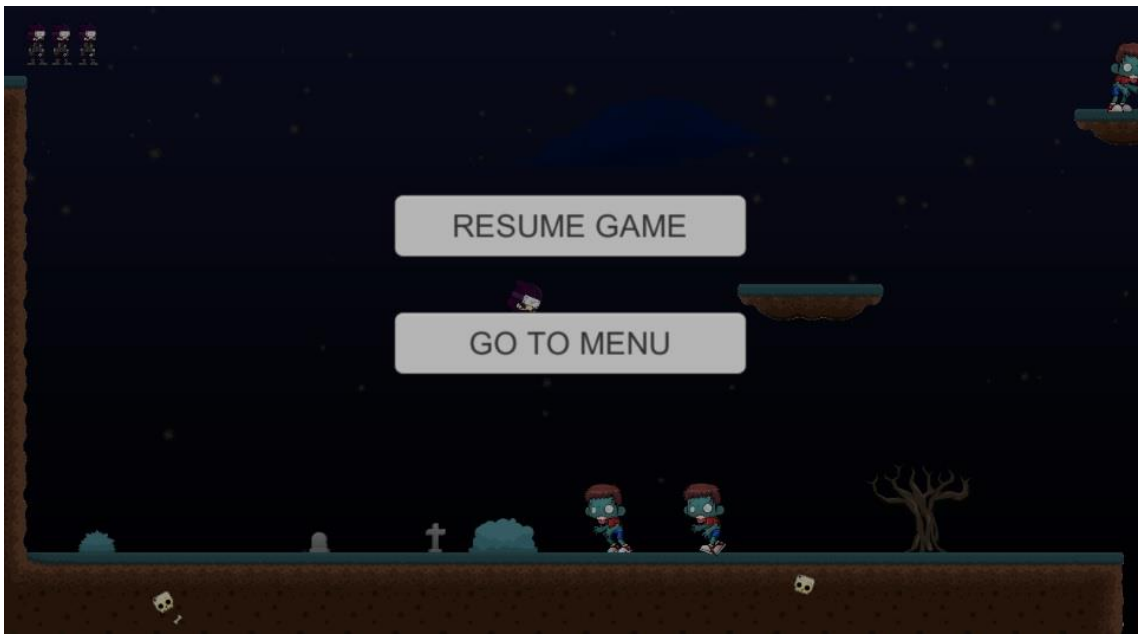
El menú d'instruccions mostra al jugador les tecles que ha d'utilitzar per a saltar i atacar, a més d'indicar com fer el doble salt i d'una altre forma de matar els zombies a més de l'atac. També mostra al jugador les senyals de fi de nivell i de *checkpoint*.

7. Instruccions

Un cop s'inicia el joc ens podem trobar amb els menús que es mostren quan es pausa el joc, quan el jugador es queda sense vides i quan el jugador arriba al final del nivell.

En el menú de pausa trobem els botons:

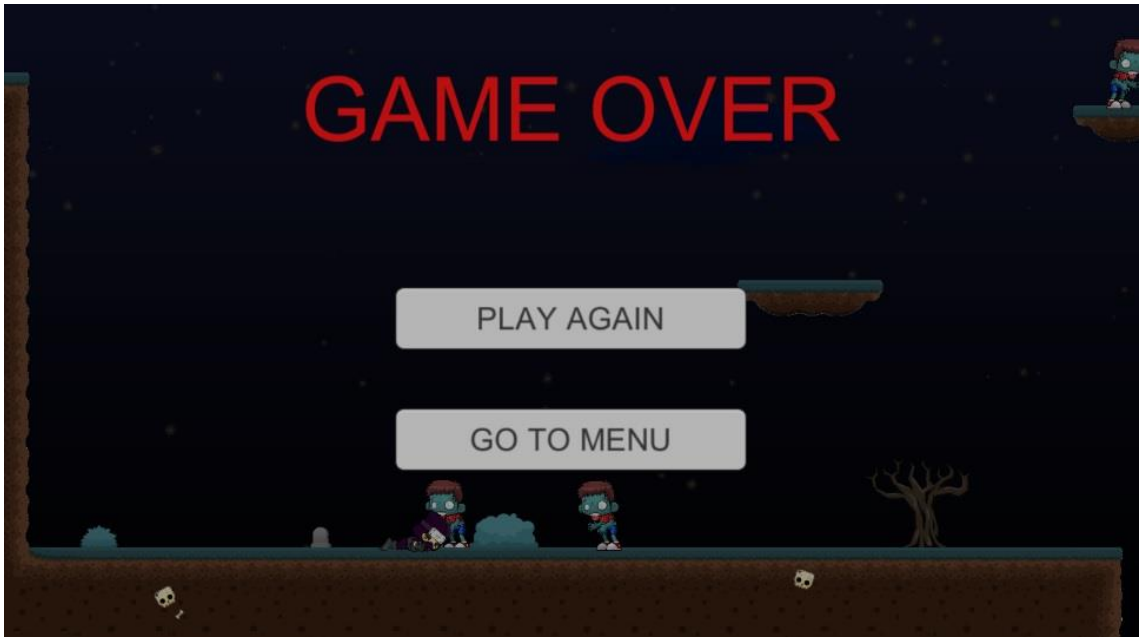
- **RESUME:** continua amb el joc.
- **GO TO MENU:** porta el jugador al menú d'inici.



8. Menú de pausa

Dintre del menú de *game over* que es mostra quan el jugador es queda sense vides trobem els següents botons:

- **PLAY AGAIN:** torna a iniciar el nivell.
- **GO TO MENU:** porta el jugador al menú d'inici.

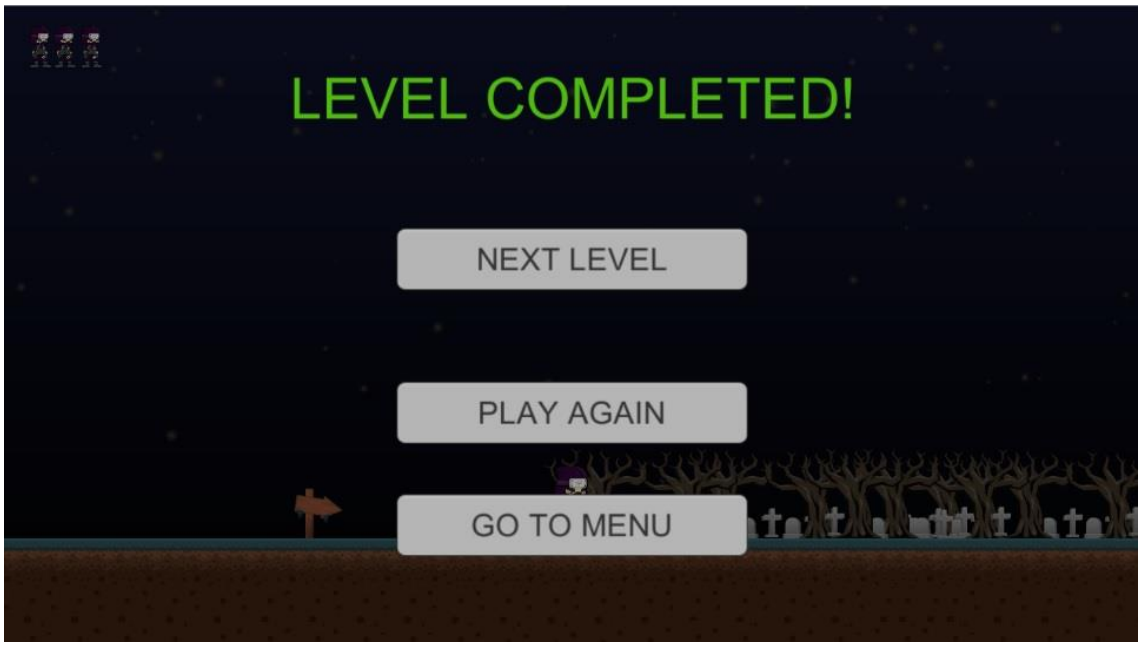


9. Menú de game over

Al menú que es mostra quan el jugador arriba al final del nivell ens trobem els següents botons:

- **NEXT LEVEL:** inicia el següent nivell.
- **PLAY AGAIN:** torna a iniciar el nivell.
- **GO TO MENU:** porta el jugador al menú d'inici.

Aquest menú solament mostra el botó per a tornar al menú d'inici quan el jugador finalitza l'últim nivell.



10. Menú de final de nivell

2.2- *Sprites*

2.2.1- Personatges

El *sprites* utilitzats per a crear les animacions del *ninja* són les següents:

Dret

Córrer

Salt

Atac

Mort

El *sprites* utilitzats per a crear les animacions del *zombie* són les següents:

Atac

Mort

Dret

Caminant

El *sprites* utilitzats per a crear les animacions del *zombie* final són les següents:

Atac

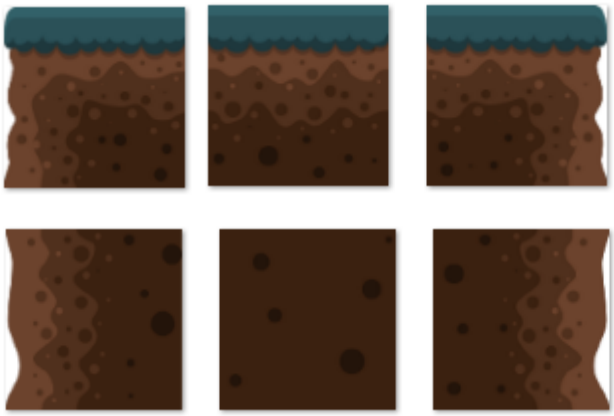
Mort

Dret

Caminant

2.2.2- Mapa

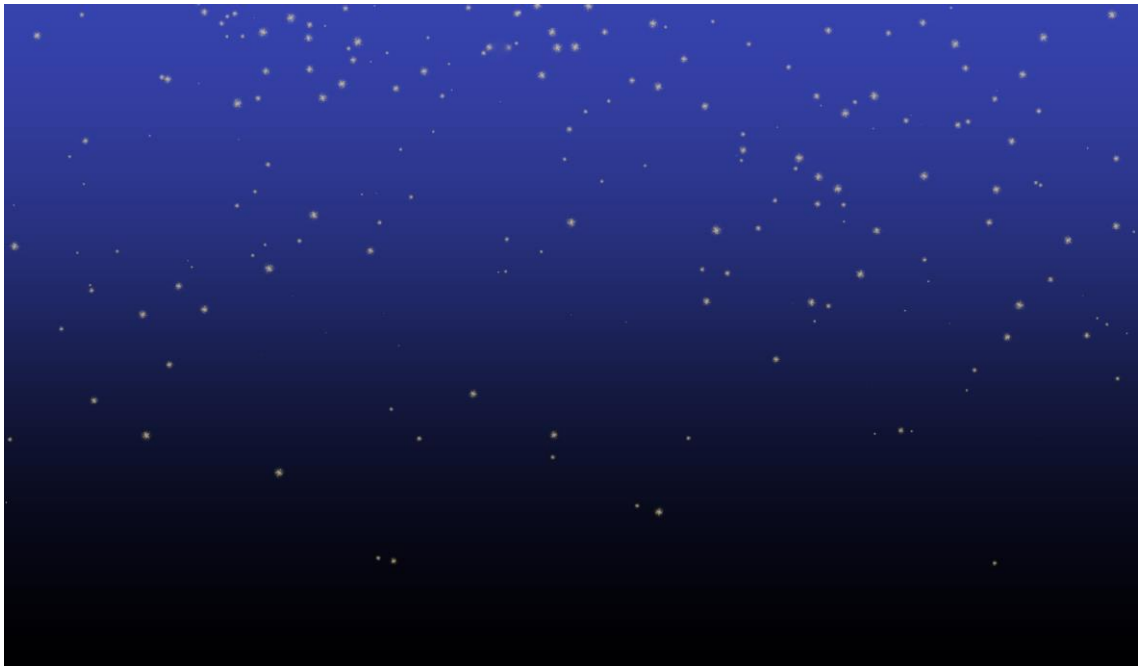
El *sprites* utilitzats per a crear el terreny dels nivells són els següents:

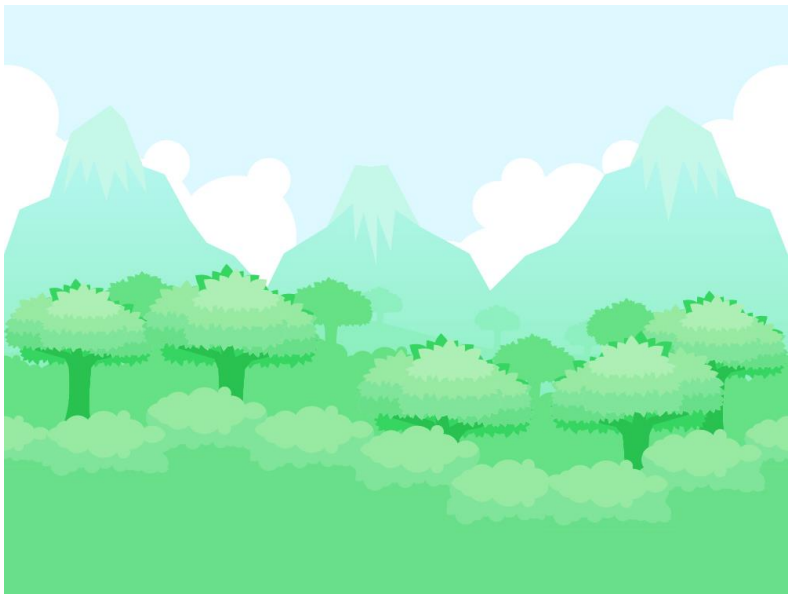


2.2.3- Decoració

El *sprites* utilitzats per a decorar el terreny dels nivells són els següents:

Els *sprites* utilitzats per a simular el fons i el cel són:





2.4- *Scripts*

2.4.1- StartMenuController

Aquesta script implementa el comportament del menú d'inici.

El primer que fa la script en inicialitzar-se és comprovar que els menús estan inactius per a evitar que en iniciar-se l'escena apareguin tots uns a sobre dels altres, i que el ratolí està visible, ja que durant la partida aquest desapareix.

```
18      // Initialization
19      void Start () {
20          menuInstructions.SetActive(false);
21          menuCredits.SetActive(false);
22          menuBeforeStart.SetActive(false);
23          Cursor.visible = true;
24      }
```

Tot seguit hi ha el mètode que tanca el joc, cridat des del menú d'inici quan es prem el botó **EXIT.**

```
26      public void closeGame()
27      {
28          Application.Quit();
29      }
30
```

A continuació hi ha els mètodes que obren i tanquen els menús que contenen les instruccions i els crèdits. Aquests, comproven si els menú al que fan referencia respectivament està actiu; si no ho està, desactiven el menú d'inici i activen el menú en qüestió i, si sí que ho està, el desactiven i tornen a activar el menú d'inici.

```
31      // Opens or close the instructions menu
32      public void MenuIntructions()
33      {
34          if (!menuInstructions.activeSelf)
35          {
36              mainMenu.SetActive(false);
37              menuInstructions.SetActive(true);
38          }
39          else
40          {
41              menuInstructions.SetActive(false);
42              mainMenu.SetActive(true);
43          }
44      }
```

```
46      // Opens or close the credits menu
47      public void MenuCredits()
48      {
49          if (!menuCredits.activeSelf)
50          {
51              mainMenu.SetActive(false);
52              menuCredits.SetActive(true);
53          }
54          else
55          {
56              menuCredits.SetActive(false);
57              mainMenu.SetActive(true);
58          }
59      }
```

El següent mètode te el mateix funcionament que els dos anteriors però amb la diferencia que abans d'activar el menú de selecció de nivell, crida el mètode *checkButons* que comprova a quin nivell ha arribat el jugador, i activa el botons que porten al jugador als nivells als que ja ha arribat.

```
61      // Opens or close the before-start menu
62      public void MenuBeforeStart()
63      {
64          if (!menuBeforeStart.activeSelf)
65          {
66              checkButtons();
67              mainMenu.SetActive(false);
68              menuBeforeStart.SetActive(true);
69          }
70          else
71          {
72              menuBeforeStart.SetActive(false);
73              mainMenu.SetActive(true);
74          }
75      }
```

I el mètode checkButtons accedeix a la *script SaveLoadLevelState,* de la que més endavant explicaré el seu funcionament, per a aconseguir el nombre enter que indica a quin nivell ha arribat el jugador i activar els botó del nivell en qüestió i els dels nivells inferiors.

```
77      // Checks the level reached by the player and activates the buttons of the levels reached
78      private void checkButtons()
79      {
80          SaveLoadLevelState.Load();
81          lvl = SaveLoadLevelState.levelReached;
82
83          switch (lvl)
84          {
85              case 2:
86                  buttonlvl2.interactable = true;
87                  break;
88              case 3:
89                  buttonlvl2.interactable = true;
90                  buttonlvl3.interactable = true;
91                  break;
92              case 4:
93                  buttonlvl2.interactable = true;
94                  buttonlvl3.interactable = true;
95                  buttonlvl4.interactable = true;
96                  break;
97              case 5:
98                  buttonlvl2.interactable = true;
99                  buttonlvl3.interactable = true;
100                 buttonlvl4.interactable = true;
101                 buttonlvl5.interactable = true;
102                 break;
103         }
104
105     }
```

2.4.2- TransitionController

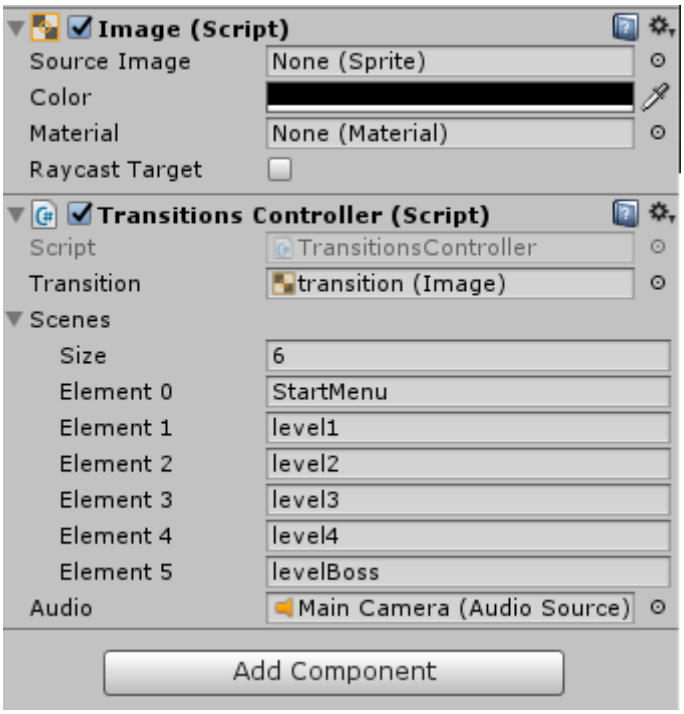
Aquesta *script* controla les transicions entre escenes.

Quan s'inicia utilitza el mètode *CrossFadeAlpha* per a fer a fer transparent de forma gradual un imatge completament negra. Imatge que es farà també de forma gradual opaca quan es crida el mètode públic *FadeOut.*

Quan es crida el mètode *FadeOut* se li passa un enter com a paràmetre que indica a quina escena, dintre del vector d'escenes, volem anar. Aquest crida una corutina per a fer el canvi d'escena que, després d'esperar el segon que tarda la imatge en fer-se completament opaca, atura la música i carrega la següent escena. Escena que en activar-se tornarà a fer transparent la imatge, donant així un efecte de transició suau.

```
14      // Initialization
15      void Start()
16      {
17          transition.CrossFadeAlpha(0, 1f, false);
18      }
19
20      public void FadeOut(int s)
21      {
22          transition.CrossFadeAlpha(1, 1f, false);
23          StartCoroutine(changeScene(scenes[s]));
24      }
25
26      IEnumerator changeScene(string scene)
27      {
28          yield return new WaitForSeconds(1);
29          audio.Stop();
30          SceneManager.LoadScene(scene);
31      }
32
33  }
```

La mida i els valors del vector que conté el nom de les escenes s'assignen des del mateix Unity per a poder modificar-ho d'una forma més senzilla a mesura que es generen noves escenes.



2.4.3- NinjaControllerScript

Aquesta script controla els moviments, animacions i estats del *ninja* que controla el jugador.

En inicialitzar-se, entre les diferents variables que inicialitza, guarda en variables el component *RigidBody2D* per a poder generar els moviments, i el component *Animator* per a poder modificar l'estat d'aquest i mostrar les animacions desitjades en cada moment.

```
68      // Initialization
69      void Start ()
70      {
71          rb = this.GetComponent<Rigidbody2D>();
72          a = this.GetComponent<Animator>();
73          attackTrigger.enabled = false;
74          fallingAttackTrigger.enabled = false;
75          playerIsDead = false;
76          timeToGO = 1f;
77          timeToCP = 2.5f;
78          levelended = false;
79          lifes = 3;
80          playingGO = false;
81          checkpoint.transform.position = transform.position;
82      }
```

A continuació entren en el bloc de la funció FixedUpdate, en el que està implementat el moviment bàsic del ninja ja que aquest no depèn dels fotogrames per segon del jugador.

El primer que hi ha en quest bloc són dos condicionals que aturen l'execució de l'script en aquell punt si el jugador està mort o si aquest ha arribat al final del nivell.

En concret, el condicional que comprova si el jugador està mort conté tres parts:

- Si el jugador està mort i no te vides retorna aturant l'execució en aquell punt.
- Si el jugador està mort però te vides restants i el temps per a reaparèixer en el *checkpoint* és més gran que zero, retorna aturant l'execució en aquest punt fins que el temps d'espera per a reaparèixer és 0 o inferior.
- Si el jugador està mort, té vides i ja s'ha ha acabat el temps d'espera per a reaparèixer resta una vida, indica que el jugador ja no està mort i crida el mètode *Respawn,* explicat més endavant, que el fa reaparèixer en l'últim *checkpoint.*

```
85      void FixedUpdate()
86      {
87
88          // Stops the movement if the player is dead
89          if (playerIsDead && lifes <= 0)
90          {
91              return;
92          }
93          else if (playerIsDead && timeToCP > 0f)
94          {
95              if (timeToCP < 1.7f && timeToCP > 1.5f)
96              {
97                  transition.CrossFadeAlpha(1f, 1f, false);
98              }
99              timeToCP = timeToCP - Time.deltaTime;
100             return;
101         }
102         else if (playerIsDead)
103         {
104             lifes = lifes - 1;
105             lifesCounter[lifes].SetActive(false);
106             playerIsDead = false;
107             a.SetBool("dead", playerIsDead);
108             Respawn();
109         }
110
111         // Stops the movement if the player reach the end of the level
112         if (levelended)
113         {
114             a.SetBool("ended", levelended);
115             return;
116         }
```

A continuació l'script comprova si el jugador està a terra o en l'aire, i si esta enganxat a una paret.

Tot seguit envia al *Animator* el valor booleà que indica si el jugador està al terra i la velocitat vertical. A partir d'aquest valors, juntament amb la velocitat horitzontal calculada a partir de l'input horitzontal, l'*Animator* calcularà l'estat en que es troba el *ninja* i mostrarà l'animació pertinent.

Tenint en compte aquest input horitzontal, donat per les fletxes del teclat o les lletres 'A' i 'D', i la velocitat màxima que he indicat, se li enviarà al *RigiBody2D* un nou vector que indicarà cap on s'ha de moure el *ninja.*

I per a finalitzar aquest bloc, hi ha un condicional que a partir de la direcció que s'indica cap a la que es mou el *ninja* i la direcció cap on mira tenint en compte l'animació, gira aquesta animació per a que mostri al ninja mirant o movent-se cap a la direcció que ho esta fent.

```
118         // Checks if the character is in the ground
119         grounded = Physics2D.OverlapCircle(groundCheck.position, groundRadius, whatIsGround);
120
121         // Checks if the character is in a wall
122         onWallUp = Physics2D.OverlapCircle(wallCheckUp.position, wallRadius, whatIsWall);
123         onWallDown = Physics2D.OverlapCircle(wallCheckDown.position, wallRadius, whatIsWall);
124         onWall = (onWallDown || onWallUp);
125
126         // Sends the boolean grounded and the vertical speed to the animator
127         a.SetBool("Ground", grounded);
128         a.SetFloat("vSpeed", rb.velocity.y);
129
130         // Gets the direction input
131         move = Input.GetAxis("Horizontal");
132
133         // Sends the input to the animator
134         a.SetFloat("Speed", Mathf.Abs(move));
135
136         // Move the character in the scene
137         rb.velocity = new Vector2(move * maxSpeed, rb.velocity.y);
138
139         // Chooses the direction to move
140         if (move > 0 && !facingRight)
141         {
142             Flip();
143         }
144         else if (move < 0 && facingRight)
145         {
146             Flip();
147         }
```

El següent bloc de la *script* és el que pertany a la funció *Update.* La que és crida un cop per cada fotograma.

S'inicia comprovant si el jugador està per sota del nivell i viu, en cas afirmatiu, indica que està mort i reprodueix el soroll de mort.

```
153     void Update()
154     {
155         // Check the conditions that kills the player
156         if (rb.transform.position.y < -2f && !playerIsDead)
157         {
158             playerIsDead = true;
159             deadSound.Play();
160         }
161
```

A continuació hi ha un bloc condicional en el que si el jugador està mort però te vides retorna per a aturar l'execució de la *script* i evitar que es mogui. En cas que no tingui vides, inicia un compte enrere per a que el jugador tingui veure l'animació de mort del *ninja*. Quan aquest arriba a 0 canvia la música i obre el menú de fi de joc o de *game over*.

```
162         // Stops the movement if the player is dead
163         if (playerIsDead && lifes <=0)
164         {
165             if (timeToGO < 0f)
166             {
167                 music.Stop();
168                 if (!playingGO)
169                 {
170                     gameOverSound.Play();
171                     playingGO = true;
172                 }
173                 menu.gameOver();
174             }
175             else
176             {
177                 timeToGO = timeToGO - Time.deltaTime;
178             }
179             return;
180         } else if (playerIsDead)
181         {
182             return;
183         }
```

Després hi ha un altre condicional que atura el joc quan el jugador ha arribat al final del nivell i avisa al *Animator* de que ho ha fet .

```
185         // Stops the movement if the player reach the end of the level
186         if (levelended)
187         {
188             a.SetBool("ended", levelended);
189             return;
190         }
```

Tot seguit hi ha la part de codi que controla els salts del *ninja*. El primer que fa és, mitjançant el condicional, posar en falç la variable *pressedSpace* si el *ninja* esta a terra. He utilitzat aquesta variable per controlar si el *ninja* estava en l'aire per haver fet un salt o per qualsevol altre motiu.

```
192         // Changes the boolean value to control the jump when the char is falling without jumping
193         if (grounded)
194         {
195             pressedSpace = false;
196         }
```

Quan el jugador prem la tecla espai per indicar que vol que el *ninja* salti hi ha 4 possibles tipus de salt:

- Salt des de terra
- Salt doble després de que el jugador faci saltar el *ninja*
- Salt quan el ninja no està enlaire però sense haver saltat.
- Salt des d'una paret.

El primer és el salt simple, si el *ninja* es troba a terra i el jugador prem la tecla espai indica al *Animator* que el *ninja* ja no està a terra, afegeix una força vertical positiva al *RigidBody2D*, inicia el soroll que fa el *ninja* al saltar, inicia un

temps d'espera per a que es pugui realitzar el salt doble i indica que es pot realitzar el doble salt i que s'ha premut la tecla espai.

```
198         // Jump if the char is grounded or perform a doble jump if the char is jumping
199         if (grounded && Input.GetKeyDown(KeyCode.Space))
200         {
201             a.SetBool("Ground", grounded);
202             doublejump = true;
203             rb.AddForce(new Vector2(0, jumpForce));
204             jumpSound.Play();
205             doubleJumpTime = cooldownDoubleJump;
206             pressedSpace = true;
207         }
```

Quan el *ninja* està enlaire hi ha dos possibilitats que es realitzaran en prémer la tecla espai. La primera es el doble salt, que es dura a terme quan si *ninja* ha realitzat un salt normal i el temps d'espera per a realitzar aquest doble salt és 0 o inferior i ell seu comportament és el mateix que en el salt normal però impossibilita fer un altre salt doble i la força d'aquest és inferior. La segona opció es dona quan el ninja esta enlaire sense haver saltat, normalment al baixar d'una plataforma o haver rebotat a sobre d'un zombie. En aquest cas solament es pot fer l'equivalent al doble salt un cop.

```
209         doubleJumpTime = doubleJumpTime - Time.deltaTime;
210
211         if (doublejump && Input.GetKeyDown(KeyCode.Space) && !onWall && doubleJumpTime <= 0)
212         {
213             a.SetBool("Ground", grounded);
214             rb.AddForce(new Vector2(0, (jumpForce)*0.6f));
215             jumpSound.Play();
216             doublejump = false;
217             pressedSpace = true;
218         }
219         else if (!doublejump && !pressedSpace && !grounded && Input.GetKeyDown(KeyCode.Space) && !onWall)
220         {
221             rb.AddForce(new Vector2(0, (jumpForce) * 0.6f));
222             jumpSound.Play();
223             pressedSpace = true;
224         }
```

La última opció que es pot donar és el salt des d'una paret. En aquest cas a la força vertical se li afeixes una en direcció contraria a la paret per a que el *ninja* es separi i pugui enlairar-se.

```
226         //Jumps if the char is on a wall
227         if (onWall && Input.GetKeyDown(KeyCode.Space) && facingRight && !grounded)
228         {
229             rb.AddForce(new Vector2((-jumpForce*3), (jumpForce)));
230             jumpSound.Play();
231         }
232         else if (onWall && Input.GetKeyDown(KeyCode.Space) && !facingRight && !grounded)
233         {
234             rb.AddForce(new Vector2((jumpForce*3), (jumpForce)));
235             jumpSound.Play();
236         }
237
```

El següent que controla la *script* és l'atac bàsic amb l'espasa, que es realitza quan el jugador prem al tecla 'K'. En fer-ho, s'activa l'animació d'atac i el

disparador que indica al *zombies* que si entren en aquella zona han de morir. A més s'inicia un compte enrere controlat pel següent condicional que evita que el jugador comenci un nou atac abans que s'acabi l'animació del primer i desactiva el disparador. Finalment s'informa al *Animator* de que ha d'activar l'animació d'atac.

```
239         // Controls the  activation of the attack trigger
240         if ( !attacking && Input.GetKeyDown(KeyCode.K))
241         {
242             attacking = true;
243             attackTime = cooldownAttack;
244             swordSound.Play();
245             attackTrigger.enabled = true;
246         }
247 
248         // Controls the  desactivation of the attack trigger
249         if (attacking)
250         {
251             if(attackTime > 0)
252             {
253                 attackTime = attackTime - Time.deltaTime;
254             }
255             else
256             {
257                 attacking = false;
258                 attackTrigger.enabled = false;
259             }
260         }
261 
262         // Changes the bool that controls the attack animation in the animator
263         a.SetBool("attack", attacking);
264
```

Tot seguit hi ha un condicional que activa el disparador que indica que el *ninja* està realitzant un atac saltant a sobre dels enemics quan la velocitat vertical es negativa.

```
265         // Controls the activation of the falling attack trigger
266         if (rb.velocity.y < 0)
267         {
268             fallingAttackTrigger.enabled = true;
269         }
270         else
271         {
272             fallingAttackTrigger.enabled = false;
273         }
274
```

I per acabar amb el bloc *Update* hi ha un condicional que crida al metode per activar i desactivar el menu de pausa quan es prem la tecla 'ESC'.

```
275         // Pauses or resumes the game
276         if (Input.GetKeyDown(KeyCode.Escape))
277         {
278             menu.pauseResumeGame();
279         }
```

Un cop tancat el bloc de la funció *Update*, ens trobem amb un mètode que controla quan el *ninja* entra en un altre *Collider2D*. I mitjançant un *switch-case* diferencia si és el final del nivell o és un *checkpoint*.

En el cas que sigui el final de nivell canviarà la música, obrirà el menu de fi de nivell, modificarà el valor de la variable *levelended* per indicar-se a ella mateixa que el jugador ha arribat al final del nivell i poder aturar l'execució dels altres blocs i crida el mètode *checkSave* que tot seguit explicaré.

En el cas que sigui un *checkpoint* es guarda en una variable la posició d'aquest i elimina el disparador que pertany al *checkpoint* per evitar que si el jugador torna enrere guardi la posició d'un *checkpoint* que ja no és vàlid.

```
284     // Controls when the player reach the end of the level or a checkpoint
285     private void OnTriggerEnter2D(Collider2D other)
286     {
287         switch (other.tag)
288         {
289             case "endLevel":
290                 music.Stop();
291                 endlevel.Play();
292                 menu.nextLevel();
293                 levelended = true;
294                 rb.velocity.Set(0f, rb.velocity.y);
295                 checkSave();
296                 break;
297             case "checkpoint":
298                 checkpoint.transform.position = other.transform.position;
299                 Destroy(other);
300                 break;
301         }
302     }
```

El mètode *checkSave* comprova si la variable que guarda el màxim nivell al que ha arribat el jugador de la script *SaveLoadLevelState* és inferior al següent nivell del jugador, si ho és, en modifica el valor i el guarda.

```
362     // Checks if this level has been reached and save if it is not reached;
363     private void checkSave()
364     {
365         if (nextlvl > SaveLoadLevelState.levelReached)
366         {
367             SaveLoadLevelState.Save(nextlvl);
368         }
369     }
370 }
371
```

El següent mètode que conté la *script* és un mètode públic que utilitzaran els *zombies* per indicar a la script que el *ninja* ha mort. Aquest indica que el jugador és mort, activa l'animació i el soroll de mort i desactiva la capsula que simula el cos del ninja.

```
305     // Public method that zombies calls to kill the player
306     public void KillPlayer()
307     {
308         playerIsDead = true;
309         a.SetBool("dead", playerIsDead);
310         deadSound.Play();
311         capsule.enabled = false;
312     }
```

A continuació hi ah el mètode que crida la *script* per fer reaparèixer el jugador en l'últim *checkpoint.* Aquest mètode envia el jugador a la posició del *checkpoint*, reinicia el comptador per a fer la reaparició per a la pròxima vegada que el *ninja* mori, fa transparent la imatge de transició i habilita la capsula.

```
314     // Puts the player on the respawn position
315     void Respawn()
316     {
317         transform.position = checkpoint.transform.position;
318         timeToCP = 2.5f;
319         transition.CrossFadeAlpha(0, 0.5f, false);
320         capsule.enabled = true;
321     }
```

Tot seguit hi ha el mètode que crida el bloc *FixedUpdate* per a modificar el sentit de les animacions tenint en compte cap on és mou el jugador. Ho fa multiplicant per -1 l'escala de l'eix X del *ninja*.

```
323     // Changes the direction of the character
324     void Flip()
325     {
326         facingRight = !facingRight;
327         Vector3 theScale = transform.localScale;
328         theScale.x *= -1;
329         transform.localScale = theScale;
330     }
```

Els següents mètodes afegeixen una força vertical positiva al jugador quan cau a sobre d'un *zombie.* En el cas del *final boss,* la mida del qual és bastant més gran que la dels altres enemics, aquesta força és el doble, per a que el ninja es separi suficient del cos d'aquest.

```
332     // Adds a force to gets rebound effect and restores the capacity to perform the second jump
333     public void Rebound()
334     {
335         rb.AddForce(new Vector2(0, 2000f));
336         pressedSpace = false;
337     }
338
339     // Adds a force to gets rebound effect after hit the boss and restores the capacity to perform the second jump
340     public void BossRebound()
341     {
342         rb.AddForce(new Vector2(0f, 4000f));
343         pressedSpace = false;
344     }
```

I l'últim mètode que hi ha en aquesta script serveix per a que les altres scripts consultin si el jugador està viu o no.

```
348         // Checks if player is dead
349         public bool isPlayerDead()
350         {
351             return playerIsDead;
352         }
```

2.4.4- CameraController

Aquesta script controla el moviment de la càmera per a que segueixi el jugador mitjançant un *LateUpdate* per a assegurar que el ninja ja ha realitzat totes les accions incloses en el seu *Update*. Aquesta, calcula la diferencia que hi ha entre la posició de la càmera i del *ninja* per a mantenir-la al llarg del nivell però amb la particularitat que la càmera no es mourà horitzontalment fins que el *ninja* no és mogui certa distancia cap a la dreta del nivell, no es mourà verticalment fins que el *ninja* no arribi a certa altura, i que aquesta no baixarà més a avall de l'altura inicial.

```
11  // Initialization
12  void Start () {
13      offset = transform.position - player.transform.position;
14  }
15
16
17  void LateUpdate () {
18      // Controls the camera when the player is in the firs 10 units long of the level
19      if (player.transform.position.x >= 10f)
20      {
21          // Controls the camera when the player is below 2 units height without let the camera goes down
22          if (player.transform.position.y <= 2f)
23          {
24              transform.position = new Vector3(player.transform.position.x, 2f + offset.y, player.transform.position.z + offset.z);
25          }
26          // Controls the camera when the player reach the 9 units height following him
27          else if (player.transform.position.y >= 9f)
28          {
29              transform.position = new Vector3(player.transform.position.x, player.transform.position.y - 2f, player.transform.position.z + offset.z);
30          }
31          // Controls the camera moving it to right and left
32          else
33          {
34              transform.position = new Vector3(player.transform.position.x, 2f + offset.y, player.transform.position.z + offset.z);
35          }
36      }
37      // Controls the camera when the player is below the limit of 10 units height
38      else
39      {
40          // Controls the camera when the player is below 2 units height without let the camera goes down
41          if (player.transform.position.y <= 2f)
42          {
43              transform.position = new Vector3(0f + offset.x, 2f + offset.y, player.transform.position.z + offset.z);
44          }
45          // Controls the camera when the player reach the 9 units height following him
46          else if (player.transform.position.y >= 9f)
47          {
48              transform.position = new Vector3(0f + offset.x, player.transform.position.y - 2f, player.transform.position.z + offset.z);
49          }
50          // Controls the camera moving it to right and left
51          else
52          {
53              transform.position = new Vector3(0f + offset.x, 2f + offset.y, player.transform.position.z + offset.z);
54          }
55      }
56  }
```

2.4.5- MenusController

Aquest script controla els menús als que el jugador pot accedir mentre està dintre dels nivells però igualment té un comportament molt similar a la script anteriorment descrita *StartMenuController.*

En inicialitzar-se desactiva tots els menús i, a diferencia de la script *StartMenuController,* en aquest cas desactiva el cursor.

```
12      // Initialization
13      void Start () {
14          menuGameOver.SetActive(false);
15          menuPause.SetActive(false);
16          menuNextLevel.SetActive(false);
17          Cursor.visible = false;
18      }
19
```

A continuació hi ha els mètodes per a activar el menu de *game over* i de fi de nivell, on en tots dos s'activa el cursor per a que el jugador hi pugui interactuar. No presenten opció per a desactivar-se ja que quan s'activen és perquè el jugador o ha de canviar d'escena o ha de reiniciar l'actual.

```
20      // Actives the game over menu
21      public void gameOver ()
22      {
23          menuGameOver.SetActive(true);
24          Cursor.visible = true;
25      }
26
27      // Actives the next level menu
28      public void nextLevel()
29      {
30          menuNextLevel.SetActive(true);
31          Cursor.visible = true;
32      }
```

I finalment hi ha el mètode que activa i desactiva el menú de pausa. En activar-se també activa el cursor i atura el temps del joc i, en desactivar-se, desactiva el cursor i torna a la normalitat el temps.

```
34      // Pauses or resumes the game
35      public void pauseResumeGame()
36      {
37          if (!menuPause.activeSelf)
38          {
39              menuPause.SetActive(true);
40              Time.timeScale = 0;
41              Cursor.visible = true;
42          }
43          else
44          {
45              menuPause.SetActive(false);
46              Time.timeScale = 1;
47              Cursor.visible = false;
48          }
49      }
```

2.4.6- ZombieController

Aquesta *script* controla el comportament dels *zombies.* Quan aquesta s'inicialitza guarda en variables el *RigidBody2D* i el *Animator* del zombie, i indica que el *zombie* està viu.

```
27      // Initialization
28      void Start() {
29          rbZombie = GetComponent<Rigidbody2D>();
30          aZombie = GetComponent<Animator>();
31          zombieIsDead = false;
32      }
```

Tot seguit, inicia el bloc de la funció *FixedUpdate*, al igual que la script *NinjaControllerScript*. I aquest ho fa aturant el moviment del *zombie* si aquest està mort o si el jugador està mort, aturant l'execució de la script en aquell punt. En el cas que és el jugador qui està mort s'informa al *Animator* que el zombie ja no es mou.

```
34      void FixedUpdate()
35      {
36          // Stops the movement if the zombie is dead
37          if (zombieIsDead)
38          {
39              return;
40          }
41          if (player.GetComponent<NinjaControlerScript>().playerIsDead)
42          {
43              aZombie.SetFloat("Speed", 0f);
44              return;
45          }
```

Tot seguit, calcula la diferencia de distancia que hi ha entre el *zombie* i el *ninja*. Mitjançant condicionals, quan el *ninja* es troba a menys de 10 unitats horitzontals i 10 unitats verticals, el *zombie* es mourà cap a ell.

```
47          // Checks the distance within the player and moves to him if he's close
48          offset = player.transform.position - transform.position;
49
50          if (offset.x < 10f && offset.x > 1f && offset.y < 10f && offset.y > -10f)
51          {
52              move = 3f;
53          }
54          else if (offset.x > -10f && offset.x < -1f && offset.y < 10f && offset.y > -10f)
55          {
56              move = -3f;
57          }
58          else
59          {
60              move = 0;
61          }
```

Si el aquest condicional dona valor a la variable *move* però el *zombie* no avança se li afegeix una força vertical positiva al seu *RigidBody2D* per a que salti el petit obstacle que hi pugui haver.

```
63          speed = rbZombie.velocity.x;
64          if (speed == 0 && move != 0)
65          {
66              rbZombie.AddForce(new Vector2(0, 18000f));
67          }
```

I mou el *zombie* adjudicant el nou vector amb la variable *move* a la seva veloctiat. També informa al *Animator* del moviment o no del d'aquest i, al igual que en la script que controla el *ninja,* es modifica la direcció de les animacions per consonar amb la direcció de moviment.

```
69          // Move the zombie in the scene
70          rbZombie.velocity = new Vector2(move, rbZombie.velocity.y);
71
72
73
74          // Sends the movement to the animator
75          aZombie.SetFloat("Speed", Mathf.Abs(move));
76
77
78          // Chooses the direction to move
79          if (move > 0 && !facingRight)
80          {
81              Flip();
82          }
83          else if (move < 0 && facingRight)
84          {
85              Flip();
86          }
87      }
```

A continuació, trobem el bloc de la funció *Update*, que en aquest cas només controla la mort del zombie quan aquest cau per sota del nivell. Cas en el que indica que el *zombie* està mort i inicia el soroll de mort d'aquest.

```
92      void Update () {
93
94          // Stops the movement if the zombie is dead
95          if (zombieIsDead)
96          {
97              return;
98          }
99
100         // Check the conditions that kills the zombie
101         if (rbZombie.transform.position.y < -2f && !zombieIsDead)
102         {
103             zombieIsDead = true;
104             dead.Play();
105         }
106
107
108     }
```

Després hi ha el mètode que controla quan el *zombie* entra a un disparador i controla de quin tipus de disparador es tracta amb un *switch-case*. Hi ha tres opcions:

- Que sigui el *ninja*. En aquest cas mostrarà l'animació d'atac, iniciarà el soroll d'atac i cridarà el mètode killPlayer per indicar a la script del *ninja* que el jugador ha de morir.
- Que sigui l'espasa del *ninja.* Cas en que, si el *zombie* està viu, mostrarà l'animació de mort, iniciarà el soroll de mort i destruirà la *BoxCollirder2D* que fa de cos del zombie.
- Que el *ninja* li caigui a sobre. Aquí es donarà la mateixa situació que amb l'espasa però a més cridarà el mètode que fa rebotar al *ninja* i la destrucció de la *BoxCollider2D* es tindrà amb un retràs de 2 segons.

```
109
110     // Controls if the zombie has to do the attack animation or the dead animation
111     private void OnTriggerEnter2D(Collider2D other)
112     {
113         switch (other.tag)
114         {
115             case "Player":
116                 if (!zombieIsDead)
117                 {
118                     aZombie.SetBool("Attack", true);
119                     attack.Play();
120                     if (!(player.GetComponent<NinjaControlerScript>().isPlayerDead()) && !zombieIsDead)
121                     {
122                         player.GetComponent<NinjaControlerScript>().KillPlayer();
123                     }
124                 }
125                 break;
126             case "Sword":
127                 if (!zombieIsDead)
128                 {
129                     dead.Play();
130                     zombieIsDead = true;
131                     aZombie.SetBool("Dead", true);
132                     Destroy(this.GetComponent<BoxCollider2D>());
133                 }
134                 break;
135             case "fallingAttack":
136                 if (!zombieIsDead && !(player.GetComponent<NinjaControlerScript>().isPlayerDead()))
137                 {
138                     dead.Play();
139                     player.GetComponent<NinjaControlerScript>().Rebound();
140                     zombieIsDead = true;
141                     aZombie.SetBool("Dead", true);
142                     Invoke("killBox", 0.2f);
143                 }
144                 break;
145         }
```

A continuació hi ha un mètode que quan un disparador deixa d'estar en contacte amb el *zombie* informarà al *Animator* que ha de deixar de mostrar l'animació d'atac.

```
149     // Reset the attack boolean when the player goes away
150     private void OnTriggerExit2D(Collider2D collision)
151     {
152         aZombie.SetBool("Attack", false);
153     }
```

El mètode creat per a fer la crida que destrueix la *BoxCollider2D* amb retràs.

```
154
155     // Destroys the BoxCollider2D
156     void killBox()
157     {
158         Destroy(this.GetComponent<BoxCollider2D>());
159     }
```

I finalment el mètode, que al igual a la script *NinjaControllerScript* modifica la escala per a girar les animacions.

```
161     // Changes the direction of the character
162     void Flip()
163     {
164         facingRight = !facingRight;
165         Vector3 theScale = transform.localScale;
166         theScale.x *= -1;
167         transform.localScale = theScale;
168     }
169
```

2.4.7- CheckPointAnimatorController

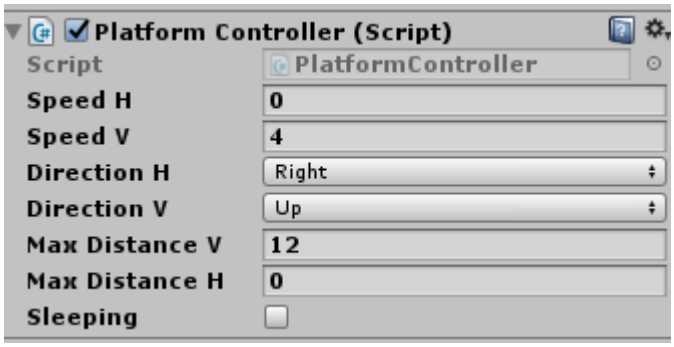
Aquesta script controla l'activació de l'animació que ha de fer la senyal de *checkpoint* quan el *ninja* hi passi.

Quan s'inicia guarda en una variable l'*Animator* i quan un disparador entra activa el soroll i indica que s'ha d'iniciar l'animació.

```
10      // Initialization
11      void Start()
12      {
13          anim = this.GetComponent<Animator>();
14      }
15
16      private void OnTriggerEnter2D(Collider2D collision)
17      {
18          audioCP.Play();
19          anim.SetBool("reached", true);
20      }
```

2.4.8- PlatformController

Aquesta script proporciona unes variables dintre d'Unity que permeten programar el moviment de les plataformes de forma senzilla i sense crear animacions. Variables que serveixen per a indicar la velocitat i la màxima distancia a la que volem que es moguin horitzontalment i verticalment, la direcció d'inici i si aquestes inicien el moviment al iniciar el nivell o quan el jugador es posa a sobre.



En aquest cas he creat dos enumeracions per a indicar la direcció.

```
5   // Enum types to choose the direction
6   public enum Horitzontal { Left, Right }
7   public enum Vertical { Up, Down }
```

Quan s'inicia guarda la posició inicial de la plataforma per a saber on ha de tornar després de recórrer la distancia màxima i inicia les variables que guarden la distancia recorreguda.

```
26  void Start () {
27
28      // Initializes all the variables to control the positions of the platform
29      pfTransform = transform;
30      returnHPosition = pfTransform.position.x;
31      returnVPosition = pfTransform.position.y;
32      movedH = 0.0F;
33      movedV = 0.0F;
34  }
35
```

Dintre ja del bloc de la funció *Update*, si hem indicat que està en estat *sleeping*, s'aturà l'execució. En cas que no hi estigui, calcula la distancia que s'ha mogut la plataforma.

```
37  void Update () {
38
39      if (sleeping)
40      {
41          return;
42      }
43      // Calculates the distances moved horitzontally and vertically
44      movedH = Mathf.Abs(pfTransform.position.x - returnHPosition);
45      movedV = Mathf.Abs(pfTransform.position.y - returnVPosition);
46
```

El següent fragment de la script modifica la direcció de moviment horitzontal si la distancia que s'ha mogut la plataforma és igual o superior a la distancia màxima que se li ha indicat.

```
47      // Changes the horitzontal direction of the platform if it reached the maxium distancie allowed to move
48      if (movedH >= maxDistanceH)
49      {
50          if (directionH == Horitzontal.Left)
51          {
52              directionH = Horitzontal.Right;
53          }
54          else
55          {
56              directionH = Horitzontal.Left;
57          }
58
59          returnHPosition = pfTransform.position.x;
60      }
```

I el mateix amb la direcció vertical.

```
        // Changes the vertical direction of the platform if it reached the maxium distancie allowed to move
        if (movedV >= maxDistanceV)
        {
            if (directionV == Vertical.Up)
            {
                directionV = Vertical.Down;
            }
            else
            {
                directionV = Vertical.Up;
            }

            returnVPosition = pfTransform.position.y;
        }
```

A continuació, modifica el valor de la velocitat en funció de la direcció cap a la que s'ha de moure la plataforma. I, per a tancar el bloc *Update*, genera el nou vector per a donar moviment a la plataforma.

```
            // Changes the horitzontal speed value to get the right direction
            if (directionH == Horitzontal.Left)
            {
                speedH = -Mathf.Abs(speedH);
            }
            else
            {
                speedH = Mathf.Abs(speedH);
            }

            // Changes the horitzontal speed value to get the right direction
            if (directionV == Vertical.Down)
            {
                speedV = -Mathf.Abs(speedV);
            }
            else
            {
                speedV = Mathf.Abs(speedV);
            }

            // Moves the platform
            pfTransform.Translate(new Vector3(speedH, speedV, 0) * Time.deltaTime);
        }
```

Finalment, ens trobem amb un mètode que en el moment que el jugador s'està a sobre de la plataforma, aquesta el fa fill seu per a que el vector anterior s'apliqui també a el *ninja* i es mogui conjuntament amb la plataforma. I un altre mètode per a que deixi de ser-ho.

```
    // Starts the platform movement when the player is on it and allows the platform and the player to move together
    void OnTriggerStay2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            other.transform.parent = transform;
            sleeping = false;
        }

    }

    // Releases the player from the platform movement
    void OnTriggerExit2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            other.transform.parent = null;
        }
```

2.4.9- DoorsController

Aquesta script controla les animacions d'obrir i tancar les portes de la sala del *final boss*. Quan s'inicia guarda l'*Animator* de les portes en una variable.

```
10
11      // Initialization
12      void Start()
13      {
14          a = GetComponent<Animator>();
15      }
```

Tot seguit te un mètode que, quan el ninja entra en el disparador, tancarà les portes i farà sonar un soroll per a cada una.

```
17      // Closes the doors when the ninja enters into the boss room
18      private void OnTriggerEnter2D(Collider2D collision)
19      {
20          if (!entered)
21          {
22              a.SetBool("entered", true);
23              sounds[0].Play();
24              sounds[1].Play();
25              entered = true;
26          }
27      }
```

I finalment, té un mètode públic que cridarà el *final boss* quan aquest estigui mort per a obrir les portes i fer sonar els seus sorolls.

```
29      // Method called to open the doors when the boss dies.
30      public void openDoors()
31      {
32          a.SetBool("killed", true);
33          sounds[0].Play();
34          sounds[1].Play();
35      }
```

2.4.10- BossController

Aquesta *script* controla les mecàniques del *final boss* i te un comportament similar a la *script* que controla els *zombies*. En inicialitzar-se guarda en variables el *RigidBody2D* i l'*Animator.*

```
39      // Initialization
40      void Start()
41      {
42          rbZombie = GetComponent<Rigidbody2D>();
43          aZombie = GetComponent<Animator>();
44          zombieIsDead = false;
45      }
46
```

A continuació inicia el bloc de la funció *Update* en el que primer que res atura la seva execució si el *final boss* està mort o si el jugador està mort.

```
47      void FixedUpdate()
48      {
49          // Stops the movement if the zombie is dead
50          if (zombieIsDead)
51          {
52              Destroy(this.GetComponent<BoxCollider2D>());
53              return;
54          }
55          if (player.GetComponent<NinjaControlerScript>().playerIsDead)
56          {
57              aZombie.SetFloat("Speed", 0f);
58              return;
59          }
```

Tot seguit, hi ha un condicional que, quan aquest es colpejat, inicia un compte enrere controlat per un altre condicional. Fins que aquest compte enrere no arriba a 0 el color del *final boss* serà roig.

```
61          if (hit)
62          {
63              timeHit = timeHit - Time.deltaTime;
64          }
65          if (timeHit < 0f)
66          {
67              hit = false;
68              transform.GetComponent<Renderer>().material.color = Color.white;
69              timeHit = 3f;
70          }
71
```

Després hi ha un altre condicional que controla el moviment d'aquest a partir de la diferencia de posicions entre ell i el *ninja.* Quan el *ninja* estigui a menys de 20 unitats del *final boss*, aquest es mourà cap a ell sempre i quan no estigui 3 unitats per sobre seu. Quan el *ninja* estigui aquestes 3 unitats per sobre seu, el *final boss* s'allunyara d'ell a una velocitat inferior que la de moviment normal. I, si aquest està en estat colpejat, s'allunyarà ràpidament del *ninja* independentment de la altura a la que es trobi aquest. Finalment, indica al *RigidBody2D* el nou vector velocitat i envia la informació a l'*Animator*.

```
72          // Checks the distance within the player and moves to him if he's close
73          offset = player.transform.position - transform.position;
74
75          if (offset.x < 20f && offset.x > 0.1f && offset.y < 3f)
76          {
77              move = 8f;
78          }
79          else if (offset.x > -20f && offset.x < -0.1f && offset.y < 3f)
80          {
81              move = -8f;
82          }
83          else if (offset.x < 20f && offset.x > 0.1f && offset.y > 3f)
84          {
85              move = -4;
86          }
87          else if (offset.x > -20f && offset.x < -0.1f && offset.y > 3f)
88          {
89              move = 4;
90          }
91          else
92          {
93              move = 0;
94          }
95
96          // Move the zombie in the scene
97          if (hit && offset.y < 3)
98          {
99             move = move * -2;
100         } else if (hit && offset.y > 3)
101         {
102            move = move * 3;
103         }
104         rbZombie.velocity = new Vector2(move, rbZombie.velocity.y);
105
106
107         // Sends the movement to the animator
108         aZombie.SetFloat("Speed", Mathf.Abs(move));
109
```

Al igual que en les scripts *NinjaControllerScript* i *ZombieController,* aquest bloc conté un condicional per a fer que les animacions tinguin la mateixa direcció que el moviment.

```
111         // Chooses the direction to move
112         if (move > 0 && !facingRight)
113         {
114             Flip();
115         }
116         else if (move < 0 && facingRight)
117         {
118             Flip();
119         }
120     }
121
```

A continuació, fora del la funció *FixedUpdate*, ens trobem amb un mètode que s'activa quan entra en un altre disparador i controla el tipus d'aquest mitjançant un *switch-case*.

En el cas que sigui el jugador, activa l'animació d'atac, informa a la script *NinjaControllerScript* que ha de matar al *ninja* i restaura les vides al seu valor inicial.

En el cas que el ninja li salti al damunt, si no no està en estat colpejat, passa a estar-ho, perd una vida i inicia el soroll de quan es colpejat. A continuació comprova si li queden vides al *final boss* i, si no li'n queden, indica que està mort, envia la informació a *l'Animator*, obre les portes i canvia la música. Si encara li queden vides, tenyeix de roig els sprites del *final boss* per indicar que està en estat colpejat. I, finalment, crida el mètode que fa rebotar al *ninja.*

```
136     // Controls if the zombie has to do the attack animation or the dead animation
137     private void OnTriggerEnter2D(Collider2D other)
138     {
139         switch (other.tag)
140         {
141             case "Player":
142                 if (!zombieIsDead)
143                 {
144                     aZombie.SetBool("Attack", true);
145                     attack.Play();
146                     if (!(player.GetComponent<NinjaControlerScript>().isPlayerDead()) && !zombieIsDead)
147                     {
148                         player.GetComponent<NinjaControlerScript>().KillPlayer();
149                         bossLifes = 5f;
150                         hit = false;
151                     }
152                 }
153                 break;
154             case "fallingAttack":
155                 if (!hit)
156                 {
157                     hit = true;
158                     bossLifes = bossLifes - 1f;
159                     dead.Play();
160                     if (bossLifes <= 0)
161                     {
162                         zombieIsDead = true;
163                         aZombie.SetBool("Dead", true);
164                         doors.GetComponent<DoorsController>().openDoors();
165                         mainTheme.Stop();
166                         afterBoss.Play();
167                     }
168                     else
169                     {
170                         transform.GetComponent<Renderer>().material.color = Color.red;
171                     }
172                 }
173 
174                 player.GetComponent<NinjaControlerScript>().BossRebound();
175 
176                 break;
177         }
```

Després hi ha el mètode que en sortir el disparador indica a l'*Animator* que ha de deixar de mostrar l'animació d'atac.

```
181     // Reset the attack boolean when the player goes away
182     private void OnTriggerExit2D(Collider2D collision)
183     {
184         aZombie.SetBool("Attack", false);
185     }
```

I finalment el mètode per a fer que les animacions mirin en la mateixa direcció a la que s'està movent el *ninja.*

```
187     // Changes the direction of the character
188     void Flip()
189     {
190         facingRight = !facingRight;
191         Vector3 theScale = transform.localScale;
192         theScale.x *= -1;
193         transform.localScale = theScale;
194     }
195
196 }
```

2.4.11- SaveLoadlevelState

Aquesta script declara una classe estàtica per a que sigui permanent i puguin cridar els seus mètodes des de qualsevol lloc i és l'encarregada de guardar el nivell màxim al que ha arribat el jugador.

El seu funcionament és guardar l'enter que se li passi mitjançant el mètode *Save* en un fitxer que perdurarà després de tancar el joc, i recuperar aquest valor quan criden el mètode *Load*.

```
7   public static class SaveLoadLevelState {
8
9       public static int levelReached = 0;
10
11      //it's static so we can call it from anywhere
12      public static void Save(int lvlR)
13      {
14          levelReached = lvlR;
15          BinaryFormatter bf = new BinaryFormatter();
16          FileStream file = File.Create(Application.persistentDataPath + "/savedState.gd");
17          bf.Serialize(file, levelReached);
18          file.Close();
19      }
20
21      public static void Load()
22      {
23          if (File.Exists(Application.persistentDataPath + "/savedState.gd"))
24          {
25              BinaryFormatter bf = new BinaryFormatter();
26              FileStream file = File.Open(Application.persistentDataPath + "/savedState.gd", FileMode.Open);
27              levelReached = (int)bf.Deserialize(file);
28              file.Close();
29          }
30      }
31
```

2.5- Música i sons

La música que he utilitzat ha estat tota creada per Matthew Pablo. L'he utilitzat amb el seu consentiment i respectant les normes d'atribució demandades.

Menú d'inici – Prologue

Nivell 1 al 4 – ArcLight

Nivell 5 – Blackmoor Colossus

Final del joc – Sunny Flight (2014)

Game Over – Irradiated Dreams

Tots els altres sons els he trobat a la web https://opengameart.org/ i tots són gratuïts i sense atribució.

2.6- Dificultats

El principal problema amb el que he tingut que lidiar ha sigut la meva inexperiència. Aquest ha sigut el meu primer videojoc, el meu primer projecte amb Unity i el primer projecte que he realitzat sense unes pautes a seguir. I, per això, he perdut més temps del necessari en molts dels problemes que m'han sorgit.

Una altra cosa que em va costar va ser trobar *sprites* i música gratis que realment m'agradessin i pogués utilitzar en el meu projecte. Sobretot em va costar trobar un fons que quedes bé amb la temàtica i el moviment de la càmera.

També he tingut que modificar molts cops part del codi perquè a mesura que he anat avançant amb el projecte he vist que no era la forma més correcta d'implementar-ho o que aquesta m'impossibilitava afegir noves funcionalitats sense complicar moltíssim el codi.

A més, m'he trobat que el tutorials dels que disposa el propi Unity a la seva pàgina web, tot i estar molt complets per a iniciar-te, els falta profunditat quan busques anar una mica més enllà.

Amb això, els tres errors que realment m'han costat de solucionar i, que més que solucionar, he trobat una forma de camuflar-los són, per una banda, que els *zombies* en morir hi ha cops que no criden la funció que fa rebotar el *ninja*; per una 'altra banda, que el *ninja* es quedava encallat a les vores de les plataformes; i, finalment, volia que en el nivell 3 el *ninja* es veies més afectat per la gravetat a mesura que els cubs inclinaven el pla on es trobava aquest i no que solament dificultessin el salt mentre el *ninja* manté la posició.

3- Conclusions

Crear un videojoc no és una cosa fàcil. En els tutorials i cursos que pots trobar per internet sempre sembla tot molt fàcil i que tot hagi de funcionar a la primera. La veritat és que no és tant fàcil quan t'enfrontes a una pàgina en blanc sense que ningú he digui el que has de fer, on no tot el que programes funciona. El moment que et dones compte que no sempre esta al teu abast implementar en el joc allò que havies imaginat per senzill que pugui semblar... Però quan estàs fent una cosa que realment t'agrada, que gaudeixes fent-la, sempre pots trobar el camí per a tirar endavant amb la feina, sigui trobant la solució al problema o pensant alternatives.

I aquest, el joc que he creat, no ha sigut una excepció. Hi ha hagut dies que han pogut ser molt frustrants, hi ha hagut d'altres que tot a sortit a la primera, però he vist que, tant en uns com en els altres, estava gaudint amb allò que feia i que tenia ganes de seguir endavant per molts cops que no sortissin les coses bé i, al final, crec que he obtingut la meva recompensa.

Sé que és un videojoc que tècnicament és mediocre, que estèticament podria millorar bastant, que el seu codi podria estar més polit i, segurament, un llarg etcètera. Tot i això, quan he jugat a la versió final m'he divertit, he tingut aquella sensació de repte que tant ens agrada als jugadors i he vist en el meu joc una mica d'aquells jocs que van fer que molts en aficionéssim al gènere.

Amb això ja em donava per satisfet amb la feina feta. Però en veure que l'altra gent que l'ha jugat també ha experimentat aquest sentiments, que realment han gaudit jugant-lo fins al punt de demanar-me que seguis ampliant-lo i que el prepares per a poder desplegar-lo en altres plataformes com Linux o Android, m'he sentit molt feliç.

És per això que crec que he complert l'objectiu que m'havia marcat quan vaig elegir fer videojoc del gènere plataformes en 2D.

4- Webgrafia

Sprites – http://www.gameart2d.com/

Música – http://www.matthewpablo.com/

Sons – https://opengameart.org/

Informació – https://docs.unity3d.com/es/current/Manual/UnityManual.html