Títol: *Estudi de la llibreria PlayN pel desenvolupament de videojocs tipus plataformes sobre HTML5*

Volum: *1/1*

Alumne: *Jordi Llamas Pons*

Director/Ponent: *Luis Solano*

Departament: *LSI*

DADES DEL PROJECTE

Títol del Projecte: Estudi de la llibreria PlayN pel desenvolupament de videojocs tipus plataformes sobre HTML5

Nom de l'estudiant: Jordi Llamas Pons

Titulació: Enginyeria Informàtica

Crèdits: 37,5

Director/Ponent: Luis Solano

Departament: LSI

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: Luis Pérez Vidal

Vocal: Gemma Reig Ventura

Secretari: Luis Solano Albajes

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Índex

Índex 4

Llista de figures 7

Acrònims 9

1. Introducció i objectius 10

1.1. Introducció 10

1.2. Objectius del projecte 11

2. Videojoc tipus plataformes 13

2.1. Descripció i característiques 13

2.2. Història dels videojocs plataformes 13

2.2.1. Plataformes sense scrolling 13

2.2.2. Plataformes amb scrolling 16

2.2.3. Plataformes en 3D 19

3. HTML5 i tecnologies relacionades 22

3.1. CSS3 24

3.2. SVG 25

3.3. WebGL 27

3.4. WOFF 29

3.5. Aplicacions web i videojocs 30

4. Llibreria PlayN i altres tecnologies 33

4.1. Llibreria PlayN 33

4.1.1. Llibreria multi-plataforma 35

4.1.2. Interfície Game 36

4.1.3. Bucle del joc 37

4.1.4. Sistema de capes 38

4.1.5. Alternatives a PlayN 39

4.2. TriplePlay 40
4.3. Box2D 42
4.4. GWT 43
4.5. Google AppEngine 45
4.6. Eclipse 45
4.7. Maven 46
4.8. Git 48
4.9. JSON 48
5. Desenvolupament 50
5.1. Anàlisis de requisits 50
5.1.1. Requisits funcionals 50
5.1.2. Requisits no funcionals 51
5.2. Especificació 52
5.2.1. Model de casos d'ús 52
5.2.1.1. Actors del sistema 52
5.2.1.2. Diagrama de casos d'ús 53
5.2.1.3. Especificació de casos d'ús 53
5.2.1.3.1. Iniciar joc 54
5.2.1.3.2. Fer una pausa 54
5.2.1.3.3. Reprendre el joc 55
5.2.1.3.4. Finalitzar joc 56
5.2.1.3.5. Iniciar editor de nivells 57
5.2.1.3.6. Afegir elements 58
5.2.1.3.7. Moure elements 59
5.2.1.3.8. Eliminar elements 60
5.2.1.3.9. Desplaçar vista del nivell 61
5.2.1.3.10. Guardar nivell 62

5.2.1.3.11. Finalitzar editor de nivells 63
5.2.1.3.12. Mostrar wireframe 64
5.3. Disseny 65
5.3.1. Diagrama de classes general 66
5.4. Implementació 67
5.4.1. Diagrama de classes del nucli del programa 68
5.4.1.1. Classe pfcjoc 69
5.4.1.2. Classe GameLogic 71
5.4.1.3. Classe GameState 72
5.4.1.4. Classe MenuGameState 73
5.4.1.5. Classe EditGameState 74
5.4.1.6. Classe PlayGameState 78
5.4.1.7. Classe Level 82
5.4.1.8. Classes EditLevel i PlayLevel 85
5.4.2. Diagrama de classes de les entitats 86
5.4.2.1. Classe Element 87
5.4.2.2. Classe StaticElement i Background 88
5.4.2.3. Classe PhysicElement 89
5.4.2.4. Classe Player 90
5.4.2.5. Altres classes derivades de PhysicElement 92
5.4.3. Diagrama de classes auxiliars 93
6. Planificació i pressupost 95
7. Conclusions 101
7.1. Objectius del projecte 101
7.2. Possibles ampliacions 102
7.3. Valoració personal 104
8. Referències 105

Llista de figures

Figura 1: Donkey Kong 14
Figura 2: Mario Bros 15
Figura 3: Prince of Persia 15
Figura 4: Super Mario Bros 17
Figura 5: Alex Kidd 18
Figura 6: Mega Man 19
Figura 7: SkyRoads 20
Figura 8: Super Mario 64 20
Figura 9: HTML5 i tecnologies relacionades [5] 23
Figura 10: Exemple d'una imatge SVG amb el seu corresponent codi al costat 26
Figura 11: Angry Birds versió Web 27
Figura 12: Zygote Body (Abans Google Body) 29
Figura 13: Classes pfcjocJava i pfcjocHtml 35
Figura 14: Interfície Game 36
Figura 15: Bucle del videojoc 37
Figura 16: Jerarquia de capes de PlayN 38
Figura 17: Inicialització del món a Box2D 43
Figura 18: Fitxer pom.xml 47
Figura 19: Fitxer level.json 49
Figura 20: Diagrama de casos d'ús 53
Figura 21: Diagrama de classes general 67
Figura 22: Diagrama de classes del nucli del joc 68
Figura 23: Classe pfcjoc 70
Figura 24: Classe GameLogic 71
Figura 25: Classe GameState 72
Figura 26: Classe MenuGateState 74
Figura 27: Classe EditGameState 76
Figura 28: Classe PlayGameState 80
Figura 29: Classe Level 83
Figura 30: Diagrama de classes de les entitats 86
Figura 31: Classe Element 87

Figura 32: Classes StaticElement i Background 89
Figura 33: PhysicElement..... 89
Figura 34: Classe Player 90
Figura 35: Altres subclasse de PhysicElement 93
Figura 36: Classe Sprites i relacionades 94
Figura 37: Sprites i JSON 94
Figura 38: Diagrama de Gantt del projecte 97
Figura 39: Costos treballadors 99
Figura 40: Desglossament dels costos del projecte 100

Acrònims

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CSS	Cascading Style Sheets
DHTML	Dynamic HTML
GPU	Graphics Processing Unit
GWT	Google Web Toolkit
HTML5	HyperText Markup Language 5
IDE	Integrated Development Environment
JIT	Just-in-time
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
OpenGL	Open Graphics Library
OpenGL ES	OpenGL for Embedded Systems
PaaS	Platform as a Service
PGML	Precision Graphics Markup Language
POM	Project Object Model
SMIL	Synchronized Multimedia Integration Language
SVG	Scalable Vector Graphics
SWT	Standard Widget Toolkit
VM	Virtual Machine
VML	Vector Markup Language
W3C	World Wide Web Consortium
WebGL	Web Graphics Library
WOFF	Web Open Font Format
XML	Extensible Markup Language
XWT	XML Window Toolkit

1. Introducció i objectius

1.1. Introducció

En els darrers anys Internet, i, més en concret, el Web, s'ha convertit en una plataforma utilitzada diàriament per milions de persones arreu del món per a obtenir informació, comunicar-se amb altres persones, accedir a continguts d'entreteniment, etc. Una de les tendències ha estat traslladar al Web aplicacions amb funcionalitats típiques d'aplicacions d'escriptori, com processadors de text, fulls de càlcul, eines per a dibuixar o retocar imatges, etc. Fins i tot videojocs. Alhora, els navegadors web, entenent que el futur passava per les aplicacions web, i que, amb les capacitats que els navegadors oferien a principis de la dècada dels 2000, no es podria igualar les aplicacions d'escriptori, han anat evolucionant molt ràpidament, proporcionant als desenvolupadors noves capacitats, com veurem a continuació, per tal que les aleshores simples pàgines web poguessin esdevenir en aplicacions web.

En aquest projecte de final de carrera s'ha desenvolupat un petit videojoc de tipus plataformes amb el seu corresponent editor de nivells. Aquest videojoc funciona en qualsevol navegador modern que suporti HTML5, canvas, WebGL, localStorage i d'altres tecnologies web. Per a dur a terme aquest projecte s'ha utilitzat la llibreria PlayN. Desenvolupada per Google per a la versió web del popular videojoc Angry Birds, PlayN permet programar en el llenguatge de programació Java i compilar per a diferents plataformes [1]: Java, HTML5, Android, iOS i Flash. Un cop generada l'aplicació web HTML5 aquest s'ha pujat a Google AppEngine, que és una plataforma de cloud computing de tipus PaaS, és a dir, Platform as a Service, i que permet córrer aplicacions web en centres de dades gestionats per Google i que ofereix, entre altres característiques, escalat automàtic segons el nombre de peticions que es reben en un moment determinat.

1.2. Objectius del projecte

- **El principal objectiu** marcat a l'hora de desenvolupar aquest projecte de final de carrera consistia en la realització d'un videojoc 2D de tipus plataformes en HTML5. Les raons per cada una d'aquestes 3 característiques les detallem a continuació. S'ha triat el desenvolupament d'un videojoc en 2D ja que és significativament més simple i ràpid que un videojoc 3D. Aspectes com la programació del motor del joc, el disseny de nivells, el disseny de l'art i altres multiplica el temps i esforç que s'ha de dedicar per a un videojoc en 3D respecte a un videojoc 2D. Com es tracta d'un projecte amb temps limitat, desenvolupat per una única persona, la tria d'un videojoc 2D és la més raonable. S'ha triat un videojoc de tipus plataformes ja que, durant una època, entre els anys 80 i principis dels 90, els videojocs de tipus plataformes eren els més populars. Cada sistema de videojocs tenia nombrosos videojocs plataformes. Les noves videoconsoles s'estrenaven amb videojocs com Super Mario, Sonic i altres per tal d'atreure els compradors. Per a un servidor aquest era el tipus de videojocs més coneguts i familiars. Amb la gairebé total presència de videojocs 3D a les videoconsoles actuals, els videojocs plataformes han perdut popularitat, tot i que adaptacions de plataformes 3D com els mencionats Mario i Sonic encara tenen un èxit relatiu. En canvi, els videojocs senzills en 2D semblen viure una segona joventut amb la popularitat de les tendes virtuals dels telèfons mòbils de darrera generació, on qualsevol persona pot desenvolupar i penjar-hi un videojoc fàcilment. Finalment, s'ha triat fer un videojoc HTML5 ja que el nou èxit dels videojocs en 2D també s'ha traslladat a la web. Amb les noves capacitats dels navegadors web ha aparegut una quantitat important de videojoc 2D per a HTML5. Sembla haver-hi una tendència a l'alça de desenvolupar aplicacions i videojocs per a la web, assegurant així compatibilitat entre diferents dispositius que d'altre manera requeririen al desenvolupador crear una versió diferent del software per cada un dels dispositius. Per tant, la creació d'un videojoc HTML5 sembla adient si tenim en compte cap a on es dirigeix el món de les aplicacions.
- **Un altre objectiu important** ha estat el disseny d'un editor de nivells. Per a la creació d'un videojoc de tipus plataformes és essencial disposar d'uns nivells que captin l'atenció del jugador. Aquest nivells venen determinats per la posició de

diferents elements com plataformes que el jugador ha de saltar, enemics que ha de derrotar, elements decoratius, etc. Tots aquests elements que formen els nivells s'han de poder situar de forma fàcil i ràpida per tal de no perdre més temps del necessari. En conseqüència, és bàsic disposar d'un editor gràfic de nivells. En aquest projecte de final de carrera s'ha acompanyat al videojoc d'un editor de nivells on l'usuari pot fer servir el cursor per a seleccionar i situar elements, esborrar-los, canviar la seva posició, guardar la configuració del nivells i, finalment, jugar en el nivell que s'ha creat.

- **El darrer dels objectius** d'aquest projecte de final de carrera ha estat la publicació del videojoc més l'editor en un servidor remot. La llibreria PlayN triada per al desenvolupament d'aquest projecte permet generar un projecte en codi HTML5 i pujar-lo directament a Google AppEngine amb relativa facilitat. D'aquesta manera, aquest projecte no només s'ha provat de manera local, sinó que qualsevol persona hi pot accedir i provar-lo des de qualsevol navegador modern. Com s'ha comentat, si en el futur hi hagués una demanda important d'usuaris accedint al joc, Google AppEngine escalaria automàticament permetent a tots els usuaris jugar-hi.

2. Videojoc tipus plataformes

2.1. Descripció i característiques

Els videojocs de tipus plataformes o, per abreviar, els videojocs plataformes, són un tipus de videojocs en els quals el jugador ha d'avançar dins una escena saltant entre plataformes aèries i sobrepassant altres obstacles també saltant per tal d'aconseguir arribar al final del nivell. La principal característica que tenen tots els videojocs plataformes és, precisament, que inclouen l'acció de saltar. També hi ha altres mecanismes que han incorporat alguns jocs plataformes per ajudar a saltar el jugador com els trampolins o les plataformes amb molles, i fins i tot barres i cordes on els jugadors es poden balancejar per agafar impuls. Exemples de trampolins i plataformes amb molles els trobem a Alpha Waves [2], i de barres enganxades a parets i sostres on balancejar-se en trobem a Ristar i Bionic Comando.

2.2. Història dels videojocs plataformes

2.2.1. Plataformes sense scrolling

Els primers videojocs plataformes, apareguts a inicis dels anys 80, es desenvolupaven en una sola pantalla o habitació, sense cap tipus de desplaçament de càmera, degut a les limitacions tècniques dels sistemes de videojocs disponibles durant la època. Donkey Kong, del 1981, és considerat com el primer joc plataformes de la història, tot i que ja havien apareguts alguns jocs amb trets semblants als jocs plataformes com els coneixem avui dia, però amb personatges que no podien saltar. Donkey Kong era un joc per a màquines recreatives que permetia als personatges saltar sobre obstacles i a través de forats entre plataformes. Una dada curiosa és que Donkey Kong és el primer joc on va aparèixer els personatge del fontaner Mario, que després protagonitzaria una sèrie de

videojocs que arriben als nostres dies, entre ells el videojoc plataformes més venut de la història, Super Mario Bros 3, i que acabaria sent la icona principal d'aquest tipus de videojocs.

Figura 1: Donkey Kong

El primer videojoc de la saga Mario va ser el Mario Bros, del 1983, que era també un plataformes d'una sola pantalla o sense scrolling. Mario Bros també permetia el joc cooperatiu entre dos jugadors a la vegada. Desenvolupat per Shigeru Miyamoto, el joc consistia en que Mario i el seu germà Luigi, fontaners, havien d'eliminar els enemics que sortien de dues canonades a la part superior del nivell. La forma d'eliminar-los era donant-los una puntada de peu quan havien voltat, cosa que s'aconseguia saltant sota les plataformes per on aquestes criatures passaven. Un altre videojoc plataformes que va agafar idees de Marios Bros com el joc cooperatiu va ser Bubble Bobble, en el qual es controla a un drac que va elimina enemics generant bombolles, les quals els atrapen, i llavors el jugador les pot rebentar. Bubble Bobble va ser també dels primers videojocs a oferir múltiples finals.

Figura 2: Mario Bros

El videojoc Pitfall! incloïa escenaris gran dividits en múltiples pantalles que es podien travessar d'una a una altra per avançar en el nivell. Tot això sense cap tipus de desplaçament de càmera o scrolling. També a principis dels 80 apareix el joc dels Barrufets, o Smurf: Rescue in Gargamel's Castle, que a més d'incloure nivells dividits en múltiples pantalles, les transicions entre aquestes pantalles es realitzava amb una vista panoràmica de desplaçament, de manera que es donava la sensació de continuïtat entre les pantalles, estàtiques, del mateix nivell.

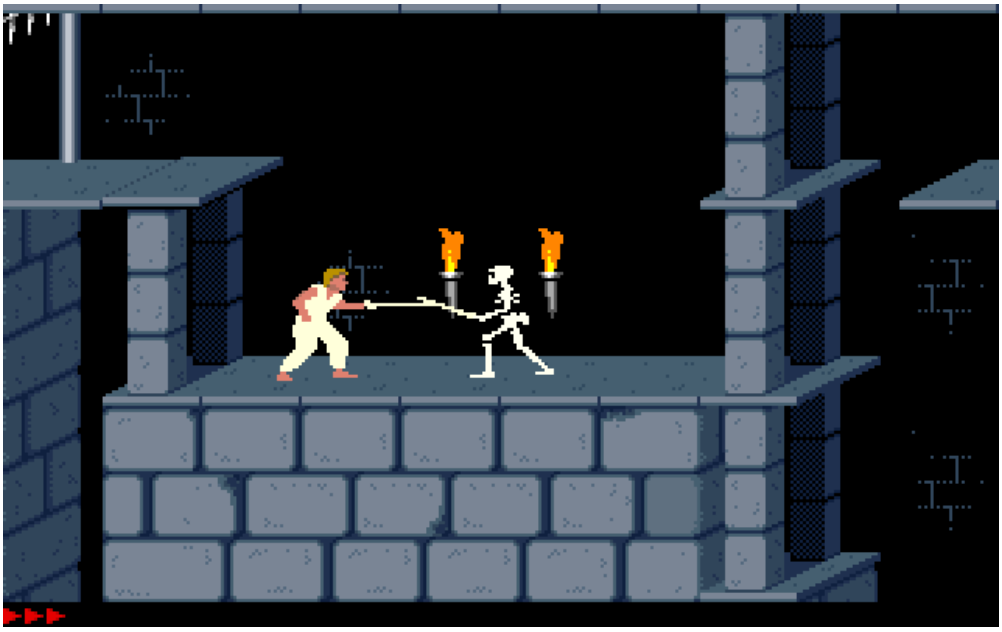


Figura 3: Prince of Persia

Un altre videojoc plataformes molt conegut d'aquella època és Prince of Persia, també sense scrolling, que oferia als jugadors exploració i unes animacions del personatge principal molt realistes degut a que el seu creador, Jordan Mechner, va fer unes gravacions del seu germà petit realitzant els moviments que després va reproduir amb molt de detall al videojoc.

2.2.2. Plataformes amb scrolling

El primer videojoc que va utilitzar scrolling en directe per a mostrar tota la escena va ser Jump Bug, que curiosament va sortir per a màquines recreatives el mateix any que el primer Donkey Kong. El joc consistia en controlar un vehicle que no parava de saltar per tal d'anar eliminant enemics amb una pistola. Un any després, el 1982, va aparèixer Jungle King, on el jugador controlava a Tarzán mentre aquest recorria la jungla i saltava entre lianes, que realitzava scrolling inclús quan el jugador saltava. També el mateix any apareix Moon Patrol, amb la principal novetat d'incloure parallax scrolling, una tècnica que consisteix en desplaçar les diferents capes gràfiques del fons de les escenes a diferents velocitats per tal de donar sensació de profunditat al jugador. El 1984 apareix Pac-Land, que evoluciona la fórmula dels plataformes, fins al moment simples, i també incloïa parallax scrolling. El mateix any apareix Dragon Buster, que va introduir conceptes com el mesurador de vida i el nivell central, o hub, que era un pseudo- nivell en el qual el jugador podia fer coses com triar nivell per jugar a continuació i altres gestions com més endavant Super Mario Land i, especialment, Super Mario Bros 3, van incloure com gestió de potenciadors i altres.

El 1985 apareix Super Mario Bros, que es convertiria en el videojoc inclòs amb videoconsola, la NES, més venut amb més de 40 milions de còpies venudes. El joc consisteix en avançar pels nivells fins al final, on s'ha de saltar el més amunt possible d'un pal de bandera. Per avançar pels nivells, el jugador ha d'anar esquivant o eliminant enemics saltant-los a sobre. Una altra manera d'eliminar enemics és disparant boles de foc, el qual es pot fer quan s'ha agafat la flor que transforma la vestimenta del Mario i li dóna el poder de llançar boles de foc. Altres enemics amb closca resistent nomes es poden eliminar llançant-los una closca de tortuga que s'aconsegueixen saltant sobre les tortugues. El jugador també pot anar agafant monedes que hi ha espargides per tot el

nivell per tal de sumar punts i vides quan s'obtenen 100 monedes. El joc també té blocs, uns amb un interrogants que quan es colpegen per sota deixen anar monedes o potenciadors, com per exemple el bolet que converteix el Mario en Super Mario i es fa més gran. Altres blocs són fets de totxos que es poden trencar només quan amb el Super Mario. Un altre potenciador és la estrella, que s'obté de blocs invisibles, i que dóna a Mario invencibilitat durant una estona, alhora que elimina els enemics només de tocar-los. Super Mario Bros inclou també nivells aquàtics, nivells amagats sota les canonades verdes, enemics al final de cada món, etc.

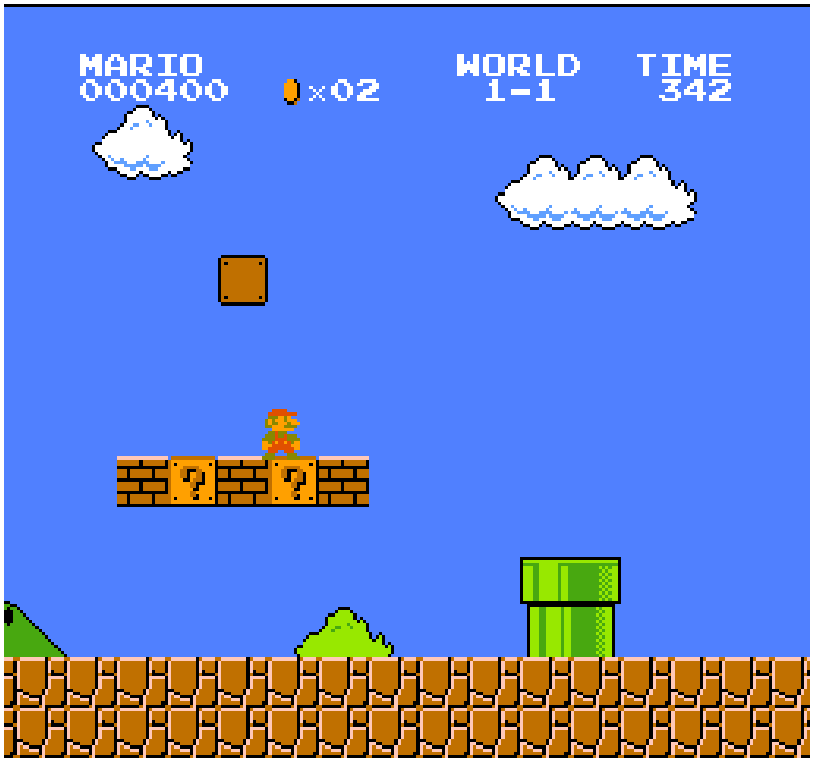


Figura 4: Super Mario Bros

Sega va respondre a l'èxit de Super Mario Bros amb Alex Kidd in Miracle World, que tenia nivells amb scrolling horitzontal i vertical, l'habilitat de colpejar amb els punys els enemics i els blocs del nivell. També tenia i tendes on comprar potenciadors, vehicles com una motocicleta o un helicòpter a pedals, etc. Sega també va desenvolupar el videojoc Wonder Boy, que incloïa parts on es podia avançar amb planxa de skateboard, cosa que donava sensació de velocitat.

Figura 5: Alex Kidd

Cal mencionar també videojocs com Metroid, del 1985, que oferia una experiència menys lineal i donava la possibilitat als jugadors d'explorar els nivells en totes direccions. Mega Man, del 1987, va introduir el concepte de joc no linear, amb un menú inicial on el jugador podia seleccionar quin dels nivells volia jugar a cada moment. Cada nivell tenia un enemic final que atorgava els seus poders a Mega Man quan els derrotava. El 1988 apareix Super Mario Bros 3, o SMB3, que amplia Super Mario Bros de totes les maneres, com afegint nous potenciadors per donar noves habilitats com poder volar, nadar més ràpid, etc. SMB3 ja inclou el nivell de selecció de nivells, amb múltiples camins per, sovint, avançar entre nivells en l'ordre desitjat pel jugador.

Amb les consoles de 16 bits, com Mega Drive i Super Nintendo, apareixen continuacions de jocs com Super Mario World i Alex Kidd in the Enchanted Castle. Sega aprofita la competència i les noves capacitats de les videoconsoles per treure Sonic the Hedgehog, el gran competidor de Mario. Sonic trencava amb el disseny més clàssic de Super Mario, oferint nivells amb desplaçaments a alta velocitat per tot tipus de terrenys com rampes corbes, loopings, etc. En aquesta època apareixen també jocs com Duke Nukem, Eartworm Jim, Aladdin, Mickey Mouse, Donkey Kong Country, etc.

Figura 6: Mega Man

2.2.3. Plataformes en 3D

Comentarem en aquest apartat una mica sobre videojocs plataformes 3D, sense aprofundir tant com en els plataformes 2D, ja que l'objectiu d'aquest projecte de final de carrera gira en torn a videojocs plataformes en dues dimensions per a navegadors web. Dit això, cal comentar la diferència entre videojocs que tenen gràfics 3D i es juguen també en tres dimensions, és a dir, on el personatge pot moure's en qualsevol direcció del eixos XYZ, i els videojocs amb gràfics 3D però on el jugador continua limitat a moure's només cap a l'esquerra i la dreta, a part de saltar, òbviament, i amb la càmera fixada i desplaçant-se només en el pla XY. Aquest últims es coneixen com a plataformes 2,5D. També hi ha una altra categoria que són els plataformes amb gràfics 2D que simulen la tercera dimensió utilitzant la perspectiva isomètrica. Alguns dels primers plataformes 3D van ser Congo Bongo, del 1983, Antarctic Adventure, del 1983, Penguin Adventure, del 1986, i Trailblaze, també del 1986. El 1987 trobem 3-D WorldRunner, en el qual el jugador avançar cap a l'interior de la pantalla, és a dir, la càmera se situa a l'esquena del jugador.

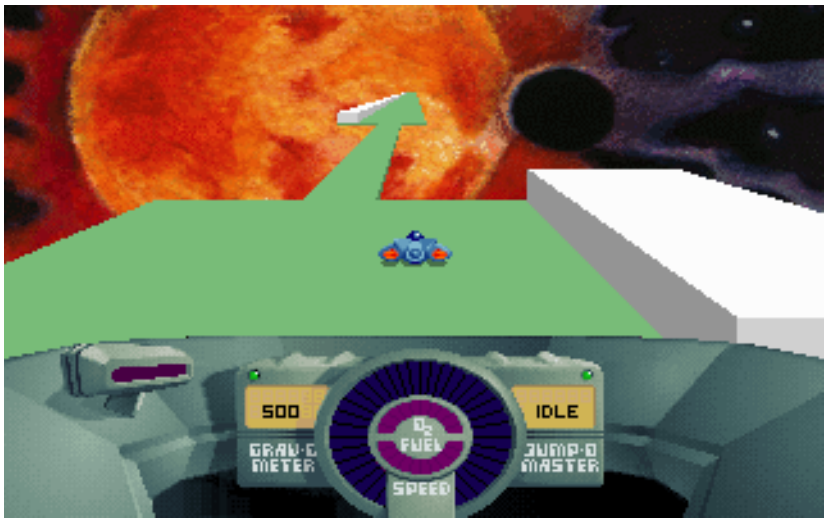


Figura 7: SkyRoads

Amb el mateix estil trobem SkyRoads, del 1993, disponible per a MS-DOS, que consistia en controlar una nau que avançava sobre una pista plena d'obstacles i forats entre plataformes que havia de saltar, com a tot plataformes, i en el qual cada cop la nau avançava a més velocitat.

Figura 8: Super Mario 64

Amb l'aparició de noves videoconsoles a finals dels anys 90, com la Nintendo 64, la PlayStation i la Sega Saturn, les quals ja tenien potència gràfica suficient per a produir

videojocs en tres dimensions amb més textures, era inevitable l'aparició de plataformes adaptats a les noves tecnologies. Apareixen jocs com Jumping Flash!, Bug!, Fade to Black o Crash Bandicoot. Ara bé, el videojoc que es pot considerar que va marcar l'inici dels plataformes 3D, i que va introduir molts conceptes que després molts altres seguirien, va ser Super Mario 64. Apareix l'any 1996, i es tracta d'un plataformes on els jugadors també han d'explorar els nivells, aprofitant les tres dimensions, aspecte que s'allunyava dels plataformes en 2D. A més, a part d'arribar al final de cada nivell i derrotar enemics, també s'introdueixen objectius paral·lels com la troballa d'estrelles. A partir d'aquí, van apareixer altres plataformes 3D com Banjoo-Kazooie, Sonic Adventure, Wario World, Donkey Kong 64, Super Mario Sunshine, etc.

3. HTML5 i tecnologies relacionades

HTML5 és la darrera versió, a meitats de 2012 encara en desenvolupament, del llenguatge estàndard HTML per a la creació de llocs i aplicacions web. En aquesta versió s'han afegit nous elements semàntics com <header>, <section>, <article> i altres per tal d'estructurar millor els documents sense tenir la necessitat de fer un abús d'elements genèrics com o per a la distribució dels elements dins un web. L'èmfasi, però, en la creació de HTML5 ha estat el desenvolupament d'aplicacions web que cada cop s'assemblin més a les aplicacions tradicionals d'escriptori, en termes de funcionalitats, rendiment, etc. Per això s'hi han afegit altres elements sintàctics, com <video> i <audio>, per poder integrar vídeos i sons dins el web sense haver de requerir d'afegits (plugins en anglès) com el Flash Player, Real player i altres.

Un altre element molt important incorporat a HTML5 és <canvas>, el qual permet definir una regió a la qual poder dibuixar a baix nivell, ja sigui en 2D o en 3D per mitjà de WebGL. L'element <canvas>, desenvolupat inicialment per Apple al 2004 [2] i adoptat més tard per altres navegadors com Mozilla Firefox i Opera, és el que ha donat una empenta a la creació de videojocs web. Els videojocs tenen molta presència en el món de les aplicacions d'escriptori des de fa molts anys i a la web, fins fa pocs anys, requerien de terceres parts com el complement Flash Player. Amb l'element <canvas> integrat a la majoria de navegadors [3], es fa possible la creació de videojocs en HTML5 que funcionin sobre qualsevol navegador modern amb tecnologies web estàndard. WebGL, per altra banda, permet gràfics 3D sobre <canvas>, de manera que inclús es pot programar videojocs web en 3D. El problema radica en que no tots els navegadors implementen WebGL. A meitats de 2012, entre Internet Explorer de Microsoft i les versions antigues dels altres navegadors, més de la meitat dels usuaris utilitzen un navegador Web que no implementa WebGL [4]. Una de les raons per les quals Microsoft rebutja WebGL és per estar basat en OpenGL, la llibreria oberta 3D d'escriptori que competeix amb Direct3D, de Microsoft.

Una altra característica important relacionada amb l'element Canvas és la interfície de programació requestAnimationFrame. Aquesta API, creada per Mozilla i adoptada per

gairebé tots els altres navegadors amb posterioritat, serveix per a la creació del bucle de joc i permet un ús més eficient dels recursos dels navegadors web en comparació amb les solucions basades en l'ús de temporitzadors amb la interfície setInterval de JavaScript. D'aquesta manera el navegador pot optimitzar vàries animacions en una sola fase de recol·locació dels elements i de repintat de la pàgina web. Un altre avantatge que ofereix requestAnimationFrame és que els videojocs o aplicacions que la utilitzin, si són en una pestanya d'un navegador no visible per a l'usuari, el bucle d'animació s'aturarà, estalviant així recursos tant de la CPU com de la GPU de la targeta gràfica.

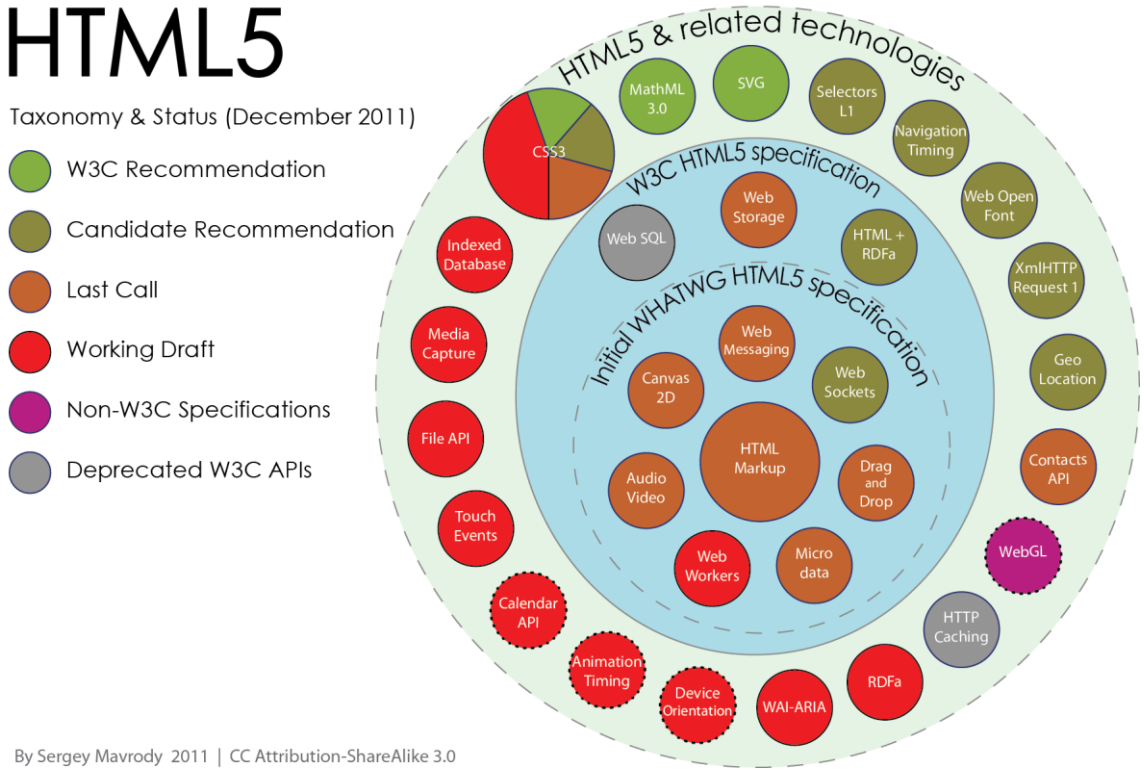


Figura 9: HTML5 i tecnologies relacionades [5]

Un fet interessant que ha succeït amb el terme HTML5 és ha deixat de significar únicament la versió 5.0 de HTML, i s'ha acabat convertint en un paraigües on s'hi ha encabit un conjunt d'estàndards i tecnologies web modernes [5] com per exemple:

- **CSS3**: darrera versió de CSS, llenguatge que s'utilitza per a descriure la presentació i l'estil dels webs HTML.
- **SVG**: format basat en XML utilitzat per a definir gràfics vectorials, molt útils per que permeten mantenir el nivell de qualitat a qualsevol nivell de zoom.
- **WOFF**: format de font per a tipografies web.

- **WebGL**: interfície de programació d'aplicacions (API en anglès) utilitzada per a mostrar gràfics en 3D i 2D. Per a fer-ho utilitza la targeta gràfica de l'usuari, el que se sol anomenar acceleració de gràfics per hardware.

3.1. CSS3

CSS (Cascading Style Sheets) és un llenguatge de full d'estils desenvolupat pel W3C (World Wide Web Consortium), que s'utilitza per a descriure l'aspecte i el format de documents escrits en llenguatges de marcatge. Habitualment aquest llenguatge de marcatge sol ser HTML, tot i que CSS es pot aplicar a qualsevol tipus de document XML, com XHTML, XML pur, SVG, etc. L'objectiu principal del llenguatge CSS és fer possible la separació entre el contingut dels documents, escrits en HTML o algun altre llenguatge de marcatge, i la presentació, que inclou conceptes com la disposició dels elements del documents, els colors, les tipografies, etc. Els avantatges d'aquesta separació són una millor accessibilitat dels continguts (per exemple per a lectors de pantalla per a persones amb problemes de visió), més flexibilitat i control en l'especificació de les característiques de la presentació, fer possible compartir un mateix estil a diferents pàgines i reduir la complexitat estructural dels documents habilitant el disseny web sense taules que tant s'havia utilitzat per a la distribució dels elements.

La primera versió del llenguatge CSS, Cascading Style Sheets en anglès, va ser publicada l'any 1996 [6] i algunes de les característiques que suportava són:

- Elecció de tipografies i canvi de colors per a text, fons i altres elements.
- Distància entre paraules, lletres i línies de text.
- Alineació de text, imatges, taules i altres elements.
- Configuració de les propietats margin, border, padding i posició per la majoria d'elements.
- Identificació única d'elements i classificació genèrica de grups d'atributs.

La segona versió de CSS va ser publicada el 1998, amb les següents noves característiques:

- Posicionament d'elements de manera absoluta, relativa i fixa.
- Z-index, que permet descriure quins elements es dibuixen sobre uns altres.

- El concepte de tipus de mitjans, que permet oferir a diferents mitjans, com una pantalla, paper, etc, diferents estils.
- Noves propietats de text com per exemple ombres, l'atribut shadow.

Finalment, la versió 3 del llenguatge CSS es va començar a escriure el 1999. A diferència de les versions anteriors, CSS3 no és una sola especificació, sinó que s'ha dividit en documents separats anomenats mòduls. En cadascun d'aquest mòduls es defineix un conjunt de noves característiques o extensions a característiques prèviament suportades. Hi ha més de 50 mòduls i a meitats de 2012 només 4 s'han publicat com a recomanacions pel W3C:

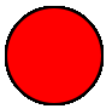
- **Media Queries**: una millora respecte a CSS2, ja que dins de cada mitjà inclou paràmetres com la mida de la pantalla, profunditat de color i relació d'aspecte.
- **Namespaces**: com en altres llenguatges de programació, permeten distingir diferents usos del mateix nom d'element.
- **Selectors Level 3**: els selectors permeten seleccionar elements dins d'un document HTML per tal d'afegir-l'hi l'estil desitjat. Els elements poden ser seleccionats a partir del seu nom, atributs, context i altres aspectes.
- **CSS Color**: mòdul que especificar aspectes relacionats amb els color dins dels documents CSS, com transparències, i varies notacions pels valors dels colors.

A finals de 2009 es va començar a escriure l'esborrany de la versió 4 del llenguatge CSS4, però cap navegador web en suporta cap de les seves característiques.

3.2. SVG

SVG (Scalable Vector Graphics) és un format de fitxer basat en XML per a descriure gràfics vectorials en dues dimensions (2D), ja siguin estàtics o dinàmics, amb o sense interacció per part de l'usuari. Desenvolupat pel W3C, el format SVG consta d'un fitxer de text amb extensió ".svg" que descriu les imatges SVG i els comportaments de les animacions SVG. És a dir, les imatges SVG es poden crear i modificar fent servir únicament un editor de text, tot i que és més pràctic i convenient l'ús d'aplicacions de dibuix vectorial que permeten guardar en format SVG, com Inkscape, Adobe Illustrator, etc.

Els navegadors web moderns com Firefox, Internet Explorer versió ≥ 9, Chrome, Opera i Safari, suporten el format SVG en diferents nivells i per tan poden mostrar imatges en format SVG directament i sense necessitat d'utilitzar un plugin o programa que s'afegeix a un navegador web per dotar-lo d'alguna funcionalitat extra. Com a exemple de plugin per a mostrar imatges SVG en navegadors web antics tenim Adobe SVG Viewer, el desenvolupament del qual va quedar aturat l'any 2009.



```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <circle cx="100" cy="50" r="40" stroke="black"
  stroke-width="2" fill="red" />
</svg>
```

Figura 10: Exemple d'una imatge SVG amb el seu corresponent codi al costat

El format SVG ha estat desenvolupat per un grup de companyies dins el W3C des de l'any 1999 [7] basant-se en els formats PGML (Precision Graphics Markup Language), alhora basat en Adobe PostScript i desenvolupat per Adobe, IBM, Netscape i Sun Microsystems, i VML (Vector Markup Language), desenvolupat per Autodesk, Hewlett-Packard, Macromedia, Microsoft i Visio

SVG suporta tres tipus d'objectes gràfics: gràfics vectorials, gràfics rasteritzats, és a dir, mapes de bits, i text. Qualsevol d'aquests objectes pot ser agrupat, modificat, transformat i composat dins d'altres objectes ja renderitzats. Des de l'any 2001, la especificació del format SVG s'ha actualitzat a la versió 1.1 i 1.2. També s'han afegit les variant SVG Mobile, SVG Tiny i SVG Basic. Aquestes tres variants estan pensades per a dispositius amb menys capacitats computacionals que un ordinador de sobretaula, per exemple telèfons mòbils.

Com ja s'ha comentat, SVG permet gràfics dinàmics i interactius. Aquests gràfics animats es poden programar en un llenguatge d'scripting com JavaScript, o es poden descriure les animacions en el llenguatge SMIL (Synchronized Multimedia Integration Language). SMIL és un llenguatge de marcatge, amb característiques similars a HTML, basat en XML, que permet definir el cronometratge dels esdeveniments, la distribució

dels elements, les animacions, transicions i la incrustació d'altres mitjans, com imatges, entre altres coses.

3.3. WebGL

WebGL (Web Graphics Library) és una interfície de programació en el llenguatge JavaScript que permet dibuixar imatges en 3D i 2D en aquells navegadors web que ho suportin sense cap necessitat d'instal·lar programari de tercers o plugins. Dissenyada i mantinguda per Khronos Group, WebGL permet realitzar efectes i processament d'imatge dins una pàgina web. Els elements de WebGL és poden combinar amb altres elements HTML i ser composats amb altres parts de la pàgina web. Per exemple, a una escena WebGL se li pot aplicar CSS com a qualsevol altre element HTML com, per exemple, afegir-li un marc amb la propietat border, o aplicar-li una transició CSS per tal que la escena sencera aparegui fent un degradat fade-in quan la pàgina web carregui.

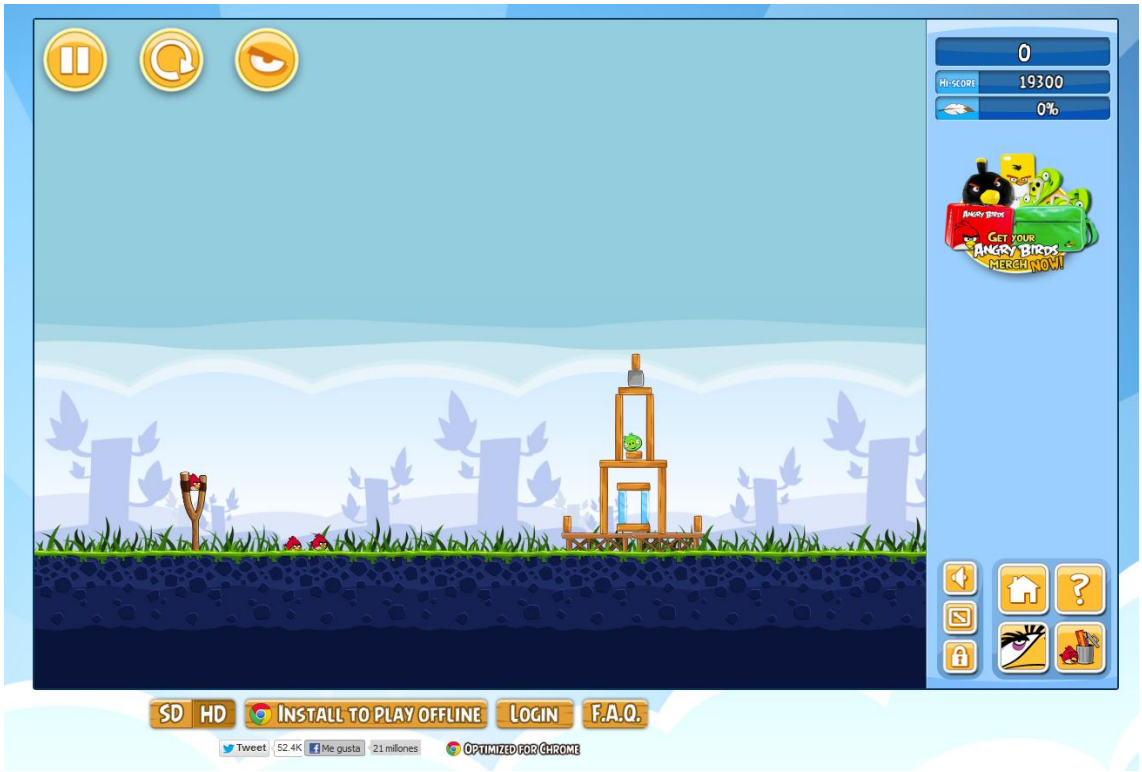


Figura 11: Angry Birds versió Web

Una aplicació que utilitzi WebGL consta del codi d'aplicació o de control en JavaScript, que conté la lògica del programa i que fa les crides a WebGL per pintar la escena, i el

codi dels shaders, el qual s'executa a la GPU (Graphics Processing Unit) de les targetes gràfiques. Els shaders són programes que s'utilitzen per a calcular els efectes de dibuixat amb bastanta flexibilitat. Les primeres targetes gràfiques tenien unes funcions fixes alhora de pintar els gràfics. Els shaders permeten realitzar modificacions diferents de les funcions fixes.

WebGL està basat en la interfície de programació OpenGL ES 2.0, la qual és un subconjunt de la interfície OpenGL dissenyada per a sistemes integrats com telèfons mòbils, tablets, consoles de videojocs, etc. WebGL utilitza l'element Canvas de HTML5 per a dibuixar les escenes desitjades. L'origen de WebGL el trobem en uns experiments realitzats per Mozilla el 2006 [4] per tal d'afegir característiques 3D a l'element Canvas. A principis de 2009 la organització sense ànim de lucre Khronos Group crea el grup de treball per a WebGL, per tal d'unificar les visions i opinions de companyies com Google, Apple, Opera, la pròpia Mozilla i altres. Així, el març de 2011 es publica la versió 1.0 de la especificació de WebGL. A meitats de 2012 l'únic navegador que no suporta WebGL és Microsoft Internet Explorer, degut, probablement, a que WebGL està basat en OpenGL, la interfície que competeix amb Direct3D de Microsoft. La resta de navegadors, Firefox, Chrome, Safari i Opera suporten WebGL, tot i que en graus diferents i, en alguns casos, amb les funcionalitats desactivades per tal de donar temps als desenvolupadors a millorar la seva implementació, com és el cas d'Opera i Safari.

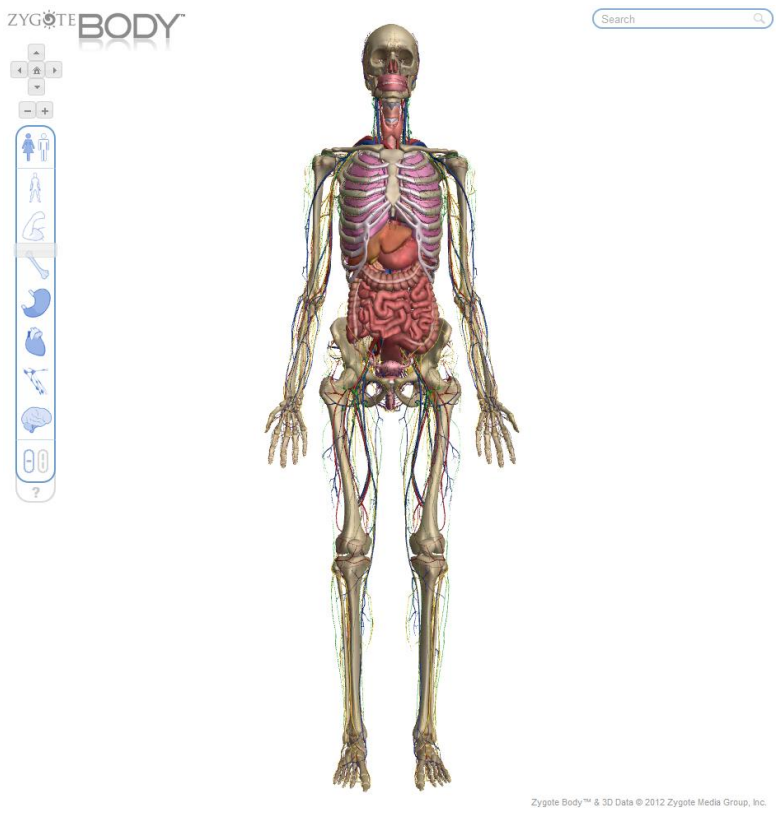


Figura 12: Zygote Body (Abans Google Body)

Algunes aplicacions web que utilitzen WebGL són Google Maps versió WebGL, Google Body (ara Zygote Body) i Angry Birds versió web, el qual utilitza la llibreria PlayN que s'ha estudiat i analitzat en aquest projecte de final de carrera. També s'han creat, al voltant de WebGL, llibreria auxiliars per tal de facilitar la programació d'escenes. Alguns exemple són Three.js, GLGE, gwt-g3d, KickJS, CubicVR.js i altres.

3.4. WOFF

WOFF (Web Open Font Format) és un format de fonts desenvolupat per a ser utilitzat dins de pàgines web. Es va desenvolupar durant l'any 2009 i està en procés de convertir-se en un estàndard recomanat pel W3C. WOFF va ser proposat per Mozilla, Opera i Microsoft al 2010, el qual pot facilitar les coses per a que es converteixi en l'únic estàndard per a fonts a la web, interoperable entre navegadors i suportat per tots els navegadors.

3.5. Aplicacions web i videojocs

Com ja hem explicat, en els darrers anys s'ha desenvolupat una gran quantitat d'aplicacions web, cada cop amb més funcionalitats. La raó d'aquesta transició cap a les aplicacions web la trobem en la quantitat d'usuaris, ja que els navegadors web han arribat a gairebé tots els sistemes informàtics, des d'ordinadors d'escriptori, passant per tabletes, fins a telèfons mòbils intel·ligents, reproductors de música portàtils, televisions, etc. Els navegadors web s'han convertit en una eina essencial en el dia a dia de moltes empreses, així com de la gent particular. Un altre motiu el trobem en la facilitat d'accés, ja accedir a una aplicació web només comporta haver d'escriure una adreça web, seleccionar-ne una de les adreces guardades, o fer un clic a un enllaç d'una altra pàgina o aplicació web. També cal mencionar que les aplicacions web no requereixen cap tipus de manteniment per part de l'usuari, ja que cada cop que s'hi accedeix ja es disposa de la última versió desenvolupada.

Les aplicacions web formen part del model de computació client-servidor. En aquest model el client i el servidor es reparteixen les tasques. El client, per exemple, mostra la interfície d'usuari amb la que interacciona l'usuari. En el cas de les aplicacions web, el client, el navegador web, es descarrega del servidor el codi que executarà la màquina client. Mentre duri la execució, el client enviarà peticions al servidor, per exemple, per obtenir noves dades, com nous correus electrònics o notícies. Inicialment les pàgines web tenien una interfície estàtica, que només canviava el seu estat quan es clicava algun enllaç per carregar una altra pàgina, o quan s'enviava un formulari de dades. Amb l'aparició del llenguatge d'scripting per a client JavaScript, el 1995 [8] per part de Netscape, els desenvolupadors de pàgines web van poder començar a utilitzar elements dinàmics que reaccionessin a les accions de l'usuari sense haver de contactar amb el servidor, per exemple, per mostrar un menú d'opcions al clicar sobre un botó.

La següent evolució en les aplicacions web va tenir com a protagonista un conjunt de tecnologies que es va venir a anomenar AJAX, Asynchronous JavaScript and XML, que consistia en poder descarregar parts d'una pàgina web per a modificar-la sense haver de baixar de nou tota la pàgina sencera. De manera asíncrona, sense interrompre a l'usuari com quan es carrega una pàgina web sencera de nou, el client pot realitzar peticions i processar les dades quan el servidor respongui. Gran part d'aquestes aplicacions web

AJAX fan servir la interfície de programació XMLHttpRequest, desenvolupada per Microsoft el 1999 [9] per al seu client de correu electrònic Outlook Web Acces, i inclosa més endavant en els navegadors Internet Explorer, Mozilla Firefox, etc.

Una altra de les evolucions dels navegadors web dels darrers anys ha estat l'increment de rendiment en la execució de programes en JavaScript [10]. Les aplicacions web cada cop fan servir més i més codi escrit en JavaScript, usant tècniques AJAX i llibreries auxiliars JavaScript per tal de desenvolupar més fàcilment aplicacions web complexes, oferirint les funcionalitats típiques de les aplicacions d'escriptori, com per exemple, menús dinàmics. Per això a finals de la dècada dels 2000 va començar una cursa entre els diferents navegadors per a optimitzar els seus motors d'execució de codi JavaScript. Com a llenguatge d'scripting, JavaScript és interpretat per una màquina virtual o VM. Per tal de millorar el rendiment d'execució de programes JavaScript, els navegadors han desenvolupat compiladors JIT, Just-in-time, de manera que el codi escrit en JavaScript es compila a llenguatge màquina a mida que es va executant i es guarda en una caché la compilació per accelerar també els futurs usos del codi.

Així, amb el temps, gràcies a aquest evolucions, han aparegut aplicacions web com:

- **Clients de correu electrònic**: com Gmail, Hotmail i altres, on es poden obrir correus electrònics sense haver de carregar de nou tota l'aplicació web.
- **Mapes online**: com Google Maps, Bing Maps i altres, on a mida que l'usuari es desplaça pel mapa, el client es descarrega dades del servidor i quan les rep les utilitza per formar el mapa. L'usuari veurà, segons la velocitat de la connexió a Internet, parts del mapa que primer són buides i que apareixen dinàmicament, sense haver de fer res més.
- **Eines ofimàtiques**: com Google Docs, Microsoft Office Web Apps i altres, que ofereixen processador textos, fulls de càlcul i creació de diapositives.
- **Calendaris**: com Google Calendar, Windows Live Calendar i altres, que ofereixen una eina per a gestionar el temps, marcar cites, crear esdeveniments, horaris, etc.
- **Allotjament de fitxers**: com Dropbox, Microsoft Skydrive, Google Drive i altres, que ofereixen espai a servidors remots per a guardar fitxers i la possibilitat de sincronitzar-los en diferents dispositius.

- **Videojocs**: com el mencionat Angry Birds, Bejeweled, Canvas Defense, Sketch Out, WordSquared, Z-Type, Canvas Rider, o la gran quantitat de jocs HTML5 disponibles en webs com www.html5games.com.

Els videojocs, com a aplicacions web modernes, estan en un moment de creixement. Les capacitats dels navegadors encara no permeten crear grans videojocs com el d'escriptori, però en canvi si que hi ha gran quantitat de jocs petits, amb una mostra d'ells donada en l'apartat anterior. Això té alhora l'avantatge que, al ser videojocs petits i normalment de durada reduïda, permeten que molta més gent pugui dedicar-hi estones curtes per a jugar-hi, en comptes que poca gent hi pugui dedicar molt de temps. D'aquesta manera l'abast d'aquests petits videojocs s'amplia, arribant a jugadors que no poden dedicar molt temps a un videojoc. Al mateix temps que apareixen nous videojocs, també s'estan desenvolupant eines, com motors per a crear videojocs en HTML5. Per exemple Mozilla està creant el Gladius [11], del qual de moment només hi ha disponibles versions preliminars en desenvolupament. Altres motors de videojocs HTML5 són: Impact, LimeJS, Isogenic Engine, Game Closure, Effect Engine, etc. Alguns d'aquest motors van ser considerats per a aquest projecte de final de carrera, com es detallarà en el proper apartat, abans de triar la llibreria PlayN, la qual no és un motor per a videojocs, sinó que proporciona les interfícies de programació necessàries per a crear un motor de videojocs propi, com s'ha acabat fent en aquest projecte.

4. Llibreria PlayN i altres tecnologies

4.1. Llibreria PlayN

La llibreria PlayN és una llibreria d'abstracció per al disseny de videojocs desenvolupada per Google des de principis de 2011 i presentada a la conferència per a desenvolupadors Google IO 2011. És de codi obert i es pot consultar a la següent adreça web: http://code.google.com/p/playn/. La principal característica de la llibreria PlayN és que permet als desenvolupadors escriure els seus programes o videojocs en llenguatge Java i compilar per a vàries plataformes. En aquest moment aquestes plataformes són:

- **Java**: aplicació Java d'escriptori que pot córrer sobre totes les plataformes que suporten Java com Windows, Linux, Mac, etc.
- **Navegadors HTML5**: aplicació web HTML5 utilitzant tecnologies explicades en apartats anteriors com WebGL, Canvas2D, CSS3, Web Audio API, etc.
- **Android**: aplicació Android per a dispositius mòbils com telèfons mòbils intel·ligents i tabletes de múltiples fabricants com Samsung, HTC, Sony, etc. Les darreres versions d'Android disposen de navegadors web moderns que farien redundant aquesta opció en un futur, excepte en alguns casos en que sigui necessari un rendiment que els motors de JavaScript no poguessin oferir.
- **iOS**: aplicació iOS per a dispositius mòbils de la companyia Apple com iPhone i iPad. De manera semblant al cas de l'aplicació Android, es pot fer la reflexió sobre la necessitat de compilar a aplicacions natives quan s'incorpora un navegador web mòbil modern.
- **Flash**: aplicació per al plugin per a navegadors web més conegut. Tot i que en diferents llocs webs i per a determinades funcionalitats les aplicacions de tipus Flash comencen a desaparèixer gràcies als navegadors web moderns, la llibreria PlayN proporciona aquesta opció que pot ser encara útil.

L'aplicació més coneguda fins al moment en utilitzar la llibreria PlayN és la versió per a navegadors web del popular videojoc per a mòbils Angry Birds, disponible a la següent

direcció: http://chrome.angrybirds.com/. Amb aquest videojoc es va presentar la llibreria PlayN al Google IO 2011 al públic. Inicialment la llibreria PlayN s'anomenava ForPlay, definint que la llibreria estava pensada per a jugar, per al disseny de jocs. Més endavant, però, se li va canviar el nom per evitar malentesos amb la paraula anglesa "foreplay", que res a veure té amb els videojocs.

La llibreria PlayN està desenvolupada sobre el framework de Google GWT (Google Web Toolkit), que permet als programadors escriure en Java i compilar per a navegadors web utilitzant tecnologies pròpies d'aquests, és a dir, sense que el client o l'usuari final necessiti instal·lar cap tipus de programa de tercers o plugin en el seu navegador. GWT típicament s'utilitza per a programar aplicacions webs de negoci, ja que permet passar una aplicació escrita en Java d'una empresa ràpidament a la web. Google va voler aprofitar tota la infraestructura que ja tenien amb GWT i construir la llibreria PlayN a sobre. Eines de desenvolupament com Eclipse, optimitzacions de rendiments en la compilació de Java a web, llibreries ja existents basades en GWT, etc. són algunes de les raons per les quals es va triar desenvolupar PlayN sobre GWT. Una d'aquestes llibreries per GWT és GWTBox2D, basada en la original en C++ Box2D, que permet realitzar càlculs físics per tal d'afegir un comportament més realista als videojocs.

El gran avantatge de la llibreria PlayN sobre altres llibreries per a plataformes específiques és que aquesta abstrau els detalls de les diferents plataformes per a les quals pot compilar el codi del desenvolupador. Aquest només ha d'utilitzar la API proporcionada per la llibreria PlayN per a programar el seu videojoc. Els detalls específics que PlayN amaga i simplifica de cara al desenvolupador són els següents:

- **Bucle del joc**: PlayN necessita que el desenvolupador implementi les funcions init(), update(delta), i paint(delta). Aquestes serveixen, respectivament, per a inicialitzar el videojoc, actualitzar l'estat del videojoc a cada volta del bucle i per a pintar l'estat del joc a cada volta del bucle.
- **Sistema d'entrada/sortida**: la llibreria PlayN facilita la feina del programador proporcionant una sola API enlloc d'una per a cada plataforma per a controlar aspectes com: el so, els gràfics, entrada de teclat, entrada de mouse, entrada tàctil, emmagatzematge i xarxes.

- **Gestió de recursos**: PlayN facilita la càrrega i descarrega dels recursos com gràfics, sons, textos, etc. per a ser utilitzats dins els videojocs. També s'encarrega de la cache per tal que si ja s'havia carregat un recurs amb anterioritat no es torni a carregar estalviant així recursos de xarxa o de computació.

4.1.1. Llibreria multi-plataforma

Com s'ha dit abans, la llibreria PlayN realitza un conjunt d'abstraccions per facilitar la feina al programador i li proporciona una API sobre la qual programar els seus videojocs per a vàries plataformes sense haver d'escriure codi específic de cap plataforma. L'única diferència en el codi de les diferents plataformes les trobem en els fitxers nomProjecteNomPlataforma.java. Per exemple, per al desenvolupament d'aquest projecte de final de carrera s'han utilitzat les plataformes Java i HTML5, especificades en els fitxers pfjocJava.java i pfcjocHtml.java:

```
public class pfcjocHtml extends HtmlGame {
  public void start() {
    HtmlPlatform platform = HtmlPlatform.register();
    platform.assets().setPathPrefix("pfcjoc/");
    PlayN.run(new pfcjoc());
  }
}
```

```
public class pfcjocJava {
  public static void main(String[] args) {
    JavaPlatform platform = JavaPlatform.register();
    platform.assets().setPathPrefix("pfc/pfcjoc/resources");
    PlayN.run(new pfcjoc());
  }
}
```

Figura 13: Classes pfcjocJava i pfcjocHtml

És important tenir en compte que la llibreria PlayN no és un motor de videojocs. Per a la programació de videojoc amb aquesta llibreria, el desenvolupador ha de desenvolupar

el seu propi motor, cosa que s'ha realitzat en aquest projecte de final de carrera a petita escala. PlayN simplement una llibreria que ofereix el bucle del joc i unes simplificacions de tal manera que el desenvolupador pot començar el desenvolupament del videojoc i del motor amb relativa facilitat. En aquest sentit, es podria comparar la llibreria PlayN amb la llibreria OpenGL, en el sentit que les dues proporcionen unes capes d'abstracció als programadors, ja que OpenGL abstrau al programador de la complexitat de programar una targeta gràfica directament en codi màquina.

4.1.2. Interfície Game

L'abstracció principal que ofereix la llibreria PlayN és la interfície Game que tot videojoc que utilitzi PlayN ha d'implementar. La llibreria PlayN implementa d'aquesta manera un motor de videojocs buit, realitzant les crides a unes funcions buides que el desenvolupador haurà d'implementar.

```
public interface Game {
	void init();
	void update(float delta);
	void paint(float alpha);
	int updateRate();
}
```

Figura 14: Interfície Game

La funcions que la interfície Game declara són:

- **init()**: codi d'inicialització del videojoc.
- **update(float)**: actualitza l'estat del videojoc a cada volta del bucle. El paràmetre delta de tipus float permet saber quants mil·lisegon han passat des de la darrera crida.
- **paint(float)**: dibuixa l'estat del videojoc tants cops per segon com la plataforma sobre la qual s'executa ho permet. El paràmetre alpha de tipus float permet saber quants mil·lisegons han passat des del darrer dibuixat.
- **updateRate()**: permet al desenvolupador especificar cada quants mil·lisegons s'ha de cridar a la funció update(float).

4.1.3. Bucle del joc

La llibreria PlayN ofereix al programador una interfície senzilla per a posar en marxa el bucle del videojoc. D'aquesta manera s'eviten els problemes d'haver de treballar amb els sistemes de temps de diferents plataformes. També s'aconsegueix separar l'actualització de l'estat del dibuixat del videojoc, de manera que es pot donar el cas que es pinti més d'un cop per cada actualització de l'estat, com veurem a continuació.

Anteriorment hem vist que el programador implementa una petita classe específica per cada plataforma que vol utilitzar, en la qual finalment acaba cridant a la funció PlayN.run(new pfcjoc()). A partir d'aquest crida el programador ja no controla el bucle del joc, sinó que aquest s'executa fins que se surt del programa. De manera conceptual podem veure el bucle així:

```
while (true) {
     game.update(...);
     game.paint(...);
}
```

Figura 15: Bucle del videojoc

El bucle del videojoc no s'executa el més ràpidament possible, ja que podria ocupar tots els recursos del processador de l'usuari, sinó que el ritme d'actualització depèn de la velocitat a la qual la plataforma en concret permet pintar, ja sigui per un límit d'aquesta plataforma o per un límit en la capacitat d'actualització de la pantalla del dispositiu.

El programador si que ha de tenir en compte que es pot donar el cas que el procés de pintat i el d'actualització de l'estat del videojoc no segueixin el mateix ritme, és a dir, que tinguin freqüències d'actualització diferents. Per a jocs amb una lògica senzilla aquesta velocitat es pot fer que sigui igual implementant la funció updateRate() per a que retorni 0, de manera que per a cada frame que es pinti es cridarà a la funció update(). Per a jocs més complexos, com el desenvolupat per aquest projecte de final de carrera, però, és necessari reduir la velocitat d'actualització de la lògica del videojoc, ja que estem treballant amb un nombre important d'elements dins un motor de física com és Box2D. En aquest videojoc s'ha triat un temps d'actualització de la lògica del joc de 25milisegons, el que correspon a 40 crides a update() cada segon. Això afegeix certa

complexitat a l'hora de programa el joc. Les crides a la funció paint() es poden produir en qualsevol moment entre dues actualitzacions del joc, fent extremadament difícil saber quin estat s'ha de tenir en compte en el moment de pintar el joc. La solució que proporciona la llibreria PlayN és que la funció paint() rebi un paràmetre entre 0 i 1 que representa en quina part del procés d'actualització ens trobem, entenent que com més a prop de 1 més a prop estem que es torni a produir una actualització de la lògica del joc. Amb aquest nombre podem realitzar una interpolació de l'estat del joc dins la funció paint() per tal que l'animació del videojoc sigui el més fluida possible, i alhora evitar pintar més d'un cop el mateix estat, cosa que malgastaria recursos i que faria que veiés uns moviments amb aturades constants.

4.1.4. Sistema de capes

La llibreria PlayN ofereix una interfície de programació per al dibuixat que permeten especificar amb detall el resultat final de les escenes. Per tal aconseguir un rendiment òptim en les diferents plataformes i en les diferents situacions que es vulguin dibuixar, PlayN proporciona diferents maneres de pintar representades en un sistema de capes.

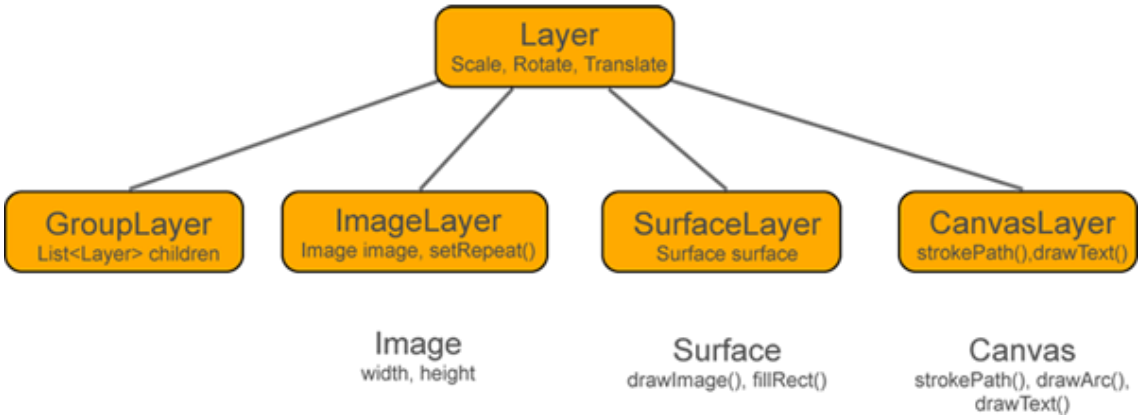


Figura 16: Jerarquia de capes de PlayN

Quan es pinta un videojoc desenvolupat amb PlayN s'utilitza un sistema jeràrquic de capes a partir de la capa arrel que proporciona el sistema. Hi ha diferents tipus de capes que proporcionen diferents interfícies de dibuixat diferent que corresponen a diferents mètodes de pintat en cada plataforma suportada per PlayN. Presentem a continuació els diferents tipus de capes disponibles:

- **GroupLayer**: capa lògica que serveix per a agrupar altres capes, podent realitzar així operacions sobre conjunts de capes tals com desplaçaments, rotacions, etc.
- **ImageLayer**: capa gràfica útil per a gràfics estàtics, pels quals no es requereix un control específic per modificar-los. És a dir, aquesta capa permet objectes que es moguin, rotin, etc. però amb els continguts de la capa sense modificar
- **SurfaceLayer i ImmediateLayer**: aquests dos tipus de capes s'han agrupat ja que les dues ofereixen una interfície de programació de dibuixat imperativa. Les funcions bàsiques que ofereix són les d'esborrar la imatge i pintar-ne una altre, útil per a implementar animació per sprites. La principal diferència entre una SurfaceLayer i una ImmediateLayer és tècnica. Una ImmediateLayer pinta directament al framebuffer de la targeta gràfica, mentre que una SurfaceLayer crea un buffer a la memòria de la targeta gràfica i pinta dins d'aquest buffer, de manera que a cada frame aquest buffer s'ha de copia dins el framebuffer. La recomanació és utilitzar ImmediateLayer si a cada volta del bucle s'ha d'esborrar i repinta la imatge de nou per evitar malgastar memòria de la targeta gràfica. En canvi, si l'actualització de les imatges es realitzarà més espaiadament en el temps és millor utilitzar una SurfaceLayer. La forma de programar-les també vària sensiblement, ja que en una ImmediateLayer el programador no controla quan es pintarà la capa, sinó que ha programar una funció de callback, la qual serà cridada cada cop que PlayN ho consideri oportú.
- **CanvasLayer**: aquest és el tipus de capa que ofereix una interfície per a pintar molt més rica que qualsevol dels altres tipus de capes, però també fa que sigui el tipus de capa que es pinta més lentament. Les capes CanvasLayer són útils quan es necessita pintar de manera procedimental, com per exemple línies, cercles, rectangles, gradients o text. També es poden pintar imatges com en les capes SurfaceLayer. La interfície de programació per a dibuixar que ofereix aquest tipus de capes és molt similar a la interfície que ofereix l'element Canvas de HTML5.

4.1.5. Alternatives a PlayN

En aquest apartat parlarem de les alternatives a la llibreria PlayN triada per a la realització d'aquest projecte de final de carrera. Principalment aquestes alternatives són

llibreries per a programar en JavaScript i HTML5 directament, sense cap tipus de compilació intermèdia. Algunes d'aquestes eines són més que simples llibreries i inclouen un motor per a videojoc. Aquestes eines alternatives es van considerar abans de triar la llibreria PlayN per al desenvolupament d'aquest projecte de final de carrera:

- **EffectGames**: era una llibreria de codi propietari que es va alliberar com a codi obert quan es va abandonar el seu desenvolupament poc després de l'inici d'aquest projecte de final de carrera. Aquesta que inclou tot un seguit d'eines per a desenvolupar videojocs i per a publicar-los com a jocs web. Entre les eines hi ha un gestor de recursos com sons, imatges, etc. També inclou un editor de nivells, un gestor d'sprites, un editor d'entorns així com eines per a publicar els videojocs creats. El llenguatge que utilitza és JavaScript. Funciona amb DHTML per als gràfics, comú fa anys, enlloc de fer servir eines modernes com l'element Canvas o WebGL. Per al so utilitza HTML5 quan es pot, recorrent a Flash quan fa falta.
- **Impact**: és un motor de videojocs que utilitza el llenguatge JavaScript. És de pagament i utilitza HTML5 i Canvas. Inclou també un editor de nivells per a videojocs en 2D, així com eines per a publicar els videojocs a l'Apple AppStore. Finalment, també ofereix eines als desenvolupadors, com un debugger per a trobar errors en el videojocs, problemes de rendiment, etc.
- **LimeJS**: és un motor per a la creació de videojocs en el llenguatge JavaScript per a navegadors web i sistemes operatius de mòbils moderns. És de codi obert.
- **Mozilla Gladius**: és un motor per a videojocs 3D en JavaScript i HTML5 , de codi obert i creat per Mozilla. Per als gràfics en 3D utilitza la llibreria CubiVR.js, que funciona sobre WebGL. Per als efectes físics utilitza Box2D.js

4.2. TriplePlay

Una de les llibreries importants que s'ha utilitzat en la realització d'aquest projecte de final de carrera és la llibreria TriplePlay de la companyia ThreeRings, la qual es va fundar l'any 2001 i l'any 2011 va ser adquirida per SEGA. La llibreria TriplePlay ofereix una col·lecció d'utilitats per a ser incorporades en videojocs desenvolupats per la

llibreria PlayN, com per exemple animacions, eines per a crear interfícies gràfiques, efectes de partícules, etc.

A continuació detallem algunes de les funcionalitats que proporciona la llibreria TriplePlay:

- **Interfícies gràfiques**: el paquet TriplePlay.UI ofereix vàries funcionalitats per a definir interfícies gràfiques:
    - **Col·locació d'elements**: com en d'altres entorns de programació d'interfícies gràfiques, es poden col·locar a la interfície elements com: botons, caselles de verificació, etiquetes, camps de text editatble, sliders, botons d'activació/desactivació, espaiadors, etc.
    - **Estil dels elements**: capacitat per a definir l'estil dels elements en cascada, de manera semblant al llenguatge CSS, però de manera programàtica enlloc de descriptiva. Algunes característiques de les que es poden configurar són: tipus de lletra per al text, color del text, color d'ombres, color de destacats, alineació d'elements a esquerra, dreta i centre, alineació del text d'una etiqueta a esquerra, dreta i centre i altres tipus de configuracions.
- **Utilitats per a videojoc**: el paquet TriplePlay.Game ofereix funcionalitats per a gestionar diferents pantalles i les transicions entre elles.
- **Animacions**: el paquet TriplePlay.Anim ofereix funcionalitats per a definir animacions per a elements.
- **Partícules**: el paquet TriplePlay.Particle ofereix funcionalitats per a generar efectes de partícules, molt utilitzades en alguns tipus de videojocs.
- **Altres utilitats**: el paquet TriplePlay.Util conté una col·lecció de vàries funcionalitats que poden ser útils per al desenvolupament de videojocs web com: funcions útils per a gestionar les capes, parsing de fitxer en format JSON, interpolació de valors, configuració per renderitzar text de vàries maneres, eines per gestionar els mètodes d'entrada com mouse/punter, generació de nombres aleatoris i altres utilitats relacionades amb probabilitats, registre d'esdeveniments en local i remot, etc.

4.3. Box2D

Box2D és una motor per a la simulació física de sòlids rígids, és a dir, cossos que no es deformen, en dues dimensions o 2D. És tracta d'una llibreria de codi obert programada en llenguatge C++, tot i que s'han realitzat conversions a Java, Flash, JavaScript, GWT i altres llenguatges. En el cas d'aquest projecte de final de carrera s'ha utilitzat la variant GWTBox2D Alguns videojocs coneguts per a mòbils i webs de videojocs en Flash que han utilitzat Box2D són: Crayon Physics, Angry Birds, Rolando, Fantastic Contraption, etc.

Les principals característiques que Box2D suporta són:

- **Col·lisions**: Box2D implementa detecció contínua de col·lisions, que és un mètode de detecció de col·lisions que a cada volta del bucle prediu la trajectòria dels objectes físics de la simulació i calcula els instants de les col·lisions amb molta precisió, a diferència de la detecció discreta de col·lisions, que a cada volta del bucle avança la simulació una mica i comprova si els objectes físics col·lideixen entre ells, de manera que es poden donar situacions en que dos objectes físics puguin solapar-se entre ells. També proporciona callbacks per als contactes, en concret per a quan aquests comencen i acaben i també per a realitzar càlculs abans i després de la resposta a una col·lisió. Altres funcionalitats relacionades amb les col·lisions suportades són els polígons convexos, cercles, formes múltiples per als cossos físics, etc.
- **Física**: respecte a les característiques físiques suportades per Box2D hi ha la simulació física continua amb resolució del temps d'impacte, la reproducció d'aspectes com el contacte, la fricció i la restitució entre objectes. Box2D també implementa el concepte d'unions o articulacions, per mitjà del qual es poden unir objectes i restringir-los els moviments un respecte un altre, per exemple creant politges, marionetes, balancins i altres. El conjunt de joints inclou: revolute joint, distance joint, pulley, gear, mousejoint, etc. Box2D també implementa la reacció dels objectes a l'aplicació de forces i impulsos.

El desenvolupador d'un videojoc una aplicació que utilitza Box2D haurà de crear una variable del tipus World, que conté totes les entitats físiques i les gestiona, i configurar-

lo amb paràmetres com la gravetat o si deixa de simular els objectes que esdevinguin inactius per tal de millorar el rendiment del programa.

```
Private World world;
Boolean doSleep = true;
Vec2 gravity = new Vec2(0.0f, 10.0f);
world = new World(gravity, doSleep);
world.setWarmStarting(true);
world.setAutoClearForces(true);
world.setContactListener(this);
```

Figura 17: Inicialització del món a Box2D

4.4. GWT

GWT, o Google Web Toolkit, és un conjunt d'eines de codi obert que permet desenvolupar aplicacions en el llenguatge de programació Java i compilar-les a JavaScript i altres tecnologies web per tal que aquestes puguin funcionar sobre un navegador web. GWT permet als desenvolupadors treballar amb l'entorn de programació que desitgin, i, gràcies als plugins GWT per a entorns com Eclipse, NetBeans, JDeveloper i altres, es facilita la realització de les tasques relacionades amb GWT dins aquests entorns com la creació de projectes, cridar al compilador GWT, ressaltat de sintaxis, etc.

GWT proporciona una gran quantitat d'utilitats als desenvolupadors. Algunes de les característiques que ofereix són:

- **Gestió de l'historial dels navegadors web**: per facilitar la gestió dels estats de les aplicacions web en resposta als botons endarrere/endavant dels navegadors web.
- **Components per a interfícies gràfiques dinàmics**: es disposa de classes ja dissenyades per a estalviar temps en la implementació de comportaments dinàmics de les aplicacions web com per exemple un seleccionador de fitxers.
- **Integració amb JUnit**: per a facilitar la prova i verificació de les aplicacions web.

- **Depuració Java**: igual que per aplicacions Java, es permet executar les aplicacions pas a pas, observar els valors de les variables en un moment determinat, etc.
- **Gestió de les diferències entre navegadors**: el desenvolupador només escriu codi en Java i el compilador genera codi que funciona sobre qualsevol navegador modern.
- **Suport per a l'element Canvas d'HTML**
- **Codi JavaScript**: els desenvolupadors poden barrejar o incrustar codi JavaScript dins el codi en Java per mitjà de JSNI, o JavaScript Native Interface. És l'equivalent a incrustar codi en assemblador dins un fitxer de codi en C, per exemple.
- **Ofuscació del codi JavaScript**: per tal de minimitzar la mida dels fitxers JavaScript i alhora amagar detalls de la implementació.
- **Múltiples llibreries**: implementades per altres grups, com per exemple, les llibreries per a interfícies gràfiques Ext GWT, GWT-Ext, GWT Widget Library, Rocket GWT, Dojo i altres

Els principals components de GWT són:

- **Compilador de Java a JavaScript**: s'encarrega de la traducció dels programes escrits en Java al llenguatge web JavaScript.
- **Mode Desenvolupador**: aquest mode permet als desenvolupador executar les aplicacions GWT com a aplicacions Java dins la JVM, o Java Virtual Machine, sense compilar prèviament a JavaScript. Aquest mode necessita que el desenvolupador instal·li un plugin al seu navegador web que farà de pont per executar el codi en Java.
- **Emulació de la llibreria JRE**: la llibreria JRE, Java Runtime Environment, conté paquets que són de molta utilitat per al desenvolupament d'aplicacions Java, com utilitats per a utilitzar vectors, llistes, utilitats per a gestionar entrada sortida de dades, per a donar format a Strings, tipus de dades com Booleans, floats, longs, etc. Un subconjunt de les classes d'aquests paquets ha estat implementat en JavaScript per tal que els programadors habituats a Java tinguin una transició més suau a GWT i se sentin més còmodes. Al següent enllaç es pot consultar totes les classes implementades en la versió més recent de GWT: https://developers.google.com/web-toolkit/doc/latest/RefJreEmulation?hl=es-ES

- **Llibreria de classe GWT Web UI**: conjunt d'interfícies i classes per a la creació de widgets per a les aplicacions GWT. S'ofereixen els habituals widgets d'entorns de programació d'interfícies d'usuari com: botons, caselles de selecció, seleccionador de data, botó d'activació/desactivació, caixa de text, caixa de text per contrasenyes, etiquetes de text, barres de menú, arbres, taules, etc.

4.5. Google AppEngine

Google AppEngine és una plataforma de cloud computing de tipus PaaS, és a dir, Platform as a Service, i que permet córrer aplicacions web en centres de dades gestionats per Google i que ofereix, entre altres característiques, escalat automàtic segons el nombre de peticions que es reben en un moment determinat. AppEngine suporta els llenguatges de programació Python, Java i Go.

A diferència d'altres serveis de computació com Amazon EC2, que permet llogar màquines virtuals, AppEngine dóna més facilitats per a desenvolupar aplicacions que es puguin escalar a moltes màquines a canvi de limitar la varietat d'aquestes aplicacions, ja que han d'estar dissenyades per a la infraestructura específica de l'AppEngine. Els desenvolupadors no s'han d'ocupar de l'administració dels sistemes com a Amazon EC2. Google s'encarrega de distribuir el codi a diferents clústers, de la monitorització i de la execució de les diferents instàncies segons sigui necessari.

4.6. Eclipse

Eclipse és un entorn integrat de desenvolupament de software, o IDE per les seves sigles en anglès Integrated Development Environment, de codi obert programat en Java i que permet el desenvolupament de tot tipus d'aplicacions en Java, per defecte, i en altres llenguatges com C, C++, Perl, PHP, Python, Ruby i altres, gràcies a diferents plugins disponibles. Inicialment, Eclipse era un projecte de la companyia IBM, fins que el 2001 es va formar un consorci per tal de continuar el seu desenvolupament com a codi obert.

Eclipse es basa en plugins per tal d'oferir totes les funcionalitats que no integra per defecte. Entre les funcionalitats extres hi trobem suport per a altres tipus de sistemes de control de versions a part de CVS, com SVN, Git, Mercurial, etc. Com abans s'ha dit, el suport per a altres llenguatges de programació també ve donat per plugins. També trobem plugins per a dissenyar diagrames UML de tot tipus, com de classes, de seqüència, de casos d'ús, etc. Per al desenvolupament d'interfícies gràfiques d'usuari, Eclipse utilitzava fins al 2011 l'eina SWT, sigles de Standard Widget Toolkit. A partir de 2012, amb la versió 4.2, Eclipse passa a utilitzar XWT, sigles de XML Window Toolkit.

4.7. Maven

Apache Maven és una eina per a la gestió de projectes de software i per a l'automatització dels diferents processos que es realitzen durant el desenvolupament de software com: compilació, generació i empaquetat de codi binari, execució de tests, etc. Maven s'utilitza habitualment per a projectes escrits en llenguatge Java, però també suporta altres llenguatges com C#, Ruby, etc.

Maven es descarrega dinàmicament les llibreries Java i els plugins d'un o més repositoris. Maven disposa del repositori central Maven 2, però també hi ha altres repositoris creats per tercers que es poden afegir a la llista local de repositoris. En la còpia local que cada usuari té d'aquest repositori també s'hi poden afegir artefactes creats per projectes locals.

Maven utilitza un fitxer en format XML per tal de descriure el projecte que s'està desenvolupant, així com les seves dependències d'altres mòduls o components externs, l'ordre en que s'ha de compilar i empaquetar tot el sistema, directoris necessaris i plugins o afegits necessaris, com per exemple una llibreria externa, per tal dur a terme la compilació del projecte.

A continuació mostrem un fitxer pom.xml utilitzat en aquest projecte final de carrera. S'hi pot observar la definició de l'identificador pel projecte, l'artefacte de compilació, la manera d'empaquetar-lo, així com les dependències d'altres artefactes com la llibreria

PlayN, la llibreria TriplePlay i la llibreria JBox2D, i la descripció dels plugins necessaris i els directoris.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.
0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>pfc</groupId>
    <artifactId>pfcjoc</artifactId>
    <version>1.0</version>
  </parent>

  <artifactId>pfcjoc-core</artifactId>
  <packaging>jar</packaging>
  <name>pfcjoc Core</name>

  <dependencies>
    <dependency>
      <groupId>com.googlecode.playn</groupId>
      <artifactId>playn-core</artifactId>
      <version>${playn.version}</version>
    </dependency>
    <dependency>
        <groupId>com.googlecode.playn</groupId>
        <artifactId>playn-jbox2d</artifactId>
        <version>${playn.version}</version>
    </dependency>
    <dependency>
        <groupId>com.threerings</groupId>
        <artifactId>tripleplay</artifactId>
        <version>1.2</version>
    </dependency>
  </dependencies>
```

```
<build>
    <plugins>
      <plugin>

<groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-source-
plugin</artifactId>
      </plugin>
    </plugins>

    <resources>
      <!-- include the source files in our
main jar for use by GWT -->
      <resource>

<directory>${project.build.sourceDirectory}
</directory>
      </resource>
      <!-- and continue to include our
standard resources -->
      <resource>

<directory>${basedir}/src/main/resources</d
irectory>
      </resource>
    </resources>
  </build>
</project>
```

Figura 18: Fitxer pom.xml

Els projectes definits amb Maven es basen en el concepte de Project Object Model, POM, que proporciona tota la configuració per a un únic projecte. La configuració general d'aquest inclou el nom del projecte, el seu propietari i la dependència respecte altres projectes. També es poden configurar les fases de la preparació del codi final, com per exemple especificar la versió del compilador de Java. Per a projectes gran s'utilitzen més d'un fitxer POM representant varis mòduls o subprojectes. Es pot definir un fitxer POM arrel que permeti compilar tots els altres submòduls amb una sola instrucció. Els fitxers POM poden heretar configuracions d'altres fitxers POM, i tots els fitxers POM hereten del fitxer Super POM per defecte, el qual proporciona configuracions per defecte per coses com directoris de codi fons, plugins per defecte, etc.

Finalment, un apartat interessant de Maven és la quantitat de plugins que s'han escrit per a diferents entorns integrats de desenvolupament, IDE per les seves sigles en anglès, de manera que des d'aquests entorns es poden compilar i executar projectes Maven fàcilment. Existeixen plugins per a Eclipse, NetBenas, Jbuilder, IntelliJ IDEA i altres. Aquests també permeten editar els fitxers POM directament i fer-los servir per a determinar el conjunt de dependències d'un projecte dins el mateix entorn de programació.

4.8. Git

Git és un sistema de control de versions de software distribuït en el que cada usuari disposa en el seu directori de treball d'un repositori complet, és a dir, amb l'historial complet de versions disponible sense necessitar d'una connexió a Internet. Git va ser dissenyat per Linus Torvals per al desenvolupament del kernel de Linux. La principal característica de Git és el seu rendiment en projectes de software molt gran amb una gran quantitat de fitxers de codi font i gran quantitat de desenvolupadors treballant en diferents branques i unint-les després amb altres branques d'altres desenvolupadors.

Git és software lliure i en l'actualitat hi ha una gran quantitat de projectes de software que l'utilitzen, com es demostra per la quantitat de desenvolupadors que utilitzen llocs web de hosting de projectes com GitHub, Google Code, BitBucket, SourceForge i altres que suporten repositoris de software de Git.

4.9. JSON

JSON, JavaScript Object Notation és un estàndard obert dissenyat per a facilitar la creació de fitxers de text llegibles a simple vista per les persones. JSON deriva del llenguatge JavaScript per tal de representar estructures de dades simples i arrays associatius o objectes. JSON s'utilitza típicament per a la transmissió d'informació entre un servidor i una aplicació web, de forma alternativa al format XML.

```
{
    "nivell": 1,
    "player": {
        "x": 192,
        "y": 552
    },
    "WordlElements": [
        {
            "type": "groundblock",
            "x": 16,
            "y": 584,
            "depth": 0
        },
        {
            "type": "groundblock",
            "x": 48,
            "y": 584,
            "depth": 0
        },
        {
            "type": "groundblock",
            "x": 80,
            "y": 584,
            "depth": 0
        },
        {
            "type": "block",
            "x": 2176,
            "y": 248,
            "depth": 10
        }
    ]
}
```

Figura 19: Fitxer level.json

En aquest projecte de final de carrera s'han utilitzat fitxers JSON per a la descripció dels nivells del videojoc. La estructura dels nivells consta d'un objecte amb el nom i nombre del nivell, un objecte amb posició inicial del jugador i finalment un objecte amb tots els elements del nivell, el qual és un array format per tots els objectes dins el nivell, on cada objecte conté un tipus, la seva posició inicial i el seu nivell de profunditat dins l'escenari 2D.

Els tipus bàsics de dades que suporta JSON són: número, String, Boolean, Array, Object i null. Els navegadors web fa temps que ofereixen funcionalitats per tal de codificar i descodificar fitxer .json per tal que els desenvolupadors web puguin utilitzar-los ràpidament i de forma segura.

5. Desenvolupament

En aquest apartat explicarem el procés que s'ha seguit per al desenvolupament d'aquest projecte. Primer hi haurà l'anàlisi de requisits, després la especificació del sistema i finalment el disseny del videojoc i l'editor de nivells.

5.1. Anàlisis de requisits

En aquest projecte de final de carrera els requisits s'han obtingut a partir de l'ús d'altres videojocs semblants i per mitjà de la experimentació mentre es desenvolupava el videojoc i l'editor de nivells. Tenim dos tipus de requisits: els funcionals i els no funcionals. Els funcionals determinen quines seran les funcionalitats ofertes per l'aplicació, i els no funcionals determinen qualitats generals que ha de tenir el sistema quan realitza la funció per la qual ha estat desenvolupat.

5.1.1. Requisits funcionals

Descrivim a continuació els requisits funcionals que s'han determinat per a aquest projecte de final de carrera:

- **Jugabilitat**: el jugador ha de tenir la possibilitat de poder desplaçar el personatge per tota la escena i tenir la sensació de llibertat, podent anar endavant i endarrere en el nivell. També ha de poder defensar-se dels enemics que li sortiran al pas durant el transcurs del videojoc pel nivell.
- **Modificar el nivell**: el jugador podrà entrar a l'editor de nivells i modificar el nivell al seu gust, col·locant els diferents blocs i decoracions on desitgi. Hi haurà diferents tipus de blocs, mòbils, fixos i decoratius. L'usuari també podrà col·locar un nombre indeterminat d'enemics on vulgui. També podrà eliminar

elements del nivell. Finalment, també és permetrà moure al jugador, el que significa moure la posició inicial del personatge.

- **Guardar el nivell**: el jugador podrà guardar les modificacions realitzades al nivell per tal de poder jugar en el nou nivell, tant immediatament com en una altra ocasió gràcies a l'emmagatzemament a disc en format JSON.

5.1.2. Requisits no funcionals

Descrivim a continuació els requisits no funcionals que s'han determinat per a aquest projecte de final de carrera:

- **Eficiència**: o rendiment de l'aplicació. És desitjable que el videojoc funcioni de manera fluida, a gran velocitat d'imatges per segon, per tal que l'usuari tingui una bona experiència del videojoc. També l'editor ha de ser ràpid al moure i mostrar els elements dels nivells, i realitzar el guardat del nivell a disc en el menor temps possible.
- **Portabilitat**: el codi d'aquest projecte de final de carrera s'ha escrit en Java, que permet ser executat en qualsevol plataforma per a la qual s'hagi desenvolupat una màquina virtual de Java. A més, com s'ha explicat, la llibreria PlayN permet que aquest codi Java es compili a diferents plataformes, sent HTML5 la que ens interessa en aquest projecte i que com s'ha comentat anteriorment es pot executar en qualsevol navegador web que sigui relativament modern.
- **Facilitat de manteniment**: en el desenvolupament d'aquest projecte de final de carrera s'han utilitzar alguns patrons de disseny de software per tal de facilitar l'ampliació futura del sistema si es requerís.
- **Usabilitat**: el videojoc disposa en el menú principal d'una petita guia de controls per tal que l'usuari sàpiga com jugar tot just començar. També s'explica breument com fer anar l'editor de nivells, que per altra banda resulta fàcil d'utilitzar al disposar d'una interfície gràfica semblant a altres editors de nivells amb una sèrie de botons i icones que es reconeixen fàcilment.

5.2. Especificació

En aquest apartat farem una descripció del comportament extern del sistema, és a dir, des del punt de vista de l'usuari.

5.2.1. Model de casos d'ús

Amb el model de casos d'ús descriurem els serveis del sistema com a seqüències d'esdeveniments que realitzen actors, els qual fan servir el sistema per extreure'n algun valor per mitjà d'un procés.

5.2.1.1. Actors del sistema

Els actors són les entitats externes al sistema que participen en algun escenari d'un o més dels seus casos d'ús. En el sistema dissenyat per aquest projecte de final de carrera podem distingir dos actors:

- **Jugador**: és l'actor que intervindrà en els casos d'ús relacionats amb el videojoc en sí mateix, i que per contra no ho farà amb l'editor de nivells del sistema.
- **Dissenyador de nivells**: és l'actor que intervindrà en pràcticament tots els casos d'ús del sistema, ja siguin els relacionats amb l'editor de nivells com els relacionats amb el videojoc, ja que necessitarà provar les seves creacions abans que el jugador hi pugui jugar.

5.2.1.2. Diagrama de casos d'ús

Figura 20: Diagrama de casos d'ús

5.2.1.3. Especificació de casos d'ús

En aquest apartat es detallen els casos d'ús formalment i completament, que representen les interaccions dels usuaris amb el sistema.

5.2.1.3.1. Iniciar joc

- **Actors principals**: Jugador, Dissenyador de nivells
- **Stakeholders i interessos**:
    - **Jugador**: vol iniciar una partida per passar una bona estona, per entretenir-se.
    - **Dissenyador de nivells**: vol iniciar una partida per provar un nivell que acaba de crear o modificar.
- **Precondició**: el joc es troba en el menú principal.
- **Activació**: el jugador o el dissenyador de nivells selecciona la opció d'iniciar el joc del menú principal.
- **Escenari principal d'èxit**:
    1. El jugador o el dissenyador de nivells selecciona la opció d'iniciar el joc.
    2. El sistema inicialitza el joc i comença a carregar el nivell, amb tots els seus elements, enemics, etc.
    3. El sistema acaba la càrrega del joc i del nivell.
    4. L'usuari té el control del personatge i pot començar a jugar.
- **Escenaris alternatius**:
    1. El jugador o el dissenyador de nivells selecciona la opció d'iniciar el joc.
    2. El sistema inicialitza el joc i comença a carregar el nivell, amb tots els seus elements, enemics, etc.
    3. El sistema no pot acabar de carregar el nivell, ja sigui per la falta d'algun recurs o per algun error no determinat com problemes amb la connexió a internet de l'usuari.
    4. El sistema mostra un missatge d'error a l'usuari.
    5. L'usuari recarrega el joc amb el seu navegador web i torna a provar-ho.
- **Requisits especials**: la càrrega del nivell s'ha realitzar en un temps no molt llarg per tal que l'usuari no es cansi d'esperar.
- **Tecnologia i dades**: els nivells es guarden en fitxers en format JSON, i són fàcilment editables sense la necessitat d'obrir l'editor de nivells. L'usuari tria la opció d'iniciar el joc amb el mouse.
- **Freqüència d'ocurrència**: quan l'usuari vulgui jugar o el dissenyador de nivells provar un nou nivell o alguna modificació.

5.2.1.3.2. Fer una pausa

- **Actors principals**: Jugador, Dissenyador de nivells

- **Stakeholders i interessos**:
    - **Jugador**: vol interrompre la partida una estona, pel motiu que sigui, però sense abandonar-la del tot, per tornar a jugar-hi després.
    - **Dissenyador de nivells**: vol interrompre la partida una estona, possiblement per anotar coses que després canviarà a l'editor de nivells, o per observar amb deteniment la distribució del nivell.
- **Precondició**: el joc està dins una partida.
- **Activació**: el jugador o el dissenyador de nivells prem la tecla corresponent per posar el joc en pausa.
- **Escenari principal d'èxit**:
    1. El jugador o el dissenyador de nivells prem la tecla corresponent per tal de posar el joc en pausa.
    2. El sistema deixa de reaccionar a les tecles de moviment del jugador i també deixa d'actualitzar les posicions de tots els elements de la escena.
    3. El sistema mostra al jugador o al dissenyador de nivells un missatge indicant que el joc està en pausa i que pot reprendre el joc en qualsevol moment.
- **Escenaris alternatius**: no n'hi ha.
- **Requisits especials**: la interrupció del joc ha de realitzar-se immediatament després que l'usuari premi la tecla de pausa, sense cap tipus de retard.
- **Tecnologia i dades**: el sistema manté en memòria totes les estructures necessàries per tal que l'usuari pugui reprendre la partida en el moment que ho desitgi.
- **Freqüència d'ocurrència**: quan el jugador o el dissenyador de nivells vulgui interrompre momentàniament la partida.

5.2.1.3.3. Reprendre el joc

- **Actors principals**: Jugador, Dissenyador de nivells
- **Stakeholders i interessos**:
    - **Jugador**: vol continuar jugant la partida que havia interromput prèviament.
    - **Dissenyador de nivells**: vol continuar la partida que amb anterioritat havia interromput.
- **Precondició**: el joc es troba en pausa.

- **Activació**: el jugador o el dissenyador de nivells prem la tecla corresponent per reprendre la partida, que és la mateixa tecla que havia premut per posar el joc en pausa.
- **Escenari principal d'èxit**:
    1. El jugador o el dissenyador de nivells prem la tecla corresponent per tal de reprendre la partida.
    2. El sistema torna a reaccionar a les tecles de moviment del jugador i reprèn les actualitzacions de posicions de tots els elements de la escena.
    3. El sistema deixa de mostrar el missatge que indica al jugador o al dissenyador de nivells que el joc està en pausa .
    4. La partida es reprèn, l'usuari recupera el control del jugador i pot tornar a jugar.
- **Escenaris alternatius**: no n'hi ha.
- **Requisits especials**: la represa de la partida s'ha de realitzar immediatament després que l'usuari premi la tecla de pausa, sense cap tipus de retard.
- **Tecnologia i dades**: el sistema manté en memòria totes les estructures necessàries per tal que l'usuari pugui reprendre la partida en el moment que ho desitgi.
- **Freqüència d'ocurrència**: quan el jugador o el dissenyador de nivells vulgui reprendre la partida que havia interromput prèviament.

5.2.1.3.4. Finalitzar joc

- **Actors principals**: Jugador, Dissenyador de nivells
- **Stakeholders i interessos**:
    - **Jugador**: vol donar per acabada la partida.
    - **Dissenyador de nivells**: vol donar per acabada la partida.
- **Precondició**: el joc està dins una partida.
- **Activació**: el jugador o el dissenyador de nivell prem la tecla corresponent per tal que el joc abandoni la partida actual i torni al menú principal. Una altra possible activació del cas d'ús seria que l'usuari navegués a una altra pàgina web, però aquest cas queda fora del sistema ja que se n'ocuparia el navegador web.
- **Escenari principal d'èxit**:
    1. El jugador o el dissenyador de nivells prem la tecla corresponent per tal que el joc abandoni la partida actual i torni al menú principal.

2. El sistema deixa de mostrar la pantalla de joc.
3. El sistema deixa de respondre a les tecles de moviment del jugador.
4. El sistema carrega el menú principal i el mostra al jugador o al dissenyador de nivells.

- **Escenaris alternatius**: no n'hi ha.
- **Requisits especials**: l'aturada de la partida i la tornada al menú principal s'ha de realitzar immediatament després que l'usuari premi la tecla de sortida corresponent, sense cap tipus de retard.
- **Tecnologia i dades**: el sistema manté en memòria les estructures de dades corresponents al menú principal per tal que no s'hagi de tornar a carregar i es pugui mostrar el més ràpidament possible.
- **Freqüència d'ocurrència**: quan el jugador o el dissenyador de nivells vulgui deixar la partida per tal de tornar al menú principal.

5.2.1.3.5. Iniciar editor de nivells

- **Actors principals**: Dissenyador de nivells
- **Stakeholders i interessos**:
  - **Dissenyador de nivells**: vol iniciar l'editor de nivells per tal de dissenyar un nivell.
- **Precondició**: el joc es troba en el menú principal
- **Activació**: el dissenyador de nivells selecciona la opció d'iniciar l'editor de nivells del menú principal.
- **Escenari principal d'èxit**:
  1. El dissenyador de nivells selecciona la opció d'iniciar l'editor de nivells.
  2. El sistema inicialitza l'editor de nivells i comença a carregar el nivell, amb tots els seus elements, enemics, etc.
  3. El sistema acaba la càrrega del joc i del nivell.
  4. El sistema carrega la interfície de l'editor de nivells, amb els seus corresponents widgets.
  5. El dissenyador de nivells té el control i pot començar a treballar en les modificacions del nivell.
- **Escenaris alternatius**:
  1. El dissenyador de nivells selecciona la opció d'iniciar l'editor de nivells.

2. El sistema inicialitza l'editor de nivells i comença a carregar el nivell, amb tots els seus elements, enemics, etc.
3. El sistema no pot acabar de carregar el nivell o la interfície de l'editor ja sigui per la falta d'algun recurs o per algun error no determinat com problemes amb la connexió a internet de l'usuari.
4. El sistema mostra un missatge d'error a l'usuari.
5. L'usuari recarrega el joc amb el seu navegador web i torna a provar-ho, seleccionant de nou la opció d'entrar a l'editor de nivells.

- **Requisits especials**: la càrrega del nivell s'ha realitzar en un temps no molt llarg per tal que l'usuari no es cansi d'esperar i, en aquest cas, per tal que dissenyador de nivells pugui realitzar proves ràpidament amb les modificacions realitzades.
- **Tecnologia i dades**: els nivells es guarden en fitxers en format JSON, i són fàcilment editables sense la necessitat d'obrir l'editor de nivells. L'usuari tria la opció d'iniciar l'editor de nivells amb el mouse
- **Freqüència d'ocurrència**: quan el dissenyador de nivells necessiti entrar a l'editor de nivells per treballar-hi.

5.2.1.3.6. Afegir elements

- **Actors principals**: Dissenyador de nivells
- **Stakeholders i interessos**:
  - **Dissenyador de nivells**: vol afegir un nou element al nivell, ja sigui un bloc, un enemic o qualsevol altre element disponible.
- **Precondició**: el joc es troba dins l'editor de nivells.
- **Activació**: el dissenyador de nivells selecciona la opció d'afegir un element nou al panell de l'editor de nivells.
- **Escenari principal d'èxit**:
  1. El dissenyador de nivells selecciona la opció d'afegir un element nou al panell de l'editor de nivells.
  2. El sistema mostra al costat del cursor una representació transparent de l'element que es vol afegir per tal que el dissenyador de nivells pugui situar-lo amb facilitat.
  3. El dissenyador de nivells situa l'element a la posició desitjada per mitjà de moviment del cursor i prem el botó principal del mouse per a fixar la posició de l'element.

4. El sistema situa definitivament l'element a la posició triada pel dissenyador de nivells.
5. El sistema deixa de mostrar la representació transparent de l'element que es vol afegir al costat del cursor.

- **Escenaris alternatius**: no n'hi ha.
- **Requisits especials**: el moviment de la representació de l'element que segueix al cursor s'ha moure a la velocitat del cursor, sense mostrar cap tipus de retard, per tal de donar una bona experiència a l'usuari.
- **Tecnologia i dades**: el sistema restringeix les possibles posicions on es poden col·locar els elements a una quadrícula virtual per tal que sigui més fàcil apilar una sèrie d'elements formant una torre, el terra, i altres tipus de construccions amb múltiples elements.
- **Freqüència d'ocurrència**: quan el dissenyador de nivells vol afegir un nou element al nivell que està modificant.

5.2.1.3.7. Moure elements

- **Actors principals**: Dissenyador de nivells
- **Stakeholders i interessos**:
  - **Dissenyador de nivells**: vol moure, canviar de posició, un dels elements que estant actualment en el nivell.
- **Precondició**: el joc es troba dins l'editor de nivells.
- **Activació**: el dissenyador de nivells selecciona la opció de moure un element al panell de l'editor de nivells.
- **Escenari principal d'èxit**:
  1. El dissenyador de nivells selecciona la opció de moure un element al panell de l'editor de nivells.
  2. El dissenyador de nivells situa el cursor sobre l'element del qual en vol canviar la posició dins el nivell i prem el botó principal del mouse per a seleccionar l'element.
  3. El dissenyador de nivells, sense deixar anar el botó principal del mouse, comença a desplaçar el cursor fins a la nova posició desitjada per a l'element seleccionat.
  4. El sistema mostra una representació transparent de l'element que està seleccionat mentre aquest es desplaça.

5. El dissenyador de nivells deixa de moure el cursor i deixa de prémer el botó principal del mouse.
6. El sistema situa definitivament l'element a la nova posició triada pel dissenyador de nivells.
7. El sistema deixa de mostrar la representació transparent de l'element que es vol afegir al costat del cursor.

- **Escenaris alternatius**: no n'hi ha.
- **Requisits especials**: el moviment de la representació de l'element que segueix al cursor s'ha moure a la velocitat del cursor, sense mostrar cap tipus de retard, per tal de donar una bona experiència a l'usuari.
- **Tecnologia i dades**: el sistema restringeix les possibles posicions on es poden col·locar els elements a una quadrícula virtual per tal que sigui més fàcil apilar una sèrie d'elements formant una torre, el terra, i altres tipus de construccions amb múltiples elements.
- **Freqüència d'ocurrència**: quan el dissenyador de nivells vol traslladar un element a una nova posició dins el nivell que està modificant.

5.2.1.3.8. Eliminar elements

- **Actors principals**: Dissenyador de nivells
- **Stakeholders i interessos**:
  - **Dissenyador de nivells**: vol eliminar un dels elements presents al nivell, ja sigui un bloc, un enemic o qualsevol altre element disponible.
- **Precondició**: el joc es troba dins l'editor de nivells i l'element que s'intenta eliminar no és el jugador.
- **Activació**: el dissenyador de nivells selecciona la opció d'eliminar un element al panell de l'editor de nivells.
- **Escenari principal d'èxit**:
  1. El dissenyador de nivells selecciona la opció d'eliminar un element al panell de l'editor de nivells.
  2. El dissenyador de nivells desplaça el cursor fins a la posició de l'element que vol eliminar.
  3. El sistema mostra l'element sota el cursor com a una representació transparent per tal que el dissenyador de nivells pugui saber fàcilment quin element senyalant.

4. El dissenyador de nivells prem el botó principal del mouse per eliminar l'element marcat.
5. El sistema elimina l'element de la escena i de les estructures de dades del nivell.

- **Escenaris alternatius**:
    1. El dissenyador de nivells selecciona la opció d'eliminar un element al panell de l'editor de nivells.
    2. El dissenyador de nivells desplaça el cursor fins a la posició del personatge del joc.
    3. El sistema mostra l'element sota el cursor com a una representació transparent per tal que el dissenyador de nivells pugui saber fàcilment quin element està senyalant.
    4. El dissenyador de nivells prem el botó principal del mouse per eliminar el personatge del joc.
    5. El sistema no elimina el personatge del nivell i manté l'estat del nivell.
- **Requisits especials**: la desaparició de l'element seleccionat un cop l'usuari prem el botó principal del mouse ha de ser pràcticament immediata, sense que l'usuari noti cap retard.
- **Tecnologia i dades**: cap comentari.
- **Freqüència d'ocurrència**: quan el dissenyador de nivells vulgui eliminar un element del nivell.

5.2.1.3.9. Desplaçar vista del nivell

- **Actors principals**: Dissenyador de nivells
- **Stakeholders i interessos**:
    - **Dissenyador de nivells**: vol desplaçar la vista del nivell per tal de poder manipular el nivell fora els límits de la finestra del joc.
- **Precondició**: el joc es troba dins l'editor de nivells.
- **Activació**: el dissenyador de nivells desplaçar la roda del mouse.
- **Escenari principal d'èxit**:
    1. El dissenyador de nivells desplaça la roda del mouse.
    2. El sistema desplaça la vista del nivell, com a representació de càmera que enfoca la escena, permeten al dissenyador de nivells visualitzar la totalitat del nivell que no hi cap dins la finestra del joc.

- **Escenaris alternatius**:
    1. El dissenyador de nivells desplaça la roda del mouse.
    2. El sistema ha arribat al límit del nivell per la esquerra i, per tant, deixa de desplaçar la vista del nivell.
- **Requisits especials**: el desplaçament de la vista del nivell s'ha de realitzar de manera fluida i suau, sense que es noti cap aturada, per tal de donar una bona experiència d'usuari.
- **Tecnologia i dades**: el desplaçament de la vista té un límit per la esquerra, però no per la dreta, i, per tant, el nivell pot créixer gairebé indefinidament.
- **Freqüència d'ocurrència**: quan el dissenyador de nivells vulgui veure una part del nivell que amb la vista actual queda fora dels límits de la finestra.

5.2.1.3.10. Guardar nivell

- **Actors principals**: Dissenyador de nivells
- **Stakeholders i interessos**:
    - **Dissenyador de nivells**: vol guardar el progrés realitzat en una sessió de treball de modificació del nivell.
- **Precondició**: el joc es troba dins l'editor de nivells.
- **Activació**: el dissenyador de nivells selecciona la opció de guardar el nivell a disc al panell de l'editor de nivells.
- **Escenari principal d'èxit**:
    1. El dissenyador de nivells selecciona la opció de guardar el nivell a disc al panell de l'editor de nivells.
    2. El sistema comença a recórrer tots els elements que formen part del nivell.
    3. Per cada element del nivell el sistema emmagatzema les coordenades de la seva posició dins el nivell i la seva profunditat.
    4. Finalment, el sistema guarda en un fitxer la representació del nivell.
- **Escenaris alternatius**:
    1. El dissenyador de nivells selecciona la opció de guardar el nivell a disc al panell de l'editor de nivells.
    2. El sistema comença a recórrer tots els elements que formen part del nivell.

3. Per cada element del nivell el sistema emmagatzema les coordenades de la seva posició dins el nivell i la seva profunditat.
4. El sistema troba algun error en el moment de guardar a disc la representació del nivell.
5. El sistema mostra un missatge d'error a l'usuari.
6. L'usuari torna a intentar guardar el nivell a disc.

- **Requisits especials**: el procés de guardar la informació a disc s'ha de realitzar el més ràpidament possible, sense oblidar que en aquest cas la seguretat que les dades s'han escrit correctament té prioritat.
- **Tecnologia i dades**: els nivells es guarden en fitxers en format JSON, i són fàcilment editables sense la necessitat d'obrir l'editor de nivells. L'usuari tria la opció de guardar el nivell amb el mouse
- **Freqüència d'ocurrència**: quan el dissenyador de nivells vol guardar les modificacions realitzades al nivell.

5.2.1.3.11. Finalitzar editor de nivells

- **Actors principals**: Dissenyador de nivells
- **Stakeholders i interessos**:
    - **Dissenyador de nivells**: vol abandonar l'editor de nivells i tornar al menú principal per entrar el joc i provar els canvis realitzats, o per abandonar el sistema del tot.
- **Precondició**: el joc es troba dins l'editor de nivells.
- **Activació**: el dissenyador de nivells selecciona la opció de sortir de l'editor de nivells del panell d'opcions dins l'editor de nivells.
- **Escenari principal d'èxit**:
    1. El dissenyador de nivells selecciona la opció de sortir de l'editor de nivells del panell d'opcions.
    2. El sistema deixa de mostrar la escena dins l'editor de nivells.
    3. El sistema deixa de mostrar la interfície de l'editor de nivells.
    4. El sistema carrega el menú principal i el mostrar al dissenyador de nivells.
- **Escenaris alternatius**:
    1. El dissenyador de nivells guarda el canvis realitzats al nivell per mitjà del cas d'ús "Guardar nivell".

2. El dissenyador de nivells selecciona la opció de sortir de l'editor de nivells del panell d'opcions.
3. El sistema deixa de mostrar la escena dins l'editor de nivells.
4. El sistema deixa de mostrar la interfície de l'editor de nivells.
5. El sistema carrega el menú principal i el mostrar al dissenyador de nivells.

- **Requisits especials**: la sortida de l'editor de nivells i la tornada al menú principal s'ha de realitzar immediatament després que l'usuari seleccioni la opció de sortir de l'editor de nivells, sense cap tipus de retard.
- **Tecnologia i dades**: el sistema no guarda els canvis realitzats al nivell si el dissenyador de nivells no els guarda manualment amb el cas d'ús "Guardar nivell".
- **Freqüència d'ocurrència**: quan el dissenyador de nivells vulgui sortir de l'editor de nivells per tal de tornar al menú principal.

5.2.1.3.12. Mostrar wireframe

- **Actors principals**: Dissenyador de nivells
- **Stakeholders i interessos**:
  - **Dissenyador de nivells**: vol veure la representació a baix nivell, en forma de vectors, dels elements del nivell.
- **Precondició**: el joc es troba en el menú principal.
- **Activació**: el dissenyador de nivells selecciona la opció de mostrar la representació a baix nivell dels elements de la escena al panell de l'editor de nivells.
- **Escenari principal d'èxit**:
  1. El dissenyador de nivells selecciona la opció de mostrar la representació a baix nivell dels elements de la escena al panell de l'editor de nivells.
  2. El sistema superposa sobre els elements del nivell una capa amb transparència on es veu la representació dels elements com a vectors.
- **Escenaris alternatius**: no n'hi ha.
- **Requisits especials**: el sistema ha de mostrar la capa d'informació de baix nivell el més ràpidament possible.

- **Tecnologia i dades**: el sistema utilitza una capa gràfica apart per cada un dels elements del nivell. Quan l'usuari retiri la visualització del wireframe el sistema eliminarà aquestes capes extra.
- **Freqüència d'ocurrència**: quan el dissenyador de nivells vulgui veure el detall de la implementació dels elements en forma vectorial.

5.3. Disseny

Per al disseny d'aquest projecte de final de carrera s'han aplicat alguns patrons de disseny de software com, per exemple, el patró expert. Aquest patró de disseny especifica que les diferents operacions o mètodes s'han d'assignar a les classes que tinguin una relació més propera amb l'àmbit dels mètodes. És, per tant, un patró que afectarà a la totalitat de les classes del sistema, i que farà més entenedor el codi i més fàcil les modificacions posteriors d'aquest. Per exemple, la classe Element conté funcions com getBody() que retorna el body de Box2D de l'element actual per que una altra classe de nivell superior el pugui modificar. No tindria sentit que la funció getBody() estigués en una altra classe ja que la classe Element és la que conté l'atribut body i és, per tant, la classe experta en aquest àmbit. Altres exemples són les funcions getHeight() i getWidth() de la classe Element, per les quals podem seguir un raonament semblant a l'anterior, ja que els atributs height i width són a la classe Element i són únics per cada element del nivell.

Un altre patró utilitzat en el disseny ha estat el patró Estat. Per a un videojoc és important tenir diferents estats que representin diferents coses dins el joc, com modes de joc, finestres d'opcions, etc. En aquest projecte els estats són únicament el modes de menú principal, editor de nivells i el mode de joc. El patró Estat ens permet que les subclasses de GameStatem, MenuGameState, EditGameState i PlayGameState comparteixin una mateixa interfície de cara a l'exterior, però realitzant tasques totalment diferents en els tres modes. D'aquesta manera, la classe GameLogic pot tractar amb estats genèrics, sense haver de conèixer les estructures internes de cada un d'ells. A més, el canvi d'estat es realitza fent simplement una nova assignació a l'atribut de GameLogic currentGameState.

L'altre patró de disseny utilitzat en aquest projecte de final de carrera ha estat el patró Singleton aplicat a cada un dels estats del videojoc. S'ha triat aquest opció per evitar que es puguin duplicar estats i per tal que l'accés a aquests estats per part de les altres classes fos més senzill i des d'un punt d'accés sempre igual: el mètode getInstance(). En concret, cada un dels estats ha de poder accedir als altres estats per tal de demanar el canvi d'estat a la classe GameLogic.

Finalment, és important comentar que no s'ha utilitzat el patró Model-Vista-Controlador ja que en un videojoc la part més important és la vista, i aquesta i la part de model estan molt estretament lligades, i afegir capes de codi entre les dues per tenir una millor organització del codi font ho faria difícil de gestió per obtenir uns beneficis mínims. A més, és probable que s'empitjorés el rendiment del sistema si no es realitzés correctament. El que acostumen a fer els dissenyadors de videojocs és separar el codi en components lògics com intel·ligència artificial, so, gràfics, etc.

5.3.1. Diagrama de classes general

A continuació mostrem una visió general de totes les classes que componen el sistema. Per qüestions de mida no és possible encabir totes les classes i els seus atributs en aquest diagrama general, i per tant és mostren únicament les classes i les relacions entre elles. En els següents apartats es detallarà el disseny per parts lògiques implementades i en l'apartat d'implementació és veuran les classes en més detall, incloent-hi els seus atributs així com una descripció d'aquests.

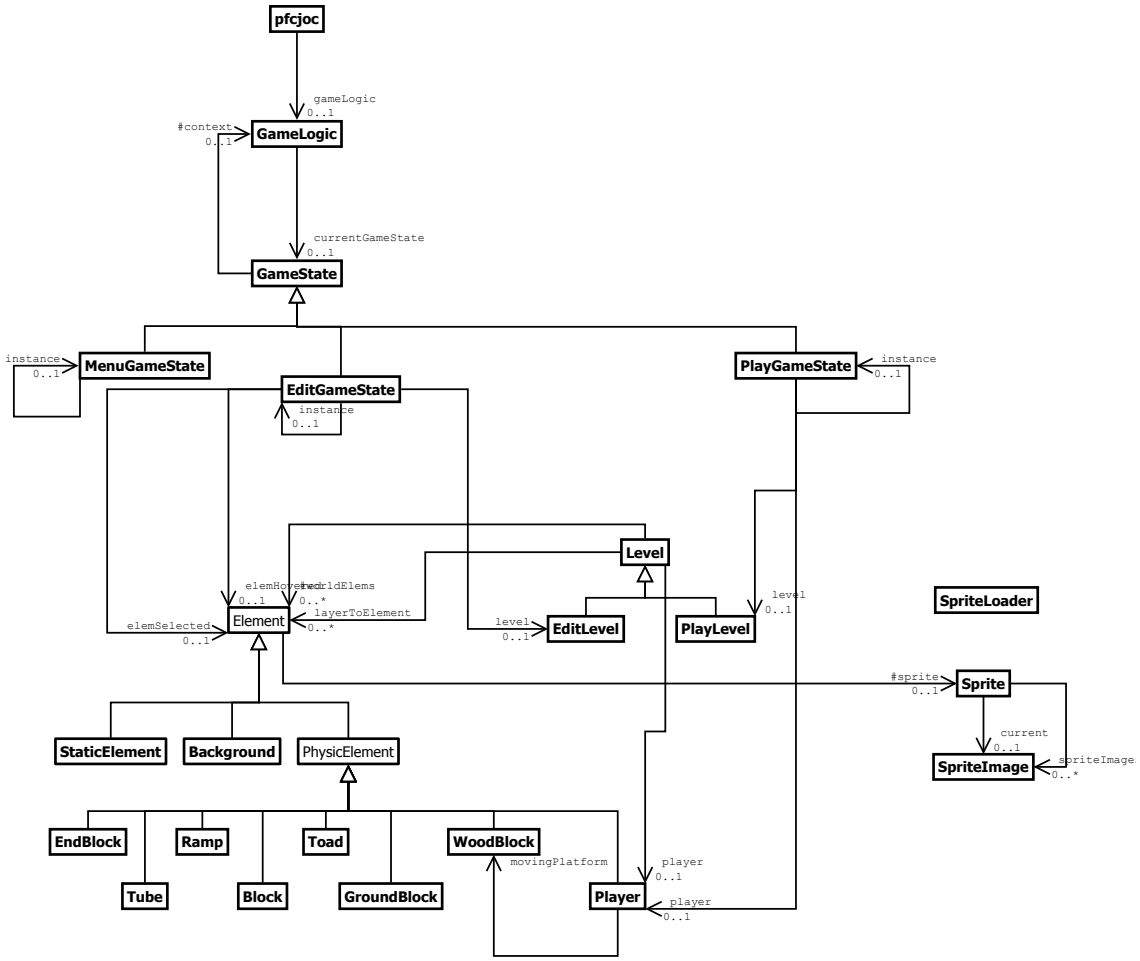


Figura 21: Diagrama de classes general

5.4. Implementació

En aquest apartat detallarem la implementació del PFC Joc. Abans de tot això, cal comentar alguns aspectes. Com ja hem comentat, estem fent servir directament dues llibreries: PlayN i Box2D. Cadascuna d'elles fa servir sistemes de coordenades diferents: el de les capes de PlayN, en unitats de nombre de píxels, i dels objectes físics del motor Box2D, en unitats de metres. Això ha provocat que s'hagin implementat unes funcions auxiliar per tal de convertir d'unes unitats a les altres, tant nombres com vectors de posicions. Hi havia algunes possibilitats per tal d'evitar aquestes constants conversions entre sistemes:

- **Passar a Box2D els nombres directament en unitats de píxels**: Això provoca que el World que el motor de física de Box2D ha de re-calcular a cada volta de

bucle del joc fos molt més gran del que és habitual i normal. Per exemple, objectes petits de 30x30 píxels tindrien una mida física de 30x30 metres. Com el motor Box2D no està optimitzar per a treballar amb magnituds tant grans [12] es va descartar aquesta solució.

- **Passar a les capes de PlayN els nombres directament en unitats físiques**: Això provoca que la finestra de joc tingui una mida molt petita, problema que es pot solucionar escalant aquesta un cop s'han acabat els càlculs a cada volta del bucle del joc. Aquesta solució es va considerar cap al final del desenvolupament del projecte, amb poc temps per a implementar-la i provar-la.

5.4.1. Diagrama de classes del nucli del programa

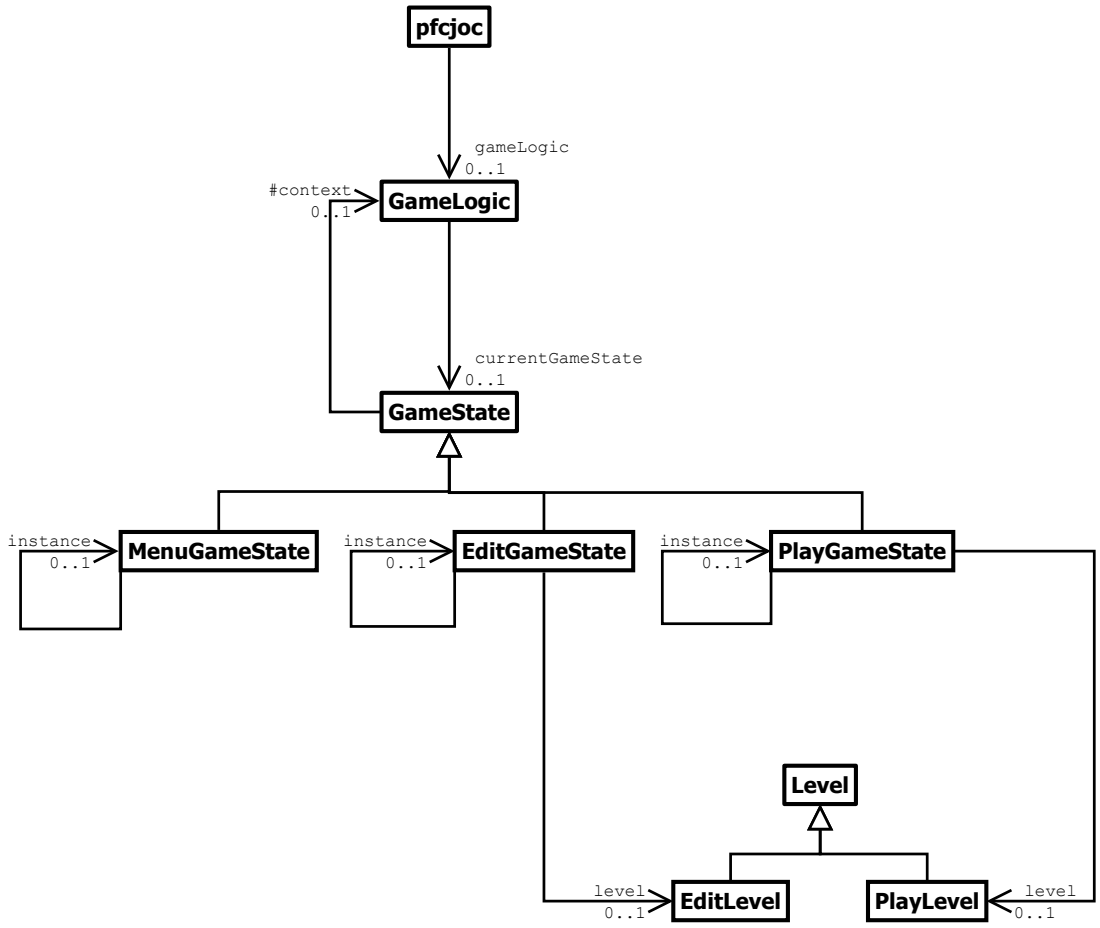


Figura 22: Diagrama de classes del nucli del joc

Mostrem aquí el diagrama de classes del nucli del programa, sense els atributs ni les operacions amb l'objectiu d'aconseguir una visió global del sistema que s'ha dissenyat. La classe pfcjoc és la encarregada de realitzar les inicialitzacions generals com la mida de la pantalla de joc i altres variables necessàries durant la execució de tot el joc. És la classe que és més a prop de la llibreria PlayN. De fet és la única classe que aquesta llibreria veu, únicament necessitant de la funció d'inicialització i dues funcions per actualitzar l'estat del joc i pintar a cada volta del bucle general. A partir d'aquí, la classe GameLogic és la encarregada de gestiona els canvis d'estat dins el programa. Aplicant el patró estat tenim una variable de tipus GameState que representa un estat genèric i a la qual podem assignar un estat de qualsevol subtipus. S'han desenvolupat tres estats diferents per a aquest videojoc: MenuGameState, EditGameState, PlayGameState. Representen, respectivament, el menú principal del joc, l'editor de nivells i el videojoc en sí o el mode de joc. Com marca el patró estat, des de cadascun d'aquests estats es pot demanar canviar a un altre estat per mitjà del context, una variable que permet accedir a la classe GameLogic. Més a baix en el diagrama de classe del nucli del programa trobem la classe Level, la qual representa un nivell del joc amb tots els seus elements. Les subclasses EditLevel i PlayLevel representen una petita diferència entre un nivell en el mode de joc o un nivell en l'editor de nivells.

5.4.1.1. Classe pfcjoc

La classe pfcjoc és la classe principal del joc, la que està per sobre de totes les demés. Aquesta classe és la que implementa la interfície Game de la llibreria PlayN, la qual s'encarregarà de cridar a cada volta del bucle del joc les funcions paint() i update(). També és la primera classe escrita pel desenvolupador en executar-se quan s'inicia el programa, ja que òbviament abans s'executa codi de la llibreria PlayN.

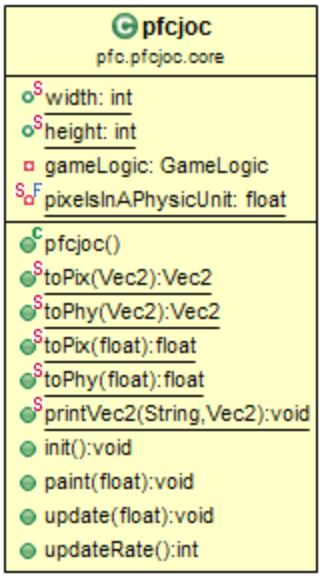


Figura 23: Classe pfcjoc

La classe pfcjoc s'encarrega de la inicialització de valors necessaris per a la resta del programa. Per exemple s'inicialitzen la mida de la finestra, o de la mida de l'element Canvas en el cas de HTML5, amb les variables width i height. També en aquesta classe s'inclouen funcions auxiliars per a realitzar les conversions d'unitats físiques a píxels i viceversa.

La classe pfcjoc conté els següents atributs:

- **width:** variable estàtica que conté el valor de l'amplada de la finestra del joc
- **height:** variable estàtica que conté el valor de l'alçada de la finestra del joc.
- **gameLogic:** conté la instància de la classe GameLogic que s'encarregarà de la lògica a alt nivell del joc, com la gestió dels estats dins el joc, i cridarà als mètodes per a pintar i actualitzar l'estat del joc corresponent a cada un dels estats en que es trobi el joc.
- **pixelsInAPhysicUnit:** conté el nombre de píxels que corresponen a una unitat física del motor de Box2D. El valor el determina el desenvolupador, i ve donat per la divisió de l'amplada/alçada de la finestra (en píxels) entre l'amplada/alçada del món de Box2D (en unitats físiques).

Algunes operacions importants de la classe pfcjoc:

- **toPix(float), toPhy(float), toPix(Vec2), toPhy(Vec2):** funcions auxiliars per realitzar la conversió entre unitats físiques i píxels, en versió float per valors simples i en versió Vec2 per a parells de valors, per exemple punts de coordenades.
- **init():** inicialitza el joc, configurant la mida de la finestra i delegant la resta de la inicialització a la classe GameLogic.
- **paint(float):** delega a la classe GameLogic la feina de pintar l'estat del joc.
- **update(float):** delega a la classe GameLogic la feina d'actualitzar l'estat del joc.
- **updateRate():** retorna en mill·isegons cada quan temps s'ha de cridar a la funció update().

5.4.1.2. Classe GameLogic

La classe GameLogic s'encarrega de la gestió de la lògica del joc a alt nivell, és a dir, gestionant els canvis d'estat del joc sense entrar en detalls de que succeeix en cada un d'aquests estats. Simplement rep les crides rebudes de la classe pfcjoc a les funcions paint() i update() i les delega a l'estat actual del joc. D'aquesta manera s'aconsegueix independència entre els estats i les seves implementacions. La classe GameLogic simplement veu un estat i ofereix una funció per facilitar el canvi d'estats.

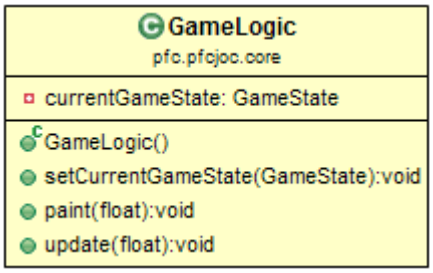


Figura 24: Classe GameLogic

La classe GameLogic conté els següents atributs:

- **currentGameState:** variable que conté la instància de la classe GameState corresponent a l'estat actual del joc.

Algunes operacions importants de la classe GameLogic:

- **setCurrentGameState(GameState):** s'encarrega de substituir l'estat actual per l'estat rebut com a paràmetre. Per a fer-ho avisa a l'estat actual que serà substituït

per tal que aquest pugui realitzar les tasques necessàries pendents, realitza al canvi, i aleshores avisa al nou estat per tal que aquest realitzi les inicialitzacions que consideri pertinents.

- **paint(float):** s'encarrega de delegar a l'estat actual la feina de pintar aquesta volta del bucle.
- **update(float):** s'encarrega de delegar a l'estat actual la feina d'actualitzar l'estat del joc aquesta volta del bucle.

5.4.1.3. Classe GameState

La classe GameState és una classe buida que representa un estat genèric del joc. No és abstracta per tal de poder tractar amb estats de joc genèrics com s'ha vist a la classe GameLogic. Si fos abstracta només es podria tractar amb estats concrets, els quals veurem més endavant. La principal funció de la classe GameState és la de declarar uns mètodes que s'implementaran en els estats de joc concrets i que es cridaran des de la classe GameLogic per tal de gestionar els canvis entre estats. Els estats concrets també afegiran altres mètodes per a realitzar funcionalitats específiques de l'estat.

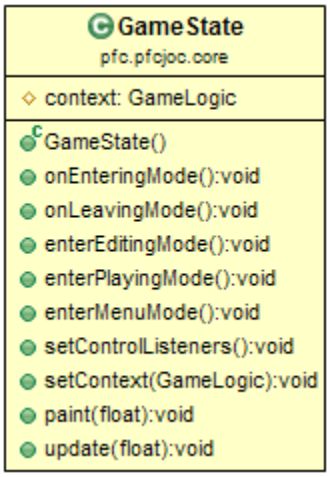


Figura 25: Classe GameState

La classe GameState conté els següents atributs:

- **context:** conté la instància de la classe GameLogic, que és el context dins el patró estat i que serveix principalment per permetre a un estat demanar un canvi d'estat.

Algunes operacions importants de la classe GameState:

- **onEnteringMode():** es crida cada cop que s'entra a l'estat actual, per tal de realitzar les inicialitzacions pertinents.
- **onLeavingMode():** es crida cada cop que se surt de l'estat actual, per tal de poder realitzar les feines de neteja o manteniment així com desfer coses fetes en el moment de la inicialització de l'estat per tal de no interferir amb el nou estat.
- **enterEditingMode():** demana que el joc passi a l'estat EditingMode.
- **enterPlayingMode():** demana que el joc passi a l'estat PlayingMode.
- **enterMenuMode():** demana que el joc passi a l'estat MenuMode.
- **setControlListeners():** funció que agrupa la configuració i inicialització dels mètodes d'entrada com teclat i ratolí. En cada un dels estats de joc concrets s'haurà d'especificar el comportament del joc en resposta a les entrades dels perifèrics d'entrada.
- **setContext(GameLogic):** assigna un nou context a l'estat actual per tal que aquest pugui demana canvis d'estat.
- **paint(float):** s'encarrega de pintar el joc en l'estat actual.
- **update(float):** s'encarrega d'actualitzar l'estat del joc actual.

5.4.1.4. Classe MenuGameState

La classe MenuGameState implementa el menú principal del videojoc. Per a definir els elements de la interfície s'ha utilitzat la llibreria TriplePlay, explicada anteriorment. El menú principal del joc conté un parell de botons, un per comença a jugar i l'altre per entrar a l'editor de nivells. També s'hi ha afegit unes etiquetes a mode de títol i d'ajuda mínima per començar a jugar.

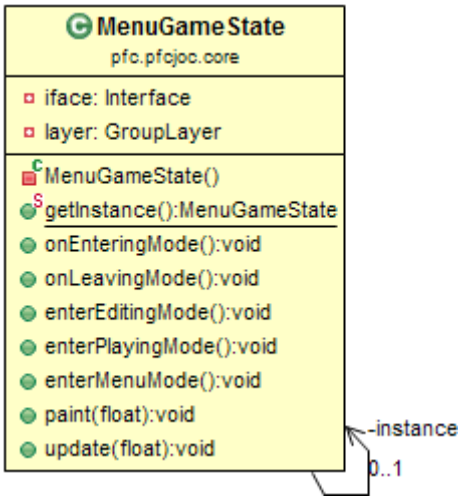


Figura 26: Classe MenuGateState

La classe MenuGameState conté els següents atributs:

- **iface**: variable de tipus Interface, el tipus principal de la llibreria TriplePlay, la qual conté l'arrel de la interfície, sobre la qual es realitzen les operacions necessàries per tal d'afegir i configurar els elements de la interfície gràfica del menú principal del joc.
- **layer**: variable corresponent a la capa lògica que contindrà les capes gràfiques amb tots els elements de la interfície generades a partir de l'arrel de la variable iface.

Les operacions implementades per la classe MenuGameState són únicament les definides per la súper classe GameState. Al tractar-se d'una classe relativament simple, en la qual només es defineixen uns elements gràfics i s'afegeixen a la interfície, no requereix cap tipus de mètode extra. La constructora privada i el mètode GetInstance corresponen al patró estat, pel qual cap classe externa pot crida a la constructora de la classe directament, sinó a través de la funció GetInstance(), la qual cridarà a la constructora si cal, i sinó simplement retornarà la classe inicialitzada anteriorment.

5.4.1.5. Classe EditGameState

La classe EditGameState implementa l'editor de nivells del videojoc. De manera similar a la classe MenuGameState, s'ha utilitzat la llibreria TriplePlay per a la implementació del menú de selecció d'opcions. Aquest menú ofereix a l'usuari seleccionar entre diverses funcionalitats com moure elements del nivell actual, eliminar elements del

nivell actual, afegir elements al nivell actual, guardar els canvis fets al nivell actual, activar i desactivar la visualització de debug i sortir al menú principal.

La implementació de l'editor de nivells consisteix en una màquina d'estats que depenent de si estem afegint elements nous, movent blocs existents, eliminant blocs existents o simplement movent el cursor per la pantalla reacciona de manera diferent als esdeveniments del mouse. D'aquesta manera, si no hi ha cap element seleccionat amb el moviment del mouse s'aniran destacant els elements sota la posició actual del mouse per donar a l'usuari la idea que amb aquest element pot interactuar. Aleshores l'usuari pot arrossegar aquest element a la posició que desitgi.

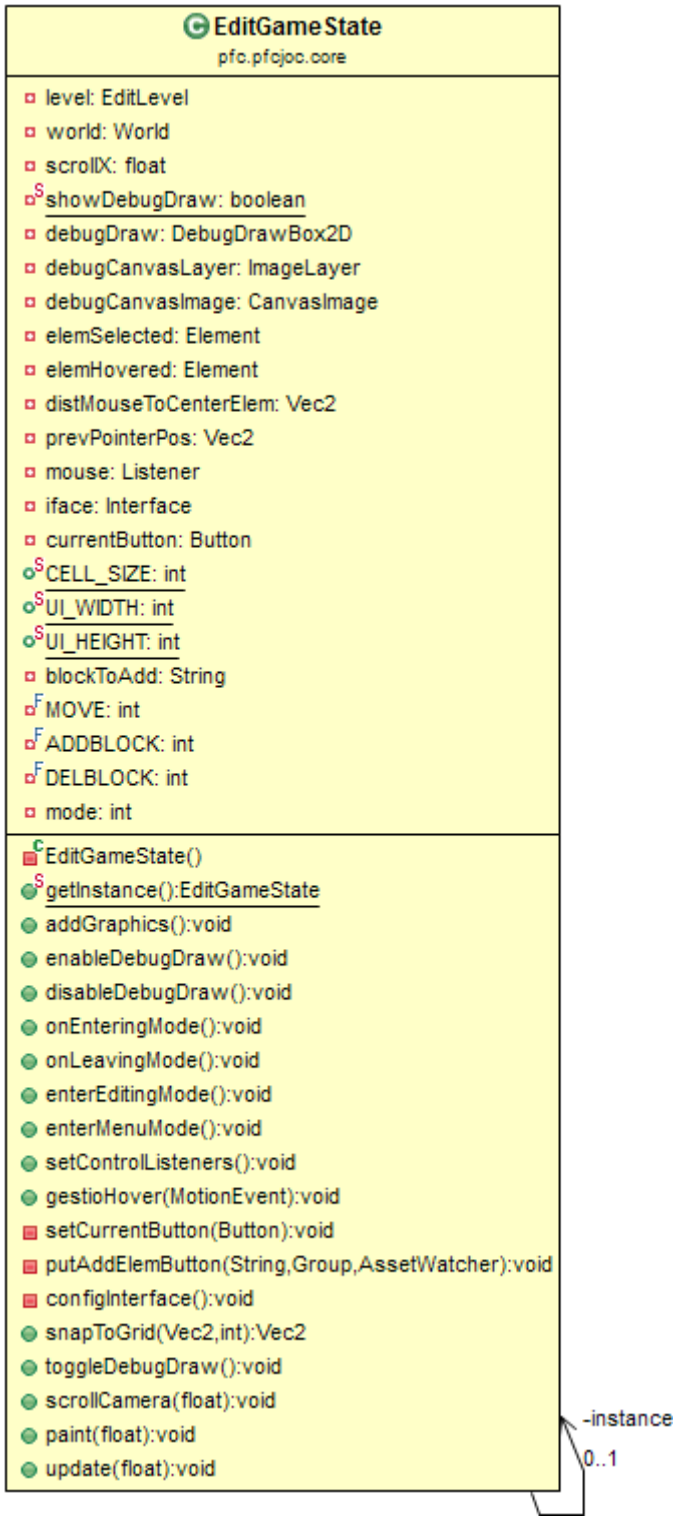


Figura 27: Classe EditGameState

La classe EditGameState afegeix els següents atributs respecte la classe GameState:

- **level**: aquesta variable conté un objecte de la classe EditLevel, que correspon a la representació del nivell amb tots els seus elements. Sobre aquesta variable es realitzen les modificacions quan l'usuari canvia alguna cosa del nivell.
- **world**: aquesta variable conté un objecte de la classe World de la llibreria Box2D. Aquesta classe World gestiona totes les entitats físiques, la simulació dinàmica i altres coses.
- **scrollX**: variable que va guardant el desplaçament de la pantalla degut a les accions de l'usuari quan utilitza la roda del mouse.
- **showDebugDraw, debugDraw, debugCanvasLayer, debugCanvasImage**: variables que serveixen per activar o desactivar la capa gràfica de debug, que mostra els límits i el centre de gravetat dels objectes físics de Box2D.
- **elemSelected**: variable que conté una instància d'un element quan se n'ha seleccionat un, o conté null quan no hi ha cap element seleccionat.
- **elemHovered**: variable que conté una instància d'un element que està sota el cursor del mouse quan, sense tenir cap element seleccionat, el cursor del mouse hi passa per sobre.
- **distMouseToCenterElem, prevPointerPos**: variables auxiliars que ajuden al moviment i col·locació dels elements dins l'editor de nivells.
- **mouse**: variable de tipus Mouse.Listener de la llibreria TriplePlay que serveix per configurar les reaccions a les accions del mouse realitzades per part de l'usuari.
- **iface**: variable de tipus Interface, el tipus principal de la llibreria TriplePlay, la qual conté l'arrel de la interfície, sobre la qual es realitzen les operacions necessàries per tal d'afegir i configurar els elements de la interfície gràfica del menú de l'editor de nivells.
- **currentButton**: variable que conté el botó de la interfície de l'editor que està actualment seleccionat.
- **blockToAdd**: variable que conté el nom del tipus d'element que s'ha d'afegir al nivell actual.
- **mode**: variable que representa el mode de funcionament, o l'estat, en que es troba l'editor en un moment donat. Els modes són: afegir bloc, eliminar bloc i moure bloc.

La constructora privada i el mètode GetInstance corresponen al patró estat, pel qual cap classe externa pot crida a la constructora de la classe directament, sinó a través de la funció GetInstance(), la qual cridarà a la constructora si cal, i sinó simplement retornarà la classe inicialitzada anteriorment. Altres operacions importants que la classe EditGameState afegeix respecte la classe GameState són:

- **addGraphics()**: afegeix les capes gràfiques necessàries per mostrar l'editor quan s'entra al mode editar del joc.
- **enableDebugDraw(), disableDebugDraw()**: funcions per activar/desactivar el mode de dibuixat de la capa de debug, ja explicat en l'apartat dels atributs.
- **gestioHover(MotionEvent)**: funció auxiliar que gestiona quan s'ha de remarca un element quan el cursor del mouse hi passa per sobre.
- **setCurrentButton(Button)**: funció auxiliar que configura un botó donat com a botó de la interfície de l'editor que està actualment seleccionat.
- **putAddElemButton(String, Group, AssetWatcher)**: funció auxiliar que afegeix a la interfície un botó que servirà per afegir nous tipus d'elements al nivell dins l'editor.
- **configInterface()**: funció que configura tots els elements de la interfície gràfica de l'editor quan s'entra en el mode editor.
- **snapToGrid(vec2, int)**: funció auxiliar que limita els moviments dels objectes dins l'editor a una quadrícula, de tal manera que sigui més fàcil encaixar objectes un al costat d'un altre.
- **toggleDebugDraw()**: funció que activar/desactiva el dibuixat de la capa de debug dins l'editor de nivells.
- **scrollCamera(float)**: funció auxiliar que s'ocupa de desplaçar tots els elements gràfics del nivell dins l'editor quan l'usuari fa scrolling amb la roda del mouse.

5.4.1.6. Classe PlayGameState

La classe PlayGameState implementa el videojoc en sí mateix, el mode de joc. Un altre cop s'ha utilitzat la llibreria TriplePlay per a afegir alguns elements d'interfície gràfica. En aquest cas la interfície és mínima i consisteix en un marcador de punts, comptador de vides restants i rellotge de temps. Quan el jugador entra en el joc, es carrega el nivell, es configura la interfície gràfica i s'inicialitzen una sèrie de variables com el nombre de vides, temps i altres i es configuren els mètodes d'entrada per teclat.

La implementació de la classe PlayGameState consisteix bàsicament en el bucle corresponent a les successives crides a la funció update(). La lectura i resposta als esdeveniments de teclat és realitzen de manera asíncrona, almenys de cara al desenvolupador, ja que quan el jugador prem una tecla s'executen les funcions especificades en el listener de teclat configurat en la funció setControlListener(). Per tant, tot el que queda fer dins el bucle és actualitzar els estats del jugador i de la resta d'elements del nivell i controlar l'estat del joc, finalitzant-lo si el jugador es queda sense vides o tornant a situar el jugador si encara li queden vides.

Un altre apartat important de la implementació de la classe PlayGameState correspon a la gestió de les col·lisions entre els elements del nivell i el player. La llibreria Box2D proporciona unes funcions buides que es criden quan s'inicia un contacte entre dues entitats, quan finalitza el contacte, i unes altres per realitzar càlculs just abans i just després que s'hagi produït un contacte. Aquestes funcions reben com a paràmetres les dues instàncies dels objectes que participen en la col·lisió i que el desenvolupador ha d'implementar amb el comportament desitjat.

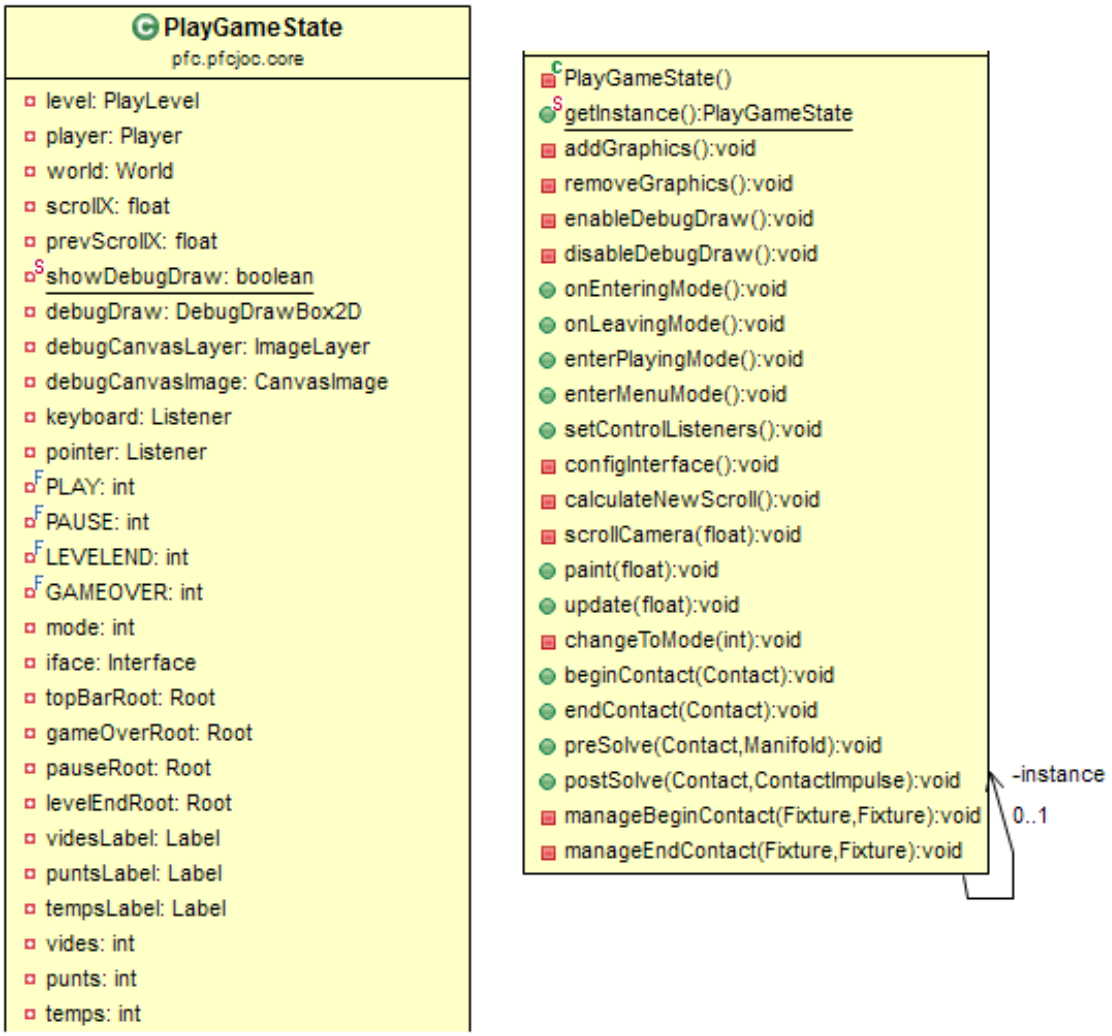


Figura 28: Classe PlayGameState

La classe PlayGameState afegeix els següents atributs respecte la classe GameState:

- **level**: aquesta variable conté un objecte de la classe PlayLevel, que correspon a la representació del nivell amb tots els seus elements. Sobre aquesta variable es realitzen les modificacions quan el jugador, per exemple, elimina un enemic i aquest ha de ser esborrat del nivell.
- **player**: aquesta variable representa el jugador del joc, l'element més important. Sobre ell es realitzen els moviments quan l'usuari prem les tecles corresponents i sobre ell es tenen en compte les col·lisions amb altres elements com enemics. La classe PlayGameState l'obté de la classe Level quan s'inicialitza.

- **world**: aquesta variable conté un objecte de la classe World de la llibreria Box2D. Aquesta classe World gestiona totes les entitats físiques, la simulació dinàmica i altres coses.
- **scrollX, prevScrollX**: variables que contenen la quantitat d'scrolling en l'eix X que hi ha actualment i en l'anterior volta del bucle.
- **showDebugDraw, debugDraw, debugCanvasLayer, debugCanvasImage**: variables que serveixen per activar o desactivar la capa gràfica de debug, que mostra els límits i el centre de gravetat dels objectes físics de Box2D.
- **keyboard**: variable de tipus Keyboard.Listener de la llibreria TriplePlay que serveix per configurar les reaccions a les accions del teclat realitzades per part de l'usuari.
- **iface**: variable de tipus Interface, el tipus principal de la llibreria TriplePlay, la qual conté l'arrel de la interfície, sobre la qual es realitzen les operacions necessàries per tal d'afegir i configurar els elements de la interfície gràfica del joc.
- **mode**: variable que representa el mode de funcionament, o l'estat, en que es troba el joc en un moment donat. Els modes són: en joc, en pausa, final de nivell i final de joc.
- **topBarRoot, gameOverRoot, pauseRoot, levelEndRoot**: variables del tipus Root de la llibreria TriplePlay, que contenen les interfícies que es mostren en els diferents modes del joc.
- **videsLabel, puntsLabel, TempsLabel**: variables corresponents a les etiquetes que es mostren durant el joc i que informen del nombre de vides, punts i temps.
- **vides, punts, temps**: variables que emmagatzemen el nombre de vides que li queden al jugador, els punts que ha obtingut i el temps de joc que ha transcorregut.

La constructora privada i el mètode GetInstance corresponen al patró estat, pel qual cap classe externa pot crida a la constructora de la classe directament, sinó a través de la funció GetInstance(), la qual cridarà a la constructora si cal, i sinó simplement retornarà la classe inicialitzada anteriorment. Altres operacions importants que la classe PlayGameState afegeix respecte la classe GameState són:

- **addGraphics(), removeGraphics()**: afegeix/elimina les capes gràfiques necessàries per mostrar tots els elements del joc, del nivell i el jugador quan s'entra al mode de joc o se'n surt.
- **enableDebugDraw(), disableDebugDraw()**: funcions per activar/desactivar el mode de dibuixat de la capa de debug, ja explicat en l'apartat dels atributs.
- **configInterface()**: funció que configura tots els elements de la interfície gràfica de l'editor quan s'entra en el mode editor.
- **calculateNewScroll()**: funció auxiliar que calcula, a cada volta del bucle, com ha de desplaçar els diferents elements de la pantalla per tal de simular scrolling quan el jugador es mou.
- **scrollCamera(float)**: funció auxiliar que s'ocupa de desplaçar tots els elements gràfics del nivell dins l'editor quan l'usuari fa scrolling amb la roda del mouse.
- **changeToMode(int)**: funció auxiliar que s'encarrega de canviar entre els diferents modes del joc. Per exemple quan el jugador vol fer una pausa es passa de mode de joc a mode pausa cridant a aquest funció.
- **beginContact(Contact), endContact(Contact), preSolve(Contact, Manifold), postSolve(Contact, ContactImpulse)**: funcions de la interfície ContactListener que la llibreria Box2D crida dins el seu bucle de resolució de col·lisions cada cop que detecta una col·lisió entre dues entitats físiques.
- **manageBeginContact(Fixture, Fixture), manageEndContact(Fixture, Fixture)**: funcions auxiliars per ajudar a gestionar les reaccions que el joc ha de fer quan es produeixen col·lisions entre entitats. Les funcions beginContact i endContact de Box2D passen el paràmetre Contact a partir del qual s'extreuen les dues entitats que col·lideixen, però com no sabem quina de les dues entitats és de quin tipus, s'hauria de repetir codi amb condicionals, cosa que es pot estalviar amb aquestes funcions auxiliars que es criden dos cops amb els paràmetres intercanviats.

5.4.1.7. Classe Level

La classe Level és la que conté tots els elements que formen part d'un nivell del videojoc, com el jugador, els enemics, els blocs que formen l'escenari, els world de Box2D, les capes gràfiques del joc, etc. És utilitzada tant per la classe PlayGameState com per la classe EditGameState, raó per la qual la classe Level conté mètodes com add(), remove() i getElemAt() que permeten afegir elements, eliminar-los i consultar-los

respectivament. També conté mètodes per a guardar el nivell i per a carregar-lo, incloent una funció per a parsejar els fitxers .json de nivells.

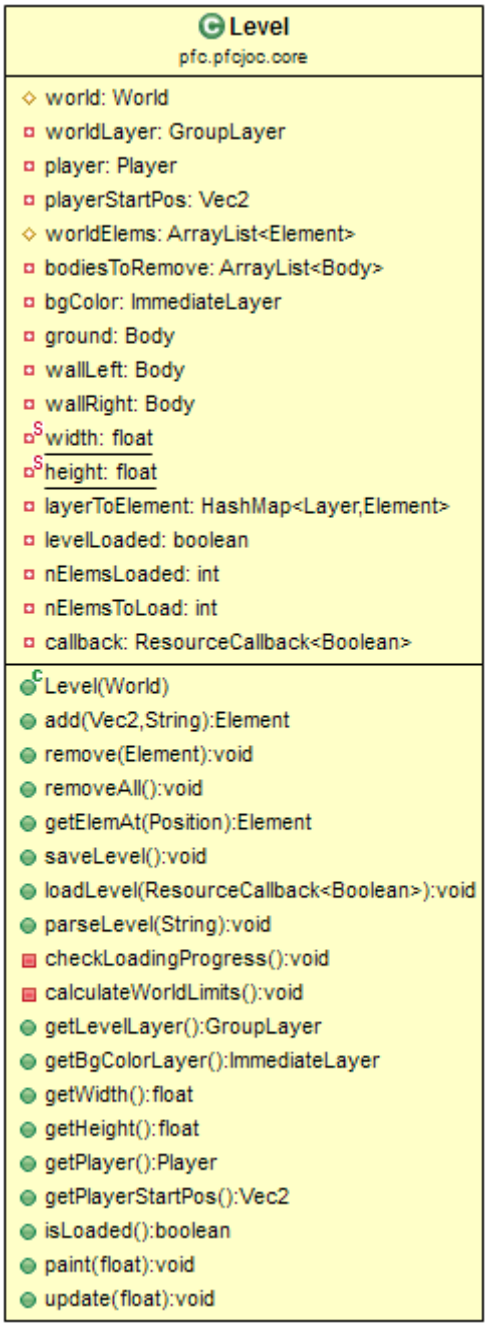


Figura 29: Classe Level

La classe Level conté els següents atributs:

- **world**: aquesta variable conté un objecte de la classe World de la llibreria Box2D. Aquesta classe World gestiona totes les entitats físiques, la simulació dinàmica i altres coses.

- **worldLayer**: variable de tipus GroupLayer que serveix per tenir agrupades totes les capes de tots els elements que formen part del nivell.
- **player**: aquesta variable representa el jugador del joc, l'element més important. Sobre ell es realitzen els moviments quan l'usuari prem les tecles corresponents i sobre ell es tenen en compte les col·lisions amb altres elements com enemics.
- **playerStartPos**: variable que emmagatzema la posició inicial del jugador dins el nivell per tal que quan se l'hagi de reviure se'l pugui situar al mateix punt.
- **worldElems**: variable que conté tots els elements del nivell i sobre la qual actuaran les operacions d'afegir, eliminar i consultar elements.
- **bodiesToRemove**: variable on es van acumulant els bodies de cada un dels elements que han de ser eliminats la propera volta del bucle. No es poden eliminar directament del world a mitja actualització del bucle perquè provoca comportaments no previstos a Box2D.
- **bgColor**: variable de tipus ImmediateLayer que s'utilitza per a pintar el color de fons de la escena quan no hi ha cap altre capa del nivell cobrint-la.
- **ground, wallLeft, wallRight**: variables de tipus body que serveixen per tenir uns límits físics i invisibles al nivell, útils per coses com saber quan el jugador ha caigut per un forat, etc.
- **widht, height**: contenen les mides del nivell en unitats físiques.
- **layerToElement**: variable de tipus mapa de Layer a Element que serveix per a detectar quan l'usuari està seleccionant un element, ja que les proves d'encert d'un click es realitzen sobre capes i ens interessa retorna l'element sencer.
- **levelLoaded**: variable que indica si el nivell s'ha acabat de carregar.
- **nElemsLoaded, nElemsToLoad**: variables per controlar el progrés de carrega del nivell.
- **callback**: variable de tipus callback que serveix per avisar de quan s'ha acabat de carregar el nivell a la classe que hagi configurat el callback prèviament.

Algunes operacions importants de la classe Level:

- **Level(World)**: constructora que inicialitza algunes variables i rep el world segons l'estat en que es troba el joc.
- **add(Vec2, String)**: afegeix un nou element al nivell, del tipus i a la posició que es passen com a paràmetre.
- **remove(Element)**: elimina del nivell l'element que es passa com a paràmetre.

- **removeAll()**: elimina tots els elements del nivell.
- **getElemAt(Position)**: retorna l'element de la posició que es passa com a paràmetre, si existeix. Sinó, retorna un element buit.
- **saveLevel()**: guarda el nivell a disc en format JSON, gràcies a les funcions que PlayN proporciona per a generar fitxers JSON.
- **loadLevel(ResourceCallback<Boolean>)**: càrrega el nivell que s'havia guardat anteriorment, o el nivell per defecte del joc si no se n'havia guardat cap. Un cop carregat el nivell s'executa el callback done(), que avisarà a la classe que l'havia configurat per tal que conegui que el nivell s'ha carregat correctament. Una de les ampliacions que es comenten més endavant és la de poder triar quin nivell carregar, guardar-ne més d'un, etc.
- **parseLevel(String)**: funció auxiliar que realitza gran part de la feina de la càrrega de nivells, ja que es la que recorre els fitxers .json i n'extreu la informació.
- **checkLoadingProgress()**: funció privada que comprova si el nivell s'ha carregat del tot cada cop que s'afegeix un nou element per tal d'actuar en conseqüència.
- **calculateWorldLimits()**: funció priva que s'executa quan el nivell s'ha carregat del tot i s'encarrega de posar els límits del nivell, les variables ground, wallLeft i wallRight a la posició que es correspon.
- **getLevelLayer(), getBgColorLayer(), getWidth(), getHeight(), getPlayer(), getPlayerStartPos(), isLoaded()**: vàries consultores útils per a altres classes externes.
- **paint(float):** s'encarrega de pintar el nivell en l'estat actual.
- **update(float):** s'encarrega d'actualitzar l'estat del nivell actual.

5.4.1.8. Classes EditLevel i PlayLevel

Les classes EditLevel i PlayLevel són dues classes que deriven de la classe Level i són les que utilitzen els modes EditGameState i PlayGameState, respectivament. La única diferència entre les dues és que en la funció update() de la classe PlayLevel es continua cridant a la funció update() de tots els elements del nivell per tal d'actualitzar-ne l'estat, les variables físiques, etc. En la classe EditLevel, com no correspon a l'estat de joc en curs, no es fan les actualitzacions ja que els objectes es mouran per l'acció dels moviments que decideixi l'usuari amb el mouse, i no per cap tipus de llei física.

5.4.2. Diagrama de classes de les entitats

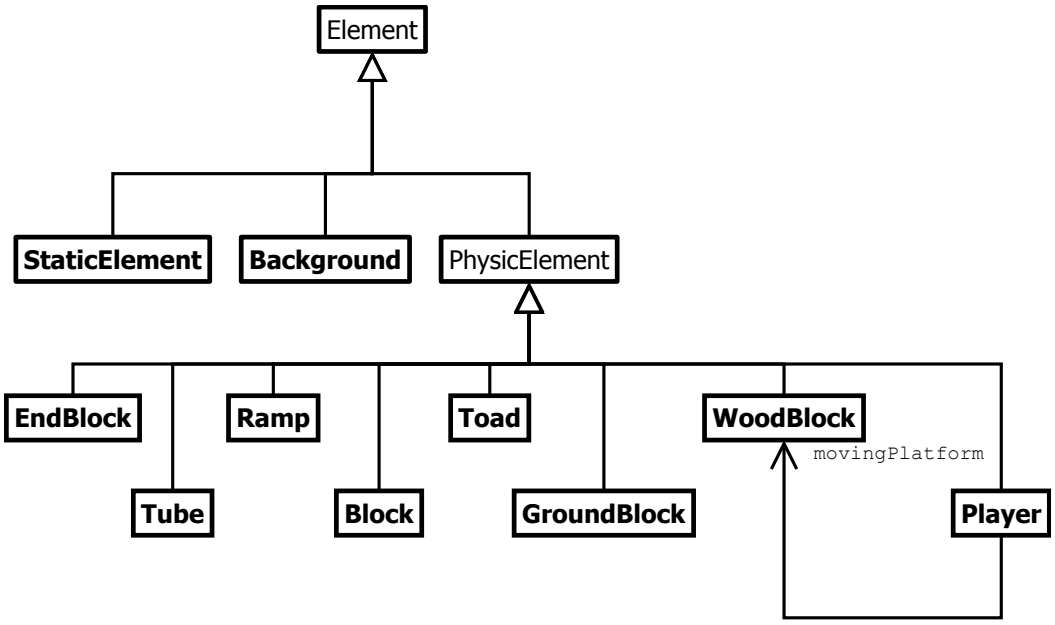


Figura 30: Diagrama de classes de les entitats

A dalt de tot de la jerarquia de les entitats que componen els nivells del joc hi tenim la classe Element. Aquesta classe Element és una classe abstracta, per tant no existiran objectes d'aquesta classe en el sistema, només de les seves classes derivades. Els objectes derivats de la classe Element són els que componen el món o els nivells del joc. La classe Level és tractarà més endavant, però com a petit resum podem dir que un nivell, un objecte de la classe Level, conté un conjunt d'objectes derivats de la classe Element que hem anomenat worldElems. La majoria de la feina que fa la classe Level correspon al tractament d'aquest elements, com per exemple:

- Afegir nous elements, com enemics i elements del món, a l'editor de nivells.
- Eliminar elements en el mode de joc quan el jugador mata un enemic.
- Consultar tots els elements en el moment de guardar/carregar el nivell a/de memòria.
- Consultats tots els elements per determinar si en una posició concreta hi ha un element, per exemple en l'editor de nivells per seleccionar i arrossegar elements.

Les classes StaticElement i Background, que són molt semblants tret que en un futur la classe Background podria configurar-se amb una velocitat d'scroll, corresponen a

elements del món que no tenen característiques físiques, és a dir, no interactuen amb la resta d'elements. Les seves funcionalitats són les de decorar l'escenari del món i també de crear efectes com el paral·lax, amb diferents elements de fons movent-se a diferents velocitats per tal de crear una sensació de profunditat de l'escenari al jugador.

5.4.2.1. Classe Element

La classe Element és la més important del grup de les classe que formen les entitats del joc. Es tracta d'una classe abstracta que proporciona uns atributs i unes operacions que són d'utilitat a un grup important de subclasses.

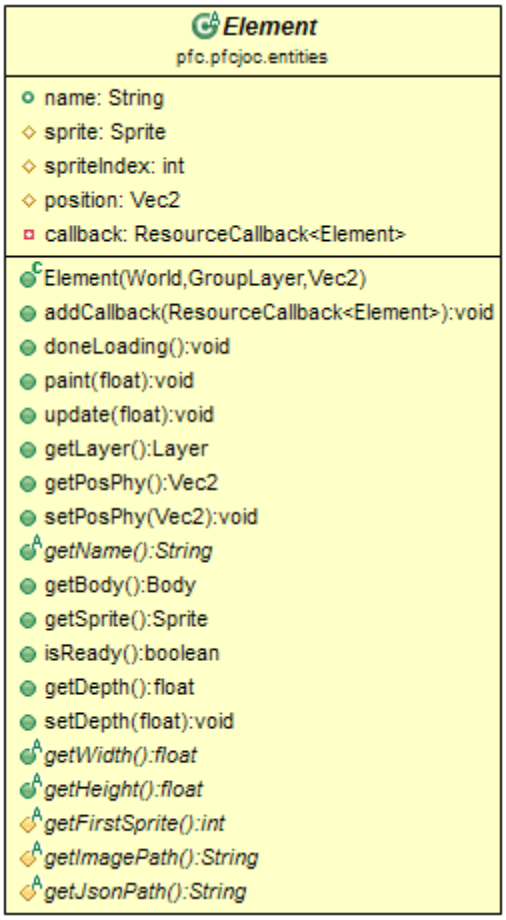


Figura 31: Classe Element

La classe Element conté els següents atributs:

- **name:** conté el nom del tipus de la classe actual. Això és útil per tal d'identificar quins tipus d'Element intervenen en una col·lisió i actuar en conseqüència. Per exemple, no és el mateix que el jugador caigui sobre un bloc de terra que sobre

un enemic. Una altra utilitat és la de poder emmagatzemar en els fitxers de nivell el tipus de objecte que hi ha a cada posició, per tal de poder carregar-lo correctament més endavant.

- **sprite:** conté el conjunt d'imatges de l'sprite corresponent a l'Element actual.
- **spriteIndex:** conté el nombre corresponent a la imatge de l'sprite que toca pintar en el següent frame del joc, en la següent execució de la funció paint().
- **position:** conté la posició en coordenades físiques de l'Element actual.

Algunes operacions importants de la classe Element:

- **addCallback(ResourceCallback<Element>)**: funció que afegeix un callback a l'element actual per avisar a la classe que l'ha configurat que aquest ja s'ha carregat del tot i està disponible.
- **doneLoading()**: funció auxiliar que es crida quan l'element actual s'ha carregat del tot. Utilitza el callback definit amb la funció anterior.
- **paint(float):** s'encarrega de pintar l'element en l'estat actual.
- **update(float):** s'encarrega d'actualitzar l'estat de l'element actual.
- **getLayer(), getPosPhy(), setPosPhy(Vec2), getName(), getBody(), isReady(), getDepth(), setDepth(float), getWidth(), getHeight(), getFirstSprite, getImagePath(), getJsonPath()**: vàries funcions consultores i alguna que assigna valors, que són útils per a classes externes.

5.4.2.2. Classe StaticElement i Background

Les classes StaticElement i Background són subclasses de la classe element que representen elements sense cap tipus d'interacció física. Són gairebé idèntiques en la seva interfície, però separades ja que la classe Background podria permetre en el futur ser configurada amb una velocitat d'scrolling, per tal de realitzar l'efecte de parallax scrolling que dóna sensació de profunditat a les escenes. Conceptualment la classe StaticElement representa element del nivell, no del fons de la pantalla, que no tenen cap interacció física amb el jugador, que són simplement decoratius.

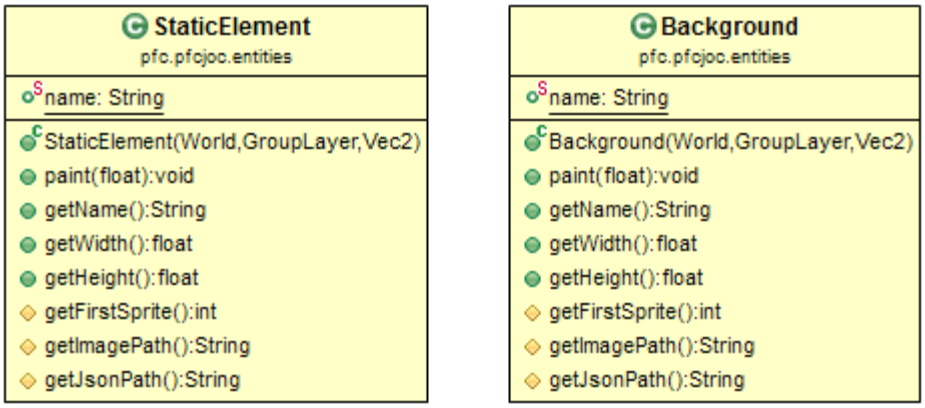


Figura 32: Classes StaticElement i Background

Les classes StaticElement i Background reimplementen algunes operacions de la súper classe Element com paint(float), getName(), getWidth(), getHeight(), getFirstSprite(), getImagePath(), i getJsonPath() per tal de retornar informacions específiques respecte l'element actual.

5.4.2.3. Classe PhysicElement

La classe PhysicElement és una subclasse abstracta de la també classe abstracta Element. Representa un element del nivell que tindrà característiques físiques, és a dir, que el jugador hi podrà interactuar, ja sigui xocant amb ell, saltant a sobre, etc. És una classe abstracta perquè els diferents elements físics necessitaran tenir diferents característiques i comportaments per mitjà de la refinació d'alguns mètodes.

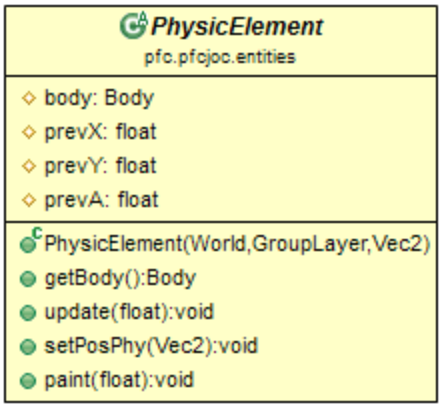


Figura 33: PhysicElement

Respecte a la classe Element s'afegeixen els atributs body, prevX, prevY i prevA, que representen el cos físic de Box2D i els valors previs de la posició i la rotació de

l'element físic, que seran útils per al procés d'interpolació a l'hora de pintar l'element actual com s'ha explicat en l'apartat de la llibreria PlayN.

Els mètodes que es redefineixen són getBody(), que ara retorna el cos físic i no buit com abans, les funcions update(float), on es guarden els valors previs comentats abans, i paint(float), que tindrà en compte la posició física per tal de pintar amb interpolació. També es refina la funció setPosPhy(Vec2).

5.4.2.4. Classe Player

La classe Player és la subclasse de PhysicElement més important, ja que representa el jugador del joc, i s'encarrega de totes les coses relacionades amb aquest, com, per exemple, moure el jugador, comptar els contactes amb el terra, enemics i altres per tal de gestionar les seves reaccions, etc. També gestiona les animacions dels gràfics o sprites que representen el jugador, canviant-los segons les circumstàncies del joc com, per exemple, canviant ràpidament d'imatges per realitzar l'animació del jugador corrent.

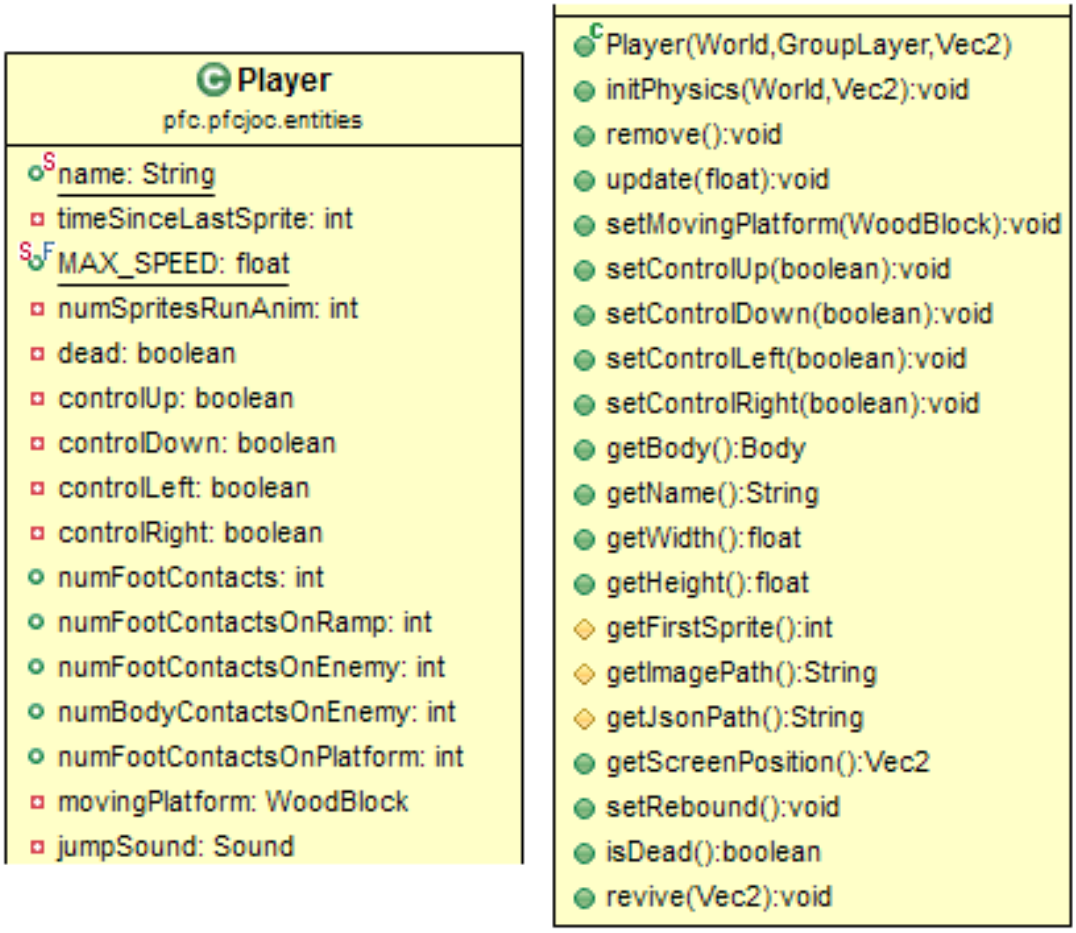


Figura 34: Classe Player

La classe Player conté els següents atributs:

- **name**: conté el nom del tipus de la classe actual. Això és útil per tal d'identificar quins tipus d'Element intervenen en una col·lisió i actuar en conseqüència. Per exemple, no és el mateix que el jugador caigui sobre un bloc de terra que sobre un enemic. Una altra utilitat és la de poder emmagatzemar en els fitxers de nivell el tipus de objecte que hi ha a cada posició, per tal de poder carregar-lo correctament més endavant.
- **timeSinceLastSprite**: conté el temps que ha passat des de l'últim cop que s'ha canviat d'sprite a l'animació, per tal de poder-ne gestiona la velocitat.
- **numSpritesRunAnim**: conté el nombre d'sprite que representen l'animació de córrer dins el full d'sprites de l'element actual.
- **dead**: indica si el jugador està mort o no.
- **controlUp, controlDown, controlLeft, controlRight**: variables que indiquen si s'està prement alguna de les tecles de direcció.
- **numFootContacts, numFootContactsOnRamp, numFootContactsOnEnemy numBodyContactsOnEnemy, numFootContactsOnPlatform**: variables que compten els contactes que s'estan produint en un moment donat en el jugador. Box2D registra quan comença i quan acaba un contacta, moments en els quals s'incrementen o disminueixen aquestes variables. Serveixen per a que el joc tingui informació del que està passant pugui reaccionar correctament.
- **movingPlatform**: conté una plataforma si el jugador és sobre una plataforma mòbil. Serveix per poder ajustar els moviments del jugador respecte els moviments de la plataforma, ja que amb Box2D no es pot fer que el player estigui enganxat a la plataforma i alhora es pugui moure per sobre, ja que per fer-ho hem d'augmentar molt la fricció entre els dos cossos.
- **jumpSound**: conté el so que s'escolta cada cop que el jugador salta.

Algunes operacions importants de la classe Player són:

- **initPhysics(World, Vec2)**: inicialitza totes les característiques físiques del jugado. Crea els diferents bodies necessaris per a simular el jugador, els peus, sensor de col·lisions, etc.
- **remove()**: elimina el cos de l'usuari del món, o world, de Box2D.
- **update(float)**: s'encarrega d'actualitzar l'estat del jugador.

- **setMovingPlatform(WoodBlock),setControlUp(boolean), setControlDown(boolean),setControlLeft(boolean), setControlRight(boolean)**: funcions que assignen la plataforma mòbil al jugador i les tecles de direcció del jugador.
- **getBody(), getName(), getWidth(), getHeight(), getFirstSprite(), getImagePath(), getJsonPath()**: vàries funcions consultores.
- **getScreenPosition()**: retorna la posició del jugador en coordenades de pantalla.
- **setRebound()**: aplica un impuls al jugador per fer-lo rebotar.
- **isDead()**: retorna si el jugador és o no mort.
- **revive(Vex2)**: reviu el jugador, és a dir, el torna a la posició inicial i el marca com a no mort.

5.4.2.5. Altres classes derivades de PhysicElement

En aquest apartat una sèrie de subclasses de la classe PhysicElement. Com tenen algunes parts en comú les descriurem breument:

- **Block**: representa un bloc estàtic del nivell. El seu ús serà per a crear plataformes fixes a partir de múltiples blocs.
- **GroundBlock**: representa un bloc estàtic del nivell. El seu ús serà per a formar el terra del nivell i, per tant, tindrà una textura diferent i unes característiques diferents del Block.
- **Tube**: representa un element estàtic del nivell. En aquest cas tindrà forma de canonada estil Super Mario Bros. Més endavant se li podrien afegir funcionalitats com la d'entrar a dins i anar a parar a un nivell secret.
- **Ramp**: representa un bloc estàtic no rectangular, amb forma de rampa, per tal de poder crear pujades i baixades en el nivell.
- **WoodBlock**: representa un bloc dinàmic que es mourà d'esquerra a dreta alternativament, fent que el jugador hagi de sincronitzar el salt per poder segui avançant en el nivell actual.
- **EndBlock**: representa el bloc que l'usuari ha de tocar per donar per acabat el nivell actual. Aquesta serà la seva única finalitat i la seva característica diferenciadora de la resta d'elements del nivell.
- **Toad**: representa un enemics en forma de gripau a l'estil de Super Mario Bros. El seu comportament serà de moure's cap a l'esquerra fins que trobi un obstacle. Aleshores es mourà cap a la dreta fins que trobi un obstacle, i així

indefinidament. El jugador el podrà matar si el trepitja per sobre. En canvi, si el jugador el toca pels costats, serà aquest qui morirà.

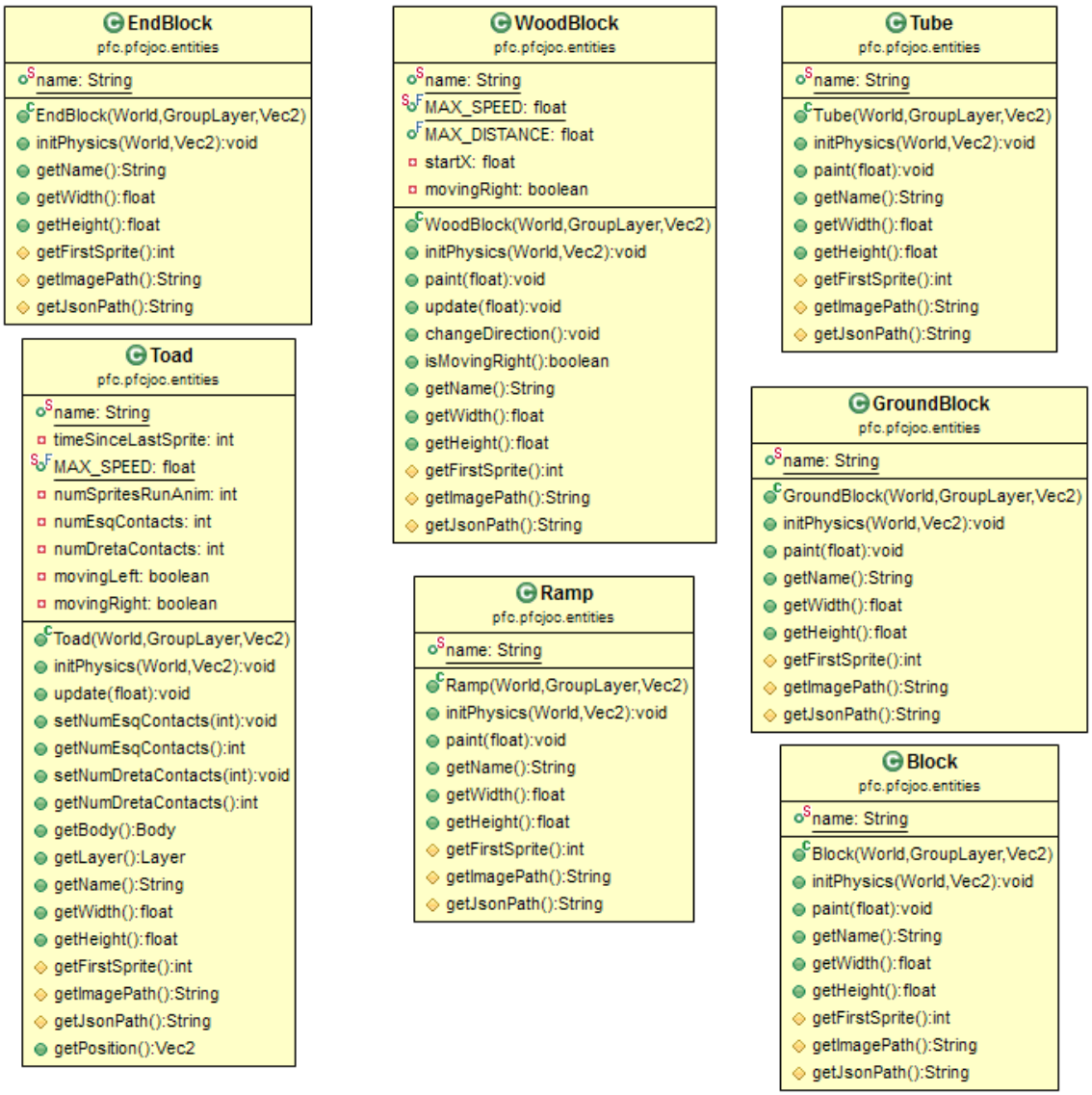


Figura 35: Altres subclasse de PhysicElement

5.4.3. Diagrama de classes auxiliars

En aquest apartat comentarem algunes classes auxiliars que s'han utilitzat en aquest projecte de final de carrera. Es tracta de la classe Sprite, SpriteImage i Sprite Loader. Aquestes classes han sortit d'un dels exemples disponibles a la pàgina web del projecte PlayN i se'ls hi ha realitzat algunes modificacions per tal de poder voltar els sprites sense haver d'incloure un sprite voltejat al full d'sprites, cosa que permet estalviar espai.

La resta és exactament igual, les classes serveixen per carregar un full d'sprites per tal de poder-los utilitzar com a animacions dins el videojoc.

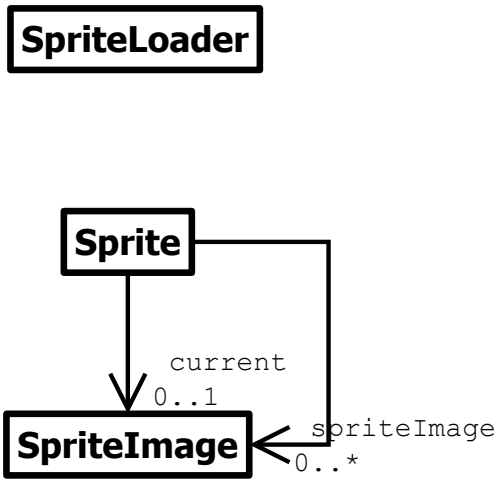


Figura 36: Classe Sprites i relacionades

El funcionament d'aquestes classes consisteix a proporcionar un fitxer amb imatges i un fitxer en format JSON que descrigui quines regions de la imatges corresponen a quin sprite. A continuació veurem un exemple:

```
{
  "sprites": [
   {"id": "sprite_0", "x": 0, "y": 0, "w": 32, "h": 32},
   {"id": "sprite_1", "x": 32, "y": 0, "w": 32, "h": 32},
   {"id": "jumping", "x": 64, "y": 0, "w": 32, "h": 32}
  ]
}
```

Figura 37: Sprites i JSON

6. Planificació i pressupost

Aquest projecte de final de carrera es va començar a desenvolupar a meitats de desembre de 2011 i s'ha acabat a finals d'octubre de 2012. Aproximadament uns 300 dies naturals. No totes les setmanes la dedicació al projecte ha estat la mateixa, i tenint em compte dies festius, vacances i altres compromisos es fa difícil poder donar un nombre exacte d'hores dedicades per setmana. Amb l'objectiu d'obtenir una mitjana d'hores per dia, podem suposar una dedicació diària de 3,5 hores, durant 5 dies a la setmana, durant les 43 setmanes que ha durat el projecte. Així obtenim una xifra de 752 hores. Altre cop, cal tenir en compte que la dedicació en algunes setmanes, segons el volum de feina que s'havia de realitzar, augmentava o disminuïa bastant i, com en molts projectes, les darreres setmanes les hores dedicades tendeixen a augmentar de manera important.

La primera tasca que es va dur a terme les primeres setmanes de projecte va consistir en fer una petita recerca sobre les diferents llibreries disponibles a Internet per al desenvolupament de videojocs en HTML5. A partir d'aquí, un cop triada la llibreria PlayN, que anteriorment es deia ForPlay, aquesta es va estudiar més en detall, per tal d'arribar a l'estat inicial de poder mostrar una finestra amb un cercle al mig de la escena. Aquí apareixen temes com la descàrrega de la llibreria, compilació de la mateixa, creació del projecte i compilació del mateix. Tots aquests passos van quedar simplificats un cop Google va posar la llibreria al repositori central de Maven, de manera que des de l'entorn Eclipse es podia buscar l'artefacte de projecte de PlayN i en pocs minuts s'obtenia un projecte buit, amb les dependències resoltes i mostrant una finestra buida. Un cop arribats a aquest punt, es van realitzar proves i experiments per aprendre el funcionament bàsic d'aquesta llibreria. Aquí entren temes com la càrrega d'imatges per a textures, la gestió del sistema de capes, el funcionament del bucle del joc, la gestió dels perifèrics d'entrada, la utilització bàsica de la física amb Box2D, etc. Amb tot això es realitza un petit prototip de videojoc on es podia moure una capsa a esquerra i dreta i fer-la saltar de manera més o menys realista. El següent pas va consistir en familiaritzar-se amb la llibreria de física Box2D, realitzant tasques com la creació del món, configuració de la gravetat, incloure cossos físics i enllaçar-los a les capes gràfiques de

PlayN, configurar els paràmetres físics d'aquests cossos com la massa, la fricció, etc. Amb els coneixements adquirits de Box2D, s'amplia el prototip anterior, afegit un terra a base d'elements físics pels quals es desplaçava un personatge ja amb una textura adequada. El següent objectiu va ser fer el disseny del videojoc, principalment el motor ja que PlayN és només una llibreria que ofereix un bucle de joc buit. Es tenen en compte coses com el canvi entre estats dins el joc, gestió del perifèrics d'entrada segons quin estat, càrrega d'imatges i sons, etc. A partir d'aquí comença la programació del videojoc i, a mida que s'avança en la implementació del joc, es fa més evident la necessitat de tenir un editor de nivells a més capacitats per tal de poder fer proves més fàcilment i poder iterar ràpidament el desenvolupament del joc en diferents escenaris. En paral·lel a tot això també es desenvolupa la escriptura d'aquesta memòria, part també molt important del projecte de final de carrera. Les darreres setmanes també s'ha dedicat temps a la preparació de la presentació del projecte davant el tribunal.

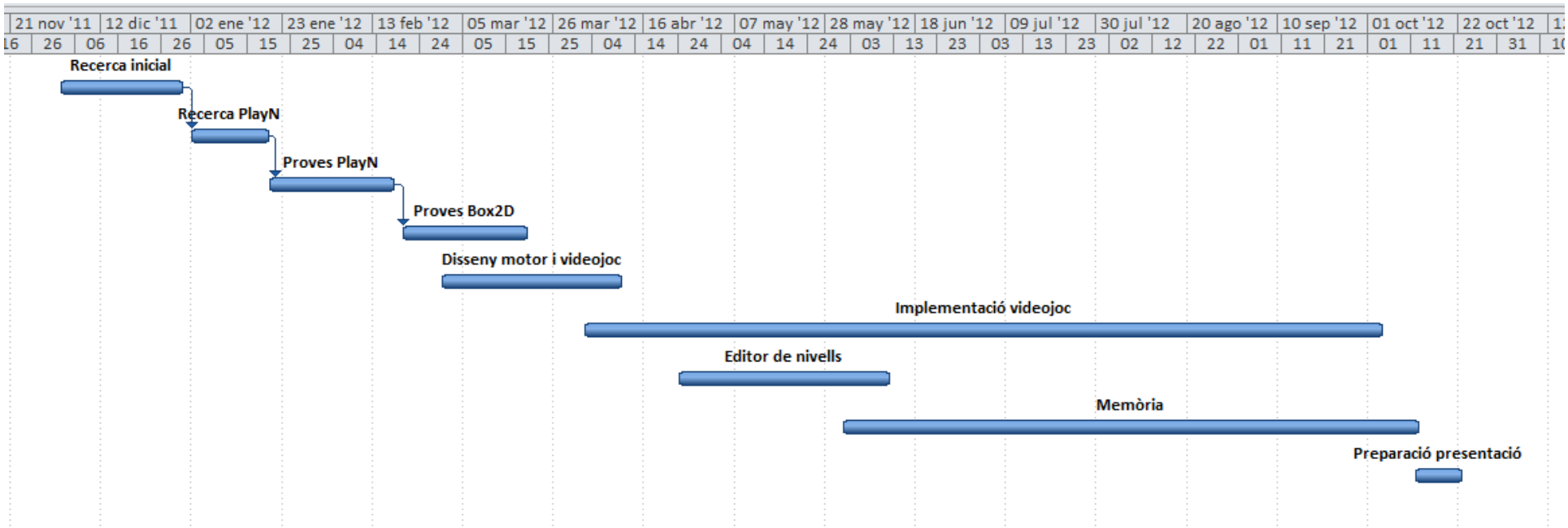


Figura 38: Diagrama de Gantt del projecte

A continuació una descripció temporal de les tasques realitzades en aquest projecte de final de carrera corresponents amb el diagrama de Gantt anterior:

- **Recerca inicial**: durant gairebé tres setmanes es busquen i s'estudien per sobre diferents llibreries o eines que serien útils més endavant. Després d'això es tria la llibreria PlayN d'entre les diferents eines disponibles per al desenvolupament de videojocs per a HTML5.
- **Recerca PlayN**: es dediquen un parell de setmanes a estudiar més a fons la llibreria PlayN, es miren demos, fòrums, el repositori on hi ha el codi font, etc.
- **Proves PlayN**: amb les demos vistes prèviament es comencen a fer proves i modificacions per tal d'entendre el funcionament real de PlayN. Es dediquen unes tres setmanes a fer proves.
- **Proves Box2D**: de manera semblant es dediquen tres setmanes a mirar codi de programes que utilitzen Box2D, es modifiquen i es creen petites demostracions per entendre el funcionament de la eina.
- **Disseny motor i videojoc**: es dedica un mes a dissenyar el motor i videojoc, que van molt lligats. Més endavant es farien retocs al disseny inicial.
- **Implementació videojoc**: la implementació del videojo s'allarga uns 133 segons el diagrama de Gantt, però tenint en compte que no tots els dies s'hi dedicava temps en concret a la implementació, sinó que es combinava amb altres tasques.
- **Editor de nivells**: l'editor de nivells havia de ser senzill però comença a prendre forma i se li afegeixen característiques i la possibilitat de ser ampliar en un futur amb certa facilitat. Se li dediquen uns 35 dies aproximadament.
- **Preparació presentació**: la darrera setmana del projecte es quan es prepara la presentació per a la defensa del projecte.
- **Memòria**: una de les parts més gran del projecte i a la qual se li ha dedicat uns tres mesos.

A continuació realitzarem una estimació del cost del projecte. Per tal de desenvolupar aquest projecte de final de carrera s'ha passat per diferents fases que, probablement, en el món laboral haguessin realitzat persones amb diferents càrrecs o llocs de treball. En el nostre cas tindrem en compte això per tal de separar les diferents tasques i aplicar un cost per hora diferent segons quin treballador la duria a terme. Càrrecs com el cap de projectes, el qual podria dedicar-se a la gestió de l'equip de treball i a assegurar-se que el projecte mantingui uns nivells de qualitat i que compleixi els terminis demanats, no té

sentit que s'inclogui en aquests càlculs ja que aquestes tasques les ha realitzar una sola persona a l'hora que la resta de tasques. Tindrem en compte l'analista, que s'encarrega dels requisits del sistema final i els transforma a casos d'ús, el dissenyador, que realitza el disseny de l'aplicació en base als inputs de l'analista, i el programador o desenvolupador que s'encarrega de transformar el disseny en codi font estructura adequadament. Tindrem en compte uns preus per hora de 60€ per l'analista, 60€ pel dissenyador i 45 pel programador. Aquests preus podrien variar en la vida real depenent de la empresa, el país i la experiència del treballador.

Tipus de feina	Hores	Cost per hora (€/h)	Total(€)
Analista	167	60	**10020**
Dissenyador	285	60	**17100**
Desenvolupador	300	45	**13500**
	752		**40620**

Figura 39: Costos treballadors

Per altra banda, s'ha de tenir en compte els recursos necessaris que s'han utilitzat per al desenvolupament del projecte. Primerament tenim l'ordinador, que podem suposar amb un preu de 1000€. Suposarem també que s'amortitza el seu preu en uns 4 anys, i tenint en compte que el temps de duració del projecte ha estat d'uns 11 mesos, tenim un cost de 230€. També tenim el sistema operatiu Windows 7 i els programes Microsoft Office 2010, Microsoft Visio 2010 i Microsoft Project 2010. Tots aquests programes són disponibles a la web de Microsoft per a estudiants DreamSpark degut a un acord entre aquestes dues institucions, però mostrarem els preus del seu cost en el mercat segons s'ha pogut observar a la web de compres de Google, que recull preus d'una gran quantitat de tendes virtuals i reals. En una empresa real els preus segurament quedarien per sota d'aquests preus, potser amb algun descompte per a empreses, però òbviament per sobre de les llicències que ofereix la FIB gratuïtament als estudiants matriculats. Amb aquests preus també aplicarem l'amortització, suposant també un període de 4 anys, com els PCs, i tenint en compte la duració del projecte. Windows 7 Pro costa uns 130€, uns 30€ per 11 mesos. Office 2010 Pro costa 699€, uns 160€ per 11 mesos. Visio 2010 costa 725€, uns 166€ per 11 mesos. I finalment Project 2010 costa 690€, uns 158€ per 11 mesos.

Nom del recurs	Preu (€)
Costos treballador	40620
PC	230
Windows 7 Professional	30
Microsoft Office 2010 Professional	160
Microsoft Visio 2010	166
Microsoft Project 2010	158
TOTAL	**41334**

Figura 40: Desglossament dels costos del projecte

També s'ha de tenir present que es podrien estalviar diners fent servir les alternatives lliures que mostrem a continuació:

- Linux com a sistema operatiu base, en lloc de Windows, per a fer córrer després l'entorn de programació Eclipse, que ha estat la principal eina utilitzada, els navegadors web Firefox, Chrome i Opera, etc.
- OpenOffice.org, en lloc de Microsoft Office, per a realitza la escriptura d'aquesta memòria.
- GanttProject, en lloc de Microsoft Project, per a realitzar els diagrames de Gantt sobre l'estat i la planificació del projecte.
- Inkscape, per fer els esquemes de xarxes i figures vectorials, en lloc de Microsoft Visio.

7. Conclusions

En aquest darrer apartat de la memòria farem un repàs dels objectius del projecte per veure en quina mesura s'han complert. També es proposaran algunes ampliacions al projecte que es podrien implementar.

7.1. Objectius del projecte

Els objectius a l'iniciar aquest projecte de final de carrera eren:

- **Desenvolupar un videojoc en 2D de tipus plataformes en HTML5**: aquest era el principal objectiu i s'ha assolit del tot. Com veurem en les ampliacions tot és millorable, però s'ha aconseguit desenvolupar un videojoc des de zero amb la llibreria PlayN. El videojoc ofereix les típiques accions de saltar, moure's en totes direccions, enemics amb una petita intel·ligència artificial que es poden matar, plataformes mòbils i forats per on poder caure. També s'ha utilitzat la llibreria Box2D per a la física del joc. Tot això programant en Java i executant el joc tant com a aplicació d'escriptori com a aplicació web dins els servidors de Google AppEngine. Quan es va començar el projecte, la llibreria PlayN feia pocs mesos que havia aparegut. Això ha provocat algunes dificultats, com que amb cada actualització d'aquest el joc s'hagués d'anar adaptant, a vegades només uns petits detalls i en altres ocasions modificacions importants. Els darrers mesos el desenvolupament de PlayN va entrar en una fase més estable i, a més, un servidor va triar no actualitzar més la llibreria per evitar problemes a mida que s'acostava el final de projecte. Cal dir que la llibreria PlayN és força interessant per a desenvolupar videojocs en 2D, ja sigui plataformes com de qualsevol altre tipus, ja que facilita tasques com la carrega de gràfics, sons i altres tipus de recursos, com els de tipus text utilitzats per als nivells en aquest joc. El fet de poder programar en Java obre les portes a molts programadors que estan més familiaritzats amb aquest llenguatge. En aquest cas la tria de la

llibreria no ha tingut res a veure amb el llenguatge Java, ja que a la FIB hem utilitzat de tot, incloent assemblador de vàries màquines, C, C++, Java, Lisp, Erlang, HTML, CSS, JavaScript, OpenGL i algun altre més.

- **Desenvolupar un editor de nivells per al videojoc**: com a objectiu secundari tenim l'editor de nivells. El primer que cal dir que amb el pas dels dies l'editor es va convertir en un objectiu no tant secundari, ja que a mida que el joc creixia es requeria poder crear un nivell amb rapidesa i facilitat per tal de poder testejar més freqüentment. Així, l'objectiu de l'editor també ha estat assolit. Disposa de funcionalitats per a afegir tot tipus d'elements al nivell, per eliminar-los i per moure'ls. També s'ha tingut en compte la interfície i la resposta de l'editor a les accions de l'usuari per fer més còmode i fàcil l'ús d'aquest. Un exemple és que quan l'usuari mou el cursor per la pantalla, els elements que queden a sota es marquen per indicar que s'hi pot interactuar. Un altre exemple el trobem en el panell, amb alguns botons que es queden marcats per indicar un canvi de mode quan fa falta. Les altres funcionalitats de l'editor són poder guardar el nivell amb un botó, poder sortir també amb un botó i mostrar wireframe, és a dir, els detalls de la implementació dels objectes, molt útil per a corregir errors en el videojoc.
- **Publicació del joc a un servidor remot**: aquest ha estat un objectiu secundari que també s'ha assolit. A la adreça http://pfc2011prova1.appspot.com/ és pot comprovar com el videojoc i l'editor funcionen, o haurien de funcionar, ja que les diferències entre navegadors i sistemes operatius pot ser molt gran, sobre qualsevol navegador web modern, és a dir, actualitzat com a molt el darrer any. Cal comentar que Google AppEngine no és una tenda d'aplicacions, tant de moda en els darrers temps, sinó d'un lloc, a vegades anomenat núvol, on poder penjar aplicacions web.

7.2. Possibles ampliacions

En aquest apartar parlarem d'algunes de les funcionalitats que no s'han implementat però que durant el desenvolupament del projecte s'han tingut en compte com a possibilitats i que es podrien afegir en un futur:

- **Selector de nivells**: ara mateix, tant al mode de joc com a l'editor de nivells només es pot carregar el nivell per defecte i quan es vol guardar el nivell editat a l'editor de nivells se sobreescriu el que hi ha carregat a memòria en el moment. Per tant, una millora possible seria implementar una classe que mostrés una llista amb els nivells disponibles a la carpeta "Levels" i que permetés carregar el que se seleccionés. De la mateixa manera, en l'editor de nivells es permetria triar amb quin nom guardar el nivell que s'ha editat i no haver d'esborrar forçosament el nivell que s'ha carregat.
- **Entrada per les canonades**: les canonades de l'estil de Super Mario Bros utilitzades en aquest videojoc no tenen cap tipus d'interacció especial. Seria interessant fer que el jugador pogués entrar-hi i que es carregués un nivell secret que el jugador hauria de passar-se.
- **Afegir més tipus d'element, d'enemics, etc**: es podrien afegir blocs que al ser colpejats donessin punts, objectes o monedes, per exemple. O blocs que es poguessin trencar. També enemics amb altres comportaments, enemics voladors, etc.
- **Poder editar comportaments de blocs en moviment, d'enemics, etc**: l'editor es podria millorar oferint la possibilitat de configurar la trajectòria o els moviments dels blocs dinàmics, o modificant algunes característiques de les reaccions dels enemics.
- **Afegir potenciadors**: un clàssic dels videojocs, els potenciadors o powerups. Són elements que donen noves habilitats al jugador, com més velocitat, habilitat per volar, invencibilitat, etc.
- **Guardar mapes a Google AppEngine**: aquesta és una possibilitat interessant, ja que per ara els nivells es guarden en local, utilitzant localStorage dels navegadors webs. AppEngine ofereix als usuaris un espai de disc que es pot utilitzar per a emmagatzemar qualsevol cosa mitjançant unes interfícies de programació específiques. Així doncs es podrien guardar els nivells remotament, al núvol, i que els usuaris els poguessin carregar des de qualsevol lloc, sense necessitar el seu navegador de casa.
- **Altres millores i detalls**: com mostrar una pantalla amb informació del progrés de la càrrega del joc i dels nivells. O un detall que molt plataformes moderns tenen, com es no fer scrolling immediatament quan es jugador es mou, sinó esperar a que s'hagi apartat dels límits de la pantalla. O també afegir un botó de

turbo per fer que el personatge es pugui moure a dues velocitats. O afegir informació de la versió dels nivells per tal que si canvia el format en un futur es pugui saber i actuar en conseqüència.

7.3. Valoració personal

No serà gaire llarg, però cal fer una petita valoració del que ha significat aquest projecte de final de carrera. Per a algú que va créixer jugant a alguns jocs plataformes com Super Mario Bros 3, Mega Man 3, Joe&Mac i Chip 'n Dale Rescue Rangers a la NES i SkyRoads, Prehistorik i Fury of the Furries a MS-DOS, poder desenvolupar un videojoc plataformes, per molt petit que sigui en comparació amb els mencionats, ha estat molt interessant. La màxima inspiració ha estat Super Mario Bros, òbviament, i ha estat revelador comprovar que hi ha una gran feinada en el desenvolupament d'un videojoc i més tenint en compte com en aquells temps les eines disponibles eren no tant nombroses ni fàcils com avui dia. És evident que se sol treballar en grups nombrosos i que aquí s'ha fet tot en solitari, però és impressionant pensar que se solien programar els jocs en assemblador. Tot i així, la gran quantitat de tasques que s'han de dur a terme, com dissenyar el motor del joc, crear gràfics i sons interessants, crear les intel·ligències de tot tipus d'enemics, crear nivells que suposin un repte per al jugador, donar al personatge un moviment i una resposta a les accions del jugador que l'incitin a seguir jugant, etc. Com hem dit, es tracta d'un projecte individual, cosa que té coses positives com la no discussió de res amb ningú i poder crear el que es vulgui, però té la part negativa d'haver de fer-ho tot un mateix, haver-se de gestionar el temps, etc. Amb tot, ha estat un projecte molt interessant de fer, i molt gratificant veure'n el resultat final.

8. Referències

[1] «PlayN Platform Status,» [En línia]. Available: http://code.google.com/p/playn/wiki/PlatformStatus.

[2] «Platform Game Wikipedia Article,» [En línia]. Available: http://en.wikipedia.org/wiki/Platform_game.

[3] «Extending HTML,» [En línia]. Available: http://ln.hixie.ch/?start=1089635050&count=1.

[4] «Canvas Support,» [En línia]. Available: http://caniuse.com/#feat=canvas.

[5] «WebGL support,» [En línia]. Available: http://caniuse.com/#feat=webgl.

[6] «HTML5 Wikipedia Article,» [En línia]. Available: http://en.wikipedia.org/wiki/HTML5.

[7] «CSS Wikipedia Article,» [En línia]. Available: http://en.wikipedia.org/wiki/Cascading_Style_Sheets.

[8] «SVG Wikipedia Article,» [En línia]. Available: http://en.wikipedia.org/wiki/Svg.

[9] «Web Applications Wikiperdia Article,» [En línia]. Available: http://en.wikipedia.org/wiki/Web_application.

[10] «XMLHttpRequest Wikipedia Article,» [En línia]. Available: http://en.wikipedia.org/wiki/XMLHttpRequest.

[11] «JavaScript Wikipedia Article,» [En línia]. Available: http://en.wikipedia.org/wiki/JavaScript_engine.

[12] «Gladius, a Modular 3D Game Engine for the Web,» [En línia]. Available: https://blog.mozilla.org/labs/2012/04/gladius-a-modular-3d-game-engine-for-the-web/.

[13] «Box2D FAQ,» [En línia]. Available: http://code.google.com/p/box2d/wiki/FAQ#What_units_does_Box2D_use?.

[14] «SVG support,» [En línia]. Available: http://caniuse.com/#feat=svg.

[15] «WebGL Wikipedia Article,» [En línia]. Available: http://en.wikipedia.org/wiki/Webgl.