Creació d'un joc 2d amb Libgdx

Rubén Polo Escudero

Resum— Actualment el sector dels videojocs, tant a nivel local com a nivel mundial es troba en molt bon estat. Això ha estat afavorit per una sèrie de causes, la proliferació de cada vegada més dispositius i més fàcil accés a aquests, smartphones, tablets, consoles, pcs, etc. i també la aparició de cada vegada més empreses independents que creen videojocs cada vegada més originals i a un preu mínim.

Tot això ha ocasionat un gran boom en el sector dels videojocs donant lloc a moltíssimes més facilitats alhora de voler desenvolupar el teu propi videojoc, a Internet es pot trobar una quantitat enorme de contingut per moltíssims frameworks i diferents plataformes, i a més mayoritariament de forma gratuïta.

Aquest projecte s'alimenta d'això, de l'esperit dels jocs independents, utilitzant totes les eines gratuïtes a l'abast i els coneixements assolits al grau per crear un joc desafiant, divertir i gratificant.

Paraules clau—Actor,Body,Tiles,Intel·ligencia Artificial,OpenSource

Abstract—Nowadays gaming industry, both local level as global level is in very good condition, it has been favored by a number of causes, the proliferation of devices and increasingly easier access to them, smartphones, tablets, consoles, pcs... and the emergence of increasingly independent companyies that create the most original games and at mínimum price.

This has caused a boom in the game industry resulting in many more facilities to develop your own game, in the web there is a huge amount of content for many diferent platforms and frameworkds, and free nearly all of them.

This projects is based on the spirit of independent games, using all the free tools available and the knowledge gained to create a challenging, fun and rewarding game.

Index Terms— Actor,Body,Tiles,Artificial Inteligence,OpenSource

1 Introducció

AQUEST treball ha sigut motivat per la passió per els videojocs, i per crear una estructura base per el desenvolupament de videojocs a partir de eines gratuïtes que es troben a la web i utilitzant les tècniques adquirides tant al Grau d'Enginyeria Informàtica, i més concretament a la menció de Computació.

Dins de l'àmbit dels videojocs, el que s'ha desenvolupat es un videojoc de plataformes 2D, d'aspecte clàssic, però adaptat a la jugabilitat trepidant dels temps moderns, creant un joc aleatori, difícil, molt gratificant i sobretot divertit. Aquest projecte es troba fortament influenciat per jocs actuals com Spelunky[7] o Rogue Legacy[8].

Els objectius d'aquest projecte es el disseny, implementació i testing d'un joc anomenat Snow passant per totes les etapes d'aquest, desde l'analisi i el disseny del software adient, estudiant quines seran les tècniques necessàries a aplicar i com es possible optimitzarles per a crear un videojoc dinàmic i que es pugui executar en la majoria d'ordinadors personals, passant també per les etapes d'implementació, entre les que es troba la implementació de l'esquelet, que és la part lògica del videojoc, i la implementació gràfica, que és tota la part visual, finalitzant amb tests tant d'aspecte intern, com d'aspecte extern utilitzant testers.

Aquest article s'estructura de la següent manera: en la secció 2 es troba la metodologia, on s'explicaran totes les eines utilitzades i com ha estat estructurat el projecte. En la secció 3 es troba el desenvolupament, on es mostraran quines han sigut totes les etapes del projecte, començant per la definició de les parts més bàsiques, continuant amb el disseny, i finalitzant amb la implementació i el test del software. En la secció 4 es troba la conclusió, on es donarà una visió de quins han sigut els resultats obtinguts, un repàs dels problemes sorgits al desenvolupament, una opinió personal d'aquest projecte i un apartat de millores.

- *E-mail de contacte: ruben.polo@e-campus.uab.cat*
- *Menció realitzada: Computació*
  Treball tutoritzat per: Enric Martí
- *Curs 2014/15*

2 METODOLOGIA

2.1 Eines utilitzades

Per a la realització d'aquest projecte s'han tingut en compte diferents eines que hi ha per a la creació de videojocs, donant prioritat a totes aquelles eines que fossin gratuïtes o OpenSource. Les eines escollides han sigut:

1. **Libgdx**
2. **Box2d**
3. **Eclipse**
4. **OpenGL**
5. **Tiled**

2.1.1 Libgdx

Framework OpenSource que treballa sobre OpenGL, oferint diversos mètodes que faciliten la programació de videojocs, ja sigui la gestió de les escenes, la creació d'actors, la gestió dels fitxers, etc. L'elecció d'aquesta eina ha vingut donada per ser un dels frameworks més coneguts alhora de crear videojocs en 2d, i a més té una wiki molt completa de totes les seves capacitats.

Libgdx consta de diferents classes en Java que ajuden a la gestió dels elements del videojoc, aquestes són:
- **Actor**: pot ser qualsevol cosa a dins d'una pantalla, en Libgdx tot està format per actors ja sigui texte, un enemic, el propi jugador, etc.
- **Stage**: classe encarregada de gestionar els inputs del jugador i donar-li els outputs adients, aquesta classe incorpora una llista de tots els *Actors* i es la que controla tota la part lògica del videojoc.
- **Screen**: classe encarregada de la implementació de les pantalles i menús del videojoc, que poden variar entre el menú principal, menú d'opcions, pantalla del joc, pantalla de game over, etc.
- **Game**: classe principal de tot joc, encarregada de gestionar les Screens.

2.1.2 Box2d

Llibreria OpenSource que implementa un motor físic 2D, és una de les més conegudes i utilitzades per implementar físiques.

Conté diferents classes per facilitar la creació d'elements i la seva gestió:
- **World**: classe encarregada de gestionar tots els moviments i les físiques, fa un rol similar a la classe *Stage* de Libgdx.
- **Body**: és un cos rígid, que pot tenir qualsevol forma, aquests cossos poden ser tan estàtics (no li afecten les forces i no es mouen), dinàmics (li afecten les forces i es mouen) i cinemàtics (no li afecten les forces i es mouen). A un body també se li pot assignar propietats com la seva densitat, grandària, fricció, gravetat, etc.

2.1.3 Eclipse

Entorn de programació amb el que es treballarà amb el llenguatge Java. S'ha escollit per ser un dels entorns més treballats al Grau, i per ser l'únic que implementa el framework Libgdx.

2.1.4 OpenGL

Una de les llibreries gràfiques més utilitzades pel desenvolupament d'aplicacions visuals.

2.1.5 Tiled

Un Tile és un bloc quadrat que serveix per crear la part gràfica d'un videojoc, com plataformes on saltar, parts de l'escenari, etc.

Tiled és una eina OpenSource que permet crear mapes de tiles amb total llibertat, que després es pot importar fàcilment al videojoc i treballar sobre aquests mapes.

3 DESENVOLUPAMENT

L'estructuració ha estat distribuïda en diverses etapes:

1. **Concepció de la idea de videojoc**
2. **Disseny del software**
3. **Implementació**
4. **Testing**

3.1 Concepció de la idea de videojoc

Definim tres característiques principals d'un videojoc:

1. **Gènere**: el gènere del videojoc es un joc de plataformes del estil de Mario Bros, però enfocat als temps moderns.
2. **GamePlay**: el gameplay es basa en els moviments bàsics amb WASD i l'utilització del ratolí com a eina per matar als enemics disparant projectils. És un gameplay adaptat a les tendències actuals dels videojocs, amb una dificultat molt elevada per la gran quantitat d'enemics que hi ha en pantalla, però una gran repetibilitat, buscant l'aleatorietat per a la creació dels enemics sobre l'escenari, i també donant un punt estratègic fent els elements de l'escenari destructibles.
3. **StoryBoard**: el videojoc es basa en la lluita entre la neu i el foc, per tant l'ambientació busca combinar els dos elements a l'escenari tant en els enemics, jugador, com els elements de l'escenari.

També s'han definit com serien els enemics, jugador, i les regles que segueixen, quins són els seus tipus de moviments, les seves habilitats, l'aspecte gràfic de l'escenari, etc.

Finalment s'ha escollit quina és la plataforma més adequada per a l'implementació, tot i tenir la capacitat de Libgdx de implementar un codi multiplataforma (PC,

Android, Html5). Es va decidir implementar per a PC, ja que ofereix una major jugabilitat que les altres plataformes.

3.2 Disseny del software

L'arquitectura software del videojoc es basa en una sèrie de mòduls com es pot veure a la figura 1:

3.2.1 Game

Aquest mòdul és l'encarregat de gestionar quina és la pantalla que es veurà en cada moment. Per fer això es comunica directament amb la part lògica, que és la que li indica segons unes determinades condicions, com per exemple si el jugador té 0 vides i ha d'ensenyar la pantalla de Game Over, quina es la pantalla corresponent a aquell moment.

3.2.2 Stage

Encarregat de gestionar el bucle principal del videojoc, rebre tots els inputs del jugador, controlar les col·lisions, gestionar els recursos. Per a poder realitzar totes aquestes accions ha de tenir un contacte directe amb els Actors.

Hi poden haver dos tipus de comunicacions entre la part lògica i els Actors.
- **Manual**: quan es rebi un input i aquest input tingui associat una certa acció, notificarà al Actor corresponent per a que realitzi la seva acció.
- **Automàtica**: certs Actors no depenen de cap input i realitzen les seves funcions a cada execució del bucle principal.

3.2.3 Actor

Cada Actor implementa les seves pròpies funcionalitats. Aquestes funcionalitats son cridades des de el mòdul Stage.

Aquets Actors es troben conectats directament amb els Body. Generalment per a cada Actor hi ha un Body, els Actors s'encarreguen de fer la funcionalitat corresponent, i li comunica els resultats tant al Body com al Stage per a mantenir actualitzat l'estat del joc.

També reben dades del mòdul gràfica, el qual l'indica quines son les textures que li corresponen i li proporciona funcionalitats per la seva visualització.

3.2.4 Body

Es l'encarregat de les físiques del videojoc, aquest Body rep les dades de l'Actor on se li indica les dades que li han sigut actualitzades, com pot ser la posició, grandària, si ha col·lisionat, i el Body s'encarrega de fer la simulació de les físiques corresponents.

3.2.5 Gràfica

Aquest mòdul es l'encarregat d'assignar a cada Actor quina es la textura que li correspon, i també li facilita funcionalitats per a poder realitzar el renderitzat d'aquesta.

3.2.6 Usuari

Encarregat de polsar els inputs corresponents per a que el jugador es mogui, es troba comunicat directament amb Stage, i aquest rep les dades i les gestiona.

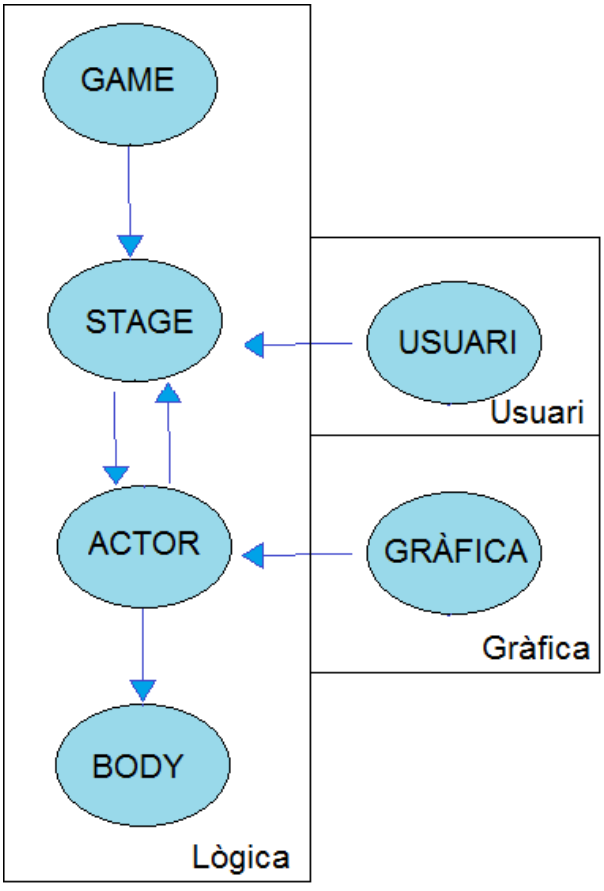

Fig. 1. Esquema en mòduls del videojoc

3.4 Implementació

Etapa en la que es realitza la implementació del concepte de videojoc, seguint el disseny del software planificat i amb la planificació estimada.

El desenvolupament es divideix en dos, una es la creació de l'esquelet del videojoc, la part lògica que inclou també els mòduls de Stage, Actors i Body plantejats al disseny, i l'altre es la creació de tota la part gràfica.

Aquestes parts s'han desenvolupat en paral·lel.

3.4.1 Lògica

Per a la part lògica es va començar implementant l'arquitectura de software planificada, i es va fer un petit test a partir de la creació de dos Actors, el Player i Ground, i també la implementació d'una càmera que segueixi al Player amb uns viewports definits.

Per a la creació dels dos Actors també es van crear els seus Body corresponents. Una de les coses importants sobre els Actors i Body és que formen part de "mòns" diferents. Els Actors de Libgdx i els Body de Box2d, la diferència principal entre aquests és en el sistemas de coordenades que utilitzen i en els sistemes de mesura.

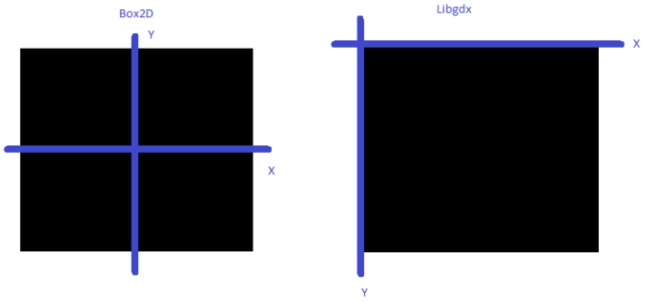

Fig. 2. Sistemas de coordinades de Box2d (esquerra) i Libgdx (dreta)

A la Figura 2 podem observar com l'eix de coordenades de Libgdx comença en la part superior esquerra i la seva y creix cap avall, en canvi per a Box2D el seu eix comença en el centre i la seva y creix cap amunt.

La diferència en el sistema de mesura es que Libgdx treballa amb píxels, i Box2d treballa amb metres.

Per tant es va haver de definir un sistema de conversió per a la transformació de píxels a metres i viceversa, es va definir a partir d'un ratio de píxels/metre i un ratio de metre/píxel per poder fer les transformacions en els dos sentits.

I també un sistema de conversió per canviar el sistema de coordinades de Libgdx.

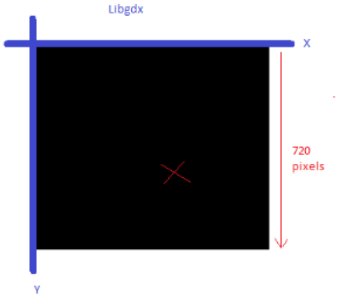

Fig. 3. Exemple d'un punt al sistema de coordenades de Libgdx

En la figura 3 es pot veure com en una llargaria de 720px, el punt marcat en vermell es troba més o menys en la coordenada 400px per al cas de Libgdx, però seguint un sistema en el que la y creix cap amunt, aquesta coordenada seria 320px.

$$\begin{aligned} \text{Box2dCoordinate} &= \text{GameHeight} - \text{LibgdxCoordinate} \\ \text{Box2dCoordinate} &= 720\text{px} - 400\text{px} \\ \text{Box2dCoordinate} &= 320\text{px} \end{aligned}$$

On Box2dCoordinate es la coordenada y del mòn Box2d, GameHeight es l'alçada total de la pantalla i per últim LibgdxCoordinate es la coordenada y del mòn de Libgdx.

Després d'aplicar aquesta fórmula s'aplica el rati de conversió desitjat per pasar de píxels a metres.

$$\begin{aligned} \text{Meters} &= \text{Pixels} * \text{WORLDtobox} \\ \text{Meters} &= 320\text{px} * 0.02\, metres/px \\ \text{Meters} &= 6.4\text{ m} \end{aligned}$$

On Meters és el sistema de mesura de Box2D, Pixels el sistema de mesura de Libgdx i per últim WORLDtobox es el rati de conversió de metres/píxel.

Aquestes conversions son molt necessàries per al desenvolupament del videojoc, ja que proporcionen l'eina adequada per poder calcular la posició exacta d'un element Body en pantalla, i a l'inversa.

El següent pas és la generació d'enemics utilitzant técniques d'IA. S'ha decidit utilitzar l'algoritme A*[4]

La raó d'aquesta elecció ha sigut deguda a que el mapa 2d que es vol desenvolupar està format completament per Tiled[3]. Això proporciona un mapa generat per Tiles, de 32 píxels cadascuna com es pot veure a la Figura 4.

Fig. 4. TileSets

Un monitor estándar actual té normalment una resolució de 1920x1080. En aplicar A* sobre aquest escenari dona una quantitat inmensa de nodes disponibles, el que fa reduir el rendiment. Però treballant amb Tiles, el mapa passa de mesurar-se amb píxels, per mesurar-se amb Tiles de 32x32, o sigui que aconseguim tenir unes 60x32 Tiles en un monitor de 1920x1080, el que equival a una gran disminució de les opcions disponibles.

Aquest escenari fa de l'algoritme A* un algoritme de cerca molt ràpid i precís.

Un cop implementat el A* i testejat el seu funcionament, de forma interna comprobant que per tots els cassos l'algoritme trovaba al jugador, es va comprovar que els resultats no eren molt bons en quant a rendiment, ja que tot i aconseguir trobar al jugador trigava un temps mitjà de uns 100ms, i el bucle principal while de la Figura 6 línea 8 feia unes 700 iteracions fins a trobar l'objectiu.

```
1 function A*(start,goal)
2     closedNodes;
3     openNodes = start;
4     map;
5     neighborNodes;
6     gCost[start] = 0;
```

```
7
8     while openNodes is not empty
9         current = first node in openNodes;
10         if current = goal
11             return getPath(came_from, goal)
12         remove current from openNodes
13         add current to closedNodes
14         for each neighbor in neighborNodes(current)
15             if neighbor in closedNodes
16           continue
17             temp_gCost = gCost[current] +
dist_between(current,neighbor)
18
19             if neighbor not in openNodes or temp_gCost <
gCost[neighbor]
20                 came_from[neighbor] = current;
21                 gCost[neighbor] = temp_gCost;
22                 fCost[neighbor] = gCost[neighbor] + heuristic-
Cost(neighbor, goal);
23                 if neighbor not in openNodes
24                     add neighbor to openNodes
25 return failure
```

Fig. 5. Pseudocodi de l'algoritme A*

Com la idea base és la de enfrontar-se a molts enemics en pantalla i a més de forma ràpida i dinàmica no es pot deixar de banda l'optimització del algoritme A* per a poder aconseguir la sensació de tenir un framerate constant.

Estudiant l'algoritme de la figura 5 es va arribar a la conclusió de que la gestió dels nodes oberts (nodes que poden ser els que ens porten a la solució) que es realitza a la linea 23, era un dels punts on l'algoritme era molt feble, ja que no hi havia cap tipus de ordre, el primer element podrà ser el més ràpid per arribar a l'objectiu, o podrà ser l'ultim.

Per la realització d'aquesta millora es busca que cada vegada que s'inserti un node a la llista, es posi al primer lloc de tots el node que tingui un menor cost, i per tant sigui més probable de ser el millor.

Es van tenir en compte dos mètodes d'ordenació de llistes, el Binary Heap[5] (complexitat O(log n)) o un mètode d'ordenació simple com el *sort* que ofereix la llibreria Collections de Java[6] (complexitat n * (log n)). Les dades extretes es mostraran conjuntament a la Figura 6, on es determina quins son els punts des d'on s'han pres les mesures, com a la Taula 1, que indica quin ha sigut el temps en ms, i el número d'iteracions per a trobar el target en cada posició.

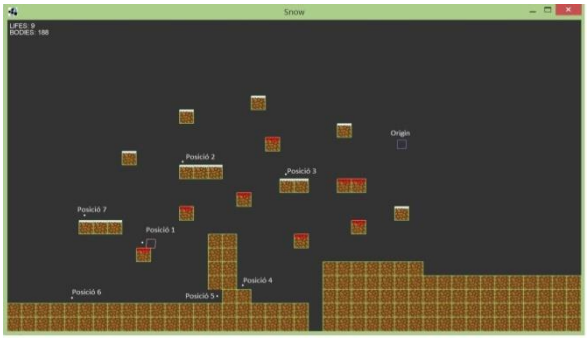

Fig. 6. Punts clau de l'entorn on s'han fet les mesures

TABLE 1
COMPARACIONS ALGORITMES ORDENACIÓ

Position	Java Sort Method		Binary Heap	
	Time (ms)	Nº Iterations to find Target	Time (ms)	Nº Iterations to find Target
1	13	110	12	80
2	12	41	9	23
3	8	23	8	25
4	24	69	35	139
5	33	191	25	332
6	22	170	22	134
7	21	145	19	115
Average	19	107	19	121

Les conclusions extretes són que el algoritme *Binary Heap* és molt més eficient en nombre d'iteracions en els cassos en els que l'objectiu al que es vol arribar sigui llunyà i sense gaires obstacles al seu voltant, però la diferencia de temps envers l'altre mètode no és massa gran.

Llavors tenint en compte l'estructura del videojoc i buscant que el comportament dels enemics sigui molt ràpid i la major part del temps perseguint al jugador a una distància molt petita, es va optar per mantenir el mètode sort que ofereix Java, que tot i tenir una major complexitat computacional, dona uns bons resultats en mapes petits i en els tipus de situacions on el jugador es trobi a prop de molts obstacles.
Si el mapa del joc sigués molt més gran i menys carregat d'obstacles el Binary Heap seria sense dubte el mètode més eficient, però al no tenir un gran nombre de nodes que gestionar dintre de la llista de nodes oberts, la seva eficiència baixa molt.

Un cop generat el primer tipus d'enemic que la seva unica acció és la de perseguir al jugador, s'implementa un segon tipus, un enemic que es trobi sempre estàtic i disparant desde posicions llunyanes al jugador, depenent de la distància a la que es trobin.

Finalment l'últim tipus d'enemic es el Boss, l'enemic més fort del nivell, i que, una vegada s'hagi guanyat s'acaba el joc.

Per a la creació del Boss s'ha tingut en compte la fusió de les funcionalitats dels altres dos enemics. El Boss es troba estàtic en un punt, però és capaç de crear enemics perseguidors i el Boss mateix també llença projectils. La principal diferència és que els llença en molta més quantitat, grandària i velocitat.

La IA del Boss és única també, ja que la seva dificultat augmenta a mesura que baixa la seva vida, millorant totes les seves característiques.

Un cop implementats tots els tipus d'enemics, falta tenir en compte com serà la seva creació dins del joc. Com es busca un joc dinàmic i sobretot molt rejugable, s'ha seguit un procés de generació dinàmica dels enemics depenent de l'escenari, i que a més aquesta generació es pugui

guardar en un fitxer, juntament amb la puntuació aconseguida, per a poder-ho utilitzar en futures partides i així superar el record, o també compartir-ho amb algú i veure qui aconsegueix més puntuació.

Per això s'ha utilitzat el mateix mapa generat que se li envia al algoritme de A*, i així poder conèixer quines són les posicions que es troben lliures i quines es troben bloquejades dins del mapa.

Amb aquestes dades, ja es poden generar tots els enemics, però el problema es la decisió de la quantitat que es volen generar, ja que aquesta quantitat no pot ser estàtica, sino ens trobaríem que amb un mapa molt petit hi hauran la mateixa quantitat d'enemics que amb un mapa gegant. Llavors es fa un càlcul, amb diferents paràmetres extrets del mapa, per a determinar la quantitat d'enemics a generar.

$$rangeOfEnemiesGeneration = ((mapXSize) - (positionPlayer.getX + FreeSpacePlayer))$$
$$numberOfEnemiesToGenerate = rangeOfEnemiesGeneration / RatioTilePerEnemies$$

On $rangeOfEnemiesGeneration$ es el nombre de Tiles que hi ha entre dos punts, ($mapXSize$) és el nombre de Tiles que té el mapa en l'eix X, $positionPlayer.getX$ és la posició inicial en l'eix X del jugador, $RatioTilePerEnemies$ és un ratio que indica el nombre de Tiles per Enemic, i $numberOfEnemiesToGenerate$ és el número final d'enemics que es generarà.

$FreeSpacePlayer$ serveix per a que no es puguin generar enemics immediatament davant del jugador. D'aquesta forma se li dona un espai lliure al jugador per pensar abans de començar. La constant $RatioTilePerEnemies$ ens serveix per escollir quina serà la quantitat de Tiles per cada Enemy, o sigui si hi han 50 Tiles, i el ratio es 2, llavors només hi hauran 25 enemics. Aquesta variable podria ser fàcilment modificable per a escollir el nombre apropiat d'enemics que es volen generar, i també dona peu a una fàcil implementació de diferents nivells de dificultat en els que es poden generar mes o menys enemics.

Una vegada determinada la quantitat d'enemics a generar, es fa un altre càlcul sobre aquest en el que es calcula la quantitat a generar d'enemics de cada tipus. Tenint en compte que del tipus d'enemic Boss només es vol generar un per partida, només cal generar la dels altres dos tipus.

$$numberOfEnemiesType1 = numberOfEnemiesToGenerate * RatioEnemiesType1$$
$$numberOfEnemiesType2 = numberOfEnemiesToGenerate * RatioEnemiesType2$$

Es generen més enemics del tipus perseguidor que dels altres per a incrementar la dificultat i la emoció del joc, ja que es més fàcil evitar un projectil que no esquivar a un enemic que no para de perseguir.

Amb la quantitat i el rati de enemics que es volen generar ja es pot implementar la part de la generació aleatòria. Aquesta part es fa mitjançant unes iteracions, i aplicant paràmetres determinats per a evitar una aglomeració molt gran d'enemics en un únic espai.

Les dades corresponents es guarden a un Array, on es troben les posicions de tots els enemics diferenciats per tipus. Cal guardar l'array en un fitxer per a tenir la funcionalitat que es vol aconseguir. Per a fer-ho s'implementa una serialització amb JSON sobre les dades de l'Array i es parseja a dintre d'un fitxer txt. Per a carregar les dades es fa el mateix però a la inversa, es deserialitzen les dades mitjançant JSON i es passen a l'Array.

Una vegada s'ha implementat aquesta funcionalitat el usuari pot escollir entre un mapa ja guardat, o jugar una partida nova.

Per poder habilitar aquesta opció a l'usuari es crea un menú principal que es pot veure a la figura 7, en el que es pot escollir el mapa sobre el que es vol jugar, i si es vol que es generin automàticament els enemics o es vol utilitzar una generació ja guardada.

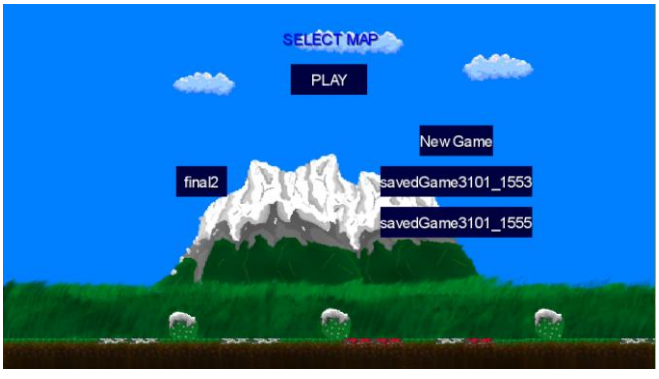

Fig. 7. Menu principal

El jugador pot eliminar els enemics, llançant projectils mitjançant el click del ratolí.

Per a fer això es va calcular quina es la velocitat de la trajectòria entre el punt des d'on dispara, fins on s'ha fet click, es calcula amb la següent fórmula:

$$Velocity = TargetPoint - PlayerCoords$$

On $Velocity$ es la velocitat que tindrà el projectil, $TargetPoint$ són les coordenades on s'ha fet click, $PlayerCoords$ són les coordenades del jugador.

Extraient informació de la fórmula es veu que quant més llunyà estigui el punt on es vulgui disparar, més velocitat tindrà el projectil. Per fer això s'ha normalitzat la veloci-

tat, i llavors s'ha donat un impuls determinat, i aquesta velocitat es calcula amb la següent formula

$$VelocityNormalised = Velocity/Velocity.lenght,$$
$$VelocityResult = VelocityNormalised * X$$

On $VelocityNormalised$ es la velocitat normalitzada del projectil, X es l'impuls donat a la velocitat normalitzada, $VelocityResult$ es la velocitat constant resultant.

Les millores que s'han implementat per l'entorn han estat diverses. Es volia un gameplay on hi hagués una lluita continua entre el foc i el gel. Per tant, el mapa es troba ple de blocs aeris que són tant de foc com de neu. Com que el jugador té el rol de ser una bola de neu que lluita contra els enemics de foc, el seu objectiu ha de ser intentar no tocar els blocs de foc, ja que això li causa danys per estar fet de gel, i ha d'aprofitar-se de les posicions estratègiques dels blocs de gel. D'aquesta manera podrà abordar als enemics saltant de bloc en bloc, i aquests alhora també aprofiten els blocs de foc com a lloc per posicionar-se i llençar projectils al jugador.

Per aprofitar els blocs es va pensar la idea de fer que els blocks aeris fossin destructibles. D'aquesta forma els enemics de foc podrien destruir els blocks de gel, però no li farien res als de foc, i a la inversa pel cas del jugador, que podria destruir els blocs de foc i no ocasionar danys als blocs de gel.

Per implementar aquesta destructibilitat es va tenir en compte un sistema de col·lisions basat en bit masks, en que cada element del joc te una category, i una mask, que serveixen per a definir quines col·lisions es volen i quines es volen ignorar, un exemple es mostra en el següent codi:

```
CATEGORY ICE = 0x0001;
CATEGORY_FIRE = 0x0002;
CATEGORY_STRUCTURE = 0x0004;
CATEGORY PLAYER = 0x0008;
CATEGORY_ICEBULLET = 0x0016;
CATEGORY FIREBULLET = 0x0032;

MASK_FIREBULLET = CATEGORY_STRUCTURE | CATEGO-
RY_ICE | CATEGORY_PLAYER;
MASK_PLAYER = CATEGORY_FIRE | CATEGORY_STRUCTURE |
CATEGORY_ICE;
```

Per a cada element (Ice, Fire, Structure, Player, IceBullet, FireBullet) se li asigna una category de bits, i per a cada element que s'hagi de filtrar les seves col·lisions s'assigna també una mascara de bits. A partir d'aquesta mascara es defineix amb quins elements pot col·lisionar, en el cas de `MASK_FIREBULLET`, l'element que contingui aquesta mascara de bits, només podrà col·lisionar amb Structure, Ice i Player.

Una cop definides totes les col·lisions caldrà implementar un sistema per a poder eliminar els blocks ja creats del mapa, i mantenir sempre actualitzat el mapa que utilitza l'algoritme A*, ja que si en algun moment es destrueix un block i queda una posició lliure al mapa no es vol que els enemics perseguidors continuïn evitant passar per aquesta posició.

Per a realitzar aquesta implementació també es va tenir en compte que els enemics també son destructibles. Per tant també s'hauran de eliminar els seus per a no consumir recursos innecessaris.

Es va crear per tant un Array en el que es van emmagatzemant tots els cossos que es volen destruir, i a cada iteració del joc es destrueixen. Llavors quan es vol eliminar un cos, ja sigui enemic, blocks aeris, projectils o qualsevol altre element només s'ha de emmagatzemar en aquest Array i la funcionalitat creada ja s'encarrega de eliminar aquests cossos per si mateixa.

Per a millorar encara més l'apartat de gameplay, es va implementar que el jugador no pogués tocar els blocks de foc, o que si els toques se li decrementin les vides, i també que si el jugador destrueix un bloc on a sobre hi ha un enemic, aquest caigués a terra i per efecte de la gravetat redueixi també els seus hit points de forma dràstica.

Com a part de les millores a la utilització de recursos es va tenir en compte que un cos que caigués per sota del mapa o qualsevol projectil que sortis de la vista de la càmera es destrueixi. D'aquesta manera es troba enormement optimitzat l'ús de recursos, ja que només es trobaran actuant els elements que es trobin dins dels límits del mapa establerts.

3.4.2 Gràfica

Com a part de la implementació gràfica només fa falta realitzar la creació dels Sprites de tots els Actors del Stage.

Per fer això s'ha seguit una técnica anomenada Pixel Art, que consisteix en fer els dibuixos píxel a píxel, donant-li un estil molt clàssic.

Per poder implementar després els Sprites al videojoc i que cada Body tingués el seu Sprite es va buscar optimitzar aquest funcionament, ja que no es volia carregar un nou Sprite cada vegada que hi hagués un enemic perquè això seria una càrrega de memòria innecessària

Es va crear un TextureAtlas, com es pot veure a la Figura 8, que agrupa totes les Textures en una mateixa imatge, crea un pack i fa que l'accés a aquestes sigui molt senzill, ja que només s'ha de conèixer a quina regió es troba de la imatge generada.

Per tant es genera un .png i un .txt on el .txt indica a quina regió es troba exactament la imatge corresponent, i a més se li assigna un nom que s'utilitza per a poder mapejar de forma fàcil aquesta Textura des de Eclipse.

Fig. 8. Texture Atlas

I el fitxer .txt conté aquestes dades per referenciar les textures:

```
snow_idle1
  xy: 182, 958
  size: 32, 32
snow_jump1
  xy: 182, 992
  size: 32, 30
background
  xy: 2, 2
  size: 1500, 720
bigFireBall_move1
  xy: 148, 724
  size: 48, 48
```

On s'indica el nom assignat a un element (snow_idle1), la posició a la que es troba dins del Atlas (xy: 182, 958) i el seu tamany en píxels (size).

3.5 Testing

Per testejar el joc ha sigut realitzat un testing tant intern com extern.

3.5.1 Testing intern

El testing intern s'ha subdividit en tres parts:

1. **Jugador**: s'han provat totes les funcionalitats en el moment de la implementació veient que no hi ha cap error tant en els moviments com en les seves habilitats.
2. **Enemics**: s'han provat les funcionalitats per separat, només provant un enemic alhora, verificant el correcte funcionament de totes les seves habilitats i la correcta gestió de memòria al eliminar els enemics.
3. **Mapa**: s'han provat les funcionalitats de destructibitat de l'entorn i de la gestió de memòria dels elements que surten fora de la pantalla.

El tests s'han realitzat tots en el moment del desenvolupament d'una nova funcionalitat, sempre intentant aïllar el cas en un principi i veure si funciona bé per separat dels altres elements del joc, i després es prova una vegada en conjunt.

3.5.2 Testing extern

El testing extern s'ha realitzat sobre persones reals, van participar 5 persones, a les que se li es va enviar el joc en un executable i van jugar al joc durant un temps de 15 minuts, i una vegada provat van omplir una plantilla basada en una sèrie de punts:

1. Gràfics
2. Gameplay
3. Duració
4. Diversió

Els resultats finals d'aquest testing han sigut aquests:
- Bon framerate durant tot el joc.
- Gràfics senzills i ben definits amb un estil retro.
- Té una curva de dificultat molt gran al començar a jugar, però es intuïtiu i s'aprén ràpidament.
- Falta de més nivells o powerups per al jugador.
- Tot i ser curt es molt rejugable.
- Falta de velocitat, tant de moviment com de projectils, del jugador.

4 CONCLUSIÓ

S'ha dissenyat un joc de plataformes 2d difícil, on el jugador aprèn a jugar morint molts cop i assimilant les mecàniques del joc a cada derrota.

S'ha dissenyat el software seguint una arquitectura basada en mòduls.

S'ha implementat un sistema per a la generació d'escenaris aleatoris a cada partida depenent del mapa de joc.

S'ha implementat un sistema per guardar les dades del mapa actual, tant les puntuacions com les posicions dels enemics.

S'ha implementat un sistema d'IA per als enemics fent ús d'algoritmes de cerca i optimitzant els resultats.

S'ha implementat destructibilitat a l'escenari del videojoc fent ús d'un sistema de col·lisions basat en mascares de bits.

S'ha realitzat un testing de software tant a nivell intern com a nivell extern.

Un dels problemes que ens hem trobat ha sigut per la diferencia entre els sistemes de coordenades de Libgdx i Box2D, ja que en cada moment que s'hagi de crear un element nou o s'hagi de traduir un click de pantalla a una coordenada en el món de Box2D, s'ha de tenir en compte fer la conversió correcta, i això ha portat alguns problemes al llarg del desenvolupament.

Altre problema ha vingut donat per la càmera, ja que al començament tot es creaba utilitzant una sola càmera i

això portaba problemes si es volia mantenir l'escenari estàtic i només es volia moure el jugador per exemple. Aquest tema es va solucionar utilitzant dos càmeres, una estàtica per a l'escenari, i un altre per els objectes en moviment.

La conclusió d'aquest treball és que ha sigut molt satisfactori tant a nivell personal com a nivell de resultats, trobo que s'ha arribat a realitzar un joc com el previst difícil, ràpid, jugable i divertit com els jocs clàssics que tantes hores de joc m'han donat.

L'arquitectura de disseny ofereix una base molt sòlida per poder fer un altre joc, tot i que sigui d'una temàtica totalment diferent, ja que no es veuria afectat en gran cosa el codi principal del joc.

Com a millores es poden destacar:
- Creació de més nivells de joc amb enemics diferents tant gràficament com a nivell de funcionalitat.
- Possibilitat d'escollir power-ups per al jugador, donant més profunditat al joc, ja que cada jugador escollirà el que més li convingui en cada moment o el que millor s'adapti al seu estil de joc.
- Noves habilitats per al jugador, donant més diversitat d'habilitats e incrementant la diversió i l'estratègia.
- Sistema online en el que tothom juga amb el mateix nivell i es fa un ranking de puntuacions, el que portarà a un sistema competitiu amb premis virtuals per als millors jugadors, millorant així la rejugabilitat.

BIBLIOGRAFIA

[1] ZECHNER, Mario. https://github.com/libgdx/libgdx/wiki https://github.com/libgdx/libgdx/wiki/Box2d , 2013. Data darrer accés, Octubre de 2014.

[2] OEHLKE, Andreas. Learning Libgdx Game Development, Publicat per Packt Publishing Ltd, 2013. Data darrer accés, Octubre de 2014.

[3] LINDEIJER, Thorbjørn. http://www.mapeditor.org/ https://github.com/bjorn/tiled/wiki,2014. Data darrer accés, Novembre de 2014.

[4] LESTER, Patrick. http://www.policyalmanac.org/games/aStarTutorial.htm, 2005. Data darrer accés, Novembre de 2014.

[5] LESTER, Patrick. http://www.policyalmanac.org/games/binaryHeaps.htm,2005 Data darrer accés, Desembre de 2014.

[6] ORACLE. http://docs.oracle.com/javase/tutorial/collections/interfaces/order.html,1995. Data darrer accés, Desembre de 2014.

[7] MOSSMOUTH, LLC. http://www.spelunkyworld.com, 2009-2013. Data darrer accés, Gener de 2015.

[8] CELLAR DOOR GAMES. http://roguelegacy.com/, 2013-2014. Data darrer accés, Gener de 2015.