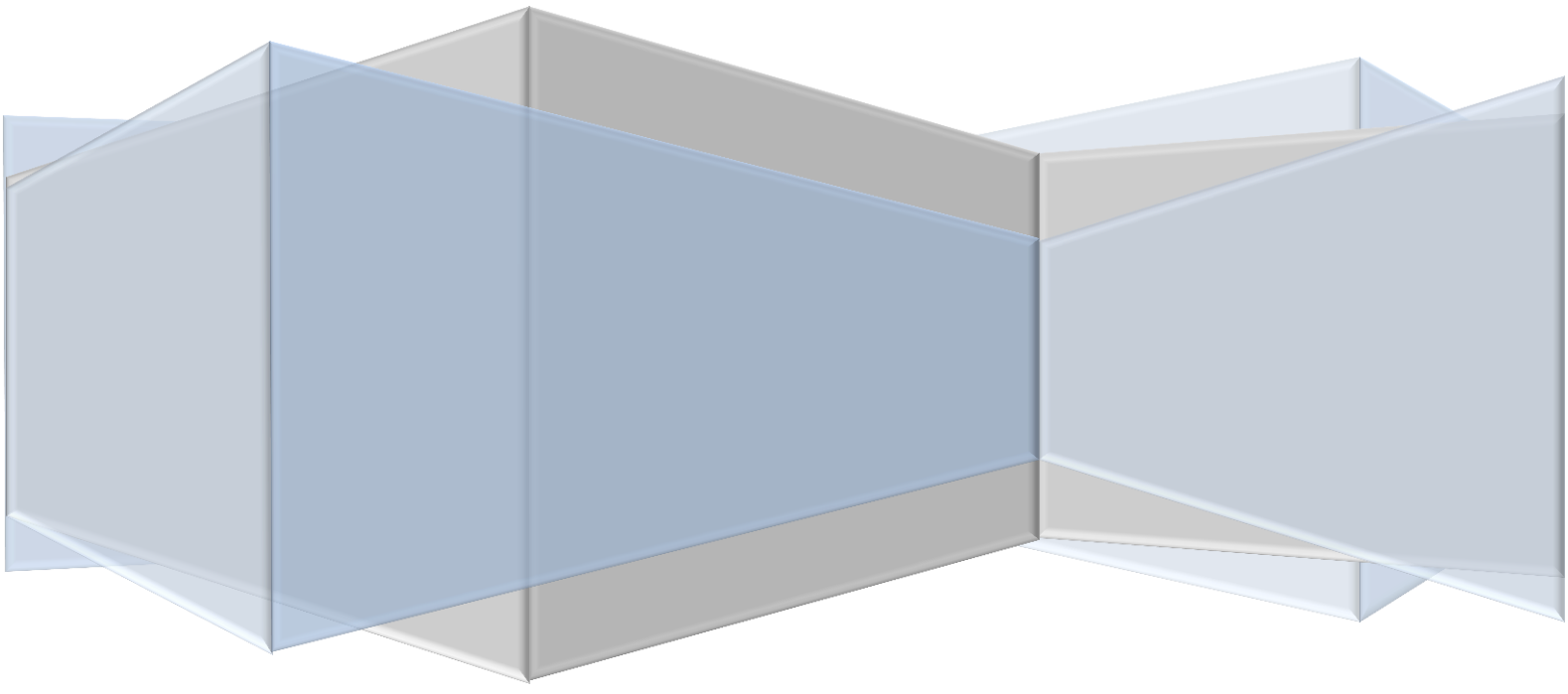
Universitat Oberta de Catalunya

Disseny i desenvolupament d'un videojoc d'artilleria

Treball Final de Carrera

Francesc Xavier Revilla Truyols



Disseny i desenvolupament d'un videojoc d'artilleria

Treball Final de Carrera

Francesc Xavier Revilla Truyols

Resum:

En el present treball de fi de carrera s'ha desenvolupat un videojoc d'artilleria per a dos jugadors amb modalitats de joc local i en xarxa LAN. El joc es basa en clàssics com Artillery, Gorillas o Worms on dos tancs separats per un mur es disparen projectils per torns per tal d'eliminar el contrincant. S'ha dotat de moviment limitat horitzontal als dos vehicles i de moviment independent rotacional per als seus respectius canons, a més d'implementar un sistema de partícules per a les explosions dels projectils a l'impactar amb algun dels elements de joc.

Per a la realització del mateix s'ha fet servir el marc XNA integrat amb l'entorn de desenvolupament Visual Studio 2010 i tot el codi s'ha escrit amb el llenguatge de programació C#. Per a la implementació del mode de joc en xarxa local s'ha decidit no utilitzar les eines que incorpora XNA per aquest propòsit i s'ha optat per a la llibreria de codi obert Lidgren per a Visual Studio amb resultats positius. S'ha aconseguit finalitzar el projecte amb les dues modalitats de joc en estat completament funcional complint així amb els objectius principals marcats en l'inici del desenvolupament.

Resumen:

En el presente trabajo de fin de carrera se ha desarrollado un videojuego de artilleria para dos jugadores con modalidades de juego local y en red LAN. El juego se basa en clásicos como Artillery, Gorillas o Worms donde dos tanques separados por un muro se disparan proyectiles por turnos con el objetivo de eliminar al contrincante. Se ha dotado de movimiento limitado horizontal a los dos vehículos y de movimiento independiente rotacional para sus respectivos cañones, además de implementar un sistema de partículas para las explosiones de los proyectiles al impactar con alguno de los elementos del juego.

Para la realización del mismo se ha usado el marco XNA integrado con el entorno de desarrollo Visual Studio 2010 y todo el código se ha escrito con el lenguaje de programación C#. Para la implementación del modo de juego en red local se ha

decidido no utilizar las herramientas que incorpora XNA para este propósito y se ha optado por la librería de código abierto Lidgren para Visual Studio con resultados positivos. Se ha conseguido finalizar el proyecto con las dos modalidades de juego en estado completamente funcional cumpliendo así con los objetivos principales marcados en el inicio del desarrollo.

Abstract:

A two player Artillery game with local and local network modes has been developed for this present Final degree Project. The game is based on classics like Artillery, Gorillas or Worms, where two tanks separated by a wall compete with each other by throwing projectiles to each other with the main objective of eliminate the opponent. Limited horizontal movement has been implemented for the two vehicles alongside independent rotational movement for their respective cannons, as well as implementing a particle explosion for the impacts made when one of the projectiles impacts with one of the elements of the game.

XNA framework integrated with Visual Studio 2010 IDE has been used for the realization of this game using C# programming language for all the coding. To implement the local network mode, open source Lindgren library for Visual Studio has been used instead of the XNA integrated tools with positive results. This project has been finished with the two main modes fully operational fulfilling the main objectives of this development.

Resum de la memòria:

En aquesta memòria s'explica el procés de creació d'un videojoc amb XNA. En el primer apartat s'introdueix al lector, després d'una breu introducció, en les motivacions que han portat a l'elaboració del mateix i es marquen els principals objectius que es volen aconseguir. Es fa un petit repàs al gènere al qual pertany el joc (Artillery Games) i s'exposen els beneficis que l'el.laboració d'aquest treball suposa per l'autor. Tot aquest apartat s'engloba dins del capítol Introducció.

Tot seguit, en el capítol Estat de l'Art, es fa un breu repàs de les últimes tendències en el desenvolupament de videojocs i es presenta detalladament els rols o funcions que poden realitzar els treballadors d'un estudi de creació de videojocs. Es podrà observar com hi ha tres grups principals de rols en la seva el.laboració (dissenyadors, artistes i programadors) i com tots ells són igual d'importants. També s'exposa una sèrie de eines actuals que es fan servir actualment junt amb els denominats *Engines* i els llenguatges de programació més usats.

En el tercer capítol (Disseny) entrem en matèria ja amb l'el.laboració del joc en sí. Es defineixen les principals característiques i funcionalitats del mateix juntament amb la presentació d'un diagrama de blocs i de classes pensats per a la seva implementació. Es realitza també una breu introducció al flux d'una aplicació realitzada amb XNA i es presenten les classes bàsiques d'aquest entorn que s'han fet servir per al desenvolupament del videojoc.

A continuació es passa a la part més feixuga del treball, el capítol de Implementació on s'explica més detalladament com s'ha realitzat la implementació del joc usant XNA i el llenguatge de programació C#. S'explica, entre d'altres temes, com s'ha realitzat la gestió d'estats del joc, com s'han animat els elements que apareixen per pantalla, com s'ha realitzat la detecció de moviments i com es fa per gestionar una partida en xarxa local.

Per acabar es mostren unes demostracions del joc. Es tanca el treball amb unes conclusions i observacions de cara al futur.

Índex

Índex de figures i il.lustracions

Capítol 1: Introducció

1.1. Motivació i idea original

Des de ben petit, m'han apassionat els videojocs. La primera màquina que vaig tenir va ser un MSX, el qual em va obrir les portes al món de l'oci electrònic. Després va venir l'ordinador Amiga, videoconsoles varies, com Master System, Megadrive, Super Nintendo, Playstation, etc.. fins avui dia que, tot i que molt menys, segueixo jugant i igual d'apassionat que el primer dia.

El paràgraf anterior reflexa la principal motivació de qui escriu aquestes paraules per a embarcar-se en un projecte com aquest, juntament amb la seva passió per programar, la qual també exerceix professionalment en el món de l'ABAP per a una petita consultora de Barcelona. La barreja de la passió pels videojocs i per la programació és la principal impulsora de que s'hagi escollit l'àrea de desenvolupament d'aplicacions de nova generació (en aquest cas un videojoc) com a treball de final de carrera per tal de poder usar-ho com a mètode d'aprenentatge i com a introducció en el món de la producció de videojocs.

Tal com s'explica en el resum inicial d'aquesta memòria, el projecte tracta sobre la creació d'un videojoc amb XNA. A continuació es mostra la idea inicial que es tenia sobre aquest projecte:

Es tracta de la realització d'un videojoc d'estil clàssic (concretament del gènere d'artilleria) on dos jugadors s'enfronten a través de la xarxa. El concepte bàsic és similar al de jocs com WORMS, SCORCHED EARTH o GORILLAS on 2 personatges separats per un terreny amb obstacles, bé sigui una ciutat, un camp o qualsevol altre tipus d'escenari terrestre, es llencen projectils l'un a l'altre tenint en compte la força i l'angle amb què es dispara. També poden haver factors externs com el vent o la inclinació del terreny, amb la qual cosa s'introdueix un factor estratègic força acusat. Cada personatge disposa d'un torn on fer les seves accions. Un cop acabada l'acció del jugador 1, es passa al torn del jugador 2.

Algunes de les possibles opcions de cara al contingut i característiques del joc són les de poder seleccionar diferents tipus de personatges, diferents escenaris, armament variat, diferents tipus de factors externs a part del vent (com per exemple objectes que poden interceptar la ruta del projectil), etc...

Per a la realització del videojoc es dubta entre dues vies. Per una banda desenvolupar-lo per a sistemes Android fent servir l'SDK d'Android y per altra banda desenvolupar-lo per a sistemes Windows fent servir un entorn de programació XNA com per exemple l'XNA Game Studio 4 (disponible per descarregar als recursos de la UOC). Com que XNA fa servir C# com un del seus llenguatges, el qual és molt similar a Java en el que jo hi estic una mica familiaritzat, he decidit desenvolupar el videojoc fent servir aquest entorn.

En quant a aspectes tècnics preliminars, comentar que el joc disposarà de senzills gràfics en 2 dimensions, efectes sonors i al menys una melodia per a la introducció o la pantalla de joc.

La idea al final del projecte es conserva força semblant al que s'havia imaginat, però s'han hagut de simplificar algunes de les idees que es tenien pensades. Finalment no es pot escollir personatge (queden limitats a un parell de tancs) ni escenari (només n'hi ha un) i tampoc s'han inclòs elements o factors aliens al jugador com pot ser el vent. Però per altra banda si que s'ha aconseguit finalitzar la base que es tenia pensada.

1.2. Paraules clau

A continuació es mostra la definició dels tres pilars bàsics en què es sustenta aquest treball. Això és l'entorn i eines que s'han fet servir per la seva el.laboració, el gènere al que pertany el joc i el multi-jugador (una de les funcions bàsiques del joc):

XNA: Es tracta de l'Xbox New Architeture, un conjunt d'eines englobades en un entorn o framework que facilita el desenvolupament de videojocs. La idea és facilitar la feina als programadors, a l'hora que els allibera de la creació de codi repetitiu. Va ser

presentat el 24 de març del 2004 i es va llençar al mercat el 14 de març del 2006. A part del framework, tenim l'XNA Game Studio que és un IDE (entorn de desenvolupament integrat) per al desenvolupament concret de videojocs. Hi han 5 versions i en el meu cas faré servir la versió 4.0 llençada al mercat el 16 de desembre del 2010. El principal llenguatge que es fa servir és el C#, el qual es podria considerar com la resposta de Microsoft al llenguatge Java (tots dos són molt similars). És un llenguatge orientat a objectes, que tot i això, no arriba als graus de complexitat del seu germà gran C++.

Artillery games: Es tracta del gènere on s'engloba el videojoc que vull realitzar. És un gènere que es remunta a l'any 1976 quan a la revista Creative Computing es va publicar un joc programat en BASIC anomenat Artillery on dos tancs lluitaven l'un contra l'altre per torns disparant-se projectils.

S'ha de dir que les primeres versions d'aquests tipus de jocs estaven basades en text i no va ser fins el 1980 on va aparèixer una versió gràfica d'Artillery per l'ordinador Apple II. A partir d'aquí, van començar a aparèixer multitud de versions i variants del joc, arribant al seu punt àlgid amb la sortida del mencionat WORMS on dos equips de cucs guerrillers s'enfronten fent servir algunes de les armes més esbojarrades. Aquesta versió a més a més, permetia el moviment limitat dels personatges durant el seu torn. Finalment, també comentar que hi han versions d'aquest gènere en 3D com Worms 3D, Scorched 3D entre d'altres.

Multijugador online: Es tracta de que dos jugadors humans puguin jugar a un videojoc cadascú des de la seva pròpia màquina ja sigui a través d'una xarxa com internet o a través d'una connexió LAN local (per exemple dos ordinadors en la mateixa sala connectats en xarxa). Principalment tenim dos tipus de modalitats online, asíncrons i síncrons. Els primers solen ser jocs d'estratègia per torns on cada jugador ha d'esperar a que el rival realitzi una acció per poder realitzar la seva. Els segons, funcionen en temps real i tots dos jugadors interactuen al mateix temps. Amb el pas dels anys s'han anat popularitzant molt els videojocs online i tenim gèneres exclusivament orientats a aquesta modalitat de joc com els MMORPG (Massive Multiplayer Online Role Playing Game) tipus World of Warcraft o FPS (First Person Shooter) tipus Modern Warfare.

1.3. Objectius principals

Objectius mínims:

El principal objectiu és aconseguir una versió jugable del joc on hi hagi una pantalla d'introducció o presentació amb un menú on escollir les principals modalitats. Aquestes serien com a mínim una modalitat per 2 jugadors en joc local, una modalitat online o en LAN, un apartat d'opcions o per a sortir del joc i una pantalla de crèdits.

En referència a la partida l'objectiu és aconseguir que se'n pugui jugar una de completa. Aquesta partida acabaria quan l'energia del jugador contrari arribi al mínim (en principi funcionarà per barres o punts de vida). El disseny dels dos jugadors com a mínim serà el de dos tancs de diferent color. Haurà de ser possible llençar els projectils inclinant el canó dins d'un rang de 90graus i aquests projectils s'hauran de comportar de manera mínimament realista (tenint en compte la força amb que es llença, la gravetat, etc...). Hi haurà d'haver al menys un escenari per seleccionar.

Objectius desitjables:

Incloure diferents dissenys de personatges per a escollir. Diferents escenaris i que puguin ser destructibles. Moviment en els personatges durant el seu torn. Idear un sistema de puntuació en funció de la dificultat o precisió del tir i poder accedir a una pantalla amb un rànquing on es mostrin les millors puntuacions aconseguides

1.4. Beneficis

Els principals beneficis que es poden obtenir en la realització d'aquest projecte són:

- Adquisició d'un bon coneixement sobre el desenvolupament de videojocs.
- Oportunitat per posar en pràctica en futures ocasions (ja sigui de manera laboral o de manera lúdica) tota la feina apresa.
- Millorar les habilitats en matèria de programació (sobretot en orientació a objectes) tant en la part tècnica com en la part de disseny analític o funcional.

Per altra banda, no hi ha cap tipus de benefici econòmic potencial ja que no hi ha ni temps ni nivell suficient per poder realitzar un treball que es pugui explotar en aquest sentit. Això no és cap problema ja que no entra dins dels objectius del projecte.

Capítol 2: Estat de l'Art

2.1. Tendències en el desenvolupament de videojocs actuals

Actualment el desenvolupament de videojocs s'ha convertit en una de les fonts d'ingressos més importants del món. El creixement que ha experimentat durant aquests darrers anys ha estat molt elevat. El mercat avui en dia està centrat en les consoles més punteres com XBOX360 i Playstation 3, les consoles més casuals com Wii, els ordinadors compatibles PC i sobretot els dispositius mòbils basats en iOS i Android (i també les tabletes com iPAD i Samsung Galaxy).

Segons les enquestes més recents, trobem que les plataformes on més desenvolupaments es realitzen, tant d'estudis grans com d'estudis més petits, són PS3 i XBOX360 amb una quota molt semblant, com podem veure en el gràfic següent:

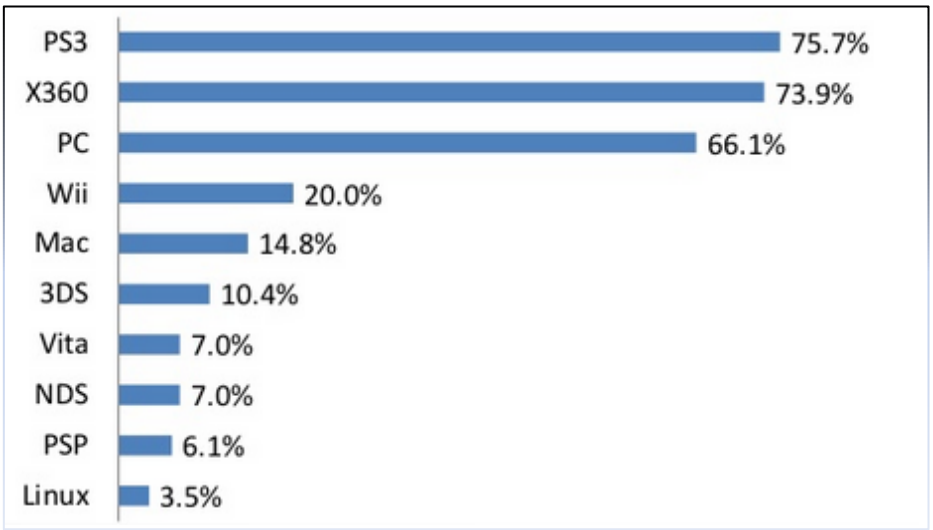


Figura 1: Repartiment en desenvolupament de videojocs l'any 2011

Dins del grup de videojocs amb gran pressupost, aquests van majoritàriament destinats als ordinadors compatibles (PC) i a les consoles de nova generació (PS3, XBOX360 i pròximament WI-U). Com que són màquines amb potència similar (tot i que

els ordinadors compatibles estan a la vanguàrdia tecnològica i amb les últimes targetes gràfiques i configuracions són molt més potents) la conversió dels videojocs entre les 3 plataformes és ràpid, fàcil de fer i maximitza l'audiència potencial a qui van dirigits aquests productes.

Com que els costos de desenvolupament aquests últims any s'han disparat degut a l'increment de recursos necessaris per tal de poder realitzar un joc amb les noves tecnologies (a més potència, més polígons, millors animacions, etc... es tradueix en més personal) el fenomen de jocs multiplataforma (editats pel màxim nombre de màquines diferents possible) ha anat en augment comparat amb temps enrere. Per tant, avui en dia trobem quasi bé els mateixos jocs en les diferents plataformes, i només els estudis first-party (estudis pertanyents a la companyia propietària de la plataforma) poden permetre's realitzar jocs exclusius per a una plataforma en concret. Com a exemple d'aquest últim cas, tenim l'estudi Naughty Dog, pertanyent a SONY que només desenvolupa jocs per a la consola PS3. El cost de desenvolupament d'un videojoc per a aquestes màquines, pot estar entre els 15 i 40 milions de dòlars actualment.
Per altra banda trobem que hi ha hagut un augment enorme en la producció de videojocs (en aquest cas de més baix pressupost) per a plataformes mòbils i per a tablets (ipad, galaxy tab, etc...). La gran inversió que suposa produir actualment un joc per a una de les plataformes més punteres tecnològicament, fa que molts estudis, i fins i tot desenvolupadors independents, s'hagin començat a centrar en la realització de videojocs per a dispositius mòbils i tablets.

2.2. Rols principals en el desenvolupament de videojocs

Dissenyadors:

Són els encarregats d'idear com serà el joc a desenvolupar. Això inclou des de l'elaboració de l'argument fins a la jugabilitat, passant pels personatges i univers on ocórrer l'acció. Cal aclarir que principalment plasmen les idees de manera escrita i

també amb l'ajuda de llenguatges d'scripting (tipus javascript o python) de manera que en fases posteriors, els programadors i artistes els hi donin vida. Per a aconseguir aquest objectiu, els dissenyadors han d'assessorar i supervisar als programadors i artistes durant el procés de producció.

Dins del grup dels dissenyadors trobem també diferents rols:

- ***Dissenyador principal:*** Principalment s'encarreguen de gestionar les tasques, planificació i documentació dels diferents equips de dissenyadors. Recullen les idees generades pels diferents equips de dissenyadors per a confeccionar un document de disseny del videojoc. També han de vetllar perquè el videojoc en producció segueixi amb la major fidelitat possible l'indicat en el document de disseny.

- ***Dissenyador de continguts:*** Són els encarregats d'el.laborar la història, univers i personatges del videojoc. La seva funció es duu a terme principalment durant l'etapa de pre-producció del joc, però sovint han d'editar la feina feta prèviament quan hi ha canvis durant la producció principal.

- ***Dissenyador de mecàniques de joc:*** Aquests dissenyadors s'encarreguen de la jugabilitat del joc (o diferents jugabilitats en funció del gènere). Per exemple en un joc tipus trencaclosques s'encarreguen de definir unes regles o un sistema de joc acord amb el gènere.

- ***Dissenyador de nivells:*** S'encarreguen de dissenyar els escenaris i la disposició dels seus elements (obstacles, plataformes, precipicis, en el cas d'un joc tipus Mario, per exemple). També de la durada dels mateixos, events que hi puguin ocórrer, enemics, etc... sempre en funció del tipus de joc, és clar.

- ***Escriptor:*** Tal com s'intueix, aquests dissenyadors s'encarreguen d'escriure els textos i diàlegs que puguin aparèixer en el joc. És clar que en funció del tipus de joc aquest rol no serà gaire necessari, per exemple en un joc de trencaclosques no fa falta gaire text i normalment no hi apareixen diàlegs. En

canvi, en un joc d'aventures (RPG, aventura gràfica, etc..) és essencial la presència d'un escriptor.

Artistes:

Els artistes donen vida sobre el paper als escenaris, personatges i diferents elements del joc. S'encarreguen de l'estil visual que tindrà de manera que més endavant els programadors ho puguin plasmar en pantalla. Hi ha diferents maneres de realitzar aquest procés, a gust d l'artista. O bé fent servir la manera tradicional amb llapis i paper, o bé fent servir maneres més modernes com una tableta per gràfics on tot el que s'hi dibuixa es veu reflectit en una pantalla d'ordinador.

Podem trobar els següents rols dins d'aquest grup:

- ***Director d'art:*** S'encarrega de gestionar (pressupost, planificació, etc..) el departament d'art a l'hora que es coordina amb l'equip principal de desenvolupament. És també l'encarregat de definir l'estètica del joc tot consultant-ho amb responsables dels dissenyadors i dels programadors.

- ***Artista principal:*** S'encarreguen de gestionar els equips artístics i defineixen els seus mètodes i eines de cara a la creació dels artworks. Controlen també que l'estil d'art utilitzat tingui consistència i qualitat (que es segueixi la línia marcada pels superiors).

- ***Artista conceptual:*** Realitzen tot l'art conceptual a partir de l'estètica definida pel director d'art. Aquest art conceptual consisteix en escenaris, objectes, personatges, moments clau en la història definida pels dissenyadors, etc... Gràcies a aquesta feina, els dissenyadors poden visualitzar les seves idees a l'hora que els altres artistes tenen un camí a seguir.

- ***Modelador:*** Aquests artistes s'encarreguen de crear les versions digitals (sobretot quan parlem de 3d) de l'art conceptual realitzat. S'ajuden d'una

àmplia varietat de programari per al modelatge. També modelen els personatges i escenaris que apareixeran al joc.

- ***Animador:*** S'encarreguen d'animar els objectes, personatges i altres elements que apareguin per pantalla.

Programadors:

Els programadors s'encarreguen de que les idees i conceptes dels dissenyadors i artistes cobrin vida a les pantalles dels ordinadors o consoles. Per a fer-ho es fan servir diversos entorns i llenguatges de programació com C++, Java, Delphi, etc... en funció de l'àrea que s'està treballant. Hi han diferents funcions dins d'aquest grup de treballadors, ja que en la programació d'un videojoc hi han àrees temàtiques ben diferenciades.

Dins d'aquest grup trobem els següents rols:

- ***Programador principal:*** Són els encarregats d'assignar i planificar la feina per als diferents equips de programadors. També supervisen la feina que realitzen aquests equips i fan reunions periòdiques amb els artistes, dissenyadors i productors principals per a tractar els problemes que puguin aparèixer durant el desenvolupament del videojoc.

- ***Programador d'intel.ligència artificial:*** Són els encarregats de programar el comportament dels elements (enemics en un joc d'acció, habitants d'un poble en un joc RPG, etc...) no controlats pel jugador.

- ***Programador de gràfics:*** S'encarreguen d'el.laborar eines que permetin als artistes poder plasmar en pantalla els seus dissenys. Fan servir matemàtiques avançades per a programar algorismes que puguin produir gràfics en 2D o en 3D. Es comuniquen molt sovint amb els artistes per poder determinar la millor manera d'incorporar el seu art dins del joc.

- ***Programador de xarxa:*** La part online (joc a través de la xarxa) és quasi bé obligatòria en tots els jocs d'avui en dia. Aquests programadors s'encarreguen d'escriure codi que ho permeti a l'hora que desenvolupen mesures de seguretat per a evitar que hi hagi jugadors que facin trampes.

- ***Programador de físiques:*** Tot i que sempre ha estat important, cada cop és més important disposar d'un bon sistema de físiques en un joc, sobretot si aquest vol simular la realitat (com en un simulador de conducció, un joc de futbol, etc...). Aquests programadors s'ajuden de codi matemàtiques avançades, per poder crear-les. També s'encarreguen de controlar els comportaments dels diferents objectes, per exemple en un xoc entre ells, el seu moviment degut a la gravetat, etc...

- ***Programador d'eines:*** Són els encarregats de programar eines que facin que el desenvolupament del joc sigui més fàcil. Una de les eines més importants en el desenvolupament és el motor gràfic. Aquest pot estar realitzat internament o bé se'n pot utilitzar un d'extern llicenciat. Aquest tipus d'eines seria un exemple del que poden realitzar aquests programadors.

- ***Programador d'interfície d'usuari:*** Els menús del joc (opcions, pantalla d'inici, pantalla de selecció, etc...) corren a càrrec d'aquests programadors. L'objectiu és que siguin el més intuïtius i fàcils d'usar possible.

Altres rols importants:

Aquests grups de rols són els més abundants en un equip de desenvolupament de videojocs, però també n'hi ha d'altres tant importants com els següents:

- ***Treballadors del so:*** Aquí s'inclou tot el personal encarregat d'el.laborar el so, diàlegs, música, etcètera que apareix en el joc.

- ***Executius:*** Els responsables de l'estudi o companyia, encarregats de les tasques de negoci per tal d'aconseguir vendre el producte o donar llum verda a nous projectes.

- ***Productors:*** Són l'enllaç entre executius i desenvolupadors. S'encarreguen de controlar el pressupost, proporcionar material i eines als desenvolupadors, assegurar-se que el joc es llença al mercat dins de la data estipulada. També s'encarrega de tasques administratives i de planificació de reunions.

- ***Beta testers o provadors de qualitat:*** Es tracta del personal que proba els desenvolupaments abans de sortir al mercat, per tal de trobar errors, bugs, falles gràfiques, problemes de jugabilitat, etc... Així, l'equip de desenvolupament pot posar-se a arreglar aquests problemes un cop detectats.

2.3. Eines principals per al desenvolupament de videojocs

Game Engines:

Es tracta d'un sistema dissenyat per al desenvolupament de videojocs. Això consisteix en un conjunt de programari el qual està dividit en les funcions de motors de renderitzat (ja sigui per gràfics en 2D o per gràfics en 3D), de físiques i de detecció de col.lisions, d'intel.ligència artificial, d'animació, de so, de xarxa, de gestió de memòria, entre d'altres. Amb aquest conjunt d'eines. Podem dir que amb tot això tenim tot el necessari per a desenvolupar un videojoc.

El conjunt d'eines que ofereixen els game engines, acostumen a venir integrats en un entorn de desenvolupament, facilitant així el procés de creació del joc. Hi ha molts tipus d'engines, per a jocs 3D, 2D, gratuïts, llicenciats, d'ús intern (moltes companyies, sobretot les més importants solen fer servir engines desenvolupats per ells mateixos), etc. A més, una de les característiques més importants és la d'independència de

l'engine respecte la plataforma on s'utilitza. És a dir, podem trobar un engine compatible amb ordinadors PC i per videoconsoles.

S'ha de dir que el fenomen dels engines llicenciats (engines creats per un estudi i posats al mercat comercialment per què altres estudis el puguin fer servir) s'ha incrementat durant els últims anys, degut sobretot a l'increment en els costos de desenvolupament. Anys enrere, els engines s'acostumaven a crear i a utilitzar de manera interna (propietaris in-house) per l'estudi desenvolupador, i tot i que avui en dia encara es segueix fent (sobretot pels estudis first-party), com dèiem abans cada cop es fan servir més engines llicenciats.

A continuació mostrem una llista dels engines més populars avui en dia:

Engines llicenciats:

- ***Unity engine:*** Possiblement l'engine més utilitzat actualment contant tot tipus de desenvoupadors (estudis grans tradicionals o estudis independents). Va ser creat per l'estudi Danès Unity Technologies l'any 2005. La versió mes recent és la Unity 3 llençada al mercat el setembre del 2010. En principi ha tingut molt èxit entre els desenvolupadors de videojocs per plataformes mòbils com iPhone o dispositius Android. Un altre dels aspectes que ha afavorit que es converteixi en un dels més usats és el fet que l'any 2009 se'n va llençar una versió gratuïta (amb menys característiques que la versió de pagament), de manera que molts desenvolupadors casuals i independents han pogut crear videojocs fent servir aquest motor. Actualment, l'engine és compatible amb les plataformes Windows, Mac, Unity Web Player, iOS, Android, Nintendo Wii, Playstation 3, XBOX360 i es preveu que també ho sigui per a la nova consola de Nintendo, Nintendo Wii-U.

- ***Unreal engine 3:*** Ha estat un dels més escollits durant els darrers anys per gran part dels estudis grans per a desenvolupar els seus videojocs. És la tercera versió del motor Unreal i va ser creat per la companyia Epic Games l'any 2004. Actualment és compatible amb les plataformes, Windows, Xbox

360, Playstation 3, Android, iOS i Mac OS X, entre d'altres. S'ha anat actualitzant al llarg d'aquests anys incorporant millores per als desenvolupadors. Alguns dels jocs més importants desenvolupats amb aquest motor són: La saga Gears of Wars (XBOX360), Batman Arkham Assylum i Batman Arkham City (XBOX360, PS3 i PC), Enslaved(XBOX360, PS3), entre d'altres.

- ***CryENGINE:*** Desenvolupat per l'estudi Crytec l'any 2004. Actualment van per la versió 3 i és un engine llicenciat.. És usat principalment per a desenvolupar jocs tipus First-person-shooter (acció en primera persona) i actualment és compatible amb les plataformes Windows, XBOX360, Playstation 3, Wii-U, iOS i Android. És un dels engines més potents que hi ha actualment i alguns dels jocs més importants desenvolupats amb aquest, són: La saga Far Cry (Multiplataforma), la saga Crysis (Multiplataforma) i Homefront (Multiplataforma).

- ***Source Engine:*** Desenvolupat per la prestigiosa companyia Valve Corporation per a utilitzar en els seus jocs, va ser creat l'any 2004. No ha tingut mai noves versions, sinó que s'ha anat millorant i ampliant al llarg dels anys. Està dissenyat per a la creació de jocs d'acció en primera persona (first person shooters), però també està pensat per altes geners, com RPG, puzzle, estratègia en temps real, etc... L'engine és compatible amb les plataformes Windows, Mac OS X, Xbox, Xbox360, Playstation3 i Linux. Alguns dels jocs més importants desenvolupats amb aquest motor són: Counter Strike, Half Life 2, Portal 1 i 2, Left 4 Dead 1 i 2, etc...

Engines d'ús intern (in-house):

- ***RAGE:*** Creat per la companyia Rockstar Games, concretament per l'estudi intern Rockstar San Diego, l'any 2006 amb l'objectiu de desenvolupar videojocs per a les noves plataformes d'aquell moment (Playstation 3, XBOX360, Wii) i per a plataformes Windows. En aquest engine va aparèixer per primera vegada l'impressionant motor d'animació Euphoria també creat per Rockstar. Va ser el

primer engine creat per aquesta companyia i entre els jocs més importants on es fa servir tenim: Grand Theft Auto 4 (i pròximament Grand Theft Auto 5), Midnight Club: Los Angeles, Red Dead Redemption i Max Payne 3. Tots ells jocs d'altíssima qualitat.

- ***Crystal Tools:*** Engine d'ús intern creat per Square Enix aproximadament l'any 2007. Es va crear amb la intenció de desenvolupar jocs per a les consoles punteres d'aquell moment Playstation 3 i XBOX360, tot i que avui dia també és compatible amb plataformes Windows i amb la consola Wii de Nintendo. Alguns dels jocs més importants creats amb aquest engine són: Final Fantasy XIII i Final Fantasy XIII-2. També està pensat per els futurs jocs Dragon Quest X i Final Fantasy Versus XIII.

A part d'aquests engines, n'existeixen multitud més, alguns de gratuïts per a facilitar la feina dels desenvolupadors més novells.

Per a l'entorn de programació XNA (el qual serà utilitzat per a la realització del projecte), podem trobar engines com SunBurn Game Engine, TorqueX, Visual3d.net, entre els comercials, i Axiom3d, IceCream (ideal per als jocs 2D), entre d'altres.

Middleware:

Podríem definir middleware com a porcions d'engine (per exemple un motor gràfic o un motor de físiques) amb el qual qualsevol companyia se'n pot ajudar per a desenvolupar videojocs. Un exemple molt clar és el motor de físiques Havok, el qual és usat per una gran quantitat de companyies, tot i que van sortint noves alternatives com Physx o Euphoria (aquest últim creat per la companyia Rockstar per implementar animacions ultra-realistes en els seus jocs).

Algunes de les eines middleware més usades actualment són:

- ***Scaleform:*** Creat per Scaleform Corporation susidiaria de Autodesk, aquest motor serveix per a renderitzar gràfics vectorials per a mostrar interfícies,

d'usuaris, textures animades, etc... Compatible amb les plataformes XBOX360, Playstation 2 i 3, PSP, 3DS, PS Vita, Wii, iOS i Android.

- ***Bink Video:*** Es tracta d'un format de vídeo creat per RAD Game Tools usat en la majoria de plataformes del mercat. Suporta des de resolucions bàsiques de 320x240 fins a resolucions d'alta definició d'avui en dia.

- ***PhysX:*** Es tracta d'un potentíssim motor de físiques desenvolupat per Ageia el 2004. Actualment pertany a la companyia koreana Nvidia i permet als desenvolupadors manejar e implementar les complexes interaccións físiques que es requereixen en un joc d'avui dia.

Wwise: En aquest cas ens trobem davant d'un motor de creació d'àudio desenvolupat per la companyia Audiokinetic. Està pensat per fer-se servir conjuntament amb altres engines complets com l'Unreal Engine, i és compatible amb totes les plataformes de videojocs actuals (PC, Xbox360, Playstation3, Wii (i la pròxima Wii-U), iOS, Android, 3DS, PS Vita, Mac,

2.4. Llenguatges de programació

Normalment s'usen aquests llenguatges amb dos propòsts: Per realitzar el disseny en sí, del qual s'encarreguen els dissenyadors fent servir llenguatges d'scripting i per a realitzar la programació en sí, ja sigui d'eines per a la creació del joc fent servir llenguatges més de baix nivell. Així doncs, trobem que els llenguatges més usats actualment són:

Llenguatges d'scripting:

- ***Lua:*** Creat al 1993 a la Universitat Catòlica Pontifical de Rio de Janeiro al Brasil, és potser el llenguatge d'scripting més usat actualment en el món dels videojocs. És un llenguatge multi-paradigma i permet als programadors incloure espais de noms, classes i d'altres característiques de la orientació a objectes

com l'herència per mitjà del sistema de meta-taules (la manera de guardar la informació que fa servir Lua).

- ***PHP:*** Més conegut per el seu ús en el desenvolupament de pàgines web, aquest llenguatge creat l'any 1995 per Rasmus Lerdorf, és també força usat en el món dels videojocs. Està implementat amb C i igual que Lua és multi-paradigma (concretament suporta programació orientada a objectes, imperativa, entre d'altres).

Llenguatges de programació:

- ***C++:*** Un dels més usats actualment i realment desde fa molts anys. Es un llenguatge orientat a objectes derivat de C (o millor dit és el llenguatge C que suporta el paradigma d'orientació a objectes). La seva flexibilitat i potència per treballar amb la memòria de la màquina (amb l'ús dels famosos punters) el fa ideal per a exprimir i optimitzar l'arquitectura sobre la que s'està treballant.

- ***C#:*** Una evolució de C++, tot i que més aviat és la resposta de C a Java. Molt similar a aquest últim, és semblant també a C++ però més simplificat en el què a maneig de memòria es refereix. Actualment és també un dels més usats i multitud de frameworks i engines el fan servir com a llenguatge base. L'entorn de desenvolupament XNA(amb el que realitzarem el projecte) és un dels que el fa servir.

També tenim d'altres llenguatges que es fan servir, com Java (sobretot per desenvolupar per plataformes Android), Objective C (Per plataformes iOS) i també trobem llenguatges que es poden fer servir tant pel propòsit d'scripting com pel propòsit de creació del joc en sí com Python.

Capítol 3: Disseny

3.1. Especificacions principals del projecte

Principalment el joc s'ha de poder jugar en local i en algun tipus de multi-jugador en xarxa. A causa de la falta de temps i la complicació de implementar un joc en xarxa (lloguer de servidors, etc...) s'opta per una modalitat de joc en xarxa local.

El jugador ha de poder accedir a una pantalla principal des d'on accedirà a la modalitat de joc local o a la de joc en xarxa. Un cop a dins el joc, ha de poder sortir per tornar a la pantalla principal bé quan s'acabi la partida o bé quan es vulgui. Aquesta funcionalitat només estarà implementada en la modalitat de joc local. En la modalitat de joc en xarxa s'acabarà la partida i es tornarà a la pantalla d'inici.

Durant la partida, el joc funciona de la següent manera: Cada jugador té un torn per moure's i disparar. El moviment és limitat a cada torn i es té un indicador de potència que indica amb quanta força surt disparat el projectil. Quan es polsa el botó de disparar, s'activa el comptador de potència i quan es torna a polsar el botó, s'atura el comptador i el projectil es dispara amb aquesta potència. Un cop han disparat, no es poden tornar a moure fins al següent torn. La partida s'acaba quan a un dels jugadors se li acaba l'energia (són un total de 10 projectils encaixats).

Escenari del joc:

L'escenari consisteix en un terra pla amb un mur situat al mig de la pantalla. A cada banda del mur es situa un jugador i no es pot creuar a l'altra banda. Per a poder tocar al jugador contrari amb un projectil, aquest s'ha de llençar per sobre del mur. Per això podrem moure el canó del tanc de manera que el projectil pugui passar per sobre del mur i caure a l'altra banda si pot ser sobre el jugador contrari. S'ha escollit fer-ho així perquè s'ha determinat que és la manera més fàcil i pràctica d'implementar el joc tenint en compte la quantitat de temps disponible per a la seva realització.

3.2. Diagrama de Blocs

Tenint en compte tot lo especificat anteriorment, tenim el següent diagrama de blocs:

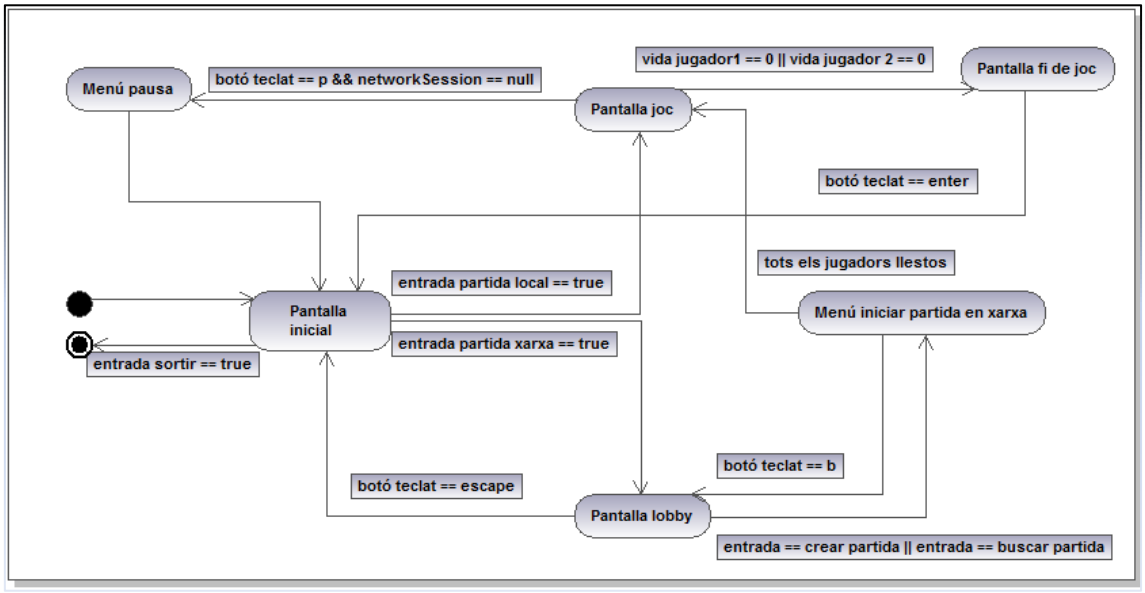


Figura 2: Diagrama d'activitats del projecte

Tal com veiem al diagrama, al començar el programa se'ns presenta la pantalla inicial o de títol. Des d'aquí podem accedir a una partida local que ens porta directament a la pantalla de joc, o bé podem escollir una partida en xarxa amb la qual cosa anirem a la pantalla de Lobby. Si estem en aquesta última pantalla podem accedir al menú per iniciar partida en xarxa bé seleccionant l'entrada crear partida o l'entrada buscar partida. Quan estan tots els jugadors llestos s'accedeix a la pantalla de joc i comença la partida en xarxa. A la pantalla de joc, tant en xarxa com en joc local, si s'acaba la vida d'algun dels dos jugadors, ens apareix la pantalla de fi de joc. En canvi, quan estem en partida local, podem aturar la partida i sortir al menú principal. Des de la pantalla de fi de joc també podem accedir a la pantalla inicial polsant la tecla 'enter'.

3.3. Diagrama de Classes:

Les classes principals que es crearan per fer servir en el joc, apart de les que proporcioni el framework XNA, es poden veure en el següent diagrama de classes (fent zoom al document s'aprecia millor):

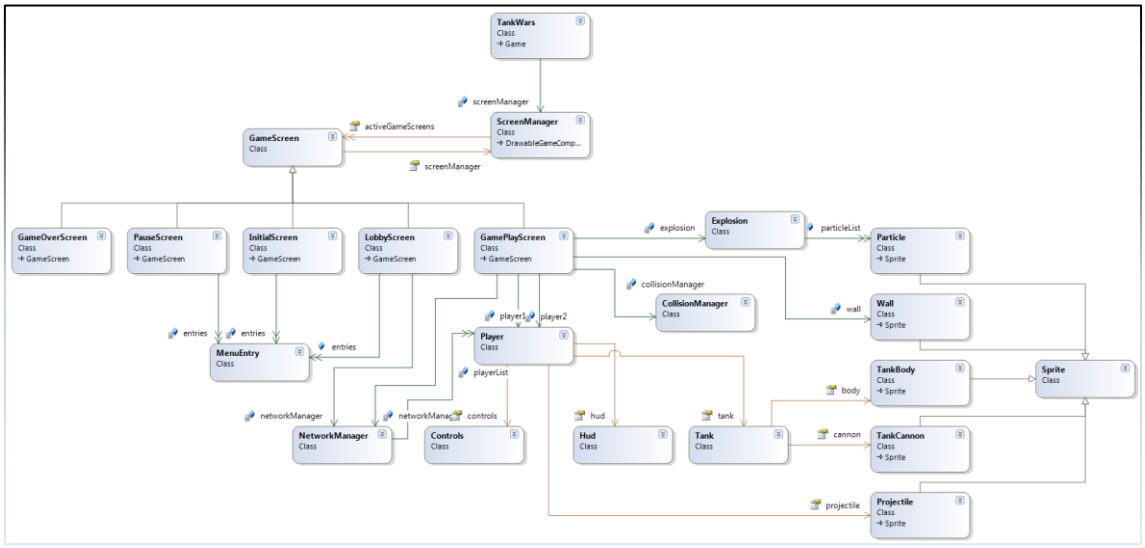


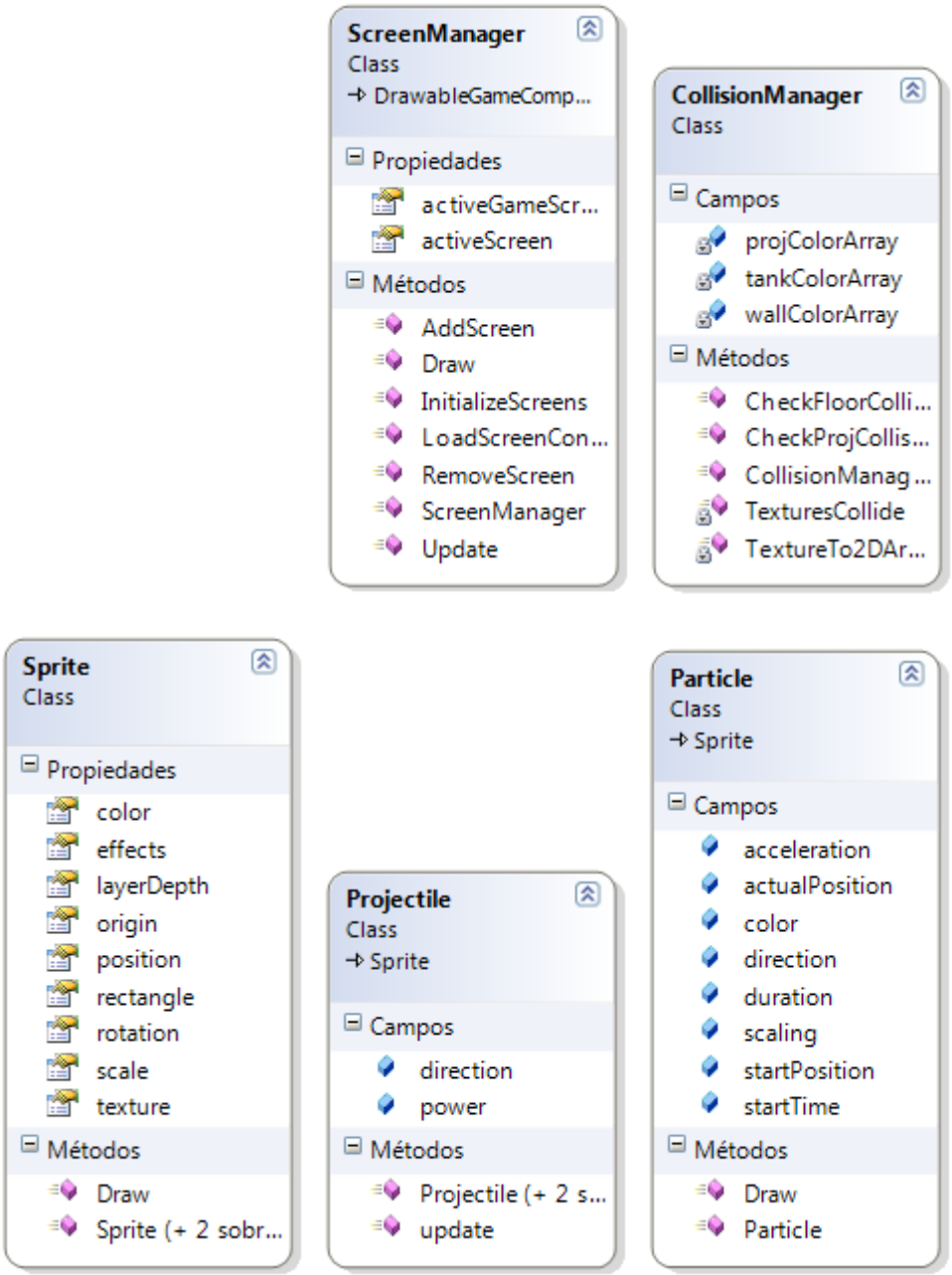
Figura 3: Diagrama de classes

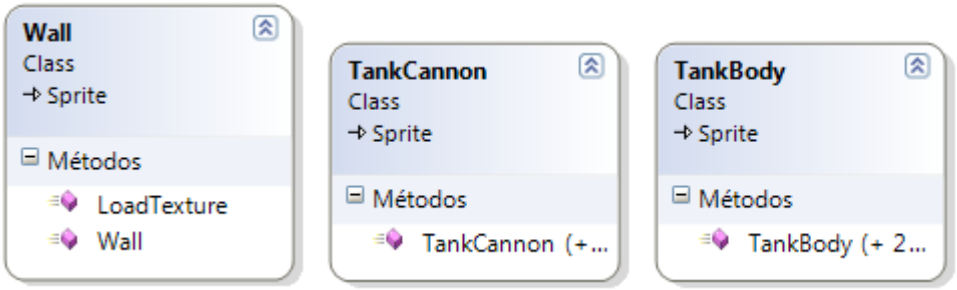
Tot seguit s'expliquen les relacions entre les classes que componen el diagrama y una breu descripció de la seva funció:

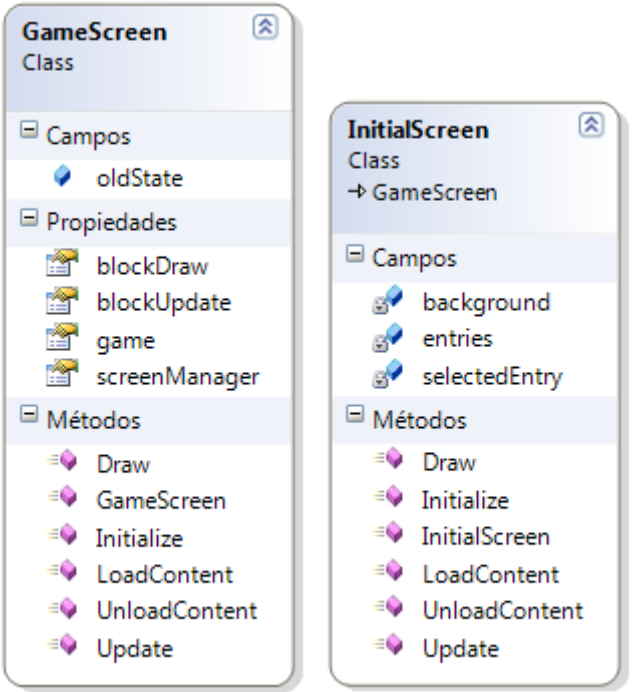
- **TankWars**: És la classe principal i en la qual arranca el joc. Controla el flux principal (inicialització, actualització i presentació per pantalla) i conté un *ScreenManager*. Aquesta classe hereda de la classe *Microsoft.Xna.Framework.Game*.
- **ScreenManager**: S'encarrega de gestionar les diferents pantalles que formen el joc. Està format per una llista de *GameScreen*.
- **GameScreen**: S'encarrega de gestionar tot el que passa en una determinada pantalla de joc. Conté un *ScreenManager* i té 5 classes filles;
    - **GameOverScreen**: Pantalla de final de partida.
    - **PauseScreen**: Pantalla de pausa. Conté una llista de *MenuEntry*.

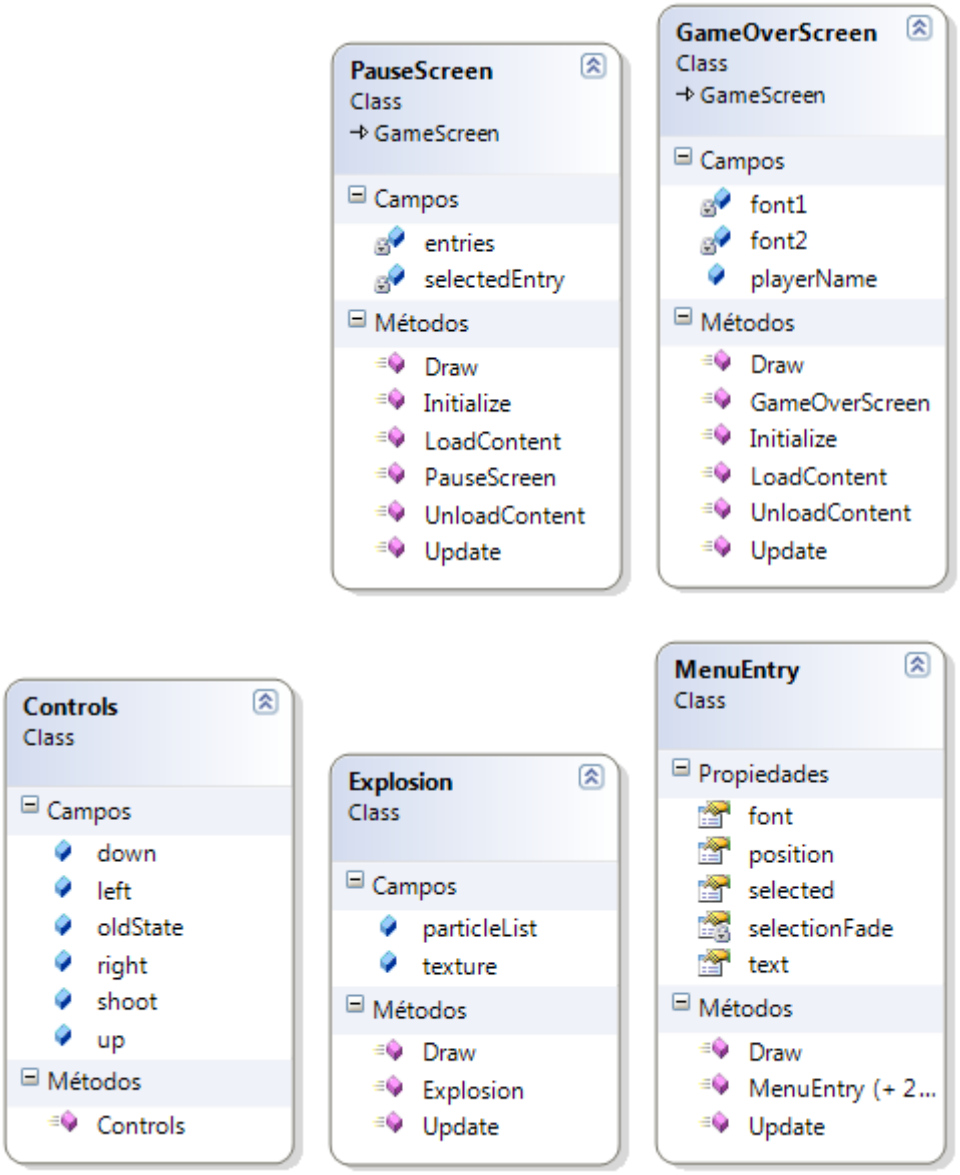
  - **InitialScreen**: Pantalla de títol o inici del joc. Conté una llista de *MenuEntry*.
  - **LobbyScreen**: Pantalla per als preparatius online. Conté una llista de *MenuEntry* i un *NetworkManager*.
  - **GamePlaySreen**: Pantalla del joc principal on s'executa la partida. Conté dos *Player*, un *CollisionManager*, un *NetworkManager*, un *Explosion* i un *Wall*.
- **CollisionManager**: S'encarrega de gestionar les col.lisions entre els diferents objectes de la partida.
- **NetworkManager**: S'encarrega de gestionar els aspectes relacionats amb les connexions de xarxa.
- **Explosion**: Gestiona tot el relacionat amb les explosions del joc. Conté una llista de *Particle*.
- **Player**: Gestiona tota la informació relacionada amb un jugador. Conté un Controls, un *Hud*, un *Tank* i un *Projectile*.
- **Controls**: Gestiona la informació relativa als controls d'un jugador.
- **Hud**: Gestiona la informació de pantalla d'un jugador (per exemple la vida, potència, puntuació etc... d'un jugador).
- **Tank**: El tanc que controla el jugador. Està format per un *TankBody* i per un *TankCannon*.
- **Sprite**: És la classe base per a la representació d'un sprite (dibuix) del joc. Té 5 classes filles:
  - **Particle**: Es tracta d'un sprite que representa una particula d'explosió. Amb unes quantes partícules es pot formar una explosió.
  - **Wall**: Un sprite estàtic amb forma de mur.
  - **TankBody**: Un sprite que representa el cos d'un tanc.
  - **TankCannon**: Un sprite que representa el canó d'un tanc.
  - **Projectile**: Un sprite que representa un projectil disparat per un tanc.

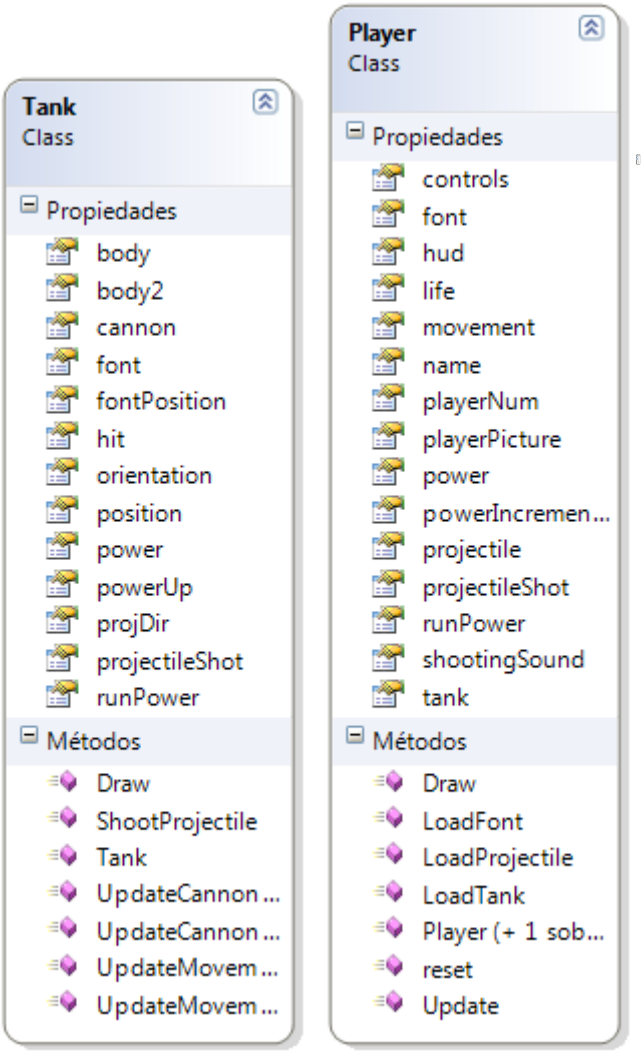
A continuació veiem els atributs i mètodes de totes les classes:




Wall
Class
Sprite
Métodos
LoadTexture
Wall
TankCannon
Class
Sprite
Métodos
TankCannon (+...
TankBody
Class
Sprite
Métodos
TankBody (+ 2...


GameScreen
Class
Campos
oldState
Propiedades
blockDraw
blockUpdate
game
screenManager
Métodos
Draw
GameScreen
Initialize
LoadContent
UnloadContent
Update
InitialScreen
Class
GameScreen
Campos
background
entries
selectedEntry
Métodos
Draw
Initialize
InitialScreen
LoadContent
UnloadContent
Update


PauseScreen
Class
GameScreen
Campos
entries
selectedEntry
Métodos
Draw
Initialize
LoadContent
PauseScreen
UnloadContent
Update
GameOverScreen
Class
GameScreen
Campos
font1
font2
playerName
Métodos
Draw
GameOverScreen
Initialize
LoadContent
UnloadContent
Update
Controls
Class
Campos
down
left
oldState
right
shoot
up
Métodos
Controls
Explosion
Class
Campos
particleList
texture
Métodos
Draw
Explosion
Update
MenuEntry
Class
Propiedades
font
position
selected
selectionFade
text
Métodos
Draw
MenuEntry (+ 2...
Update



3.4. Flux d'un Joc XNA:

La classe principal en un projecte desenvolupat en XNA és la classe *Microsoft.Xna.Framework.Game*. Aquesta classe conté entre d'altres 5 mètodes, que podem sobreescriure per adaptar-los a les nostres necessitats, essencials pel funcionament del programa. Aquests són:

- Initialize(): La seva funció és la d'inicialitzar els atributs de la classe que creguem convenients. S'executa una sola vegada a l'iniciar el joc.

- LoadContent(): Aquí és on s'han de carregar contingut extern (gràfics com textures per sprite, fitxers d'àudio, etc...) per a poder-ho utilitzar en el joc. S'executa una sola vegada a l'iniciar el joc.
- UnloadContent(): En aquest mètode s'eliminen les referències carregades en el mètode LoadContent(). S'executa una sola vegada al finalitzar el joc.
- Update(GameTime gameTime): Un dels mètodes principals on s'actualitzen dades del joc en una freqüència de 60 Hz (cada segon es passa 60 vegades pel mètode. S'executa en bucle indefinidament fins que es surt del joc.
- Draw(GameTime gameTime): L'altre mètode principal on es dibuixa per pantalla tot allò que li diguem. Ho fa també en una freqüència de 60 Hz i s'executa en bucle indefinidament fins que es surt del joc.

Aquest flux s'executa seguint el següent diagrama:

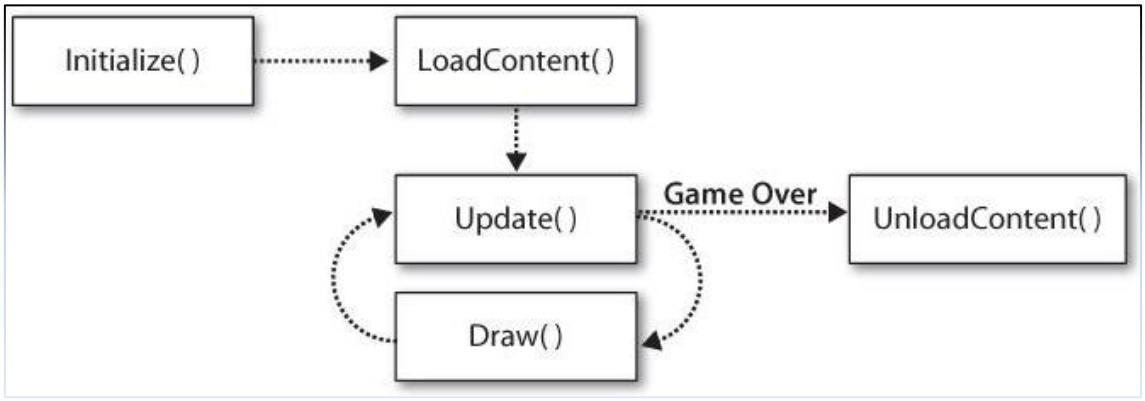


Figura 4: Flux d'un joc XNA

Així doncs, s'ha basat el disseny del joc tenint en compte aquest funcionament. Com es veu en el diagrama de classes mostrat anteriorment, tenim que la classe principal és la classe TankWars. Aquesta classe hereda de Microsoft.Xna.Framework.Game i s'adaptarà a les necessitats del projecte.

La idea és que hi hagi una màxima independència dins de cada classe (que cadascuna es faci la seva feina, com carregar les textures i elements que li pertanyin, etc...) sempre que es pugui. Hi haurà casos que aquesta independència degut al disseny escollit no serà possible.

3.5. Classes bàsiques en XNA:

A part de la classe Game, per a l'el.laboració d'un joc en XNA són essencials les següents classes, les quals expliquem el seu funcionament tot seguit:

- **GraphicsDeviceManager:** S'encarrega de la configuració i gestió del dispositiu de gràfics.
- **GraphicsDevice:** Realitza renderitzat bàsic, crea recursos, maneja variables a nivell de sistema, ajusta els nivells de gamma i crea shaders (que en quest projecte no es fan servir).
- **SpriteBatch:** Permet que un grup d'sprites o imatges es dibuixin per pantalla fent servir la mateixa configuració. S'acostuma a iniciar amb el mètode Begin(), per tot seguit dibuixar tot el que vulguem amb el mètode Draw(), i es tanca el procés amb el mètode End().
- **GameTime:** Guarda l'estat del temps del joc, i o ho pot expressar tant en temps real com en temps de joc.
- **ContentManager:** És el gestor encarregat de carregar textures, sons i altres elements cap al programa.

Capítol 4: Implementació

4.1. Gestió d'estats del joc

Un cop vist el diagrama d'activitats i el diagrama de classes, passem a explicar el funcionament bàsic de la gestió d'estats del joc. El fil principal el porta la classe *TankWars* que segueix el flux explicat en el punt anterior. Per tal de poder gestionar els estats del joc s'ha decidit d'usar un gestor de pantalles. Es tracta de la classe *ScreenManager* en la qual es guarden totes les pantalles actives del joc. Aleshores, un cop iniciada la classe principal, dins a cada mètode es crida la classe *ScreenManager* perquè executi la seva versió d'aquest mètode tal com veiem en el codi corresponent a la classe *TankWars*:

```
public class TankWars : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    GraphicsDevice device;
    SpriteBatch spriteBatch;
    ScreenManager screenManager;

    public TankWars()
    {
        this.graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
        Components.Add(new GamerServicesComponent(this));
        this.screenManager = new ScreenManager(this);
    }

    protected override void Initialize()
    {
        this.graphics.PreferredBackBufferWidth = 1200;
        this.graphics.PreferredBackBufferHeight = 500;
        this.graphics.IsFullScreen = false;
        this.graphics.ApplyChanges();
        this.spriteBatch = new SpriteBatch(GraphicsDevice);
        this.device = graphics.GraphicsDevice;
        this.Window.Title = "Tank Wars";
        InitialScreen initialScreen = new InitialScreen(this.screenManager);
        this.screenManager.AddScreen(initialScreen);
        this.screenManager.InitializeScreens();
        base.Initialize();
    }

    protected override void LoadContent()
    {
```

```
            this.screenManager.LoadScreenContent(Content);
        }

        protected override void UnloadContent()
        {
        }

        protected override void Update(GameTime gameTime)
        {
            this.screenManager.Update(gameTime, Content, this.device);
            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            this.screenManager.Draw(gameTime, spriteBatch, device);
            base.Draw(gameTime);
        }
    }
```

Com que *screenManager* pot tenir més d'una pantalla, en els seus mètodes d'inicialització, actualització, càrrega de continguts i presentació, es fa un recorregut a la llista de pantalles actives que té i es processen, tal com podem veure en el següent fragment (pertanyent al mètode Draw()):

```
        public void Draw(GameTime gameTime,
                         SpriteBatch spriteBatch,
                         GraphicsDevice device)
        {
            foreach (GameScreen screen in this.activeGameScreens)
            {
                if (screen.blockDraw == false)
                {
                    screen.Draw(spriteBatch, device, gameTime);
                }
            }
        }
```

Veiem que sempre s'executarà el codi corresponent a cada pantalla quan es recorri el llistat. Ara bé, ens pot interessar mantenir una pantalla en el llistat del gestor, però que no es dibuixi o no s'actualitzi. Per això en la classe *GameScreen*, de la qual hereden les altres pantalles, hi tenim dos atributs booleans que indiquen si es pot actualitzar o dibuixar (blockUpdate i blockDraw).

Ara ve una de les parts més delicades. Com passem d'un estat a un altre? En aquest cas s'ha decidit implementar la funcionalitat en les pròpies pantalles. Per a fer-ho, s'assigna com a atribut a la classe *GameScreen*, un objecte *ScreenManager*. Així,

quan estiguem dins d'una pantalla, podrem dir al gestor de pantalles que posi o tregui les pantalles que es cregui convenient. Per exemple, Veiem en el codi de la classe *TankWars*, com l'*ScreenManager* s'inicialitza amb la pantalla inicial *InitialScreen*. A partir d'ara, des d'aquesta classe es controlarà si es vol passar a un altre estat (per exemple iniciar una partida local). En la següent mostra de codi veiem com ho fa dins del seu mètode Update():

```
if (this.oldState.IsKeyUp(Keys.Enter)
    && keyboard.IsKeyDown(Keys.Enter))
{
    if (this.entries[selectedEntry].text == "Local Game")
    {
        GamePlayScreen screen =
               new GamePlayScreen(this.screenManager);
        screen.Initialize();
        screen.LoadContent(content);
        this.screenManager.AddScreen(screen);
        this.screenManager.activeGameScreens.Remove(this);
        Thread.Sleep(1000);
    }
```

El codi mira si estem situats sobre l'entrada joc local i si s'ha polsat la tecla 'enter'. En cas afirmatiu es crea una pantalla de joc i s'afegeix a la llista del gestor, després d'inicialitzar-la i de carregar-hi el contingut. Seguidament, com ja no volem per res la pantalla inicial, la traiem del llistat perquè només es processi el nou estat (partida local). A partir d'ara només es processarà la pantalla de joc fins que s'acabi la partida o se'n vulgui sortir, moment en el qual es procedirà a realitzar un procés similar al què hem vist.

En aquest cas, i en la majoria del programa, només es té una pantalla en el llistat del gestor. Tot i això, en el cas de l'estat de pausa, o en el de final de joc, es tenen dues pantalles per tal que es pugui seguir veient la pantalla de joc i a més es vegi superposada a sobre la pantalla de pausa amb les opcions de continuar o sortir. Per a evitar que la pantalla de joc es segueixi actualitzant es fa ús d'un dels atributs de la classe *GameScreen*, blockUpdate (posant-lo a true), que hem mencionat anteriorment. D'aquesta manera, quan el gestor de pantalles passi pel mètode d'actualitzar, només actualitzarà les pantalles de la llista que tinguin aquest atribut posat a false.

4.2. Funcionament del joc principal

Un cop vist com es gestionen els estats passem a veure el funcionament principal d'una partida. L'encarregat de gestionar el joc aquí és la classe *GamePlayScreen*. Aquesta classe conté un CollisionManager (encarregat de la gestió de col.lisions), dos Player (els jugadors de la partida), un Wall i una Explosion (objecte que representa una explosió). A part també té les textures de l'escenàri i d'altres elements com el mur i les explosions. Altres elements com els tancs i els projectils s'inicialitzen i es carreguen dins la classe Player.

Un cop inicialitzats tos els elements, i seguint el flux que ens marca el gestor de pantalles, s'entra en el bucle d'actualització i presentació fins que s'acabi la partida o s'aturi el joc i s'indiqui que es vol sortir. En el següent fragment de codi veiem com funciona el mètode LocalUpdate() (mètode per gestionar la modalitat local de joc) d'aquesta classe:

```
public override void LocalUpdate(GameTime gameTime,
                                 ContentManager content,
                                 GraphicsDevice graphics)
{

    KeyboardState keyboard = Keyboard.GetState();
    // Per sortir del joc
    if (this.end == false)
    {
        if (keyboard.IsKeyDown(Keys.P))
        {
            PauseScreen screen = new PauseScreen(this.screenManager);
            screen.Initialize();
            screen.LoadContent(content);
            this.screenManager.AddScreen(screen);
            this.blockUpdate = true;
        }

        //Actualització de les accions del jugador 1
        if (this.playerTurn == 1)
        {
            this.player1.Update(keyboard, gameTime);
        }

        //Actualització de les accions del jugador 2
        if (this.playerTurn == 2)
        {
            this.player2.Update(keyboard, gameTime);
        }
```

```
//Detecció de col.lisions en cas d'un projectil disparat pel
   jugador 1
//Si hi ha col.lisió es canvia de torn
if (this.player1.projectileShot == true)
{
    if (checkCollision(this.player1, this.player2, gameTime)
        == true)
    {
        this.playerTurn = 2;
    }
}

//Detecció de col.lisions en cas d'un projectil disparat pel
   jugador 2
if (this.player2.projectileShot == true)
{
    if (checkCollision(this.player2, this.player1, gameTime)
        == true)
    {
        this.playerTurn = 1;
    }
}

if (this.player1.life <= 0)
{
    this.explosion =
    new Explosion(this.player1.tank.body.position,
                  15,
                  120.0f,
                  4000.0f,
                  gameTime,
                  Color.Red,
                  this.particleExplosionTexture);
    this.player1.reset();
    this.player2.reset();
    this.end = true;
    this.winner = this.player2.name;
    this.playerTurn = 0;
}

if (this.player2.life <= 0)
{
    this.explosion =
    new Explosion(this.player2.tank.body.position,
                  15,
                  120.0f,
                  4000.0f,
                  gameTime,
                  Color.Red,
                  this.particleExplosionTexture);
    this.player2.reset();
    this.player1.reset();
    this.end = true;
    this.winner = this.player1.name;
    this.playerTurn = 0;
}
```

```
            }

            if (this.explosion != null)
            {
                if (this.explosion.particleList.Count > 0)
                {
                    this.explosion.Update(gameTime);
                }
                else
                {
                    if (this.end == true)
                    {
                        GameOverScreen screen =
                         new GameOverScreen(this.screenManager, this.winner);
                        screen.Initialize();
                        screen.LoadContent(content);
                        this.screenManager.AddScreen(screen);
                        this.blockUpdate = true;
                        Thread.Sleep(1000);
                    }
                }
            }
        }
```

Aquest mètode gestiona l'estat principal de la partida. Primer mira quin és l'input del teclat i actualitza els jugadors(si és el seu torn) amb aquest input (en el mètode de la classe *Player*, update() on es controla el moviment del tanc i s'hi es dispara algun projectil). Tot seguit mira si algun dels jugadors ha disparat un projectil i en cas afirmatiu es procedeix a la detecció de col.lisions. Aquest mètode (checkCollision()) mira si el projectil col.lisiona amb els elements de la pantalla (en aquest cas el propi jugador, el jugador rival, el mur central i el terra). El mètode també crea una explosió nova en funció de la col.lisió i actualitza els nivells de vida dels jugadors en cas de rebre l'impacte. Finalment mira si algun dels jugadors s'ha quedat sense vida per llençar l'explosió final i quan aquesta acaba, si la variable end val true, es passa a la pantalla de final de joc.

Per altre banda el mètode Draw té la següent forma:

```
        public override void Draw(SpriteBatch spriteBatch,
                                  GraphicsDevice device,
                                  GameTime gameTime)
        {
            spriteBatch.Begin();
            Rectangle screenRectangle1 =
              new Rectangle(0,
                            0,
                            device.PresentationParameters.BackBufferWidth,
```

```
                device.PresentationParameters.BackBufferHeight);
        Rectangle screenRectangle2 =
          new Rectangle(0,
                0,
                device.PresentationParameters.BackBufferWidth,
                device.PresentationParameters.BackBufferHeight);
        spriteBatch.Draw(this.backGround, screenRectangle1, Color.White);
        spriteBatch.Draw(this.terrainTexture, screenRectangle2, Color.Gray);
        this.wall.Draw(spriteBatch);
        if (this.player1.life > 0)
        {
            this.player1.Draw(spriteBatch);
        }
        if (this.player2.life > 0)
        {
            this.player2.Draw(spriteBatch);
        }
        spriteBatch.End();
        if (this.explosion != null)
        {
            spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.Additive);
            explosion.Draw(spriteBatch);
            spriteBatch.End();
        }
    }
```

En aquest cas veiem que es dibuixen tots els elements, menys els jugadors que s'han quedat sense vida i les explosions que encara no s'han creat.

4.3. Gestió de menús

Algunes pantalles del projecte, com la pantalla d'inici, el lobby per al joc en xarxa, o la pantalla de pausa, tenen una sèrie d'entrades per seleccionar. Per a fer-ho s'ha creat la classe *MenuEntry* per aquest projecte. Aquestes pantalles contenen una llista d'entrades (MenuEntry) i un atribut selectedEntry que indica quina és la entrada seleccionada de la llista mitjançant un enter que fa d'índex. Quan una entrada està seleccionada, aquesta apareix d'un altre color i va creixent i decreixent creant un efecte d'atenció.

4.4. Moviment dels elements en la pantalla de joc

En aquest joc podem dir que només hi ha 3 elements mòbils si només tenim en compte un dels tancs que apareixen per pantalla.

Moviment del tanc:

Aquest element està format per un objecte *TankBody* i per un objecte *TankCannon*. La idea principal és que el tanc es pugui moure dreta i esquerra, i el seu canó es pugui moure per apuntar. Com són dos animacions diferents, es fan servir aquests dos objectes per a crear el tanc. Quan moguem el tanc a l'esquerra o a la dreta el canó també es mourà en aquella direcció donant la impressió que tots dos elements en formen un de sol. Per altra banda podrem rotar el canó fins a 90º. A continuació veiem els elements originals i el seu resultat en el joc:

Il·lustració 1: Cos del tanc

Il·lustració 2: Canó del tanc

Il·lustració 3: Tanc en el joc

Ara bé, un cop fet això, és fàcil moure tot el conjunt dreta i esquerra, però es complica més quan el que volem moure és el canó per apuntar. Per a fer-ho ens ajudem d'una de les característiques de la classe SpriteBatch. Quan es crida al mètode Draw() d'aquesta classe per a dibuixar els elements per pantalla, se li pot passar un paràmetre indicant el grau de rotació amb el que volem dibuixar la imatge. També se li passa la posició on es dibuixarà i el punt origen de la imatge (a partir del qual aplicarà el punt de rotació). En aquest cas el punt d'origen del canó es troba al punt (alçada de la imatge / 2). Així la rotació serà uniforme. Aleshores per moure amunt o avall el canó simplement hem d'afegir i treure graus a l'atribut 'angle' de la classe TankCannon per què en el moment que es dibuixi per pantalla ho faci amb la nova inclinació. A

continuació veiem el codi corresponent a l'actualització de moviment del canó cap amunt:

```
public void UpdateCannonUp()
{

    this.cannon.rotation += this.orientation * MathHelper.ToRadians(-1);
    if (this.orientation == -1)
    {
        if (this.cannon.rotation > MathHelper.ToRadians(-90))
        {
            this.cannon.rotation = MathHelper.ToRadians(-90);
        }
    }
    else
    {
        if (this.cannon.rotation < MathHelper.ToRadians(-90))
        {
            this.cannon.rotation = MathHelper.ToRadians(-90);
        }
    }
}
```

Podem observar l'angle el guardem en radiants i podem veure que la posició en vertical del canó es representa amb -90 graus i que no es pot moure més enllà (si movem el canó cap avall, tampoc es pot moure més enllà dels 0 graus). Això és així per la característica que hem comentat abans sobre l'eix Y en XNA que sempre es troba a dalt a l'esquerra de la imatge, per tant ho hem de posar en negatiu ja que va a l'inrevés. Podem veure a la següent captura com queda el tanc amb el canó en rotació 45º:

Il·lustració 4: Tanc amb el canó en rotació

Moviment del projectil:

Aquest element és el més difícil d'animar ja que hem de tenir bastantes coses en compte quan aquest és disparat per un dels jugadors. La principal dificultat es troba en

esbrinar el punt d'on ha de sortir el projectil i la direcció a la que s'ha de moure. Com que aquest element surt de la punta del canó, hem de calcular la posició i la direcció a partir de l'angle del canó i de la seva longitud. Per tant, en el fons és tant senzill com aplicar una mica de trigonometria. Així doncs, coneixent la posició del canó (recordem que ho sabem a partir del punt origen del mateix i que aquest està situat a la base de la textura), la seva longitud i el seu angle podem trobar la posició inicial del projectil. Per altra banda ens faltarà conèixer amb quina potència sortirà disparat. Farem servir l'atribut power de la classe *Player* per a saber amb quina potència sortirà. A continuació veiem en el següent fragment de codi el procediment seguit per a esbrinar la posició inicial del projectil quan aquest és disparat:

```
public void ShootProjectile(Projectile projectile, float power)
{
    // Direcció del projectil
    float degree = -MathHelper.ToDegrees(this.cannon.rotation);
    double radians = MathHelper.ToRadians(degree);
    double adjacent = Math.Cos(radians) * power / 10;
    double catet = Math.Sin(radians) * power / 10;
    Vector2 direction = new Vector2(((float)adjacent), (-(float)catet));
    projectile.direction = direction;
    // Posició origen del projectil
    adjacent = Math.Cos(radians) * this.cannon.texture.Width;
    catet = Math.Sin(radians) * this.cannon.texture.Width;
    projectile.position =
      this.cannon.position + new Vector2(((float)adjacent),
                                         (-(float)catet));
    projectile.power = power;
}
```

Recordem que XNA representa la posició Y = 0 a dalt de tot a l'esquerra i per tant, els angles s'han de posar en negatiu. Per aquest motiu obtenim els angles de rotació i els passem a negatiu per poder calcular bé la direcció i la posició. Veiem que per a calcular-ho hem de trobar la longitud del catet i l'adjacent del triangle rectangle que forma el canó amb l'eix X. Aplicant les relacions trigonomètriques sabem que el cosinus de l'angle de la hipotenusa per la longitud de la hipotenusa ens dona la longitud de l'adjacent i el sinus de l'angle de la hipotenusa per la longitud de la hipotenusa ens dona la longitud del catet. En el cas que ens ocupa, la hipotenusa és la longitud del canó i aquesta la trobem a partir de l'amplada de la seva textura:

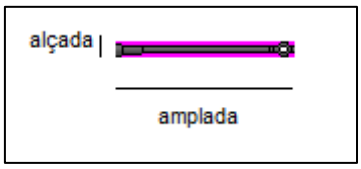


Figura 5: Amplada del canó

Un cop trobada la posició i direcció(en funció de la potència) del projectil hem de tenir en compte que l'efecte de la gravetat farà que aquest es vagi movent progressivament cap a sota a mida que vagi avançant fins a tocar al terra. Per a fer-ho es fa servir la senzilla tècnica d'anar sumant al component Y del vector direcció un valor molt petit constant perquè vagi caient. Hem dit sumar al component Y ja que en XNA el component Y va de dalt a baix de la pantalla tal com hem explicat abans. En el següent fragment de codi pertanyent al mètode Update() de la classe *Projectile* ho veiem:

```
public void update()
{
    this.direction += (new Vector2(0, 1) / 5);
    this.position += this.direction;
}
```

Dividim el resultat per un factor de 5 per tal que la baixada no sigui molt brusca i aconseguim un efecte més realista. Finalment actualitzem la posició del projectil amb la seva direcció.

4.5. Detecció de col.lisions

Un cas especial i dificultós en el desenvolupament de qualsevol videojoc és el de la detecció de col.lisions. Hi ha diverses tècniques per a tractar-les que s'exposaran tot seguit. Finalment es descriurà el procés seguit per aquest projecte:

Tècniques de detecció de col.lisions:

- **Detecció mitjançant rectangles:** Es tracta d'envoltar el sprites que col.lisionaran amb un rectangle. Aleshores hi haurà col.lisió quan aquests

rectangles facin una intersecció. Els avantatges d'aquesta tècnica és que consumeix molts pocs recursos i els inconvenients són principalment que la detecció és molt poc precisa.

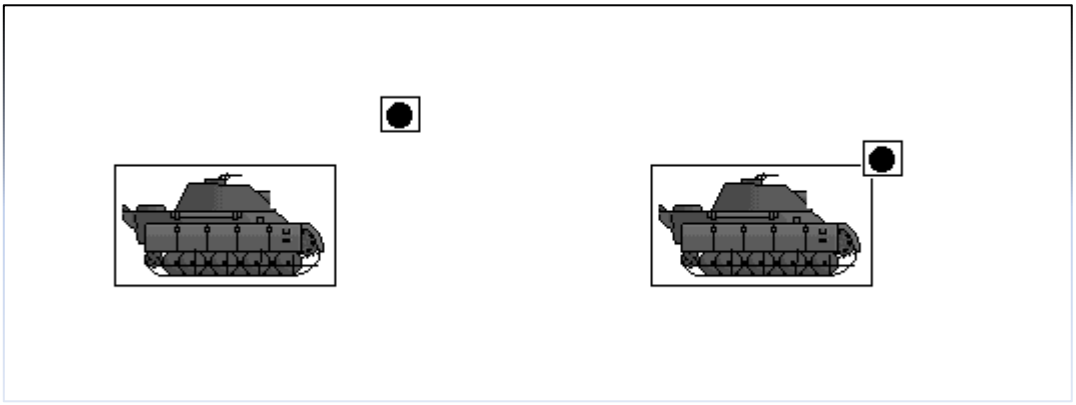


Figura 6: Col.lisió mitjançant rectangles

- **Detecció mitjançant cercles:** És molt semblant a l'anterior, però es fa servir figures circulars per a la detecció de col.lisions. Té els mateixos avantatges que el cas anterior i els seus inconvenients.

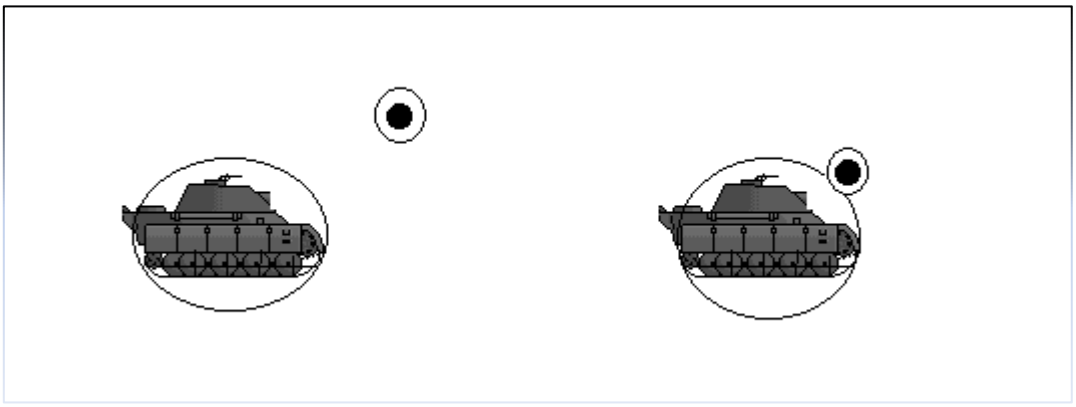


Figura 7:Col.lisió mitjançant cercles

- **Detecció píxel per píxel:** Aquest és el millor sistema de col.lisions ja que es tracta de mirar píxel per píxel si les imatges es toquen. La idea és mirar la matriu de colors que formen les textures i comparar-les píxel per píxel. Si en la mateixa posició hi ha dos píxels i cap d'ells és transparent, aleshores tenim

col.lisió. El principal inconvenient d'aquesta tècnica és que consumeix molts recursos i en jocs força avançats pot resultar perjudicial pel rendiment.

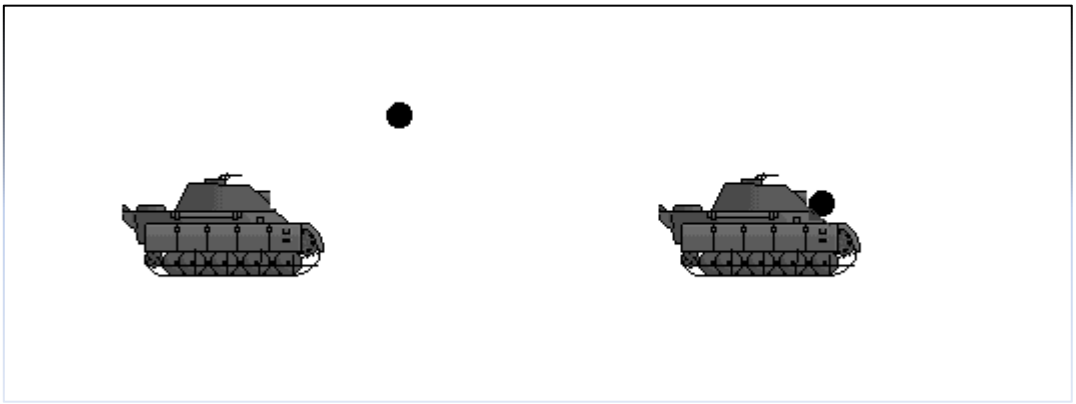


Figura 8: Col.lisió píxel per píxel

Apart d'aquestes tècniques també trobem tècniques híbrides on es fa servir el millor de cadascuna. Per exemple fer servir el sistema per rectangles i quan es detecta una col.lisió passar al sistema píxel per píxel per millorar la precisió. D'aquesta manera no es consumeixen tants recursos i no es perd tanta precisió.

Pel present projecte s'ha decidit implementar el sistema de col.lisions de píxel per píxel ja que al tractar-se d'un joc per torns i amb només dos jugadors, els recursos consumits són suficients com per aplicar aquesta tècnica. També es té en compte que només s'activarà el sistema quan un jugador dispari un projectil. Per tant, pot estar pocs segons executant el codi corresponent a la detecció de col.lisions.

Punts a tenir en compte sobre les textures:

En XNA, una textura té una posició i un origen. La posició indica on es troba la textura en la pantalla. La forma de representar-ho és la d'un vector en dos dimensions x i y. Per altra banda, l'origen indica el punt de referència a dins de la textura partir del qual s'indica la posició de la textura en la pantalla. Aquest punt origen es situa per defecte en la posició (0, 0) que en XNA vol dir que es troba a dalt a l'esquerra de la textura ja que la posició 0 de l'eix Y en XNA sempre comença a dalt de tot. A mida que baixa va

creixent el número. Si una textura té una alçada de 30 píxels, l'origen (0, 30) es trobarà a baix a l'esquerra de la textura:

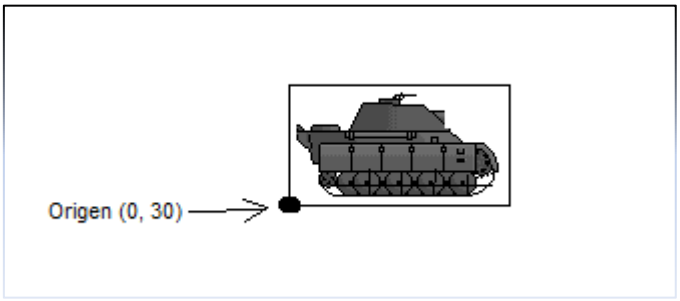


Figura 9: Punt origen en una textura

La classe *CollisionManager*:

A continuació s'explicarà la implementació i algorismes utilitzats en la gestió de col.lisions mitjançant aquesta classe.

Per a crear una instància d'aquesta classe (es pot veure a la part de codi de la GamePlayScreen que s'ha posat anteriorment), se li passen 3 textures (cos del tanc, mur i projectil) les quals seran amb les que treballarà la classe. En el constructor d'aquesta es fa servir un mètode intern per a obtenir els arrays de colors de les textures i s'inicialitzen els atributs corresponents:

```
public CollisionManager(Texture2D wallTexture,
                        Texture2D projTexture,
                        Texture2D tankTexture)
{
    this.wallColorArray = this.TextureTo2DArray(wallTexture);
    this.projColorArray = this.TextureTo2DArray(projTexture);
    this.tankColorArray = this.TextureTo2DArray(tankTexture);
}
```

Per a la detecció de col.lisions es fan servir 3 mètodes públics de la classe. Un per detectar la col.lisió d'un projectil amb un jugador, un altre per detectar la d'un projectil amb un mur i finalment un altre per detectar la col.lisió d'un projectil amb el terra. Cadascun d'aquests mètodes excepte l'últim, obté una matriu per cada element on es guarda informació de la posició de cada píxel de la textura de l'element. Tot seguit

crida al mètode intern TexturesCollide passant-li les matrius de posició i els arrays de colors de les textures corresponents. En el següent mètode podem veure el què hem comentat:

```
public Vector2 CheckProjCollision(Projectile projectile, Player player)
{
    Matrix projMatrix =
      Matrix.CreateTranslation((-projectile.texture.Width / 2),
                               (-projectile.texture.Height / 2),
                               0) *

      Matrix.CreateTranslation(projectile.position.X,
                               projectile.position.Y,
                               0);

    Matrix tankMatrix =
      Matrix.CreateTranslation(
                           (-player.tank.body.texture.Width / 2),
                             0,
                             0)*

      Matrix.CreateTranslation(player.tank.body.position.X,
                                 player.tank.body.position.Y,
                                 0);

    return TexturesCollide(this.projColorArray,
                           projMatrix,
                           this.tankColorArray,
                           tankMatrix);
}
```

Per a obtenir la matriu de posicions de píxel hem de multiplicar les matrius identitat (que en el fons val 1 i per això no es veu en el mètode), la matriu del punt origen (on està situat el punt d'origen de la textura) i la matriu de la posició de la textura. Recordem que la posició origen original del punt de referència de totes les textures és la de (0, 0) on Y = 0 té la particularitat que es troba a dalt de tot a l'esquerra de la textura. Aquest procés es fa per tornar la textura a la seva posició original i així poder comparar-ho bé amb l'array de colors. En el mètode TexturesCollide es fa la comparació píxel per píxel de les dues textures amb l'ajuda de l'array de colors i la matriu de posicions corresponent.

4.6. Gestió d'explosions:

Un altre apartat difícil és el de la creació d'explosions de manera dinàmica. Per a fer-ho aquest projecte es basa en un sistema de partícules on cada explosió està formada per una llista d'aquestes partícules, les quals es mouen i s'expandeixent en diferents direcció seguint un generador aleatori. Cada partícula està formada per la mateixa textura, però es van superposant una sobre l'altra i desapareixent en ordre de manera que creen un efecte òptic on cada explosió sembla diferent i més real. És una manera més vistosa i a l'hora més còmode que la de fer servir un sprite-sheet per a animar una explosió.

La classe *Explosion*:

Per a realitzar aquestes explosions s'ha decidit crear la classe *Explosion*. Aquesta classe conté una llista de *Particle* on s'hi van afegint particules amb una direcció, angle de rotació i acceleració aleatòries. Cada partícula a més té els atributs startTime (temps quan apareix la particula), duration(duració de la partícula), startPosition(posició inicial), actualPosition(posició actual). La idea és que a cada fracció de temps de joc, cada particula es mogui i roti en funció de la direcció, acceleració i angle generats aleatòriament en el constructor de la classe *Explosion*. També, a cada fracció de temps es va variant el color de la partícula perquè es vagi difuminant progressivament i també es modifica el seu escalat (canvi de tamany) en funció de l'espai recorregut. Finalment a mida que la duració de les textures es va acabant, es van traient de la llista. Podem veure el procés al mètode Update() de la classe *Explosion*:

```
public void Update(GameTime gameTime)
{
    float now = (float)gameTime.TotalGameTime.TotalMilliseconds;
    //Comprovem la duració de les partícules i les treiem de la llista si
    //pasen de la duració màxima
    for (int i = particleList.Count - 1; i >= 0; i--)
    {
        Particle particle = particleList[i];
        float timeAlive = now - particle.startTime;

        if (timeAlive > particle.duration)
```

```
            {
                particleList.RemoveAt(i);
            }
            else
            {
                float relAge = timeAlive / particle.duration;
                particle.actualPosition =
                       0.5f * particle.acceleration *
                       relAge * relAge + particle.direction *
                       relAge + particle.startPosition;

                float invAge = 1.0f - relAge;
                particle.color =
                       new Color(new Vector4(invAge,
                                             invAge,
                                             invAge,
                                             invAge));

                Vector2 positionFromCenter =
                 particle.actualPosition - particle.startPosition;
                float distance = positionFromCenter.Length();
                particle.scaling = (50.0f + distance) / 200.0f;

                particleList[i] = particle;
            }
        }
    }
```

4.6. Modalitat en xarxa

Per a la modalitat de joc en xarxa definitivament s'ha optat per l´ús de la llibreria de codi obert Lidgren. La idea inicial era la de fer servir els serveis que ofereix XNA per a gestionar dades de xarxa, però s'han trobat dificultats a l'hora de muntar un executable que faci servir aquets serveis en un altre ordinador. El problema és que si no es té subscripció Live Gold, només es pot implementar funcions de xarxa en mode de prova. Això vol dir que al crear un executable del joc, aquest no funcionarà en un altre ordinador ja que l'XNA distribuïble que es descarrega quan s'instal.la el joc no porta els serveis incorporats. La única manera de poder-ho fer seria tenir instal.lat tot el kit de desenvolupament XNA. Tot i així, després de varies proves sense èxit es va descartar aquesta via i tal com s'ha comentat abans finalment es va optar per la llibreria Lidgren. Aquesta llibreria disposa d'una sèrie de classes molt útils per a la gestió de xarxa. Està pensada per usar-se amb .Net i també s'ha pensat per poder-se usar amb XNA.

Classes principals de la llibreria Lidgren:

A continuació es mostren les principals classes de la llibreria Lidgren que s'han usat per a el.laborar la part de xarxa del joc:

- NetServer: Aquesta classe representa un servidor i ofereix totes les funcionalitats d'aquest. Conté una llista de les connexions que s'hi ha fet i pot enviar i rebre tot tipus de missatges (des de bytes, fins a strings, passant per enters de 32 bits, de 64 bits, etcètera).
- NetClient: Aquesta classe representa un client i ofereix totes les funcionalitats d'aquest. Pot connectar-se a un objecte servidor i transmetre i rebre tot tipus de dades igual que fa NetServer.
- NetOutgoingMessage: Conté informació sobre els missatges sortints d'un client o d'un servidor que anirà per la xarxa.
- NetIncomingMessage: Conté informació sobre els missatges entrants que rep un client o un servidor provinents de la xarxa.

Fent ús bàsicament d'aquestes 4 classes més alguna constant o mètodes estàtics d'altres classes de la llibreria s'ha aconseguit implementar un sistema de joc en xarxa local de manera satisfactòria.

Tipus de models de comunicació:

Principalment avui en dia es fan servir dos models de comunicació en xarxa per a videojocs. Són els següents:

- ***Peer-to-Peer(P2P):*** Es tracta de la modalitat en xarxa punt a punt. Amb aquest model, cada màquina es gestiona les seves dades i les envia a l'altra màquina connectada. Cadascú es gestiona l'estat de joc.
- ***Client-Servidor:*** En aquesta modalitat, hi ha un servidor al que se li connecten uns clients (jugadors) i aquest gestiona l'estat de joc. Rep dades dels clients,

les processa i les hi torna a tots els clients. D'aquesta manera tots els clients tindran la mateixa informació.

El model utilitzat per al joc ha estat el de Client-Servidor tot i que es pot considerar una mica híbrid amb peer-to-peer ja que alguns elements com les explosions es gestionen a cada màquina. Per altra banda, en aquest cas el Servidor és un dels jugadors, concretament el que crea la partida i a l'hora també fa de client. Aquest no envia informació al servidor tot i que en rep juntament amb l'altre jugador que sí que és un client pur i dur (envia i rep informació del servidor).

Funcionament de la modalitat en xarxa:

Per a accedir al mode de joc en xarxa fan falta dos ordinadors connectats dins la mateixa xarxa local. Un cop dins la modalitat xarxa local, el jugador que seleccioni l'entrada "Create Session" serà el host de la partida. Per començar la partida haurà de teclejar la tecla 'A' a dins la sala d'espera i esperar que el segon jugador es connecti mitjançant l'entrada "Join Session". Un cop el segon jugador teclegi la tecla 'A' després d'entrar a la sala d'espera començarà la partida en xarxa.

Per implementar aquesta modalitat s'ha creat una classe NetworkManager que s'encarrega de la gestió de missatges entre el client i el servidor. Aquesta classe conté un objecte NetServer, un objecte NetClient i una llista de jugadors (que sempre seran 2). L'objecte NetServer només s'inicialitza en la màquina servidor(host), és a dir, la màquina del jugador que crea la sessió. Un cop a dins la pantalla de joc (la pantalla GamePlayScreen), es crida a un nou mètode per actualitzar el mètode Update de la pantalla. En aquest cas és el mètode NetworkUpdate, el qual és molt semblant al mètode LocalUpdate, però tenim que la part d'actualització dels jugadors queda així:

```
// Enviem informació al servidor sobre els jugadors
if (this.networkManager.IsHost() == true)
{
    //Actualització de les accions del jugador 1
    if (this.player1.turn == true)
    {
        this.player1.Update(keyboard);
    }
}
```

```
            else
            {
                if (this.player2.turn == true)
                {
                    //Actualització de les accions del jugador 2
                    this.networkManager.UpdatePlayerInput(keyboard, 1);
                }
            }

            // Actualitzem els estats dels jugadors en el joc
            this.networkManager.UpdateGamePlayScreen();
```

El que es fa és només enviar informació del jugador 2 al servidor ja que el jugador 1 fa de servidor i no cal enviar la seva informació. El mètode UpdatePlayerInput de la classe NetworkManager, interpreta la informació del teclat i la hi envia al servidor. Aleshores, en el mètode UpdateGamePlayScreen() de la classe NetworkManager, si el servidor està actiu, llegeix la informació que envia el client i tot seguit el client llegeix la informació que pugui enviar el servidor. Mirem tot seguit el mètode per captar la informació del teclat i enviar-la al servidor:

```
public void UpdatePlayerInput(KeyboardState keyboard, byte index)
{

    if (keyboard.GetPressedKeys().Length <= 1)
    {
        //Enviem primer un missatge indicant el jugador
        NetOutgoingMessage om = this.client.CreateMessage();
        byte code = (byte)MessageClasses.NoAction;

        if (keyboard.IsKeyDown(Keys.Left))
        {
            code = (byte)MessageClasses.MoveLeft;
        }
        if (keyboard.IsKeyDown(Keys.Right))
        {
            code = (byte)MessageClasses.MoveRight;
        }
        if (keyboard.IsKeyDown(Keys.Up))
        {
            code = (byte)MessageClasses.CannonUp;
        }
        if (keyboard.IsKeyDown(Keys.Down))
        {
            code = (byte)MessageClasses.CannonDown;
        }
        if (keyboard.IsKeyDown(Keys.Space))
        {
            code = (byte)MessageClasses.Space;
        }

        om.Write(code);
```

```
            om.Write(index);
            this.client.SendMessage(om, NetDeliveryMethod.Unreliable);
        }
    }
```

Per a gestionar els missatges que s'envien el client i el servidor s'ha creat una enumeració que descriu quin tipus de missatge s'ha d'enviar. Es pot veure com en funció de la tecla apretada, es prepara un codi (que és un byte per a ocupar menys espai) que s'envia al servidor.

Per altra banda tenim que el servidor rep el següent:

```
public void ReadFromClient()
{
    // read messages
    NetIncomingMessage msg;
    while ((msg = this.server.ReadMessage()) != null)
    {
        switch (msg.MessageType)
        {
            case NetIncomingMessageType.DiscoveryRequest:
                // Server received a discovery request from a client;
                //send a discovery response (with no
                //extra data attached)
                server.SendDiscoveryResponse(null, msg.SenderEndPoint);
                break;
            case NetIncomingMessageType.Data:
                // server sent a position update
                Byte code = msg.ReadByte();
                int index;
                Keys[] keys;
                KeyboardState keyboard;
                List<NetConnection> connections =
                                        this.server.Connections;
                switch (code)
                {
                    case (byte)MessageClasses.Ready:
                        this.readyPlayers += 1;
                        break;
                    case (byte)MessageClasses.MoveLeft:
                        index = msg.ReadByte();
                        keys = new Keys[1]{Keys.Left};
                        keyboard = new KeyboardState(keys);
                        this.playerList[index].Update(keyboard);
                        break;
                    case (byte)MessageClasses.MoveRight:
                        index = msg.ReadByte();
                        keys = new Keys[1]{Keys.Right};
                        keyboard = new KeyboardState(keys);
                        this.playerList[index].Update(keyboard);
                        break;
                    case (byte)MessageClasses.CannonUp:
```

```
                        index = msg.ReadByte();
                        keys = new Keys[1]{Keys.Up};
                        keyboard = new KeyboardState(keys);
                        this.playerList[index].Update(keyboard);
                        break;
                    case (byte)MessageClasses.CannonDown:
                        index = msg.ReadByte();
                        keys = new Keys[1]{Keys.Down};
                        keyboard = new KeyboardState(keys);
                        this.playerList[index].Update(keyboard);
                        break;
                    case (byte)MessageClasses.Space:
                        index = msg.ReadByte();
                        keys = new Keys[1]{Keys.Space};
                        keyboard = new KeyboardState(keys);
                        this.playerList[index].Update(keyboard);
                        break;
                    case (byte)MessageClasses.NoAction:
                        index = msg.ReadByte();
                        keyboard = new KeyboardState();
                        this.playerList[index].Update(keyboard);
                        break;
                }
                break;
            }
        }
    }
```

En funció del tipus de missatge rebut, el servidor simula una classe teclat amb la tecla que s'ha premut i tot seguit fa que el jugador que toqui (indicat amb un byte que pertany a la posició de la llista en el NetworkManager) s'actualitzi.

Un cop fet aquest procés el servidor ha de comunicar cada cert temps quin és l'estat dels jugadors (les seves posicions, si han disparat un projectil, la posició d'aquest projectil, etc...). Per a fer-ho s'ha posat un ritme d'enviament de dades de 30 vegades per segon per tal de millorar el rendiment. Això provoca que la partida en xarxa sembli més lenta i brusca tot i que és possible de jugar. El servidor realitza aquesta funció amb el mètode SendUpdates() de la classe NetworkManager. Aquest mètode es crida dins del mètode UpdateGamePlayScreen() de la classe NetworkManager tal com veiem a continuació:

```
public void UpdateGamePlayScreen()
{
    if (this.server != null)
    {
        this.ReadFromClient();
        double now = NetTime.Now;
```

```
            if (now > this.nextSendUpdates)
            {
                this.SendUpdates();
                this.nextSendUpdates += (1.0 / 30.0);
            }
        }

        this.ReadFromServer();
    }
```

Finalment amb el mètode ReadFromServer(), els dos clients reben la informació de les posicions dels jugadors i els seus elements (projectils, etc...) i ho actualitzen en els seus respectius jocs.

Capítol 5: Proves i Experiments

D'alguna manera, tot aquest projecte ha estat una espècie d'experiment. La part més d'experimentació pròpiament dita ha estat en la fase inicial abans fins i tot de començar el disseny de classes, etc... De fet, moltes de les classes no hi eren en la versió Alpha que es va penjar al fòrum de l'assignatura. En un principi es va experimentar amb el moviment del tanc i amb una manera d'aconseguir moviment independent del canó. Per a fer-ho es va optar tal com s'ha explicat en el capítol 4 per a fer servir dos elements que formen el tanc (el canó i el cos). També es va experimentar bastant amb la detecció de col.lisions i el sistema d'explosió de partícules.

Per altra banda, principalment, la part on s'han hagut de realitzar més proves ha estat la modalitat en xarxa. Primer de tot es va provar de programar aquesta modalitat fent servir la classe GamerServicesComponent que ofereix XNA, però els resultats no van ser satisfactoris ja que a l'executar-ho en una altra màquina donava problemes. Això s'explica millor en l'apartat 4.6 del Capitol 4 d'aquest document. En una màquina local si que es va poder connectar dos clients i es tenia l'avantatge que amb la classe mencionada abans, s'automatitzava molt el procés. A l'iniciar el joc s'informava al jugador de un inici de sessió tal com es pot veure en la següent figura:

Figura 10: Prova d'inici amb la classe GamerServicesComponent

Finalment es va optar per a realitzar la part en xarxa amb la llibreria de codi obert Lidgren.

Per a fer les proves en aquesta modalitat s'ha fet de dues maneres: Executant dos projectes de Visual Studio 2010 en la mateixa màquina per una banda i executant dos executables publicats a partir del projecte en dues màquines pertanyents a una mateixa xarxa. Les proves han estat satisfactòries en ambdós casos. En la següent captura veiem una instantània d'un moment del joc en xarxa en acció:

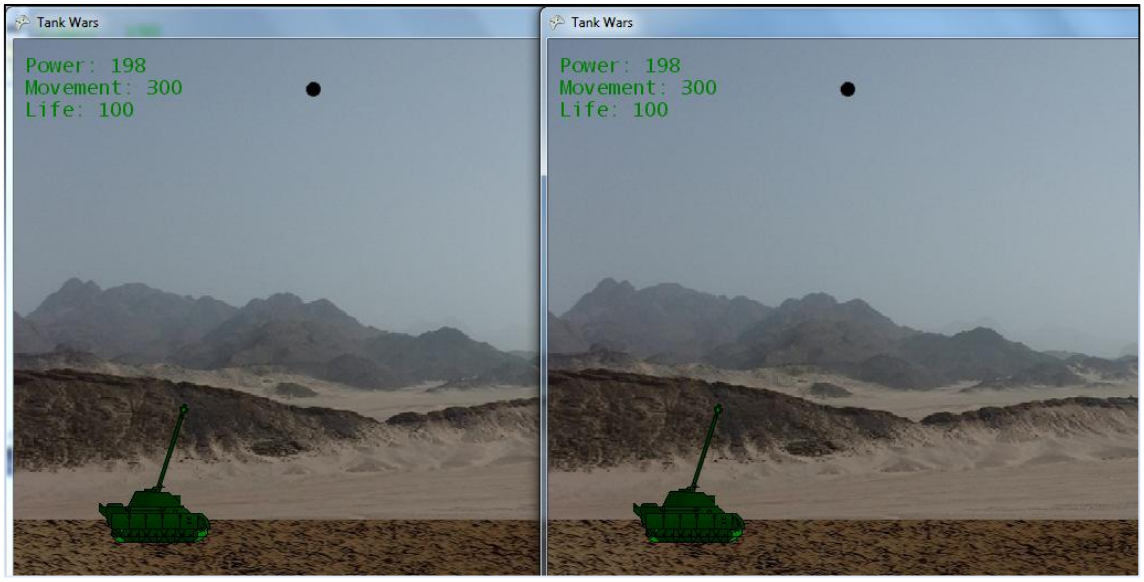


Figura 11: Captura de pantalla de la modalitat en xarxa

Aquesta captura està realitzada en una sola màquina executant dues instàncies del joc. Es pot comprovar com es reflexa el mateix en les dues pantalles, des de els valors dels marcadors fins les posicions dels elements en pantalla.

Capítol 6: Conclusions i línies de futur

L'experiència en el desenvolupament d'aquest videojoc ha estat molt enriquidora per qui escriu aquestes paraules. S'han vist les dificultats que comporta el desenvolupament d'un joc i s'ha obtingut consciència de la vital importància que tenen tots els camps que engloben un joc. Tot i que sembla que la base d'un joc és la programació, realment la part de disseny i la part artística són com a mínim igual d'importants en la seva creació. L'objectiu principal ha estat la d'aconseguir un joc en el qual es pugui realitzar una partida completa, és a dir que s'hi pugui jugar, i es pot concloure que s'han aconseguit complir els objectius marcats. Per aquest motiu, la satisfacció per la feina feta és màxima, tot i que hi ha àrees, com la part artística i de disseny d'interfícies del joc juntament amb alguns errors que a vegades es poden donar dins la modalitat de joc en xarxa (tot i que en cap cas impliquen que no es pugui acabar una partida), que es podrien haver millorat amb una mica més de temps.

També s'ha de destacar que les eines que proporciona XNA faciliten la tasca de programació de manera que en poc temps es poden tenir elements movent-se per la pantalla, però alhora permet una gran flexibilitat per a realitzar coses molt potents i professionals. De fet, es poden arribar a realitzar jocs en 3D força espectaculars i sense gaire a envejar a produccions realitzades per estudis professionals.

Definitivament, es tracta d'un món apassionant i, a qui escriu aquestes paraules li agradaria poder endinsar-s'hi i veure si en un futur poder arribar a dedicar-s'hi. Així, el següent pas a fer seria el d'intentar desenvolupar algun joc fent servir un llenguatge de programació més complex com per exemple C++ (molt usat en el món professional de la creació de videojocs) i fins i tot provar amb alguna cosa en 3 dimensions. També cal dir que estaria molt bé provar amb algun gènere tipus aventura. El camí és lent i costós, però amb constància es pot aprendre tot el que una persona es proposi.

De cara al futur relacionat amb el projecte realitzat, es podrien aplicar una sèrie de millores que es podrien separar en 3 grups:

Millores de disseny:

- Plantejar el joc per equips: Fer el joc per més de dos jugadors i que s'enfrontin per equips (4 contra 4) per exemple.
- Disseny d'escenaris: Afegir desnivells, posar diferents tipus de murs centrals (un on pugui passar un projectil per un forat al mig, per exemple), escenaris destructibles, etc...
- Afegir power-ups: Afegir objectes(que caiguin del cel aleatòriament, per exemple) que donin funcions especials als tancs (projectils més poderosos, més resistència als projectils, energia extra, projectils que puguin travessar el mur central per un temps, etc...).

Millores artístiques:

- Elements de joc: Millorar l'apartat gràfic de tots els elements del joc (escenari, tancs, projectil).
- Marcadors del joc: Millorar els marcadors del joc, (per exemple fent servir barres de vida en comptes de números, etc...)
- Pantalles del joc: Millorar en general l'aspecte de totes les pantalles de joc i dels seus menús.

Millores tècniques:

- Efectes del joc: Afegir nous efectes, com per exemple fum al disparar un projectil, zoom cap a fora de la pantalla per poder seguir el projectil, etc...
- Animacions: Millorar el moviment dels tancs fent-lo més realista (que al començar a moure's ho faci més lent, que tingui inèrcia al frenar, algun tipus de retrocés al disparar, etc...)
- Joc en xarxa: Millorar el joc en xarxa. Aconseguir que es mogui igual que en la modalitat local, eliminar la lentitud, etc...

Annexos

Captures de pantalla:

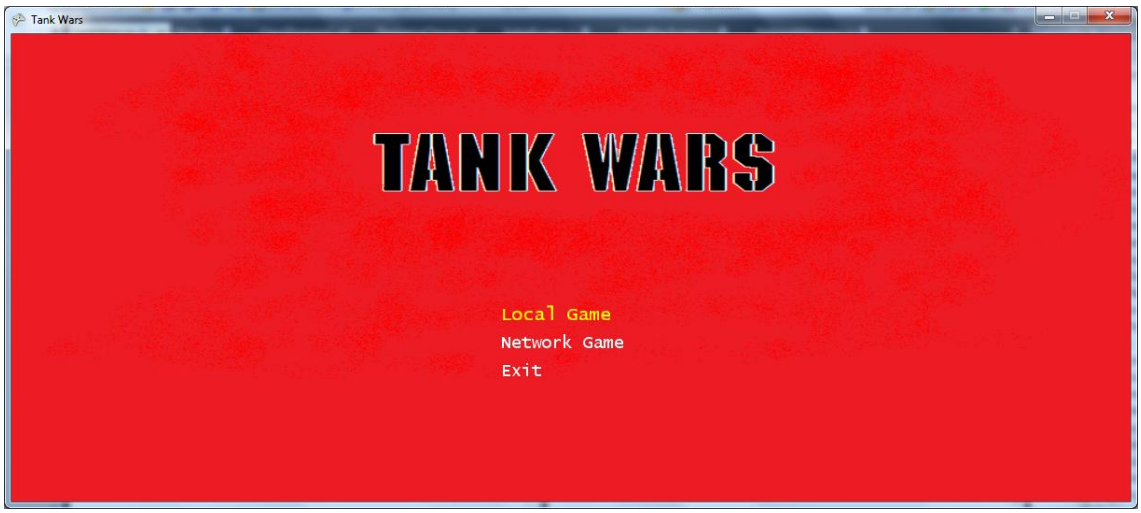


Figura 12: Pantalla d'inici de Tank Wars

Figura 13: Pantalla de joc de Tank Wars

Instruccions de la modalitat per joc local:

Aquesta és la modalitat de joc per dos jugadors en una mateixa màquina. L'acció del joc es pot pausar en qualsevol moment prement la tecla 'P'. Des del menú pausa es pot sortir al menú principal o continuar amb el joc.

La partida funciona per torns on els dos jugadors s'alternen. A cada torn, el jugador que li toca pot moure el tanc fins que el seu indicador *Movement* arribi a 0. Per a disparar un projectil s'ha de pitjar la tecla d'acció moment en el qual s'activarà l'indicador de força del jugador. Un cop activat, si es torna a pitjar la tecla d'acció l'indicador s'atura i el projectil surt disparat amb la potència indicada. Si el projectil encerta al jugador rival, aquest perd 10 punts d'energia. Si aquesta arriba a 0, s'acaba la partida i guanya el jugador que quedi amb vida. Cada cop que es dispara un projectil es finalitza un torn i el jugador contrari passa a controlar el seu tanc.

Controls del jugador 1:

- Moure a l'esquerra: cursor esquerra
- Moure a la dreta: cursor dreta
- Moure el canó cap amunt: cursor amunt
- Moure el canó cap abaix: cursor abaix
- Acció: tecla espai

Controls del jugador 2:

- Moure a l'esquerra: Tecla 4 del teclat numèric
- Moure a la dreta: Tecla 6 del teclat numèric
- Moure el canó cap amunt: Tecla 8 del teclat numèric
- Moure el canó cap abaix: Tecla 2 del teclat numèric
- Acció: tecla Intro

Instruccions de la modalitat per joc en xarxa:

Aquesta modalitat funciona exactament igual que la modalitat local, exceptuant que no es pot pausar la partida i que tots dos jugadors tenen els mateixos controls. Concretament:

Controls dels jugadors 1 i 2:

- Moure a l'esquerra: cursor esquerra
- Moure a la dreta: cursor dreta
- Moure el canó cap amunt: cursor amunt
- Moure el canó cap abaix: cursor abaix
- Acció: tecla espai

Per iniciar partida, un dels jugadors ha de crear la partida seleccionant l'entrada "Create Sesion". Un cop fet això entrarà al seu lobby. Per altra banda, després que el primer jugador hagi creat sessió, el jugador 2 seleccionarà l'entrada "Join Session" i entrarà al seu lobby. Un cop els dos jugadors estiguin cadascun en el seu Lobby, si premen la tecla 'A' començarà la partida en xarxa.

Crèdits del joc:

- Idea original, disseny i implementació: Francesc Xavier Revilla Truyols
- Fons de pantalla de l'escenari principal proporcionat per Marta Mas Pardo
- Disseny original dels sprites dels tancs: Kayos de GameMakerCommunity

Bibliografia

- Comunitat de desenvolupadors GameDev:
  - http://www.gamedev.net
- Creació d'un videojoc amb XNA:
  - http://upcommons.upc.edu/pfc/bitstream/2099.1/11830/1/file9.pdf
- Desenvolupament de videojocs (Wikipèdia):
  - http://en.wikipedia.org/wiki/Video_game_development
- Desenvolupament XNA:
  - http://www.xnadevelopment.com
- Dificultats en la creació d'un videojoc:
  - http://0-sag.sagepub.com.cataleg.uoc.edu/content/40/5/688.full.pdf+html
- Enquesta sobre Motors gràfics:
  - http://0-search.proquest.com.cataleg.uoc.edu/docview/864185625
- Game Maker Comunity:
  - http://gmc.yoyogames.com/index.php?act=idx
- Llenguatge LUA:
  - http://en.wikipedia.org/wiki/Lua_(programming_language)
- Llenguatge PHP:
  - http://en.wikipedia.org/wiki/PHP
- Llibreria Lidgren
  - http://code.google.com/p/lidgren-network-gen3/
- Llista de motors per a jocs:
  - http://en.wikipedia.org/wiki/List_of_game_engines
- Motor fisiques PhysX
  - http://en.wikipedia.org/wiki/PhysX
- Motor RAGE:
  - http://en.wikipedia.org/wiki/Rockstar_Advanced_Game_Engine
- Motor UNITY3D:
  - http:// http://unity3d.com
- Motor Source de Valve:
  - http://source.valvesoftware.com/

- Motor UnrealEngine:
  - http://www.unrealengine.com/
- Networking en XNA
  - http://msdn.microsoft.com/en-us/library/bb975645(v=xnagamestudio.31).aspx
- Programació amb C#
  - http://www.programmersheaven.com/ebooks/csharp_ebook.pdf
- Tutorials XNA rbwhitaker
  - http://rbwhitaker.wikidot.com/xna-tutorials
- Videojoc 2D per a android:
  - http://www.recercat.net/handle/2072/183802
- XNA 3.0 Game Programming Recipes: A Problem-Solution Approach
  - Llibre editat per Paperback