TecnoCampus

Centres universitaris adscrits a la

upf. Universitat Pompeu Fabra Barcelona

Grau en Disseny i Producció de Videojocs

PROGRAMACIÓ D'UN JOC TYCOON AMB UNITY3D

Memòria

ARNAU LAFUENTE FEDERICO

TUTOR: MARCO ANTONIO RODRÍGUEZ

2017 - 2018

TecnoCampus
Mataró-Maresme

Dedicatòria

A Joan Carles Lafuente, el meu pare

Agraïments

A Maria Meneses per tota l'ajuda i suport que m'ha aportat.

A Marco Antonio i Antonio José Planells pel suport brindat durant tot el projecte.

A Núria Federico per estar sempre al meu costat i haver-me donat aquesta oportunitat.

I, finalment, al meu germà Bernat i la seva parella Karlota per tota l'ajuda al llarg del procés.

Abstract

This final project is focused to implementing a *tycoon* video game design. This design has been created by Eduard Álvarez as a final degree project. This game will be implemented with *Unity3D* (game engine), it will have an axonometric view and 2D graphics. The key in this game is the artificial intelligence. In this project, a part of this intelligence will be implemented with scripting by the use of Lua. *Unity3D* accepts various programming languages, but in this project, C# object-oriented will be used.

Key words: *tycoon, Unity3D*, programming, video game, artificial intelligence, isometric, C#, Lua, object-oriented programming

Resum

Aquest treball està enfocat a la implementació del disseny d'un joc *tycoon*. El disseny ha estat desenvolupat per Eduard Álvarez al llarg del seu treball de final de grau. El joc estarà implementat amb el motor de jocs *Unity3D*, tindrà una vista axonomètrica i els seus elements gràfics seran 2D. El punt clau del joc és la intel·ligència artificial una part de la qual, en aquest projecte, es desenvoluparà utilitzant *scripting* amb Lua. Unity3D accepta diferents llenguatges de programació, però aquest projecte es desenvoluparà amb C# orientat a objectes.

Paraules clau: *tycoon, Unity3D*, programació, videojoc, intel·ligència artificial, isomètric, C#, Lua, programació orientada a objectes

Resumen

Este trabajo está enfocado a la implementación del diseño de un juego *tycoon*. El diseño ha sido desarrollado por Eduard Álvarez a lo largo de su trabajo de fin de grado. El juego estará implementado con el motor de juegos *Unity3D*, tendrá una vista axonométrica y sus elementos gráficos serán 2D. El punto clave del juego es la inteligencia artificial una parte de la cual, en este proyecto, se desarrollará utilizando scripting con Lua. Unity3D acepta diferentes lenguajes de programación, pero este proyecto se desarrollará con C # orientado a objetos.

Palabras clave: *tycoon, Unity3D*, programación, videojuego, inteligencia artificial, isométrico, C#, Lua, programación orientada a objetos

ÍNDEX

Índex de figures

Índex de taules

Glossari de termes

IA	Intel·ligència artificial (AI en anglès)
TFG	Treball Final de Grau
Actor	Personatge o objecte situat dins del joc
HUD	Heads-Up Display
RTS	Real-Time Strategy games
FPS	First person shooter
Spritesheet	Imatge composta per vàries imatges, s'utilitza per estalviar memòria
SUR	Sistema universal de referència
SRO	Sistema de referència de l'objecte
CMS	Construction and management simulations
POV	Points of visibility
VR	Virtual Reality
BIM	Building Information Modeling
Script	Fitxer contenidor de codi

1. INTRODUCCIÓ

Actualment, la indústria dels videojocs està sent més acceptada per la societat gràcies a, entre altres factors, el creixement de la indústria (Benito García, 2005). S'utilitzen jocs en molts aspectes diferents de la vida quotidiana, com per exemple, en teràpies, en l'educació, en dinàmiques de grup, etc. (Mangiron & Orero, 2012). Cada vegada més, els videojocs s'estan convertint en fenòmens capaços de mobilitzar a milers de persones, fins arribar al punt de fer competicions a nivell mundial que s'emeten per televisió i aconsegueixen audiències de milions de persones (McTee, 2014).

Al llarg dels anys els videojocs han explorat diverses temàtiques i gèneres. Un dels seus principals objectius és proporcionar entreteniment i diversió, però actualment també s'utilitzen per aportar coneixement i ajudar a nivell pedagògic (Mangiron & Orero, 2012).

El resultat d'aquest projecte està dirigit a un públic interessat en la gestió i direcció d'un negoci o bé en la creació d'estratègies per aconseguir un objectiu concret. Aquest treball es centra en aquesta temàtica, agafant el disseny desenvolupat per Eduard Álvarez i convertir-lo en un videojoc jugable. Abans de començar el projecte, juntament amb l'Eduard, es va arribar a un acord dels mínims d'aquest treball. Es va definir que seria un joc del gènere *tycoon,* enfocat a ser el dirigent d'un negoci, que combina la restauració amb l'oci dels videojocs i l'objectiu principal del joc consistirà en tenir èxit en el negoci.

El joc d'aquest treball està creat amb el motor de jocs Unity3D, a més, s'ha desenvolupat amb programació orientada a objectes, de manera que permet la reutilització d'elements i facilita la incorporació o adaptació de noves mecàniques o continguts del joc. També s'ha optat per inserir diferents eines com *l'scripting* per ajudar a l'equip a treballar de forma paral·lela. El desenvolupament d'un joc com a mínim consta de 3 grans disciplines: disseny, programació i art. Com ja s'ha comentat anteriorment, aquest projecte implementa a nivell de programació el disseny creat per Eduard Álvarez, però a nivell d'art està previst que Carolina Garrido desenvolupi tots els elements necessaris.

Al llarg del document, s'explicaran els diversos conceptes bàsics necessaris per a dur a terme la implementació, així com les eines i metodologies utilitzades a la industria dels videojocs. Aquest projecte segueix una estructura que permet una lectura i comprensió fàcil al lector.

El primer apartat consisteix en explicar els referents del projecte, és a dir, en realitzar un repàs de les obres o productes similars que inspiren la nostra feina, a més d'analitzar aquestes obres i explicar quin són els aspectes més remarcables pel projecte actual.

Seguidament, en el marc teòric, es fa una recerca dels gèneres relacionats amb el *tycoon* i les mecàniques que s'aplicaran al projecte. Dins el mateix apartat, es fa un repàs per les bàsiques de la intel·ligència artificial i les tècniques més utilitzades als videojocs. Els punts

restants del marc teòric fan un estudi a les eines i metodologies que s'utilitzaran al llarg del projecte.

A continuació, s'expliquen els objectius del projecte i què es pretén aconseguir al final el treball.

A l'apartat de la metodologia, es descriu detalladament quina metodologia ha estat l'escollida per utilitzar al llarg del projecte així com la planificació del pla de treball.

Al llarg de l'apartat de resultats s'explicarà la organització de la programació i el funcionament d'aquesta. Serà a les conclusions on es farà un repàs per comprovar si s'han acomplert els objectius marcats i una valoració personal del resultat del projecte.

Finalment, en els annexos es poden trobar els diagrames de classes i informació extra que ajuda a comprendre el treball realitzat.

2. REFERENTS

En aquest apartat s'analitzaran diferents jocs o productes que han servit com a referent a l'hora de desenvolupar el joc. Aquests anàlisis s'han realitzat al jugar a aquests mateixos.

Existeixen més jocs que s'han utilitzat com a referents, però al ser de la mateixa saga no s'han incorporat a l'anàlisi. Alguns dels conceptes introduïts seran explicats més endavant, al marc teòric.

2.1. Populous (1989)

És del gènere *god game* i ha sigut un dels referents per l'estil de càmera i les accions automàtiques que es realitzen al llarg del temps. *Populous* (Bullfrog Productions,1989). és un videojoc publicat l'any 1989 per l'empresa Bullfrog. El principal objectiu és crear una gran població que pugui enfrontar a la població d'una deïtat enemiga i vèncer-la.

Des d'una perspectiva isomètrica, el jugador realitza accions de forma directa amb el món i, accions indirectes als seguidors de la deïtat per garantir la prosperitat i el creixement de la seva gent.

Figura 2-1 *Screenshot* de *Populous* (camp)

Font: (Electronic Arts, 2018)

Figura 2-2 *Screenshot* de *Populous* (castell)

Font: (Electronic Arts, 2018)

Veient que el jugador no té un domini real sobre les activitats individuals de la gent, el joc proporciona diverses eines que permeten fer un seguiment ràpid de l'entorn i l'estat del món. Així doncs, la part més interessant pel projecte és la forma que el jugador té d'interactuar amb el joc, ja que és la intel·ligència artificial qui defineix com evoluciona el joc i el seu món a partir de les decisions preses per l'usuari.

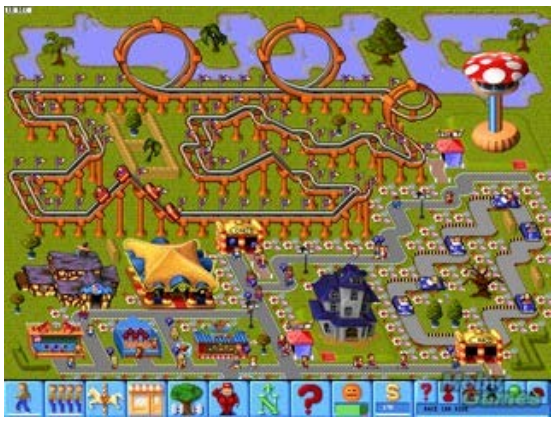
El joc ofereix la possibilitat al jugador de millorar el terreny, per ajudar a ampliar la civilització i així fer front a l'enemic. I, també li permet llançar encanteris per obstaculitzar l'avenç de l'enemic. Els personatges del joc tenen una intel·ligència artificial que els fa prendre decisions segons els aspectes que hagi modificat el jugador. Totes aquestes accions es fan a través del ratolí i botons de la interfície d'usuari del joc.

2.2. Theme Park (1994)

És un joc de l'any 1994, creat per Bullfrog, i es podria considerat del gènere *tycoon*. És interessant d'analitzar per la interacció que tenen els personatges amb intel·ligència artificial amb la resta d'objectes situats al món. A més, el jugador ha de tenir en compte l'estat de les instal·lacions, de les plantes, etc. ja que si estan en mal estat els usuaris del parc no es senten satisfets i això afecta a la reputació del parc.

Dins d'aquesta definició, entre altres, podríem incloure jocs de la mateixa saga i també les sagues de *Roller Coaster Tycoon* (Chris Sawyer, 1999), *SimCity* (Maxis, 1989).

Figura 2-3 *Screenshot* de Theme Park



Font: (Games Nostalgia, 2018)

Figura 2-4 Screenshot botiga Theme Park

Font: (Games Nostalgia, 2018)

L'objectiu principal de *Theme Park* (Bullfrog Productions, 1994) és construir el millor parc d'atraccions prop de Londres. L'usuari pot seleccionar les atraccions, les botigues i els accessoris que vulgui per situar-los al terreny de la forma que cregui més adequada. També se li dóna la possibilitat de crear camins, zones per les cues de les atraccions, etc.

El sistema de construcció és senzill i s'ha pres com a exemple, entre altres, per aplicar-lo al projecte. Consisteix en escollir dins d'un menú de botiga l'element que es vol construir. Un cop seleccionat, l'usuari veu una previsualització de com quedaria l'element mentre va desplaçant el ratolí pel mapa, si aquest prem el ratolí es construeix l'element, aleshores seguint el mateix procediment de previsualització, l'usuari ha de col·locar la resta d'elements del conjunt que hagi seleccionat abans de sortir del mode construcció. Un exemple és construir una atracció, els seus elements conjunts són la taquilla i l'entrada a l'atracció.

En aquest referent també s'ha observat el comportament de la intel·ligència artificial ja que les seves accions són molt similars a les que es volen emular al projecte. Un comportament bàsic dels personatges regit per les seves necessitats bàsiques i preferències.

2.3. Theme Hospital (1997)

Joc desenvolupat el 1997 per Bullfrog i consisteix en ser el dirigent d'un hospital. El jugador ha de construir consultes, serveis i contractar personal perquè atenguin a pacients amb malalties imaginàries. L'objectiu principal del joc és sanar la major quantitat de pacients possible. El jugador ho aconsegueix dissenyant correctament la disposició d'elements del mapa.

És interessant analitzar aquest joc ja que està basat en el control i construcció d'un negoci Com es va acordar amb l'equip de disseny el joc que es desenvoluparà durant aquest projecte està enfocat a dur a l'èxit un negoci que combina restaurant amb videojocs, *Theme Hospital* (Bullfrog Productions, 1997) és bastant proper a aquestes mecàniques.

Els elements que destaquen a l'hora d'analitzar aquest joc són la intel·ligència artificial, el canvi de comportament d'aquesta segons estigui l'entorn de joc, i com està incorporada l'economia interna i el sistema de construcció.

Figura 2-5 *Screenshot* de Theme Hospital

Font: (Electronic Arts, 2018)

Figura 2-6 *Screenshot* Theme Hospital (stats)

Font: (Electronic Arts, 2018)

El sistema de construcció és bastant similar al referent anterior ja que construeixes per sales i cada sala té els seus elements imprescindibles. Com s'ha comentat anteriorment és el referent més semblant al resultat que es vol obtenir, no només en mecàniques sinó que també amb el comportament de la intel·ligència artificial. Així doncs també ens hem basat en aquest per dissenyar un comportament similar.

3. MARC TEÒRIC

Al llarg d'aquest apartat s'expliquen diferents conceptes necessaris per la realització del projecte. És un treball enfocat a la implementació del disseny d'un videojoc i no en dissenyar-lo, tot i així, s'explicaran diferents aspectes del disseny per tal d'entendre com funcionen i com s'han d'implementar de manera adequada.

Aquest capítol es divideix en 10 grans seccions, la primera (3.1) fa un estudi del gènere *tycoon* i els relacionats amb aquest. El seu objectiu és d'entendre el gènere, les mecàniques utilitzades i els problemes que pot ocasionar implementar-les.

Tot seguit es fa una recerca de la implementació de l'art i el mapa del joc. L'estudi comença des de les diferents formes de representar art 2D en 3D (3.2) i continua explicant les tècniques necessàries per aplicar-lo al joc (3.3).

A continuació s'explica més detalladament la mecànica principal del joc (3.4). És una visió de com han aplicat un sistema de construcció modular a diferents jocs o projectes i quines complicacions i solucions s'han trobat els desenvolupadors.

Un altre aspecte clau d'aquests jocs és el comportament de les intel·ligències artificials. En el punt 3.5 es fa un estudi de com s'aborda el tema dins la indústria dels videojocs. És important tenir en compte com ho fan al sector ja que existeixen tècniques i mètodes optimitzats pel desenvolupament de jocs.

Als videojocs és molt important informar al jugador el seu estat actual (*HUD*) i també introduir-li el joc a través de diverses pantalles (menú d'inici, menú de pausa...) aquests aspectes són incorporats als jocs de moltes maneres, al punt 3.6 s'explora quines són.

Al 3.7 s'explica en què consisteix un motor de jocs i es fa una comparativa entre els diferents motors de jocs existents actualment al mercat.

Finalment al 3.8, al 3.10 i al 3.8 s'exposen diferents eines i metodologies utilitzades pel desenvolupament de software. Aquest punt intenta explicar les diferents opcions que existeixen a la indústria a l'hora de produir un programa informàtic.

3.1. El gènere *tycoon*

L'objectiu dels jocs ve definit per les normes, per tant és arbitrari, ja que aquestes normes són dictades tal i com els dissenyadors volen (Adams, 2010). Adams explica que l'objectiu principal de la majoria dels jocs és obtenir una condició de victòria, però no tots inclouen aquesta característica. Alguns jocs només tenen condició de derrota i, exposa l'autor, és aquella que indica el final del joc. Segueix remarcant que, quan es dóna el cas de no haver victòria, però si derrota, el joc no pot ser guanyat només abandonat. Adams posa l'exemple

de *Roller Coaster Tycoon* (Chris Sawyer, 1999), en el qual l'usuari pot perdre el joc quan es queda sense diners i recursos per seguir mantenint el seu parc d'atraccions.

> ***Tycoon*** *[nom]. Un tycoon és una persona d'èxit en un negoci i, per tant, ha aconseguit ser rica i poderosa.*[1]

Considerant la definició de diccionari exposada, un *tycoon* és una persona i no una acció o una temàtica com la majoria de gèneres que coneixem (estratègia, aventures, plataformes, etc.), per això és difícil que els autors arribin a un acord. La definició que més s'apropa és la de jocs de simulació de construcció i gestió, un subgènere dels jocs de simulació; ja que, segons Wolf al llibre *The Medium of the Video Game* (2000) aquests donen al jugador el repte de balancejar l'ús limitat de recursos per construir o ampliar un imperi, una empresa o una comunitat, mentrestant l'usuari ha de combatre conra les forces internes i externes a la institució, imperi o negoci.

Adams (2010) explica que els jocs de temàtica hibrida són la combinació de dos o més gèneres i proporcionen al jugador una mescla dels diferents objectius i reptes de cada tòpic. Al seu llibre *Andrew Rollings and Ernest Adams on Game Design* (2003), escrit juntament amb Rollings, remarca que existeixen jocs que es poden categoritzar com a jocs de simulació pura de negocis i no tenen la part de construcció (per exemple *Theme Park World* (Bullfrog Productions, 1999), i jocs híbrids entre CMS i jocs de guerra (com per exemple *Age of Empires* (Ensemble Studios, 1997). Aquests jocs inclouen una part d'estratègia econòmica molt interessant inclosa dins els jocs d'estratègia.

3.1.1. Jocs de simulació de construcció i gestió (CMS)

Com s'ha comentat anteriorment, és el gènere que més similituds té amb la definició de *tycoon*, ja que el jugador pren el rol d'un magnat que controla un imperi, una ciutat, etc.

> *"Els videojocs de simulació de construcció i gestió són jocs sobre processos. L'objectiu del jugador no és derrotar a un enemic, sinó construir alguna cosa en el context d'un procés en curs. Com més bé el jugador entengui i controli el procés, més èxit tindrà en la construcció." – (Adams & Rollings, 2003, p. 417)*[2]

---

[1] Tycoon. (2017). A Cambridge Dictionary.

Recuperat de https://dictionary.cambridge.org/dictionary/english/tycoon

"*A person who has succeeded in business or industry and has become very rich and powerful*" (Traducció de l'autor)

[2] "*Construction and management simulations are games about processes. The player's goal is not to defeat an enemy, but to build something within the context of an ongoing process. The better the player understands and controls the process, the more success he will have at building*." (Traducció de l'autor)

Els jocs CMS ofereixen al jugador una interessant varietat de condicions i objectius. Explica que el més comú és donar a l'usuari un espai en blanc, i permetre-li construir com vulgui dins les limitacions de l'economia i normes del joc. Tot i així, alguns jocs proporcionen una construcció parcial al jugador des de la qual ha de seguir avançant (Adams, 2010).

Adams i Rollings (2003), exposen que l'economia interna ve definida per les normes del joc i és el sistema amb el qual els recursos són produïts, consumits i intercanviats. Cada recurs es genera en una font diferent i s'emmagatzema de forma infinita o limitada depenent del joc i el recurs.

Com s'ha mencionat anteriorment, en un joc CMS els recursos són gastats. Existeixen dues maneres comuns d'utilitzar-los, la primera depèn directament del jugador, és ell qui decideix quan gastar-los (per exemple quan construeix un edifici); i la segona pot dependre del jugador, però normalment és automàtica i consisteix en el manteniment (sous de treballadors, aigua, etc.). També existeix la conversió de recursos amb la qual el jugador transforma uns recursos en un objecte (aquestes accions reben el nom de *converters*) o en una altra font de recursos (aquestes accions reben el nom de *deadlocks*) (Adams & Rollings, 2003).

Els autors Adams i Rollings (2003) fan especial èmfasi a certs aspectes dels CMS que s'han de tenir en compte. Indiquen que els jocs d'aquest gènere simulen el comportament de grups de gent i que, aquests comportaments, necessiten d'un disseny que normalment inclou escales de felicitat i necessitats per cada individu del grup. Finalment, remarquen que s'han de tenir en compte els avisos *pop-up* per informar al jugador de certs moments crítics del joc i la informació constant sobre l'estat del joc a través de la interfície d'usuari (UI).

3.1.2. Jocs d'estratègia

Adams (2010) exposa que un joc d'estratègia premia la planificació i que el gènere, normalment, es divideix entre els jocs d'estratègia en temps real (RTS) i els jocs d'estratègia basada en torns (TBS). Aquest gènere proporciona informació a l'usuari a mesura que va jugant i l'ajuda a millorar les seves tècniques i estratègies (Apperley, 2006).

> *"Els jocs d'estratègia reten al jugador a aconseguir la victòria a través de planejar, i específicament a través de plantejar una sèrie d'accions contra un o més oponents." – (Adams, 2010, p. 419)*[3]

Adams (2010) explica que, segons aquesta definició, els jocs d'estratègia són diferents als jocs de puzle ja que els últims plantegen problemes amb absència de conflicte. També dista dels CMS ja que aquests requereixen planificació però sense oponent.

---

[3] "*Strategy games challenge the player to achieve victory through planning, and specifically through planning a series of actions taken against one or more opponents.*" (Traducció de l'autor)

> *"L'eventual èxit o fracàs dels jugadors del joc ve determinat per la seva habilitat per integrar i contextualitzar les diverses activitats que intervenen dins de les regles físiques de la simulació." – (Apperley, 2006, p. 14)*[4]

Apperley (2006) remarca que al joc *SimCity* (Maxis, 1989) el jugador ha d'integrar la informació donada per fer els canvis pertinents a diferents sectors del joc, per així, fer efectiu el procés de desenvolupament que té en marxa. És en aquesta observació on els jocs d'estratègia entren dins la definició de *tycoon*.

Com s'ha mencionat prèviament, els jocs d'estratègia consten d'un conflicte. Adams i Rollings (2003) expliquen que aquest conflicte o la seva solució, tot i ser el més comú, no ha de perquè ser militar. Existeixen jocs com *Civilization III* (Firaxis Games, 2001) que utilitzen la diplomàcia per resoldre conflictes de guerra o els jocs de les sèries *tycoon* que resolen els seus conflictes comercials a través d'acords.

A diferència dels jocs CMS, en els jocs d'estratègia el jugador normalment té el rol de comandant o dirigent. El més comú, segueixen explicant els autors, és que el jugador tingui una visió de déu i controli varis exercits de cop. Però, apunten, els jocs d'estratègia basats en un model de negoci no tenen una representació física del jugador o les seves forces (aquest model es pot observar al joc de *Roller Coaster Tycoon* (Chris Sawyer, 1999)).

3.1.3. *God Games*

Al seu llibre, Adams (2003), indica que el gènere *god games* és un subgènere de jocs CMS (p. 45) i més endavant també afirma que és un subgènere de jocs RTS (p. 55). En el primer cas, afirma que els jocs tendeixen a estar basats a simular un model de negoci, però el subgènere afegeix un element de fantasia. En el segon cas fa referència al joc *Populous* (Bullfrog Productions, 1989) el qual també té un factor de fantasia.

Els jocs CMS també poden ser categoritzats com a *god games* (Perani, Moraes, & Sanches, 2017). Els autors defineixen els *god games* com els jocs de simulació on el jugador té poders divins o sobrenaturals i els utilitza per modificar l'entorn d'una forma personalitzada causant una seria de conseqüències al joc.

Els *god games* tenen dues atraccions bàsiques com les de *SimCity* (Maxis, 1989) (Poole, 2000). La primera consisteix en dirigir i controlar un entorn amb una població aparentment autònoma. La segona, continua l'autor, és poder alterar el temps a voluntat es pot accelerar o disminuir la seva velocitat per visualitzar més fàcilment les interaccions de les dades al llarg del temps.

---

[4] "*The players' eventual success or failure at the game is determined by their skill at integrating and contextualizing the various activities involved within the physical rules of the simulation.*" (Traducció de l'autor)

3.1.4. Comparativa

A continuació s'exposa una taula comparativa resumint els continguts prèviament explicats als punts anteriors. Observant la taula (Taula 3-1) es pot veure que el joc *tycoon* conté diversos elements en comú amb els 3 gèneres analitzats, però, com ja s'ha vist a l'estudi anterior, el gènere més proper és el de jocs de simulació de construcció i gestió (CMS).

Taula 3-1 Taula comparativa de gèneres

	CMS	**Estratègia**	***God games***	***Tycoon***
Rol del jugador	• Magnat que controla un negoci, un imperi o una ciutat.	• Comandant que controla un grup de gent.	• Divinitat o ésser sobrenatural	• Magnat que controla un negoci, un imperi o una ciutat.
Elements de construcció	• Fonts de recursos • Edificis, modificar el terreny, etc. • Objectes i eines	• Fonts de recursos • Objectes i armes	• Fonts de recursos • Edificis, modificar el terreny, etc. • Objectes i armes	• Fonts de recursos • Objectes i armes • Edificis, modificar el terreny, etc.
Conflictes	• Inexistents o no bèl·lics	• Normalment bèl·lics	• Bèl·lics i no bèl·lics	• Inexistents o no bèl·lics
Accions	• Dirigir un negoci, ciutat o imperi • Controlar treballadors o vilatans • Gestionar recursos	• Dirigir un imperi, ciutat o imperi • Controlar un exèrcit • Gestionar recursos	• Controlar una civilització • Controlar vilatans • Controlar un exèrcit • Defensar civilització	• Dirigir un negoci, ciutat o imperi • Controlar treballadors o vilatans
Intel·ligència artificial	• Grups de persones movent-se pel mapa	• Grups de persones movent-se pel mapa	• Grups de persones movent-se pel mapa	• Grups de persones movent-se pel mapa
Victòria	• No existeix condició de victòria	• Vèncer la força enemiga	• Vèncer la força enemiga	• No existeix condició de victòria

Font: Elaboració pròpia

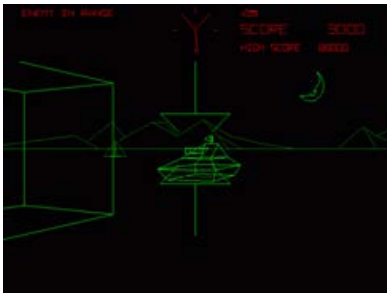
3.2. Representació en 3D d'un objecte 2D

> *"[...] una projecció ortogonal mostra una sola cara de l'objecte representat i és fàcil de realitzar, però a vegades per arribar a entendre un objecte tridimensional és necessari mostrar més d'una cara d'aquest "– (Carlbom & Paciorek, 1978, p. 472)*[5].

Existeixen diferents procediments, anomenats projeccions, per representar un objecte tridimensional en un pla bidimensional, cadascuna de les representacions té un efecte visual diferent (Krikke, 2000). L'autor també exposa que amb l'aparició dels motors 3D ha disminuït considerablement l'ús d'aquests recursos, però segueixen sent populars entre els jocs.

Mark J.P. Wolf (2009) indica que dins la indústria dels videojocs s'han utilitzat diferents maneres de representar una vista tridimensional. Fins el 1980 els jocs només utilitzaven un únic punt de perspectiva, però amb l'aparició de les vistes axonomètriques, al 1982, es va començar a optar més per aquestes tècniques. Les projeccions axonomètriques van obrir noves possibilitats a l'espai tridimensional, com van ser el desplaçament diagonal de l'escenari o l'alçada indicada per l'escala de l'objecte i/o la seva ombra (Wolf M. , 2009).

Figura 3-1 Projecció perspectiva, gràfics vectorials



Font: *Battlezone* (Atari Inc., 1980)

Figura 3-2 Projecció perspectiva + top-down

Font: *Axelay* (Konami, 1992)

La perspectiva isomètrica proporciona al jugador una visió propera a la vista d'ocell i li permet tenir una visió dels laterals dels edificis i/o objectes, per tant el jugador es sent més involucrat al joc (Adams, 2010). L'autor exposa que per aquesta raó els jocs d'estratègia i els de construcció i gestió utilitzen la vista isomètrica o una perspectiva semblant amb models 3D.

[5]"*[...] an orthographic projection shows the exact shape of one face of an object and is easy to draw. However, such a projection usually requires more than one view to represent the whole object and a great deal of experience is necessary to visualize its three-dimensional shape.*" (Traducció de l'autor)

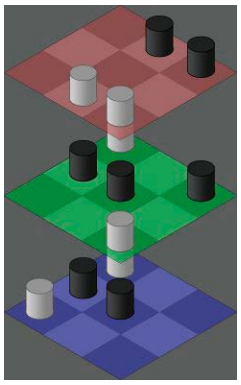
3.3. Construcció d'un món isomètric

Com s'ha esmentat prèviament, els primers jocs isomètrics van aparèixer els anys 80, abans de la disponibilitat dels acceleradors pel *hardware*. Aquests jocs utilitzen imatges 2D per crear una projecció isomètrica (Krikke, 2000). Els mons isomètrics, continua l'autor, estan constituïts de múltiples quadrats amb una rotació aplicada. Quan aquests múltiples quadrats tenen la rotació correcta, proporcionen al jugador una sensació d'espai per on poden desplaçar-se els personatges dels jocs.

Els angles de projecció poden ser ajustats, però la típica projecció isomètrica està formada per una quadricula en què les línies estan a 30 i 60 graus entre elles. Pile (2013) afirma que aquesta quadricula pot ser utilitzada per crear una graella amb funcionalitat de mapa (*tile map*), la qual té l'avantatge de poder incorporar un component d'alçada.

A la Figura 3-3 es poden apreciar diferents alçades dins d'un mateix món isomètric construït sobre línies a 45 graus en comptes de 30/60. És una imatge d'un prototip de Pile, l'autor explica que no existeix un límit a l'hora d'afegir altures al món.

Figura 3-3 Concepte de joc isomètric amb nivells



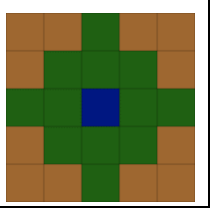
Font: (Pile Jr, 2013, p. 90)

3.3.1. *Tile maps*

Al llibre *Game Programming Algorithms and techniques* (Madhav, 2013) s'explica que existeixen jocs en dues dimensions amb grans escenaris, una manera de dibuixar tots aquests escenaris és utilitzar *tile maps*. L'autor puntualitza que els *tile maps* es formen al dividir el món en quadrats o altres polígons iguals. Tot seguit, remarca que cada quadrat conté informació vària, entre aquestes dades sempre hi ha una referència a la imatge corresponent. Aquesta referència pot ser, per exemple, un valor numèric relacionat amb una imatge concreta.

Com s'ha indicat anteriorment, per desenvolupar un videojoc amb un món isomètric s'utilitza un *tile map* amb la rotació adequada, per així aconseguir vista isomètrica. Per aplicar aquesta rotació s'han de realitzar una sèrie de transformacions explicades al següent punt (3.3.2).

Taula 3-2 Exemple de taula de referència d'un *Tile Map*

0, 0, 1, 0, 0 0, 1, 1, 1, 0 1, 1, 2, 1, 1 0, 1, 1, 1, 0 0, 0, 1, 0, 0	0 – Terra 1 – Herba 2 – Aigua	

Font: Elaboració pròpia

3.3.2. Transformacions lineals

Segons Ricardo Ramos (2011) per poder visualitzar objectes aquests han d'estar ubicats en un sistema universal de referència (SUR). Explica també, que la informàtica gràfica acostuma a utilitzar la notació matricial per variar la posició i/o mida dels objectes, aquestes variacions són anomenades transformacions lineals. Ramos (2011) exposa que per aplicar aquestes transformacions s'utilitzen els sistemes de coordenades homogenis i que aquests sistemes són els resultants d'afegir una dimensió extra al sistema de referència donat.

Com expliquen Carlbom i Paciorek (1978) les transformacions d'un objecte poden ser implementades a través d'una translació (*T*), una rotació ® i finalment una projecció (*P*) sobre el pla principal. Els autors assignen la *x* al punt a projectar i el punt projectat rep el nom de x_1. Un cop anomenats els paràmetres afirmen que $x_1 = x \cdot T \cdot R \cdot P$ on *P* és una matriu d'identitat.

3.3.2.1. Translació

Ramos (2011) explica que la translació d'un objecte consisteix en desplaçar-lo a certa distància i en una direcció determinada. Tot seguit aclareix que per desfer un desplaçament s'ha d'aplicar la translació inversa (T^{-1}).

Taula 3-3 Matriu de translació

Matriu de translació	
$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d \cdot n_1 & d \cdot n_2 & d \cdot n_3 & 1 \end{bmatrix}$	A la fórmula expressada per Calborn i Paciorek (1978) es considera d com la distància més curta des del pla fins l'origen. A la fórmula el vector (n_1, n_2, n_3) és la unitat del pla de projecció normal. Quan fem la inversa (T^{-1}) el vector canvia de signe $(-n_1, -n_2, -n_3)$

Font: (Carlbom & Paciorek, 1978; Ramos, 2011)

3.3.2.2. Rotació

Tot seguit, l'autor indica que la rotació és l'encarregada de girar el pla sobre un o varis eixos. També afirma que l'ordre en que són aplicades les rotacions és important, per tant, el producte de matrius no és commutatiu. Existeixen dos tipus de gir: el gir relatiu que és aplicat sobre un eix partint de la posició actual de l'objecte; i el gir absolut que és aplicat des de l'estat 0, és a dir quan el sistema de referència de l'objecte (SRO) està alineat amb el SUR (Ramos, 2011).

Taula 3-4 Matriu de rotació

Matriu de rotació	
$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $R_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $R_z = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	A la fórmula expressada per Ramos (2011) es considera α com l'angle que es vol rotar sobre l'eix de les X, β l'angle sobre l'eix de les Y i γ l'aplicat sobre l'eix de les Z. Per fer la inversa o desfer la rotació s'han de canviar els angles de signe (Ramos, 2011).

Font: (Ramos, 2011)

3.4. Sistema de construcció modular

Segons van comentar a la *122nd ASEE Annual Conference & Exposition* (Castronovo, Zappe, Messner, & Leicht, 2015) el procés de construcció està ple d'incògnites impredictibles i difícils de solucionar. Aquestes incògnites fan que el procés de de construcció es converteixi en un problema mal definit, i, un problema d'aquest estil, pot no tenir solució o múltiples diferents.

A la conferència pronunciada per Álvarez, Mazo i Moreno (2017) al V Congrés Internacional de videojocs i Educació (CIVE'17), es va explicar que dins la indústria dels videojocs existeixen varis jocs que utilitzen el component de construcció com a mecànica de joc, i entre ells destaquen *The Sims* (Maxis, 2000) i *Minecraft* (Mojang AB, 2009).

A nivell d'estructures arquitectòniques, dins del projecte explicat per Álvarez, Mazo i Moreno (2017), expliquen que cada edifici (o element) està segmentat en blocs individuals anomenats unitats constructives. Els autors remarquen que aquestes unitats constructives contenen informació individual sobre el seu estat, per així tenir un control més complert.

Altres autors, com Rüppel i Schatz (2011) fan un estudi sobre un *serious game* que vol explorar els efectes de l'estat d'un edifici sobre l'ésser humà durant un procés d'evacuació. Durant aquesta investigació repassen el programari utilitzat, aquest software té una secció dedicada a la informació de modelatge d'un edifici (BIM).

> *"Les connexions (joints) entre elements es poden definir fins a un total de sis graus de llibertat. La simulació de la interacció entre elements està basada en la detecció de col·lisions i la simulació dinàmica. Aquests dos components bàsics desencadenen determinades reaccions sobre els objectes." - (Rüppel & Schatz, 2011, p. 9)*[6]

El software utilitzat distingeix entre elements i les connexions que existeixen entre aquests. A la cita anterior es pot observar que els elements tenen sis graus de llibertat, el què indica que són capaços de moure's i girar sobre tres eixos (x, y i z). Aquests graus de llibertat, juntament amb la interacció entre objectes, dicten com reaccionen els objectes.

Rüppel i Schatz (2011) indiquen que una estructura (*CuboidStructure* al seu article), segons el software explicat, es divideix en múltiples cubs (a la cita els dóna el nom de *Cuboids*), i que cada cub té les diverses cares corresponents. Aquesta divisió coincideix amb la idea de la conferència d'Álvarez, Mazo i Moreno (2017) que prèviament ha estat comentada.

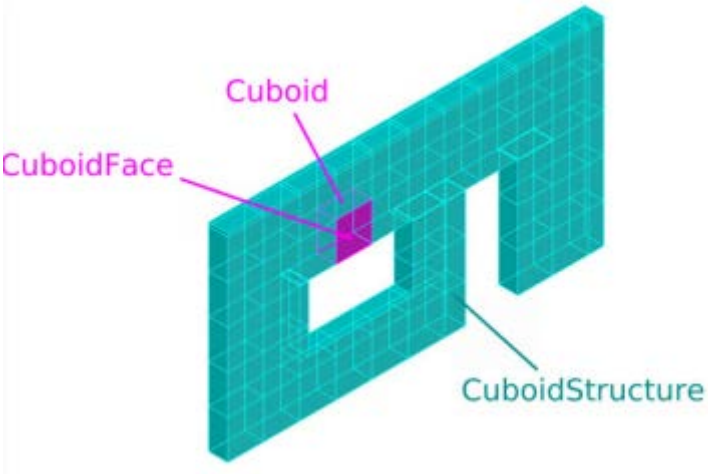
---

[6] *Connections (joints) between bodies can be defined with up to six degrees of freedom. The simulation of the interaction between objects is based on collision detection and dynamics simulation. These two core components trigger certain reactions on the objects*. - (Traducció de l'autor)

> *"Una CuboidStructure inclou totes les parts d'un element estructural, el qual representa un element estructural com una paret, una columna o un feix i amplia el seu objecte paramètric BIM. Una CuboidStructure està composta per diversos Cuboids, que a la vegada estan formats per diverses cares d'un element." - (Rüppel & Schatz, 2011, p.9)*[7]

Resumint les aportacions dels autors, un model de construcció modular conté diferents estructures, a la vegada, aquestes estan formades per varis elements, els quals es relacionen entre ells i amb el món, reaccionant de diferents formes. Cada element conté certa informació que indica al jugador, o al programa, l'estat d'aquest element i els estats disponibles que pot tenir a continuació.

Figura 3-4 Estructura dins un model de construcció modular



Font: (Rüppel & Schatz, 2011, p. 608)

[7] "*CuboidStructure includes all parts of a structural element, which represents a structural element like a wall, a column or a beam and extends their parametric BIM object. A CuboidStructure is composed of several Cuboids, which in turn have several CuboidFaces to describe the faces of a body.*." - (Traducció de l'autor)

3.5. Intel·ligència artificial 2D

3.5.1. Què és la IA?

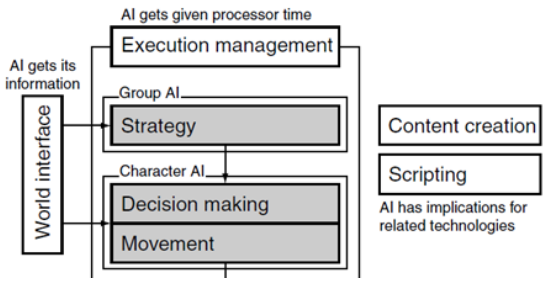
> *"La intel·ligència artificial (IA) és una branca de la informàtica que permet als ordinadors prendre decisions d'una forma semblant als humans i animals." – (Millington & Funge, 2009, p. 4)*[8]

Resumint l'explicació de Buckland (2005), existeix una diferència important entre la intel·ligència artificial acadèmica i l'aplicada als videojocs. Tot i que la IA acadèmica es pot dividir en IA forta i IA dèbil, ambdós subcamps es centren sobretot en resoldre els problemes de manera òptima, sense tenir molt en compte les limitacions de hardware. Així com la IA forta està enfocada en crear sistemes que intenten imitar els processos que realitza el cervell humà, la IA dèbil està focalitzada en resoldre situacions a la vida real. Per contra, la IA aplicada als videojocs necessita respectar unes limitacions i ha de tenir en compte que jugador gaudeixi del joc. Els desenvolupadors d'IA de videojocs són els encarregats de realitzar els algoritmes dins d'aquests paràmetres (Buckland, 2005).

3.5.2. Model de la IA d'un joc

Segons Millington i Funge (2009), la intel·ligència artificial es divideix en tres grans seccions: moviment, presa de decisions i estratègia. Les dues primeres treballen actor per actor[9], la tercera en canvi opera sobre un equip sencer. Recolzant aquest model hi ha un conjunt d'infraestructures addicionals (Figura 3-5).

Figura 3-5 Model d'Intel·ligència artificial



Font: (Millington & Funge, 2009, p. 9)

[8] "*Artificial intelligence is about making computers able to perform the thinking tasks that humans and animals are capable of.*" (Traducció de l'autor)

[9] Quan es parla d'un actor en videojocs es refereix a un personatge o objecte dins del joc.

3.5.3. Moviment

> *"Moviment es refereix als algoritmes que converteixen les decisions en desplaçament. [...] Els algoritmes de moviment poden ser més complexos que no pas moure un actor d'un punt a un altre"- (Millington & Funge, 2009, p. 9)*[10].

Buckland (2005) defineix un agent autònom com un actor que interacciona amb l'entorn del joc segons l'estat d'aquest. Aquesta definició coincideix amb la de Millington i Funge (2009), els quals expliquen que un agent és aquell actor que agafa informació del joc, pren decisions sobre quina acció dur a terme i realitza aquestes accions.

Segons Buckland (2005), un agent ha de ser capaç de resoldre una situació inesperdada i, per aconseguir-ho el moviment dels agents autònoms es divideix en tres parts. Per poder iniciar un moviment imprescindible seleccionar l'acció que es durà a terme. Per exemple saber on ha de dirigir-se. Un cop coneguda la destinació és essencial conèixer un camí que dugi l'agent al seu destí. Finalment l'agent ja pot desplaçar-se de la forma que més li interessi (Buckland, 2005).

3.5.3.1. Moviment bidimensional

Millington i Funge (2009) expliquen que avui en dia existeixen videojocs en dues dimensions i altres en tres dimensions, però moltes de les seves IA funcionen bidimensionalment. La majoria dels moviments en IA s'aconsegueixen treballant en 2D, i molts dels seus algoritmes estan definits només per aquest cas (Millington & Funge, 2009).

Tots els agents neessiten conèixer la seva posició i orientació al món del joc. Millington i Funge (2009) anomenen estàtics els algortimes i funcions que modifiquen aquests paràmetres, degut a que no contenen cap informació sobre el moviment del personatge.

Com s'ha explicat previament, s'utilitzen tècniques 2D que simulen el moviment 3D per simplificar els càlculs a realitzar i, en molts jocs, els agents estan sota la influència de la gravetat, per això es pot obviar una dimensió la major part del temps.

Millington i Funge (2009) remarquen que, degut a les lleis cinemàtiques de Newton, la velocitat d'un agent no pot canviar instantàniament. El tres autors (Buckland, 2009; Millington & Funge, 2005) coincideixen en que, a l'hora d'implementar el moviment cinemàtic d'un agent, és aconsellable crear una estructura base que s'encarregui d'actualitzar els paràmetres encarregats del desplaçament. També inclouen la necessitat d'una estructura per retornar els valors resultants dels algoritmes de moviment cinemàtic.[11]

---

[10] *"Movement refers to algorithms that turn decisions into some kind of motion. [...] Movement algorithms can be more complex than simply homing in."* (Traducció de l'autor)

[11] Als annexos es troben els pseudocodis corresponents.

Taula 3-5 Lleis del moviment o d'inèrcia de Newton

1. *Tot cos es manté en estat de repòs o en moviment rectilini uniforme, excepte si una força actua sobre ell obligant-lo a canviar d'estat.*
2. *El canvi de moviment és proporcional a la força aplicada i es produeix en la direcció cap on s'ha aplicat aquesta.*
3. *Sempre que un cos exerceix una força sobre un altre, aquest segon cos exerceix una força igual i de sentit contrari sobre el primer.*

Font: (Scheck, 2010, p. 2)[12]

3.5.3.2. Algoritmes de moviment cinemàtic

Existeixen diferents algoritmes simples que, combinats, formen els moviments complexes d'una intel·ligència artificial. Aquests algoritmes prenen els paràmetres estàtics de l'agent i retornen la velocitat desitjada (Millington & Funge, 2009).

Segons Millington i Funge (2009), la majoria dels jocs simplifiquen la part de programació i forcen l'orientació del personatge cap a la direcció del desplaçament. Aquesta orientació forçada s'aconsegueix a partir de la velocitat del personatge calculada amb l'algoritme.

Resumint molt les explicacions de Millington i Funge (2009), els dos algoritmes bàsics *Seek* (buscar) i *Wander* (deambular) són la base per la creació de la resta de moviments, els quals són aconseguits a través d'adaptacions que ajuden a obtenir altres comportaments interessants. Millington i Funge (2009) assenyalen que el comportament *Seek* calcula la velocitat a partir de les dades actuals del personatge i de la destinació desitjada. Expliquen

[12] "*I. Every body continues in its state of rest or of uniform rectilinear motion, except if it is compelled by forces acting on it to change that state.*

II. The change of motion is proportional to the applied force and takes place in the direction of the straight line along which that force acts.

III. To every action there is always an equal and contrary reaction; or, the mutual actions of any two bodies are always equal and oppositely directed along the same straight line." (Traducció de l'autor)

que, per aconseguir-ho, primer es fa la diferència entre la posició de destinació i l'actual del personatge, obtenint així, la direcció de la velocitat.

Un cop s'ha definit el comportament *Seek*, els autors amplien que es pot adaptar per aconseguir varis comportaments, entre ells el *Flee* (fugir) i l'*Arriving* (*Flee* però amb una aproximament suau).

Seguidament, Millington i Funge (2009) expliquen com realitzar el comportament de *Wander* i que aquest comportament s'utilitza per controlar el personatge sense cap objectiu en concret. Els autors sostenen que si s'assigna un valor aleatori a la direcció del personatge aquest vaga sense rumb per l'escenari, però de forma molt erràtica. Expliquen que amb certes variacions com col·locar un objectiu "en orbita" al personatge i moure'l aleatòriament es poden aconseguir moviments menys aleatoris. Els dos autors (Millington & Funge, 2009) expliquen detalladament diferents comportaments que es poden aconseguir modificant certs paràmetres i algoritmes dels explicats fins al moment.

3.5.4. *Pathfinding*

Existeixen diferents maneres d'aconseguir que una IA es desplaci per l'escenari fins un punt determinat, tal com expliquen Millington i Funge (2009) si es tracta d'un escenari senzill és molt fàcil assignar una ruta, però es complica quan el mapa té formes intricades o objectes escampats. Quan es complica l'escenari normalment s'utilitza el *pathfinding* (o *path planning*) per calcular el recorregut correcte (Pile Jr, 2013).

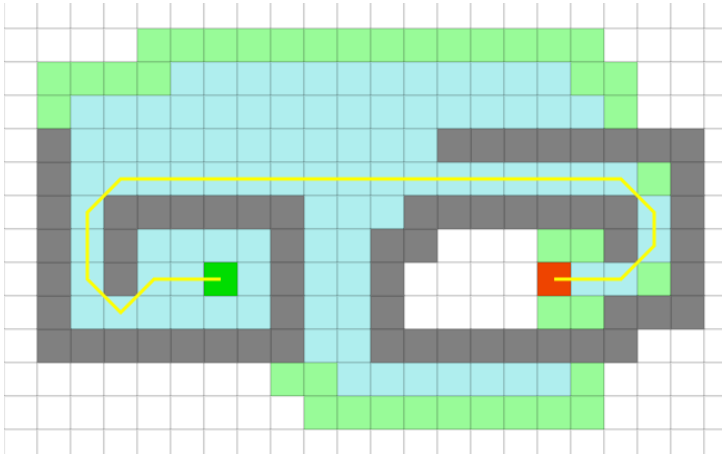
El *pathfinding* es troba actualment a totes les IA, i s'ajuda d'un gràfic que interpreta l'entorn per aplicar l'algoritme (Millington & Funge, 2009).

Els mètodes més utilitzats per obtenir la informació espacial del nivell, en els jocs moderns, són els basats en el *tile map*, en els punts de visibilitat, en la geometria ampliada o en el *navmesh* (Buckland, 2005).

3.5.4.1. Gràfic basat en *tile map*

Com s'ha mencionat anteriorment, els *tile maps* estan formats per graelles on cada cel·la conté certa informació. Buckland (2005) explica que cada node del gràfic representa el centre d'una cel·la del *tile map*, i que aquestes cel·les estan connectades amb les adjacents a través de les vores (*edges*). Depenen del tipus de terreny que representi la cel·la, passar per ella tindrà un cost o un altre, aquest cos serà el què, a través d'algoritmes, determinarà el camí més òptim (Buckland, 2005).

Figura 3-6 *Pathfinding* basat en *tile map*

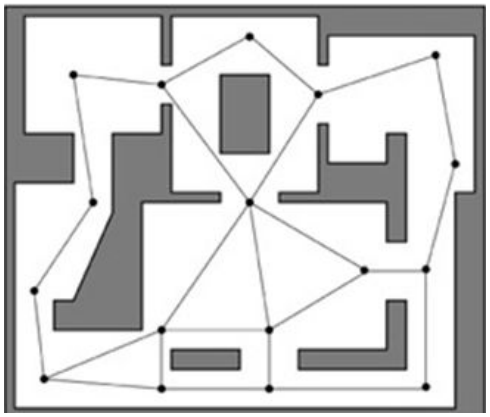


Font: (Xu, 2011)

3.5.4.2. Gràfic basat en punts de visibilitat (POV)

Un gràfic de navegació basat en punts de visibilitat (*points of visibility*) està format per nodes, normalment col·locats a mà, als punts importants de l'escenari i mínim tenen una connexió amb un altre node (Buckland, 2005). L'autor desaconsella aquest mètode per mapes de generació aleatòria.

Figura 3-7 *Pathfinding* basat en punts de visibilitat

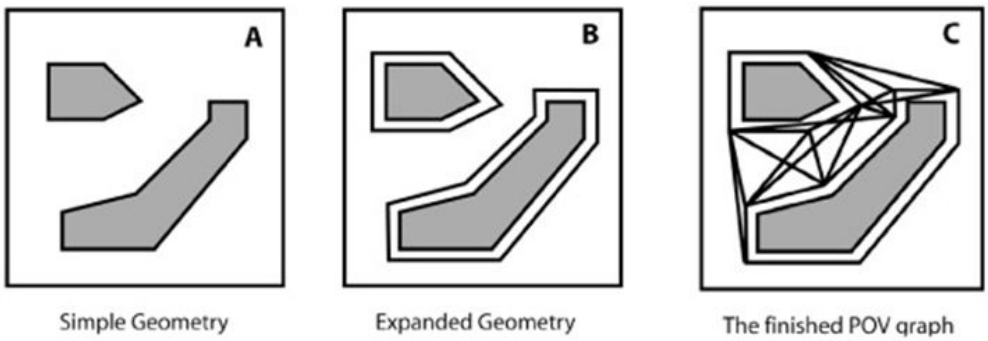


Font: (Buckland, 2005, p. 334)

3.5.4.3. Gràfic basat en geometria ampliada

El nivell d'un joc està construït a partir de polígons i és possible utilitzar-los per crear un gràfic POV (Buckland, 2005). L'autor explica que, ampliant els polígons de forma proporcional i unint els nous vèrtex trobats, es pot crear un gràfic POV i aquest pot ser calculat a temps real.

Figura 3-8 *Pathfinding* basat en geometria ampliada



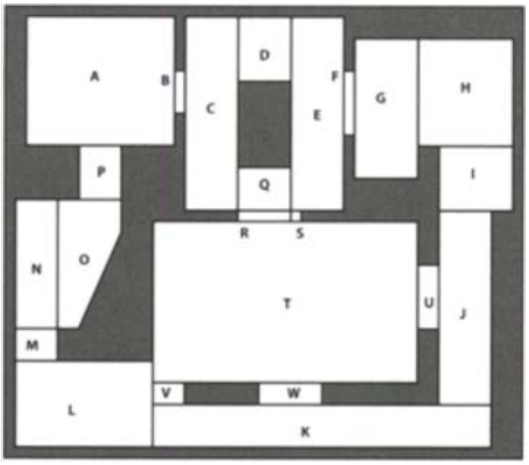
Font: (Buckland, 2005, p. 335)

3.5.4.4. Gràfic basat en *navmesh*

Entre els programadors de videojocs està creixent la tendència a utilitzar una xarxa de polígons convexes, anomenada *navmesh*, per definir les àrees transitables del nivell (Buckland, 2005).

Buckland (2005) explica que aquest mètode és eficient i molt ràpid a l'hora d'accedir la informació desada.

Figura 3-9 *Pathfinding* basat en navmesh



Font: (Buckland, 2005, p. 336)

3.5.4.5. Algoritmes de *pathfinding*

Buckland (2005) exposa que existeixen varis algoritmes per trobar un camí correcte a l'hora de desplaçar un personatge d'un punt a un altre. Puntualitza també, que no tots els camins que troben aquests algoritmes són òptims o els més curts a realitzar. Buckland (2005) aclareix que l'algoritme creat per Edsger Dijkstra i la seva adaptació (l'algoritme d'A*) cerquen el recorregut més curt. Ambdós algoritmes necessiten un gràfic en el qual estiguin assignats els costs de transitar entre nodes, també necessiten que estiguin assignats el punt inicial i la destinació desitjada (Millington & Funge, 2009).

Millington i Funge (2009) expliquen que l'algoritme Dijkstra treballa a partir d'iteracions. Cada iteració té en compte un node del gràfic i les seves connexions i, a través de calcular el pes o cost de cada connexió, l'algoritme és capaç d'escollir el node més proper a la destinació i desestimar els seus veïns (Millington & Funge, 2009).

Buckland (2005) indica que l'algoritme A* és una modificació del Dijkstra i que a través de càlculs permet saber si s'estan calculant els nodes en la direcció correcta, per tant evitar analitzar més nodes dels necessaris. Per aconseguir aquesta optimització l'A* s'ajuda de l'heurística, per tal d'estimar un cost provisional des del node observat fins la destinació (Buckland, 2005).

3.5.5. Presa de decisions

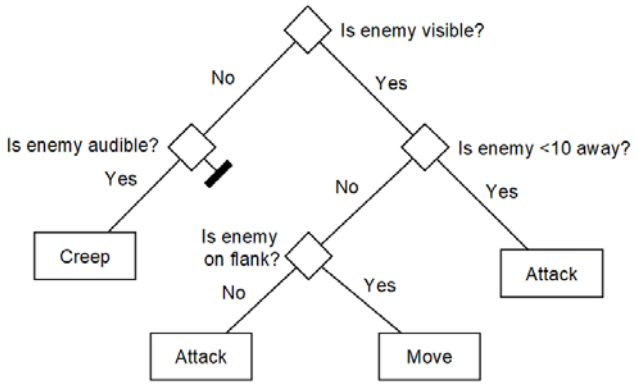
Resumint l'explicació de Millington i Funge (2009), els personatges controlats per una intel·ligència artificial processen informació per dur a terme una acció, aquest processament se li dóna el nom de presa de decisions. La majoria de jocs utilitzen sistemes molt senzills per fer la presa de decisions, normalment aquestes tècniques són els arbres de decisió (*Decision Tree*), les màquines d'estat (*State Machines*) i els arbres de comportament (*Behaviour trees*). Els últims anys s'han estudiat eines més potents per dur a terme aquestes decisions (Millington & Funge, 2009).

3.5.5.1. Arbres de decisions

Els autors puntualitzen que la tècnica més senzilla d'entendre i implementar són els arbres de decisions (Figura 3-10). Segons expliquen, tenen l'avantatge de ser molt modulars i fàcils de crear. Però l'inconvenient de tenir una assignació entre els valors d'entrada i les accions de sortida bastant complexa (Millington & Funge, 2009).

Un arbre de decisions està format per diferents decisions encadenades entre si; a mesura que es van prenent decisions s'arriba a una acció o a una altra (Millington & Funge, 2009).

Figura 3-10 Arbre de decisions



Font: altra (Millington & Funge, 2009, p. 296)

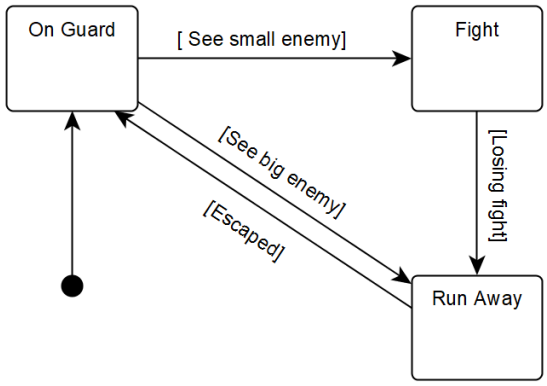
3.5.5.2. Màquines d'estat

Segons Rabin (2002), el mètode més utilitzat per la presa de decisions, en la intel·ligència artificial destinada a videojocs, són les màquines d'estat (Figura 3-11). Rabin (2002) explica que la popularitat d'aquesta tècnica és deguda a la seva fàcil implementació i comprensió, a més, tenen l'avantatge de ser fàcils de *debugar*. Segons l'autor l'inconvenient de les màquines d'estat es troba en la tendència dels programadors a no seguir una estructura coherent. Milliginton i Funge (2009)recolzen aquesta afirmació explicant que cada

programador les aplica de diferent forma i no existeix un únic algoritme definit per les màquines d'estat.

Una màquina d'estats està formada per estats i transicions. Així les defineixen els autors Millington i Funge (2009) i continuen explicant que cada transició connecta un estat amb un altre. Quan el personatge de la IA està en un estat, i es compleix la condició de la transició, aquest passarà a estar en el següent estat corresponent.

Figura 3-11 Màquina d'estats simple



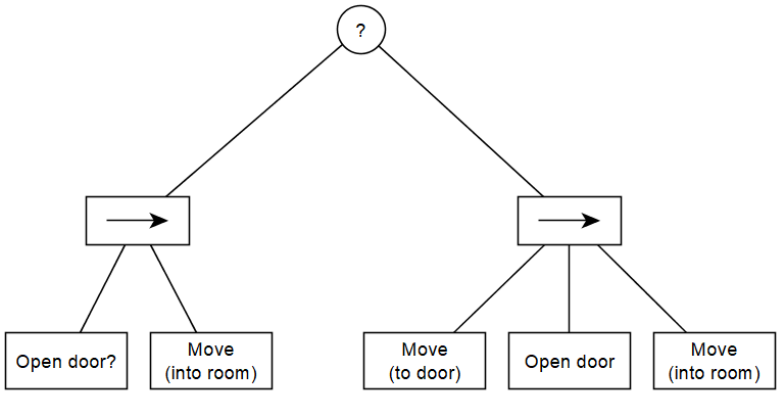
Font: altra (Millington & Funge, 2009, p. 310)

3.5.5.3. Arbres de comportament

Millington i Funge altra (2009) exposen una recent popularització dels arbres de comportament com a eina per crear personatge amb intel·ligència artificial. Afirmen que aquesta tècnica té moltes coses en comú amb les màquines d'estat però, en comptes d'estats, els blocs principals són tasques.

Un arbre de comportament està compost per tres elements bàsics: Condicions, Accions i Composts (Millington & Funge, 2009). Els autors descriuen les condicions com un element que comprova un propietat del joc. Les accions són definides com un component que altera l'estat del joc, ja siguin animacions, moviment, canvi de paràmetres del personatge, un canvi en l'àudio, etc. Per acabar els autors expliquen que els composts són els encarregats de gestionar i connectar entre si les accions i les condicions. Tot i haver varis composts, n'existeixen dos de bàsics: els selectors i les seqüències i ambdós decideixen l'orde en què s'executen la resta de les tasques (Millington & Funge, 2009).

Figura 3-12 Arbre de comportament amb nodes composts



Font: altra (Millington & Funge, 2009, p. 337)

A la Figura 3-12 pot observar un arbre de comportament, el primer pas consisteix en un selector que divideix la decisió en dues possibles branques. La primera comprova una condició: la porta és oberta? En cas positiu el personatge acabaria d'executar la primera seqüència i es mouria dins l'habitació. Per contra, en cas negatiu, l'arbre procediria a executar la segona seqüència i es mouria cap a la porta, l'obriria i finalment entraria dins l'habitació.

Els arbres de comportament, com tots els models explicats fins al moment, poden ser molt simples, com el de l'exemple, o arribar a ser molt complexes (Millington & Funge, 2009).

3.5.6. *Blackboards*

> *"Tots els sistemes socials [...] requereixen d'una certa coordinació d'acció. Cada individu necessita tenir un rol específic, i els objectius d'alt nivell han de ser atesos per accions no cooperatives, de baix nivell." (Isla & Blumberg, 2002).*[13]

Isla i Blumberg (2002) expliquen que l'arquitectura de *blackboards* fa un apropament a aquesta coordinació, ja que tots els individus comparteixen un espai on consultar i actualitzar informació. Millington i Funge (2009) afegeixen que una *blackboard* està constituïda de tres parts: un equip d'experts que prenen les decisions (eines de presa de decisions dedicades a un aspecte diferent del joc cada una), una pissarra (*blackboard*) i un àrbitre.

Segons les definicions i explicacions explicades prèviament, les *blackboard* són una eina on cada individu pertanyent a un grup bolcarà la informació que reculli, i d'on cada individu, a

[13] "*Any type of social system-whether a swarm of bees, a pack of wolves, a sports team, or an army-requires some coordination of action. Individuals need to take certain roles, and high-level goals need to be served by occasionally nonobvious, low-level, cooperative action.*" (Traducció de l'autor)

més, agafarà la informació necessària per prendre decisions. També es pot remarcar que, segons les definicions, existeix un àrbitre que controla aquest tràfic d'informació.

3.5.7. *Scripting*

El llenguatge *scripting* és qualsevol llenguatge de programació que hagi sigut creat per simplificar una tasca particular de programació (Berger, 2002).

A mesura que la producció de videojocs es va anar fent més complexa es va necessitar separar contingut del motor de jocs. Millington i Funge (2009) expliquen que els dissenyadors necessitaven retocar certs aspectes del comportament dels personatges i per això es van crear tècniques com *l'scripting*, per fer-ho possible. Els autors indiquen que l'*scripting* va portar amb ell efectes secundaris i inesperats, com obrir als jugadors la possibilitat de crear el seu propi comportament pels personatges del joc.

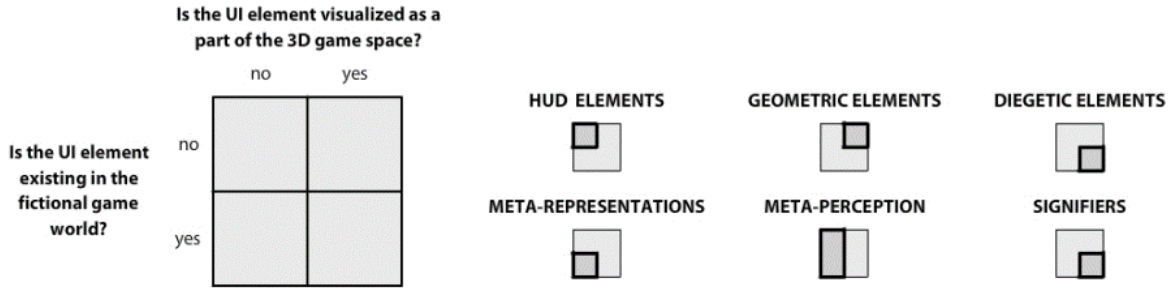
Entre altres, els avantatges d'utilitzar *scripting* són la velocitat d'execució que guanya el joc, ja que alguns llenguatges utilitzats (com LUA) són *byte-compiled* i permeten un format més compacte de codi (Millington & Funge, 2009). A més, permet a l'equip treballar en tasques paral·lelament i és una eina fàcil d'utilitzar que permet ampliar ràpidament certes parts de la IA del joc (Tozou, 2002).

3.6. Interfície d'usuari (UI)

Adams (2010) exposa que la interfície d'usuari es troba entre el jugador i el joc, és coneixedora de totes les entrades i sortides del hardware i l'encarregada de transmetre-ho. Existeixen diferents tipus d'informacions necessàries pel jugador, així com són elements visuals i d'àudio.

Aquestes idees també són exposades per Fagerholt i Lorentzon (2009), els quals diferencien en diferents categories de *UI* segons la seva integració al món, tan visual com narrativament (Figura 3-13): elements de *HUD*, meta-percepció, meta-representacions, elements geomètrics, elements diegètics i significants.

Figura 3-13 Categories d'elements de IU



Font: (Fagerholt & Lorentzon, 2009, p. 51)

Els autors es refereixen al HUD com a elements no-diegètics i aquests són els que no pertanyen ni narrativament ni espacialment al món del joc, posen d'exemple el *HUD* típic dels FPS.

Els elements de meta-percepció transmeten informació sobre l'estat intern del joc. No tenen una representació espacial dins del món del joc, però a vegades tenen una connexió amb la narrativa d'aquest. Per exemple la sang que embruta la pantalla al ser ferits al *Killzone 2* (Guerrilla Games, 2009) (Fagerholt & Lorentzon, 2009).

En canvi, expliquen els autors, les meta-representacions són entitats existents al món narratiu, però es visualitzen de forma no espacial al món del joc, un exemple clar és el mòbil de *Gran Theft Auto IV* (Rockstar North, 2008) que és un objecte de l'inventari, però que es presenta de forma no diegètica.

Existeixen també els elements geomètrics, aquests són representacions espacials d'informació, però sense pertànyer al món narratiu del joc, per exemple els perímetres dels personatges a *Left 4 Dead* (Valve Software, 2008) (Fagerholt & Lorentzon, 2009).

Una forma d'incloure els elements de manera més immersiva al joc, exposen els autors, és posar-los com a elements diegètics, aquests són pertanyents del món físic i narratiu del joc. Un exemple molt clar és la interfície de *Dead Space* (EA Redwood Shores, 2008). Els significants són un subgrup dels elements diegètics, però transmeten la informació de forma subtil perquè el jugador la interpreti.

3.6.1. Heads-up Display (HUD)

> *"Un head-up display, o HUD, és qualsevol visualització transparent que presenti dades sense obligar a l'usuari a modificar el seu punt de vista habitual. [...]. Als jocs, el terme HUD fa referència al mètode pel qual la informació es transmet visualment al jugador mentre un joc està en procés. El HUD s'utilitza normalment per mostrar simultàniament diverses dades, com ara la salut del protagonista, els objectes i els indicadors de progrés i objectius del joc. - (Fagerholt & Lorentzon, 2009, p. 1)*[14]

Segons la conferència pronunciada per Peacocke, Teather i Carette (2015),existeixen diferents formes de mostrar la informació de l'estat actual del joc. Aquesta afirmació coincideix en part amb la categorització feta per Fagerholt & Lorentzon (2009) i la conferència DiGRA'11[15] on Llanos i Jørgensen (2011) expliquen que tot i que els jocs digitals moderns solen integrar part de la interfície d'usuari al món del joc, encara s'utilitza el concepte HUD per referir-se a les parts d'interfície d'usuari que no estan integrades al món, sinó superposades.

De fet, Peacocke, Teather i Carette (2015) exposen que un dels mètodes consisteix en mostrar la informació a la capçalera (*heads-up display* o *HUD*), i aquest mostra la informació al jugador a través de mètriques, barres, icones o números. I un altre mètode, en canvi, incorpora la informació al mateix món del joc. A vegades la indústria es refereix a aquest últim com diegètic, ja que pertany al mateix món i no és extern a ell.

Com molt bé explica Alejandro Selma (2017) al seu treball final de carrera, existeixen diferents maneres d'incorporar un *HUD* amb un motor de jocs. Exposa que en el cas concret de *Unity3D* existeixen dues maneres de fer-ho, la primera està limitada en funcionalitat, però la segona és molt flexible a l'hora d'incloure nous elements més complexes.

3.6.2. Menús

Al seu llibre, Adams (2010) explica que existeixen diferents estats del joc. Per exemple si el jugador pren decisions o realitza accions que afecten al món del joc, es troba en el mode de joc. Tot i així, actualment la majoria de jocs tenen altres modes que permeten al jugador realitzar altres canvis (variar la resolució, el volum, llengua, guardar, posar en pausa el joc, etc.) i aquests modes s'anomenen menús.

---

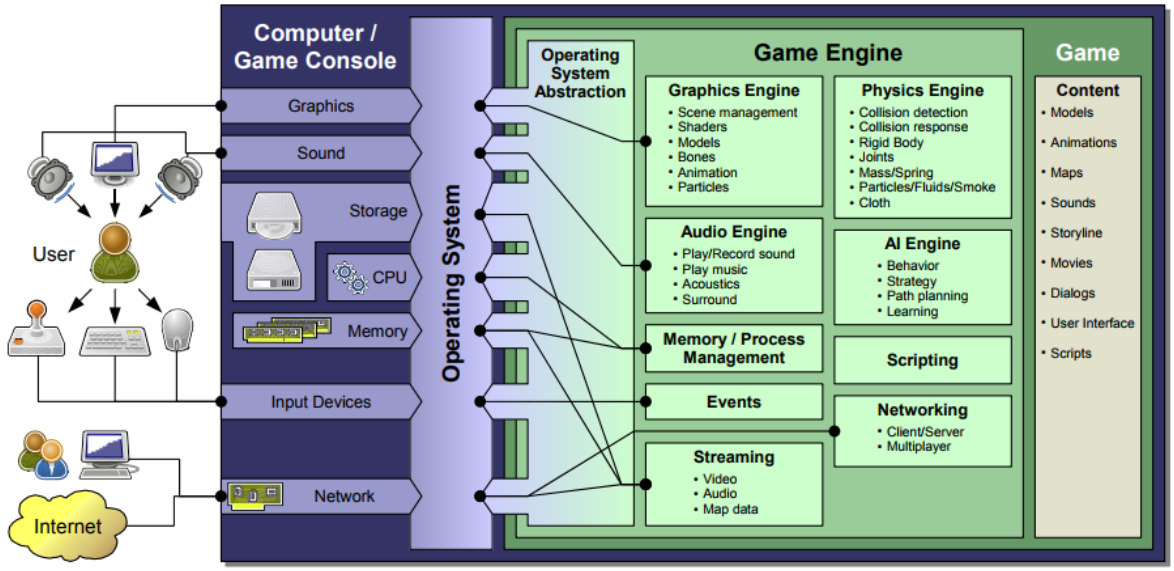
[14] *"A head-up display, or HUD, is any transparent display that presents data without requiring the user to look away from his or her usual viewpoint. [...]. In games, the term HUD refers to the method by which information is visually conveyed to the player whilst a game is in progress. The HUD is frequently used to simultaneously display several pieces of information such as the main character's health, items, and indicators of game progression and goals."* – (Traducció de l'autor)

[15] 2011 DiGRA International Conference: Think Design Play

3.7. Motors de joc

Software complex necessari pel desenvolupament de videojocs. Així defineixen Marks, Windsor i Wünsche (2007) un motor de jocs. Expliquen que els motors de joc moderns es poden dividir en diferents blocs funcionals.

Figura 3-14 Blocs funcionals d'un motor de jocs



Font: (Marks, Windsor, & Wünsche, 2007, p. 275)

Com es pot veure a la Figura 3-13, existeix una part encarregada de controlar l'apartat gràfic del joc, un bloc versat en els càlculs matemàtics per simular les físiques, una part dedicada a la gestió de sons i una secció la qual controla els esdeveniments d'entrada que pot generar l'usuari. També existeix una part encarregada de la gestió de memòria, que és la responsable de netejar d'aquesta el contingut que ja no s'utilitza per deixar lloc pel nou. Finalment es poden veure dos blocs dedicats a la connexió online (*streaming* i *networking*), una part exclusiva per la intel·ligència artificial i una secció per l'*scripting*.

3.7.1. Unreal Engine 4 (UE4)

> *"Unreal Engine 4 és un conjunt complert d'eines de desenvolupament de jocs realitzades pels desenvolupadors de jocs, per als desenvolupadors de jocs.- (EpicGames, 2018)*[16]

Segons el lloc web oficial (EpicGames, 2018), UE4 està en contínua revisió per estar sempre actualitzat amb les noves tendències, a més, tot i només poder desenvolupar en un sistema operatiu Windows, MAC OS X o Linux permet desenvolupar per diferents plataformes de llançament.

Taula 3-6 Plataformes e llançament d'UE4

PLATAFORMES DE LLANÇAMENT			
PC	WINDOWS PC	Dispositius mòbils	iOS
	MAC OS X		Android
	LINUX	Realitat Virtual (no limitat a aquests dispositius)	SteamVR/HTC Vive
CONSOLA	PlayStation 4		Oculus Rift
	Xbox One		PlayStation VR
	SteamOS		Google VR/Daydream
Web	HTML5		OSVR
			Samsung Gear VR

Font: Elaboració pròpia a partir de les dades extretes de la font oficial

La font oficial indica que, entre altres avantatges, UE4 és gratuït fins que un joc publicat per l'usuari sobrepassi els 3.000 dòlars[17]. Seguint amb els avantatges, UE4 és eina de fàcil utilització pels dissenyadors, ja que utilitza *blueprints* per programar visualment i existeix molta documentació. Però també és una eina molt potent pels programadors ja que poden ampliar codi amb C++ i, a més, tenen accés al codi font. No només facilita la programació amb *blueprints*, sinó que té diverses classes prefabricades que ajuden a dur a terme certes

[16] "*Unreal Engine 4 is a complete suite of game development tools made by game developers, for game developers.*" (Traducció de l'autor)

[17] Passats els 3000$ l'usuari haurà d'aportar el 5% dels beneficis a EpicGames (empresa desenvolupadora de UE4).

parts de programació com per exemple la creació d'un personatge, la càmera i les cinemàtiques, etc., també té un ampli ventall d'eines pel desenvolupament de la IA.

3.7.2. Unity3D

> *"Un editor altament flexible i amb múltiples prestacions. Disponible en Windows i Mac, l'editor de Unity és el centre creatiu on artistes, dissenyadors i desenvolupadors treballen junts" - [18] - (Unity, Editor, 2018)*

Unity3D no només és un motor de jocs gratuït que ofereix la possibilitat de crear els teus jocs des de zero, sinó que, com també ho fa UE4, l'usuari pot descarregar-se projectes ja començats, o inclús acabats, per modificar-los com ell vulgui.

El motor de jocs ofereix diferents eines que faciliten la implementació de la IA i les físiques, a més, en les últimes actualitzacions han afegit diferents utilitats que ajuden al jugador amb la realització de cinemàtiques, monetització, etc.

Segons el lloc web oficial (Unity, Unity3D, 2018), com ja hem vist amb el motor anterior, *Unity3D* està en contínua revisió per estar sempre actualitzat amb les noves tendències. A diferència d'UE4, només pot desenvolupar en un sistema operatiu Windows i MAC OS X, però permet desenvolupar per més varietat de plataformes de llançament.

Taula 3-7 Plataformes de llançament de Unity3D

PLATAFORMES DE LLANÇAMENT			
PC	WINDOWS PC	Realitat Virtual	SteamVR/HTC Vive
	MAC OS X		Oculus Rift
	LINUX		PlayStation VR
CONSOLA	PlayStation 4		Google VR/Daydream
	PSVita		Google Cardboard
	WiiU		Samsung Gear VR
	3DS		Windows mixed reality
	Xbox One	Realitat augmentada	Windows mixed reality
	SteamOS		Apple ARKit
	Nintendo Switch		Google ARCore

[18] "*Un editor altamente flexible y con múltiples prestaciones. Disponible en Windows y Mac, el editor de Unity es el centro creativo donde artistas, diseñadores y desarrolladores trabajan juntos.*" (Traducció de l'autor)

Dispositius mòbils	iOS		Vuforia
	Android	Web	Facebook Gameroom
	FireOs		WebGL
		SmartTV	AndroidTV
			Samsung SmartTV
			tvOS

Font: Elaboració pròpia amb dades extretes de la font oficial

3.7.3. Comparativa de motors

Com ja s'ha pogut veure a l'anàlisi anterior, els dos motors són molt semblants, però cadascun té unes característiques diferents. Al fixar-se en les plataformes de llançament es pot veure que el ventall és més ampli pel motor de jocs *Unity3D*, però en canvi *UE4* permet desenvolupar jocs amb el sistema operatiu Linux.

Comparant la informació de les pàgines oficials (EpicGames, 2018; Unity3D, 2018), es pot veure que Unity destaca per la versatilitat a l'hora de desenvolupar en 2D o 3D, en canvi *UE4*, tot i permetre el desenvolupament en 2D, està més enfocat als jocs en tres dimensions. Tot i tenir una corba d'aprenentatge més elevada al principi d'utilitzar el motor de l'empresa EpicGames, *Unreal* és una gran eina per realitzar prototipatge ràpid, ja que els propis dissenyadors poden construir-lo gràcies al visual *scripting* d'*Unreal*: els *blueprints*.

A la taula de continuació es poden veure diferents característiques d'ambdós motors comparades entre si.

Taula 3-8 Taula comparativa UE4 Vs. Unity3D[19]

	UE4	Unity3D
Preu	Gratuït fins als 3000$	Gratuït fins 200.000$
Sistema Operatiu	Windows, MAC OS X, Linux	Windows, MAC OS X
Fàcil *sourceControl*	SI	SI
2D	SI	SI
3D	SI	SI
Ajuda amb IA	SI	SI
Productes de suport	Marketplace	AssetStore
Editor de materials	SI (*blueprints*)	SI (*scripting*)
Llenguatge de programació	Blueprints, C++	C#, UnityScript
Accés al codi font	SI	NO
Suporta realitat virtual	SI	SI
Suporta realitat augmentada	NO	SI
Documentació accessible	SI	SI
Físiques dins el motor	SI	SI
Editor d'animacions	SI	SI
Multijugador	SI	SI

Font: Elaboració pròpia amb dades extretes de les fonts oficials

[19] Tota la informació ha estat extreta de les fonts oficials (EpicGames, 2018; Unity3D, 2018) l'any 2018, com es tracta de software que esta en contínua actualització, algunes d'aquestes característiques variaran amb el temps.

3.8. Llenguatges de programació

Milena Vujosevic-Janicic i Dusan Tosic (2008) introdueixen, al seu article, que als últims 50 anys s'han creat milers de llenguatges de programació, però tots ells es basen en els mateixos conceptes de programació i, a més, no tots aquests llenguatges han sigut capaços de sobreviure al llarg dels anys.

Cada llenguatge de programació té la seva sintaxis i característiques pròpies i, aquestes, s'han de tenir en compte a l'hora d'escollir un llenguatge o un altre. Com ja s'ha comentat existeixen molts llenguatges de programació i contínuament s'estan creant de nous, en aquest apartat es farà un repàs per alguns dels més destacats als videojocs els últims anys. Tenint en compte que explicar un llenguatge de programació pot ser molt extens, només es farà una breu ullada als seus orígens i característiques més importants.

3.8.1. C++

> *"Bjarne Stroustrup de AT&T Bell Labs va crear el llenguatge C++ a mitjans dels anys 80. El C++ és una ampliació del llenguatge de programació C, un producte de AT&T Bell Labs de principis dels anys 70. El C va se desenvolupat per escriure el sistema operatiu Unix, i el C és àmpliament utilitzat per al programari de sistema i el desenvolupament de sistemes incrustats."* – (Halterman, 2017, p. 6)[20]

Richard L. Halterman (2017) exposa que els programes desenvolupats amb C++ han de tenir una sintaxi en particular, i si no és correcta el compilador generarà missatges d'error i no transformarà el codi el llenguatge màquina.

3.8.2. C#

Svetlin Nakov (2013) explica que la primera versió del llenguatge C# va ser desenvolupada per Microsoft entre els anys 1999 i 2002 i, oficialment, llançat al mercat al 2002 com una part de .NET[21]. L'autor amplia que és un llenguatge de programació d'ús general, orientat a objectes i d'alt nivell. I que, tot i provenir del C i el C++, hi ha moltes característiques d'aquests llenguatges que no les suporta per tal de simplificar-lo.

---

[20] "*Bjarne Stroustrup of AT&T Bell Labs created C++ in the mid 1980s. C++ is an extension of the programming language C, a product of AT&T Bell Labs from the early 1970s. C was developed to write the Unix operating system, and C is widely used for systems-level software and embedded systems development.*" – (Traducció de l'autor)

[21] La plataforma .NET pretén facilitar el desenvolupament de programari per a Windows proporcionant un nou enfocament de qualitat a la programació, basat en els conceptes de la "màquina virtual" i el "codi administrat".

3.8.3. LUA

Roberto Ierusalimschy (2006) explica que els orígens del llenguatge de programació Lua es remunten al 1993, quan Waldermar, Luiz i ell mateix van començar a desenvolupar-lo. Al principi només era per crear dos projectes concrets però, actualment i contra tot pronòstic, és utilitzat en molts àmbits. Això és degut a la seva senzillesa, fàcil ampliació, portabilitat i eficàcia com a llenguatge *scripting*.

> *"Hem dissenyat Lua, des del principi, per integrar-se amb programari escrit en C i en altres idiomes convencionals. Aquesta integració comporta molts beneficis. Lua és un llenguatge mínim i senzill, en part perquè no intenta fer el C ja realitza correctament, com ara el rendiment, les operacions de baix nivell i la interfície amb el programari de tercers. Lua confia en C per a aquestes tasques. El que Lua ofereix és el que no és bo per a C: una bona distància del maquinari, estructures dinàmiques, sense redundàncies, facilitat de prova i depuració."* – (Ierusalimschy, 2006, p. XIII)[22]

Explica també que la majoria de la força del llenguatge prové de les seves llibreries, la tipificació dinàmica, la gestió automàtica de memòria, etc. però també la seva gran extensibilitat. Lua funciona, com diu l'autor, com un llenguatge cola ja que és capaç d'unir components escrits en llenguatges diferents.

3.8.4. JavaScript

Segons explica David Flanagan al seu llibre *JavaScript The definitive Guide* (2011), *JavaScript* és el llenguatge de programació dels lloc web, l'autor indica que la majoria dels lloc web moderns, i navegadors web, estan fets amb aquest llenguatge.

JavaScript és un llenguatge interpretat d'alt nivell, dinàmic i sense tipificació; és molt adequat per programació orientada a objectes i funcionals. *JavaScript* té elements provinents de diferents llenguatges, la seva sintaxi ve de Java, les seves funcions de *Scheme* i la seva herència està basada en *Self*.

---

[22] "*We designed Lua, from the beginning, to be integrated with software written in C and other conventional languages. This integration brings many benefits. Lua is a tiny and simple language, partly because it does not try to do what C is already good for, such as sheer performance, low-level operations, and interface with third-party software. Lua relies on C for these tasks. What Lua does offer is what C is not good for: a good distance from the hardware, dynamic structures, no redundancies, ease of testing and debugging.*" (Traducció de l'autor)

3.9. Altres formats d'arxius

3.9.1. XML

> "*XML va ser desenvolupat per un grup de treball dedicat al XML (originalment conegut com SGML Editorial Review Board) format sota els auspicis del World Wide Web Consortium (W3C) el 1996.*" - (Bray, et al., 1997, p. 1)[23]

Els autors del llibre *Extensible Markup Language (XML)* expliquen que l'XML és un llenguatge de marques. Els documents XML són conformes amb els documents SGML[24] i estan formats d'unitats d'emmagatzematge anomenades entitats, i aquestes contenen dades analitzades i sense analitzar.

Figura 3-15 Exemple d'un document XML amb declaració de tipus de document

```
<?xml version="1.1"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

Font: (Bray, et al., 1997, p. 10)

3.9.2. CSV

El format de valors separats per coma (*comma-separated values* o *CSV*) s'utilitza per intercanviar i convertir dades entre diferents programes en poc temps (Shafranovich, 2005).

Tot i que existeixen diferents especificacions i implementacions pel format CSV, no existeixen especificacions formals amb un ampli marge d'interpretacions.

Figura 3-16 Exemple arxiu CSV

```
30;"unemployed";"married";"primary";"no";1787;"no";"no";"cellular";19;"oct";79;1;-
1;0;"unknown";"no"
33;"services";"married";"secondary";"no";4789;"yes";"yes";"cellular";11;"may";220;1;339;4
;"failure";"no"
35;"management";"single";"tertiary";"no";1350;"yes";"no";"cellular";16;"apr";185;1;330;1;
"failure";"no"
30;"management";"married";"tertiary";"no";1476;"yes";"yes";"unknown";3;"jun";199;4;-
1;0;"unknown";"no"59;"blue-
collar";"married";"secondary";"no";0;"yes";"no";"unknown";5;"may";226;1;-
1;0;"unknown";"no"
```

Font: Apunts Anàlisi de dades (Bernadó, 2017)

---

[23] "*XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996.*" – (Traducció de l'autor)

[24] Llenguatge de marques generalitzat [ISO 8879]

3.10. Paradigmes de programació

Segons el diccionari[25], un paradigma és un exemple o model, segons l'article de Vujosevic-Janicic i Tosic (2008) la ciència computacional defineix paradigma com un conjunt coherent de mètodes que s'han trobat efectius per resoldre un problema. Contrastant les dues definicions es pot extreure que un paradigma en el desenvolupament de software és un conjunt de normes que serveixen com a model a l'hora de programar quelcom.

Vujosevic-Janicic i Tosic (2008) expliquen que durant els anys han sorgit molts paradigmes de programació diferents, i que, cadascun d'ells, implica una forma de pensar i uns llenguatges de programació diferents. Destaquen els quatre més importants i aquests són l'imperatiu, l'orientat a objectes, el funcional i el lògic.

3.10.1. Paradigma imperatiu

Com expliquen Vujosevic-Janicic i Tosic (2008) el paradigma imperatiu està basat en l'arquitectura d'ordinadors Von Neumann, introduïda el 1940. Aquesta arquitectura consisteix en un únic processador, separat de la memòria, i amb dades que es van passant del processador a la memòria.

> *"Els programes imperatius són seqüències d'ordres (o comandes) per realitzar una acció. Els programes imperatius es caracteritzen per seqüències d'enllaços (canvis d'estat) en què un nom pot estar vinculat a un valor en un punt del programa i, posteriorment, enllaçat a un valor diferent. Atès que l'ordre dels enllaços afecta el valor de les expressions, un problema important és la seqüenciació adequada de les vinculacions." - (Vujosevic-Janicic & Tosic, 2008, p. 69)*[26]

Així doncs, fent referència a la cita anterior, un paradigma imperatiu destaca per l'execució seqüencial de les instruccions. Els autors remarquen que els llenguatges de programació més populars per aquest paradigma són el Pascal i el C.

---

[25] Paradigma. A Diccionari.cat. Recuperat de http://www.diccionari.cat/lexicx.jsp?GECART=0164005

[26] "*Imperative programs are sequences of directions (or orders) for performing an action. Imperative programs are characterized by sequences of bindings (state changes) in which a name may be bound to a value at one point in the program and later bound to a different value. Since the order of the bindings affects the value of expressions, an important issue is the proper sequencing of bindings*". – (Traducció de l'autor)

3.10.2. Paradigma orientat a objectes

Seguint amb les explicacions de Vujosevic-Janicic i Tosic (2008), el paradigma orientat a objectes és una generalització del paradigma imperatiu. Segons els autors, aquest model es desenvolupa a partir de la simulació d'esdeveniments. Cada objecte encapsula les dades i operacions destinades a simular una entitat involucrada en el fenomen que es vol representar. Com en el món real, els objectes interactuen entre ells i, en la programació orientada a objectes, s'utilitzen els missatges per simular aquest fet.

Aquest paradigma destaca pel seu enfocament al reaprofitament de codi escrit. Els llenguatges de programació que suporten aquest paradigma s'anomenen *object-based* i els més comuns són l'Smalltalk, el C++ i el Java - (Vujosevic-Janicic & Tosic, 2008).

3.10.3. Paradigma funcional

> "El paradigma funcional es basa en la teoria de les funcions matemàtiques, més concretament en el càlcul lambda". - (Vujosevic-Janicic & Tosic, 2008, p. 73)[27]

Els autors de la cita segueixen explicant que aquest fet permet als programadors pensar en el problema a un nivell més alt d'abstracció, el què els permet dividir els problemes que han de resoldre en problemes més petits i solucionar-los més fàcilment.

Segons Vujosevic-Janicic i Tosic (2008), l'execució d'aquest paradigma està enfocada a dues accions bàsiques. La primera s'anomena *binding*, i consisteix en associar uns valors amb uns noms concrets; la segona són les funcions (*function*) i el seu objectiu és calcular els nou valors a través d'algoritmes encapsulats a les funcions. La principal característica d'aquest paradigma, segons els autors, és reduir o eliminar l'impacte dels efectes secundaris.

Els llenguatges més utilitzats en aquest paradigma són el *LISP*, *l'ML*, el *Scheme*, el *Miranda* i el *Haskell*.

[27] "*The functional programming paradigm is based on the theory of mathematical functions, more precisely on the lambda-calculus.*". – (Traducció de l'autor)

3.10.4. Paradigma lògic

> "El paradigma de programació lògica es basa en el càlcul de predicats de primer ordre. Aquest estil de programació fa èmfasi en la descripció declarativa d'un problema en lloc de la descomposició del problema en una implementació algorítmica." - (Vujosevic-Janicic & Tosic, 2008, p. 75)[28]

Segons els autors (Vujosevic-Janicic & Tosic, 2008), el paradigma lògic és una col·lecció de declaracions lògiques que descriuen com ha de resoldre's el problema abarcat. En la programació lògica és el programador qui s'encarrega d'especificar les relacions lògiques bàsiques, però no especifica la manera en què s'apliquen les normes d'inferència[29].

Els llenguatges més utilitzats que suporten aquest paradigma són el *Prolog*, el *Güdel* i el *Curry*, essent aquest últim un llenguatge multi paradigma ja que barreja elements del paradigma funcional i el paradigma lògic.

---

[28] "*The logic programming paradigm is based on first-order predicate calculus. This programming style emphasizes the declarative description of a problem rather than the decomposition of the problem into an algorithmic implementation.*.". – (Traducció de l'autor)

[29] Inferència: Raonament mitjançant el qual hom passa d'una o més proposicions, acceptades com a veres o falses, a una altra proposició, la veritat o falsedat de la qual hom suposa que depèn de la veritat o falsedat de la primera o primeres proposicions.

Inferència. A Diccionari.cat. Recuperat de http://www.diccionari.cat/lexicx.jsp?GECART=0077057

3.11. Metodologies de disseny de software i videojocs

Segons Ian Sommerville (2011), el procés de desenvolupament de software, és un conjunt d'activitats relacionades entre si que, juntes, condueixen a la producció d'un producte de software. Sommerville (2011) explica que existeixen molts processos diferents a l'hora de desenvolupar programari, però tots han d'incloure quatre activitats fonamentals: especificació del programari, disseny i implementació, validació i evolució.

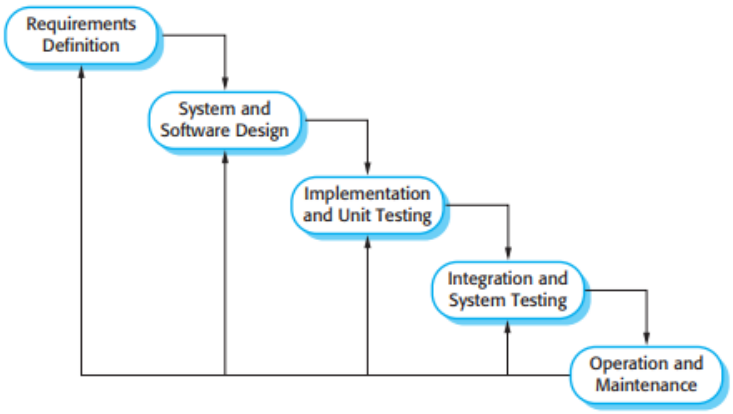
Com explica Sommerville (2011), no existeix el procés de desenvolupament ideal a l'hora de crear software. Però algunes vegades es classifiquen com a processos planificats i altres com a àgils. Resumit, els processos planificats són totes les activitats del procés que han estat planificades amb antelació, en canvi, en un procés àgil la planificació és incremental i és més fàcil de canviar en mig del procés si el client ho requerís.

3.11.1. Cascada (*Waterfall model*)

> *"Pren les activitats fonamentals del procés d'especificació, desenvolupament, validació i evolució i les representa com fases de procés separades, com ara especificació de requisits, disseny de programari, implementació, proves, etc."* – (Sommerville, 2011, p. 29) [30]

L'autor explica que aquest model deriva d'un model de desenvolupament de software més general. És un exemple de model de planificació ja que, en principi, ja que necessites una planificació i un horari previs a l'inici de la feina (Sommerville, 2011).

Figura 3-17 Metodologia en cascada



Font: (Sommerville, 2011, p. 30)

[30] "*This takes the fundamental process activities of specification, development, validation, and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing, and so on.*" – (Traducció de l'autor)

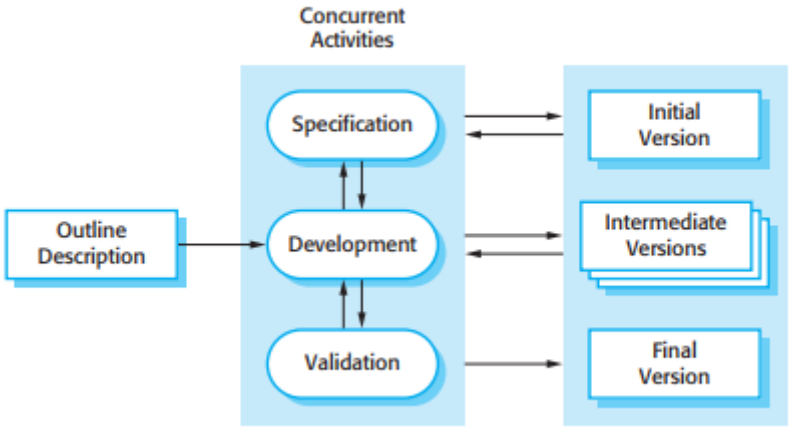
Com es pot veure a la Figura 3-14, aquest model consta de 5 parts. A la primera part es defineix què es vol aconseguir com a producte final, seguidament, a la segona etapa, es dissenya el programa. Un cop s'ha dissenyat el programa, és hora d'implementar-lo, aquí s'entra a la tercera activitat del procés, es creen petites unitats del programa i es comprova que funcionin correctament. Finalment, per acabar el producte, s'han d'unir (integrar) totes les unitats prèviament creades i comprovar el correcte funcionament. Un cop s'ha creat el producte queda la cinquena fase del projecte, el manteniment, que consisteix en arreglar els errors, millorar el programa, etc.

3.11.2. Incremental o àgil (*Incremental development*)

> *"Aquest enfocament intercala les activitats d'especificació, desenvolupament i validació. El sistema es desenvolupa com una sèrie de versions (increments), amb cada versió afegint funcionalitat a la versió anterior."* - (Sommerville, 2011, p. 30) [31]

Sommerville (2011) explica que aquest model és una part fonamental de les metodologies àgils i que és la més utilitzada pel desenvolupament d'aplicacions o sistemes. Està basat en la idea de desenvolupar quelcom inicial i exposar-li a l'usuari/client. A partir de la seva opinió o requisits es realitzen moltes versions i millores del producte, fins que s'obté un producte adequat (Sommerville, 2011).

Figura 3-18 Metodologia incremental (àgil)



Font: (Sommerville, 2011, p. 33)

A la Figura 3-15 es pot observar gràficament el procés incremental, la seva fase inicial consisteix en una descripció del producte desitjat, un cop s'ha establert es comença a

[31] "*This approach interleaves the activities of specification, development, and validation. The system is developed as a series of versions (increments), with each version adding functionality to the previous version.*" . (Traducció de l'autor)

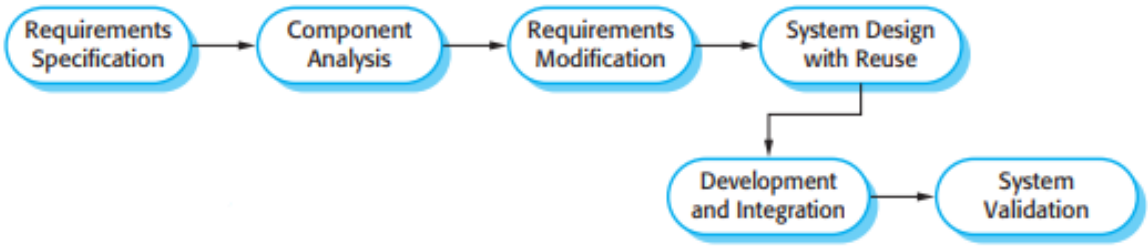
treballar en les fases d'especificació, desenvolupament i validació. Aquestes tres activitats són iteratives per cada vegada que es demana opinió al client o usuari. Cada versió del programa incorpora una funcionalitat nova que l'usuari desitja, les més importants són incorporades a les primeres versions, i a les últimes s'afegeixen les menys urgents pel desenvolupament.

3.11.3. Orientada a la reutilització (*Reuse-oriented*)

> "*Aquest enfocament es basa en l'existència d'una quantitat significativa de components reutilitzables. El procés de desenvolupament del sistema es centra en integrar aquests components en un sistema en comptes de desenvolupar-los des de zero.*" - (Sommerville, 2011, p. 30) [32]

Segons Sommerville (2011), la majoria de projectes de programari contenen part de reutilització. Sovint això succeeix de manera informal quan els treballadors coneixen el dissenys o parts de codi similars al que estan fent. Aleshores, busquen aquests dissenys o parts de codi i les modifiquen segons els sigui necessari.

Figura 3-19 Metodologia orientada a la reutilització



Font: (Sommerville, 2011, p. 35)

Com es pot observar a la Figura 3-16, el procés general d'aquesta metodologia és molt semblant a les explicades anteriorment, varien les fases intermèdies d'anàlisi de components on es fa una recerca de components que ja implementin l'especificació requerida. La següent fase és la de modificació de requeriments, aquesta fa un anàlisi més profund dels components que, anteriorment, s'han trobat i els modifica. Durant la pròxima activitat, si no s'ha trobat un marc reutilitzable, es dissenya el marc del programa. I, per acabar les parts diferents a les altres metodologies, a l'última fase s'uneixen tots els components creant un programa nou.

[32] "*This approach is based on the existence of a significant number of reusable components. The system development process focuses on integrating these components into a system rather than developing them from scratch.*." . (Traducció de l'autor)

4. OBJECTIUS

L'objectiu principal del joc és implementar el disseny de joc creat per Eduard Álvarez al llarg del seu TFG per tal d'obtenir un videojoc jugable. Des d'un principi es va tenir en compte la magnitud del projecte i el temps disposat que, juntament amb els horaris de i entregues de classes, no n'hi ha prou per desenvolupar un videojoc complet d'aquest estil. Per això, com s'ha comentat, s'ha optat dividir-lo en dos TFG: el de disseny i el de programació o implementació.

Per tal d'estructurar la feina s'ha decidit proposar-se diferents objectius més específics dins del principal.

4.1. Objectius tècnics

Entre els objectius específics, més concretament, enfocats a la part tècnica hi ha l'objectiu de desenvolupar el joc de forma reutilitzable, és a dir, que en un futur pugui ser modificat o ampliat si es donés el cas.

També s'ha decidit mantenir un rendiment òptim a l'executar el videojoc, ja que en alguns projectes anteriors ha sigut un problema que no s'ha tingut en compte.

4.2. Objectius acadèmics

Dins dels objectius específics també podem trobar uns objectius menys tècnics com poden ser els acadèmics, aquests objectius estan més enfocats al coneixement.

Un dels principals motius d'aquest TFG era la creació de la intel·ligència artificial i l'economia del joc, però no es tenien els coneixements suficients per la seva realització, així doncs aprendre més profundament sobre la IA i l'economia interna del joc era un objectiu en ment.

5. DISSENY METODOLÒGIC I CRONOGRAMA

Al llarg d'aquest apartat s'explicaran quines han estat les decisions preses per desenvolupar el projecte. Es farà una breu explicació de cada part decidida i el perquè s'ha escollit aquesta opció i no una altra. També en aquest apartat, s'exposarà la organització i planificació prevista al llarg del projecte.

5.1. Unity3D

Com s'ha explicat al marc teòric (3.7), un videojoc està construït sobre un motor de jocs, és molt car a nivell de temps, conceptes , etc. crear un motor de joc i, a més de falta de temps, l'equip no té els coneixements tècnics com per desenvolupar-ne un. Per aquesta raó s'ha optat per utilitzar una eina ja creada amb aquest objectiu. En el cas d'aquest projecte s'ha decidit utilitzar el motor de jocs *Unity3D*. No només per la seva facilitat, sinó per l'experiència que l'equip té utilitzant-lo.

A més, com s'ha comentat a l'apartat 3.7.2, Unity3D proporciona eines i facilitats a l'hora de desenvolupar un joc, com són les físiques del joc, la implementació de la interfície d'usuari, l'exportació a múltiples plataformes, etc.

5.1.1. Isometria

El disseny acordat del projecte exposa que el videojoc serà del gènere *tycoon* i especifica que tindrà una vista isomètrica. A més, com s'ha vist a la secció 3.1, aquest gènere normalment té una càmera d'estil isomètric.

Existeixen diferents productes a la botiga de *Unity3D* dedicats al desenvolupament de jocs isomètrics, però degut al seu preu i/o qualitat s'ha optat per realitzar un món isomètric des de zero a partir dels continguts explicats (3.2). Això permetrà a l'equip tenir un control més ampli del projecte i combinar les diferents parts d'una forma més personalitzada.

Com s'ha estudiat a l'apartat 3.3 el món isomètric es construirà a partir de *tile maps*, ja que permet dividir el nivell en una graella i tenir el control de cada imatge posada a aquest i de la informació individual de cada casella (fet interessant pel sistema de construcció modular i la intel·ligència artificial).

5.1.2. Sistema de construcció modular

El disseny acordat també contempla una mecànica de construcció a l'estil semblant als jocs CMS, la botiga de *Unity3D* també ofereix la possibilitat d'utilitzar petites productes creats pels usuaris en aquest àmbit però, com s'ha comentat al punt anterior, la qualitat o el preu ha fet desestimar inicialment aquesta opció.

Basant-nos en la informació exposada a l'apartat 3.4 i el fet de construir el món a partir d'un *tile map*, es desenvoluparà el sistema de construcció modular dividint les grans estructures en elements més petits. S'aprofitarà la divisió prèviament feta pel *tile map*, però guardant la informació en una taula de dades diferent.

5.1.3. Intel·ligència artificial

Quan s'han estudiat els gèneres (3.1) s'ha fet referència a la intel·ligència artificial dins d'aquests. S'ha explicat que en aquests gèneres acostuma a haver grups o "ramats" d'individus que han d'interactuar amb l'entorn i entre ells.

Al explicar els diferents mètodes d'IA (3.5), s'ha vist que existeixen vàries maneres de realitzar una intel·ligència artificial, sobretot a l'àmbit de presa de decisions i de recerca d'un camí (*Pathfinding*). Per aquest projecte s'ha decidit utilitzar l'estructura explicada a l'apartat 3.5.3 per desenvolupar el moviment de cada individu. Per la recerca del camí s'ha decantat per l'aplicació de l'algoritme A* en una graella, ja que el món està dividit en *tile maps* i, a més, és un mètode que prèviament s'ha estudiat a classe.

Com s'ha explicat prèviament, al joc hi haurà grups de personatges controlats per una IA, cadascun d'aquests individus necessita rebre informació de l'entorn i analitzar-la, per això, s'utilitzaran les *blackboards* o pissarres al llarg d'aquest projecte.

I, per tal de facilitar la feina i agilitzar-la, també s'introduirà l'eina d'*scripting* a l'hora de desenvolupar la IA.

5.1.4. Interfície d'usuari

Per la implementació de la interfície d'usuari, ja siguin menús o HUD, s'utilitzarà l'eina proporcionada pel mateix motor de jocs *Unity3D*.

5.2. Paradigmes i llenguatges de programació

Com s'ha comentat prèviament (3.7.3), *Unity3D* ofereix la possibilitat de programar en dos llenguatges de programació diferents. Al estudiar els llenguatges de programació (3.8), s'ha vist que *C#* és un llenguatge orientat a objectes, per això s'utilitzarà aquest paradigma a l'hora de programar. El motor de jocs també permet programar amb un llenguatge propi similar al *JavaScript*, però com l'equip està més familiaritzat amb el *C#* s'ha decidit no utilitzar-lo.

Com s'acaba de comentar, s'utilitzarà el paradigma de l'orientació a objectes, a més, l'estructura del codi està pensada per ser reutilitzable i, així, poder ampliar i/o modificar el projecte en un futur.

A l'apartat 3.5 s'ha exposat que es pot utilitzar l'*scripting* per facilitar el treball en equip i, alhora, treballar paral·lelament en diferents tasques. Aquesta eina consisteix en externalitzar

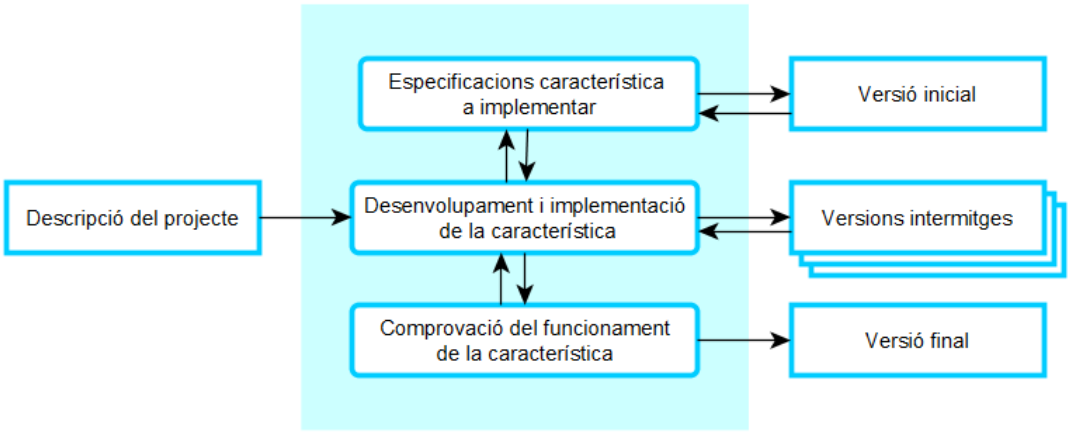
parts del procés de desenvolupament, al 3.8 s'explica que el llenguatge de programació *Lua* permet aquest fet, ja que s'utilitza com a "cola" entre llenguatges.

A més, s'ha comentat durant el disseny la possibilitat de que el propi dissenyador incorpori certes parts de la creació del *tile map*. Aquesta característica seria implementada amb el llenguatge de programació XML.

5.3. Metodologia escollida

Durant el desenvolupament del projecte s'utilitzarà metodologia àgil, ja que el disseny es realitzarà a la vegada que la implementació. Així doncs, referint-nos a la nomenclatura utilitzada al 3.10, utilitzarem una metodologia incremental.

Figura 5-1 Metodologia incremental del projecte



Font: Elaboració pròpia a partir de la Figura 3-15

La primera versió del projecte es crearà a partir de la breu definició del joc (*abstract*) i s'implementaran les mecàniques bàsiques que s'hagin acordat com són la construcció d'estructures, la interacció de l'usuari amb els elements del joc. També es desenvoluparà, en aquesta primera fase, part de la intel·ligència artificial. Un cop fetes les comprovacions corresponents es començarà a comparar amb el document de disseny ja avançat i, aleshores, s'afegiran les característiques i mecàniques restants. En alguna de les versions del joc, s'implementarà l'art que es realitzarà de forma externalitzada.

Com s'ha vist prèviament el procés consisteix en anar "fent créixer" el projecte poc a poc, a més de permetre una continua comprovació de la feina, aquesta metodologia permetrà assegurar-se entregues jugables del joc en cada una de les versions.

5.4. Planificació

En aquesta secció, com s'ha comentat prèviament, està plantejada la organització i planificació que es durà a terme al llarg del projecte. Primer s'exposaran les taules de tasques, ja dividides per entregues per fer més còmoda la interpretació. Seguidament s'explicarà a través d'un cronograma com s'organitzarà al llarg del temps.

Aquesta planificació s'ha realitzat tenint en compte les dates d'entrega del projecte, els exàmens i els dies de festa previstos.

5.4.1. Taula de tasques

En aquest apartat podem trobar les tasques del primer, segon i tercer prototip funcional del joc. Ens referim com a prototip funcional a aquell programa que, encara i no tenir totes les característiques finals implementades, funciona correctament.

El primer i segon prototip (Taula 5-1) implementaran els conceptes més bàsics del videojoc que es realitzarà. Aquests són la construcció del món isomètric, la base del sistema modular, la base de la intel·ligència artificial i una breu implementació de la IU.

Taula 5-1 Taula de tasques (1)

Primer prototip funcional	**Segon prototip funcional**
Construcció del món isomètric	**AI II**
Implementació matemàtiques isomètric	*Pathfinding*
Creació graella lògica	*Blackboard*
Creació graella visual	Recull de dades
Mapa provisional	**Sistema de construcció modular II**
Sistema de construcció modular I	Modificar elements del mapa
Incorporació de nous elements al mapa	Inventari
Selecció d'objectes al mapa	**Interfície d'usuari**
AI I	Inventari
Comportaments de moviment	HUD
25 de Gener del 2018	**10 de Febrer del 2018**

Font: Elaboració pròpia

El tercer prototip (Taula 5-2), tot i incorporar elements bàsics, ja conté elements més avançats del joc. En aquesta versió s'implementaran la presa de decisions de la intel·ligència artificial juntament amb l'anàlisi de l'entorn i l'economia del joc. A més es farà la primera incorporació de menús i esdeveniments causats per elements del món.

Taula 5-2 Taula de tasques (2)

Tercer prototip funcional
AI III
Anàlisi de l'entorn
Economia del joc
Presa de decisions (*Scripting*)
Sistema de construcció modular III
Esdeveniments dels elements
Interfície d'usuari II
Menús
Lògica del joc
Incorporació de noves característiques
27 de Febrer del 2018

Font: Elaboració pròpia

A partir del quart prototip ja s'incorporaran els elements que dicti el document de disseny i, per tan, no es veuen reflectits en aquest apartat de moment.

5.4.2. Cronograma

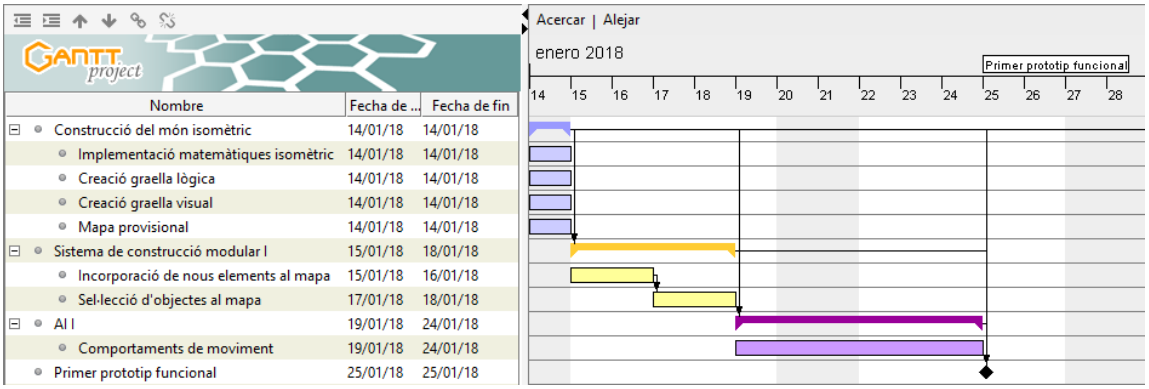
Seguidament exposem un cronograma[33] per cada un de les dates límits imposades per nosaltres mateixos.

Taula 5-3 Lliuraments imposats

Lliurament memòria intermèdia	19 de març (data límit)
Lliurament documentació final	21 de maig (data límit)

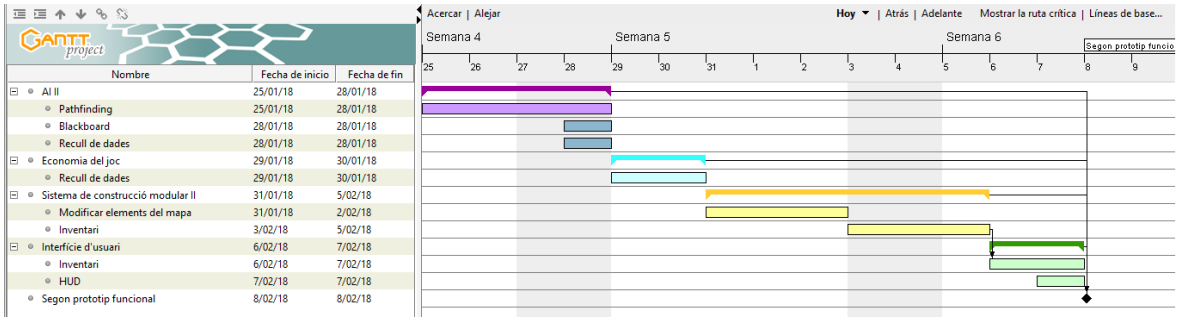
Font: Informació proporcionada per la institució

Figura 5-2 Gantt del primer prototip funcional



Font: Elaboració pròpia

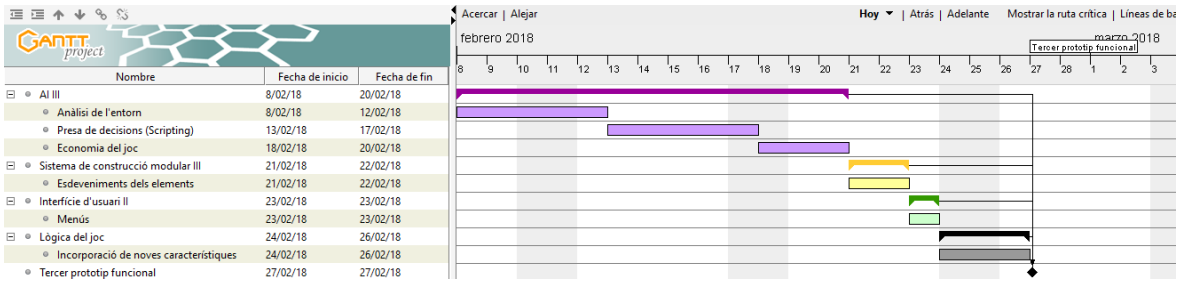
Figura 5-3 Gantt del segon prototip funcional



Font: Elaboració pròpia

[33] Cronograma fet a partir de les taules de tasques, ampliació als annexos.

Figura 5-4 Gantt del tercer prototip funcional



Font: Elaboració pròpia

6. RESULTATS DEL TREBALL

En aquest capítol s'exposen els diferents procediments que s'han dut a terme al llarg del procés; així com l'organització del codi i l'explicació d'aquest. Durant el desenvolupament no hi ha hagut gaires canvis sobre la metodologia escollida, però a causa de diversos problemes i l'evolució del projecte s'han variat certs aspectes explicats a continuació.

Pel que fa el motor de jocs s'ha seguit utilitzant l'escollit (*Unity3D*), però el sistema isomètric ha variat a ser un sistema axonomètric ja que s'ha adaptat a la nova vista proposada per l'equip d'art. A l'utilitzar *Unity3D* s'ha optat per seguir utilitzant C# com a llenguatge principal de programació i s'ha intentat seguir un paradigma de programació orientat a objectes i reutilitzable.

A més, s'ha conservat la idea de externalitzar una part de la programació. Mantenint per intel·ligència artificial el llenguatge de programació LUA, però modificant l'eina per construir un mapa *tile map* ja que principalment era amb XML i ha passat a ser amb CSV, ja que l'equip de disseny es sent més còmode treballant amb fulls de càlcul.

6.1. Organització del codi

Per desenvolupar el joc s'ha realitzat una organització basada en la programació orientada a objectes (explicada breument a l'apartat 3.9.2). El joc consta de varis *scripts*. Aquests es poden dividir en tres grans grups: els controladors (6.1.1) , els sistemes (6.1.2) i les eines d'ajuda (6.1.3). Així com cada controlador és un sol *script* des del qual es donen les directrius desitjades, els sistemes i les eines d'ajuda poden estar composts d'un o varis *scripts*.

6.1.1. Controladors

Dins d'aquest subapartat s'explicarà en què consisteix cada controlador. Com el seu nom indica, els controladors són els encarregats de gestionar i controlar els sistemes del programa. Tots els controladors segueixen el patró *Singleton* [34] per tenir una referència global dels objectes que s'han utilitzat durant el joc.

6.1.1.1. Controlador de joc

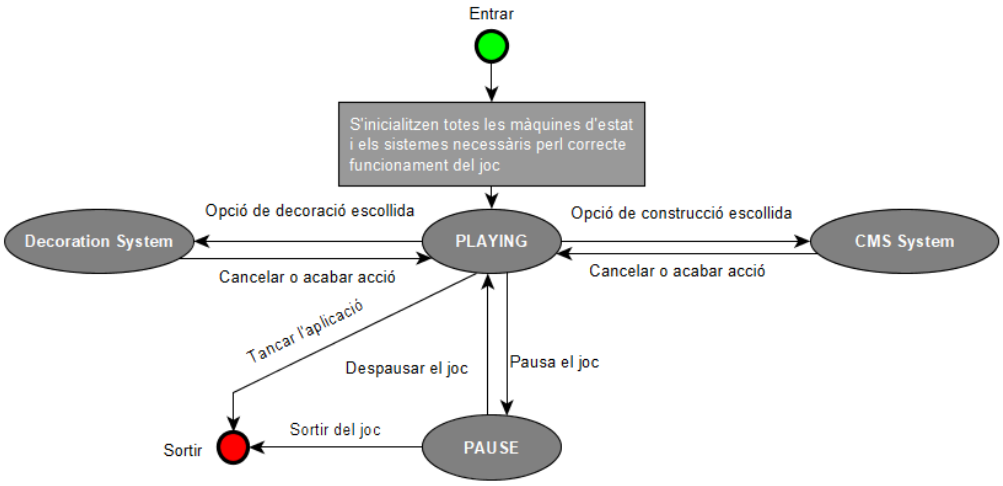
Dins del codi rep el nom de *GameManager*. És el responsable de gestionar tots els sistemes i controladors del joc. La seva estructura bàsica consisteix en un conjunt de funcions públiques encarregades de comunicar els diferents sistemes i controladors, i en una màquina

[34] Segons la documentació tècnica de Microsoft un patró *Singleton* garanteix que una classe tingui només una sola instància i proporciona un punt d'accés global a ella. – Font: https://msdn.microsoft.com/es-es/library/bb972272.aspx#EDAA

d'estats (Figura 6-1 Màquina d'estats del *GameManager*) que conté dues màquines d'estats més: una dedicada del sistema de construcció (*CMS System*) i una del sistema de decoració (*Decoration System*).

La màquina d'estats és l'eina bàsica per dirigir el joc, ja que a través d'aquesta s'activen uns sistemes o uns altres. Aquesta màquina conté 4 estats bàsics: PAUSE, PLAYING, CMS i DECORATING i es va intercanviant l'estat actual segons les accions del jugador.

Figura 6-1 Màquina d'estats del *GameManager*



Font: Elaboració pròpia

6.1.1.2. Controlador d'escenes

L'*ScenesManager* és un objecte permanent al llarg de tot el joc, és a dir que no es destrueix encara que canviem d'escena. La seva funció dins del joc consisteix en gestionar el canvi d'escenes i les modificacions que han de realitzar-se durant el procés.

6.1.1.3. Controlador d'entrades de teclat

Com el seu nom indica el controlador *PlayerInputs* té la funció de gestionar les entrades de teclat i ratolí del jugador. El motor de *Unity3D* ja té una classe encarregada d'aquestes funcions, així doncs, el controlador d'entrades personalitzat que s'ha desenvolupat per aquest projecte és una ampliació de les funcions bàsiques del que ja ve de base.

Totes les funcions que té el controlador són estàtiques, el que vol dir que poden ser invocades globalment des de qualsevol altre part del programa.

6.1.1.4. Controlador del món

Al joc existeixen dos tipus de mapes: el modificable, sobre el qual els usuaris jugaran i contenen la lògica de construcció i decoració; i el permanent, que és l'encarregat de

proporcionar un *setting* narratiu al joc i millorar visualment el joc. El *WorldManager* és l'encarregat de gestionar aquests mapes.

El mapa permanent és una o un conjunt d'imatges fixes que el jugador no pot variar, però si és necessari per codi és possible de modificar. Sobre d'aquest mapa només existirà una graella lògica pel correcte funcionament de la intel·ligència artificial. Per muntar el nivell a l'inici del joc es farà directament dins l'escena del motor de jocs amb la imatge proporcionada pels artistes.

El mapa modificable del joc està basat sobre una graella. La graella de l'inici del joc es crea a partir de fitxers de dades (*.csv[35]), dels qual s'extrauran la imatge de cada cel·la i les mides de la graella.

6.1.1.5. Controladors de fitxers

Des del controladors de fitxers es duen a terme totes les operacions que tenen a veure amb la gestió d'arxius. Actualment existeixen dos controladors dedicats a interpretar fitxers: ReadMapFiles i ReadLUAFiles.

El primer es dedica a llegir els arxius **.csv* i seleccionar les imatges corresponents. Una de les seves funcions principals és identificar una imatge concreta dins d'una imatge composta per varies (d'ara en endavant *spritesheet*).

El segon controlador es dedica a llegir els arxius LUA (desats amb l'extensió **.txt*) i, utilitzant *MoonSharp*[36], interpretar el codi de cada arxiu.

6.1.1.6. Controlador de càmera

Com el seu nom indica el controlador de càmera és l'encarregat de dirigir el comportament de la càmera. Aquests comportaments són controlats directament des del controlador de joc.

6.1.1.7. Controlador de la interfície d'usuaris

Dins de Unity3D la interfície d'usuaris és molt senzilla d'implementar. En aquest projecte hi ha dues classes encarregades de la interfície d'usuaris: *HUD_Script* i *HUD_SubMenu*.

HUD_Script conté les funcions necessàries per interactuar amb els botons de l'escena, i juntament amb el Controlador de botiga, de crear les botigues a temps real.

---

[35] Els fitxers CSV són un tipus de document en format obert utilitzat per representar dades en forma de taula, en què les columnes se separen per comes o punt i coma, i les files per salts de línia.

[36] MoonSharp és un intèrpret Lua escrit completament en C # per a la màxima compatibilitat en. NET, Mono, Xamarin i Unity. - Font: http://www.moonsharp.org/about.html

HUD_SubMenu en canvi, no està relacionat directament amb els botons ja instanciats en escena, sinó que s'utilitza per fer interactuables els botons instanciats durant l'execució.

6.1.1.8. Controlador de botiga

És l'encarregat de la creació de les botigues a temps real, les seves funcions són invocades pel controlador d'interfície d'usuari.

Com s'acaba d'explicar, el principal objectiu és crear una finestra de botiga amb els seus elements necessaris al lloc correcte, dins d'aquest controlador es gestionen les posicions i mides d'aquestes finestres. Per saber el contingut de cada botiga el programa busca dins la carpeta d'objectes quins són els desitjats.

La carpeta d'objectes és una carpeta del projecte de *Unity3D*, on els membres de l'equip poden crear objectes i decidir de quin tipus són, la mida d'aquests, i altres propietats bàsiques.

6.1.1.9. Controlador de la intel·ligència artificial

El controlador de la intel·ligència artificial s'encarrega de gestionar el sistema d'IA del joc. Des d'ell s'invoquen les funcions necessàries per crear la graella de *pathfinding* i es controla la creació de personatges en escena.

6.1.2. Sistemes

Segons el criteri que s'ha seguit al llarg del procés, es poden definir sistemes com un conjunt d'elements de codi que, interactuant junts, són capaços de dur a terme una funcionalitat concreta.

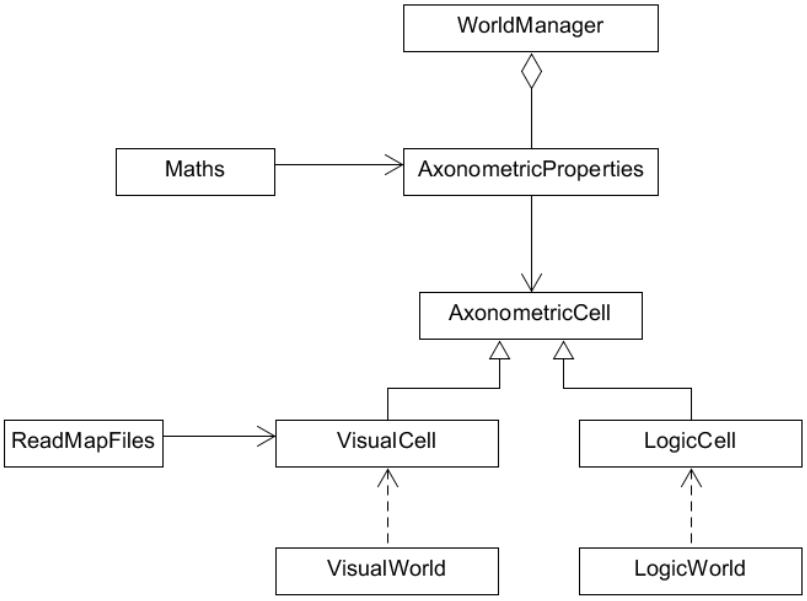
Els sistemes programats dins d'aquest projecte funcionen conjuntament amb un o més d'un controlador que dirigeix o gestiona les accions a fer.

La interacció total entre sistemes i controladors es pot observar al diagrama de classes complet (6.1.4), però dins de cada apartat s'explicarà el funcionament del sistema corresponent.

6.1.2.1. Axonomètric

Tenint en compte les explicacions de Rüppel i Schatz (2011) a l'apartat 3.4, s'ha dissenyat el sistema axonomètric com el conjunt de classes que defineixen les propietats i bases de l'axonometria al projecte. Dins del sistema, podem trobar dues classes encarregades dels càlculs matemàtics (3.3.2) per aplicar les propietats de transformació dels objectes (posició, rotació i escala), aquestes són *AxonometricProperties* i *Maths*; i també podem trobar cinc classes més destinades a definir els espais dels objectes de l'escena.

Figura 6-2 UML simple del sistema axonomètric



Font: Elaboració pròpia

AxonometricCell és la base de la qual *VisualCell* i *LogicCell* són subordinades. I és la principal encarregada de definir les propietats dels objectes axonomètrics. Ambdues classes subordinades amplien la informació que es pot desar en una cel·la axonomètrica, cada una

d'elles especialitzada en un aspecte concret, ja que no totes les cel·les necessiten totes les propietats.

Finalment hi ha les dues classes de món: *VisualWorld* i *LogicWorld*. La seva funció és construir una graella de les cel·les corresponents. Els seus mètodes són invocats globalment (estàtics) i retornen la graella desitjada.

Taula 6-1 Classes del sistema axonomètric

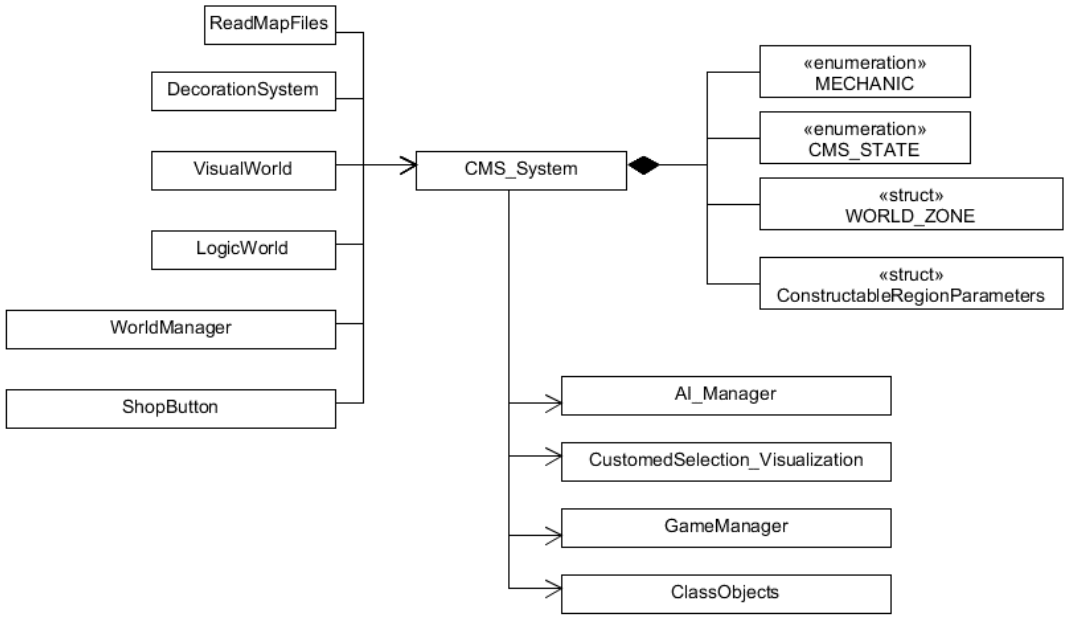
Scripts implicats	
Nom	**Breu descripció**
AxonometricProperties	Conté les propietats necessàries per definir el sistema de coordenades que s'utilitzarà al sistema axonomètric.
AxonometricCell	Consisteix en la classe primitiva de les cel·les que s'utilitzaran al sistema axonomètric, així com les funcions bàsiques necessàries.
Maths	Conté les funcions matemàtiques necessàries per realitzar les transformacions lineals desitjades.
LogicCell	Classe subordinada de *AxonometricCell*, conté un objecte de col·lisió i una booleana que indica si es pot construir a la seva posició.
LogicWorld	Crea una graella de *LogicCell*, aquesta classe serveix per gestionar aquesta graella.
VisualCell	Classe subordinada de *AxonometricCell*, conté un atribut de tipus *Sprite* i dos més de tipus *string* per desar el nom de la imatge i la *spritesheet*.
VisualWorld	Crea una graella de *VisualCell*, aquesta classe serveix per gestionar aquesta graella.

Font: Elaboració pròpia

6.1.2.2. Sistema de construcció

Basant-se amb les mecàniques observades als referents, sobretot al *Theme Hospital* (Bullfrog Productions, 1997) a l'apartat 2.3. Aquest sistema principalment consta de dos *scripts* i, la interacció d'ambdós, formen un sistema de construcció que permet realitza diferents mecàniques al jugador.

Figura 6-3 UML simple del sistema de construcció



Font: Elaboració pròpia

Dins de la classe *CMSSystem* existeix una estructura que defineix àrees del món (WORLD_ZONE). Aquesta estructura desa les propietats necessàries per interactuar amb una àrea creada. Totes aquestes zones són guardades a la llista corresponent depenent de què defineixin.

Taula 6-2 Classes del sistema de construcció (CMS)

Scripts implicats	
Nom	**Breu descripció**
CMSSystem	Conté la màquina d'estats que controla el sistema de construcció. Aquesta màquina d'estats està funcionant com a estat del *GameManager*.
CustomedSelection_Visualization	Conté les funcions necessàries pel control de partícules i *LineRender* dedicats a la visualització de la selecció.

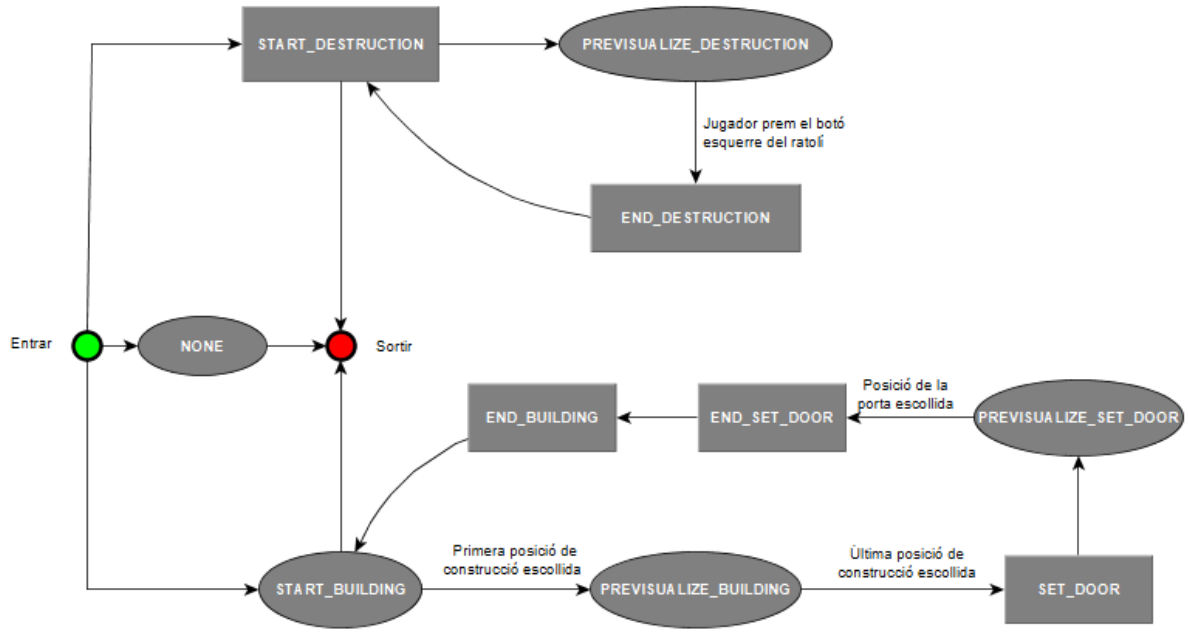
Font: Elaboració pròpia

El funcionament bàsic del sistema de construcció consisteix en la creació de les *WORLD_ZONES*, aquestes zones tenen diferents atributs que desen la informació necessària per la creació d'habitacions.

Quan es crea una habitació es desa la *WORLD_ZONE* en una llista que serà necessària per destruir-la o desar-la.

El funcionament bàsic del sistema consisteix en una màquina d'estats que, dirigida des del *GameManager*, gestiona les accions que es poden dur a terme en una situació del joc o en una altra. Bàsicament consta de tres estats (*Building*, *Destroying* i *SetDoor*) subdividits en tres sub-estats més: *Start*, *Previsualize* i *End*. També hi ha l'estat anomenat *None* i és l'encarregat de deixar la màquina en repòs.

Figura 6-4 Màquina d'estats del sistema de construcció

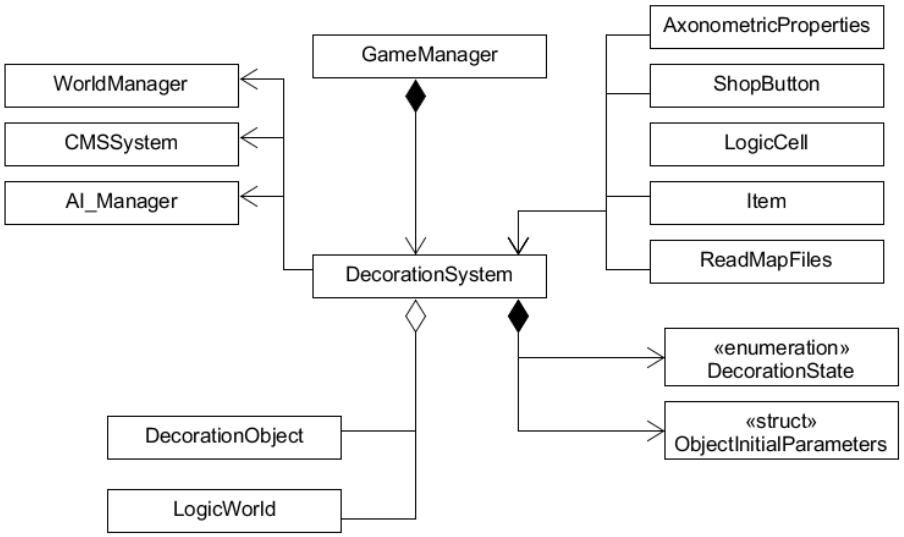


Font: Elaboració pròpia

6.1.2.3. Sistema de decoració

Aquest sistema funciona de forma semblant al sistema de construcció, però necessita utilitzar una classe d'objecte diferent, ja que aquest han de guardar més informació que una *VisualCell* i una *LogicCell*.

Figura 6-5 UML simple sistema de decoració



Font: Elaboració pròpia

La classe *DecorationSystem* consta d'una estructura amb la funció de desar els paràmetres necessaris per instanciar un objecte, és a dir aquells que defineixen l'objecte en si i els que especifiquen la posició d'aquest.

Taula 6-3 Classes del sistema de decoració

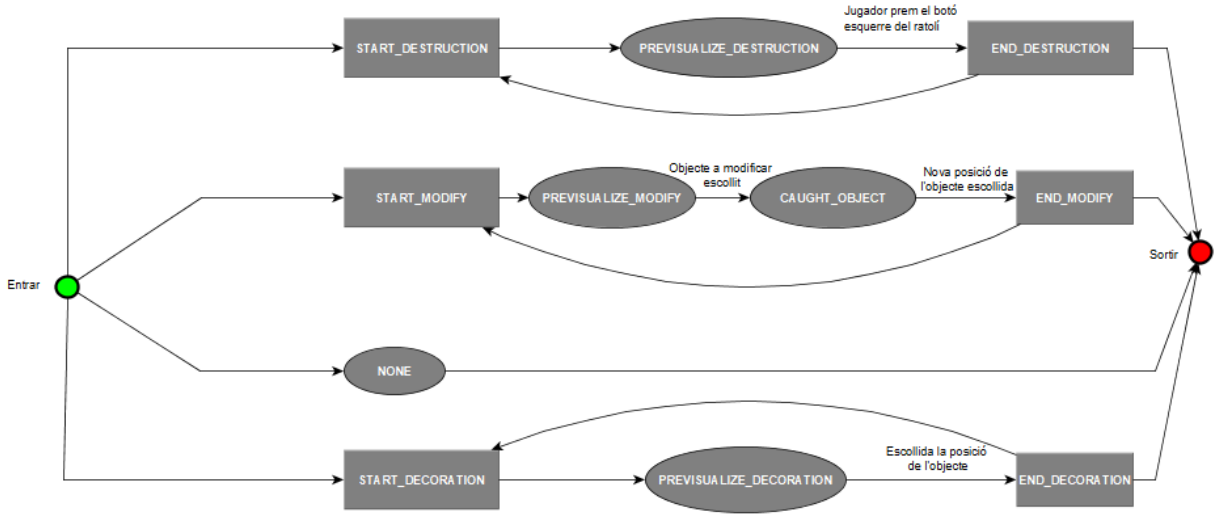
Scripts implicats	
Nom	**Breu descripció**
DecorationSystem	Conté la màquina d'estats que controla el sistema de construcció. Aquesta màquina d'estats està funcionant com a estat del *GameManager*.
DecorationObject	Conté la informació necessària d'un objecte utilitzat en el sistema de decoració.

Font: Elaboració propia

Com s'ha mencionat anteriorment, el funcionament bàsic de la decoració és bastant semblant al de construcció. Principalment consta d'una màquina d'estats amb quatre estats base, i tres d'aquests formats per varis sub-estats més.

L'acció de col·locar i destruir objectes de decoració és, a grans trets, igual a construir. L'estat de *None* té la mateixa funció de deixar en repòs la màquina d'estats. En aquest sistema s'ha afegit l'acció de modificar la posició d'un objecte ja col·locat, on a part de seleccionar l'objecte durant l'estat *Previsualize_Modify*, el jugador ha d'escollir on el recol·loca durant l'estat de *Caught_object*.

Figura 6-6 Màquina d'estats del sistema de decoració



Font: Elaboració pròpia

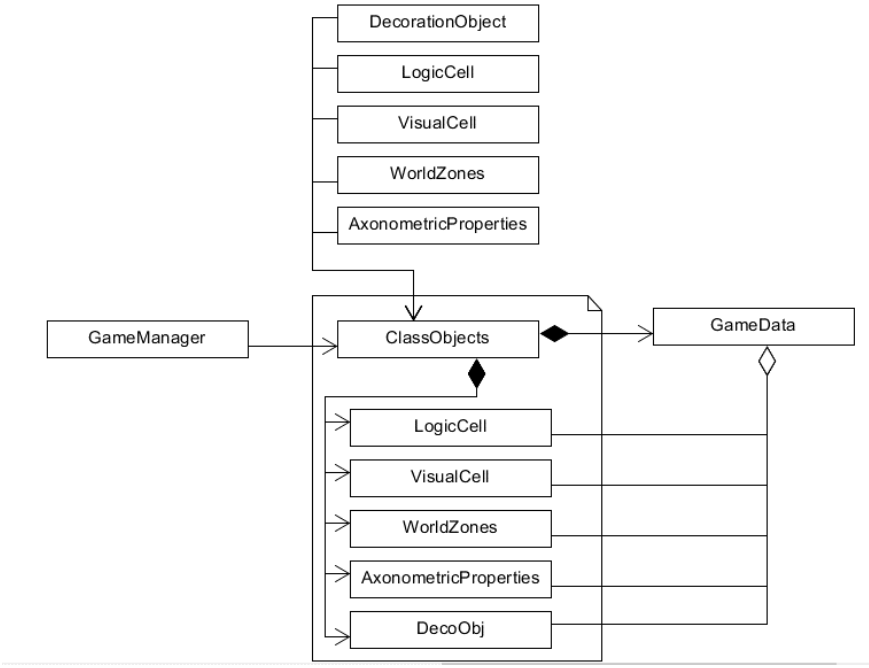
6.1.2.4. Sistema de guardat

Unity3D té un sistema de guardat, però com que cada joc és únic necessita desar diferents dades i de distinta forma. Així doncs, és necessària la definició d'aquestes dades i estructura.

Es creen classes serialitzables (classes que l'ordinador pot desar) de les dades que es desitgen guardar. Les classes estan compostes de paràmetres senzills com són els enters, cadena de caràcters, els números decimals i les booleanes.

Per aquest joc s'han creat diferents classes dins l'*Script* anomenat *ClassObjects*, però encara fa falta afegir algunes classes més que anteriorment no s'havien plantejat. Les classes creades tenen el mateix nom que les classes o estructures que guarden, sent aquests: *AxonometricParameters*, *VisualCell*, *LogicCell*, *WORLD_ZONES*, *DecoObj*.

Figura 6-7 UML simple sistema de guardat



Font: Elaboració pròpia

Per guardar o carregar una partida s'ha creat un format d'arxiu, aquest format està definit a l'*Script* anomenat *GameData*. En aquest programa es desen les classes desitjades, en el cas concret del projecte existeix una llista d'elements de cada classe abans mencionada excepte per la d'*AxonometricParameters* que només desa una instància.

Taula 6-4 Classes del sistema de botiga

Scripts implicats	
Nom	**Breu descripció**
ClassObjects	Conté totes les funcions necessàries per dur a terme el guardat de la partida. Existeixen quatre tipus de classe que es desaran, aquesta classe conté les classes creades per desar les desitjades.
GameData	Conté l'estructura de la informació que es desarà.

Font: Elaboració pròpia

6.1.2.5. Sistema de botiga i UI

La botiga es la part del HUD on el jugador escull quin objecte o estructura vol col·locar a l'escenari. Aquests objectes es disposen a la botiga corresponent de forma automàtica utilitzant els objectes creats per l'equip de desenvolupament amb una eina d'ajuda (6.1.3.1).

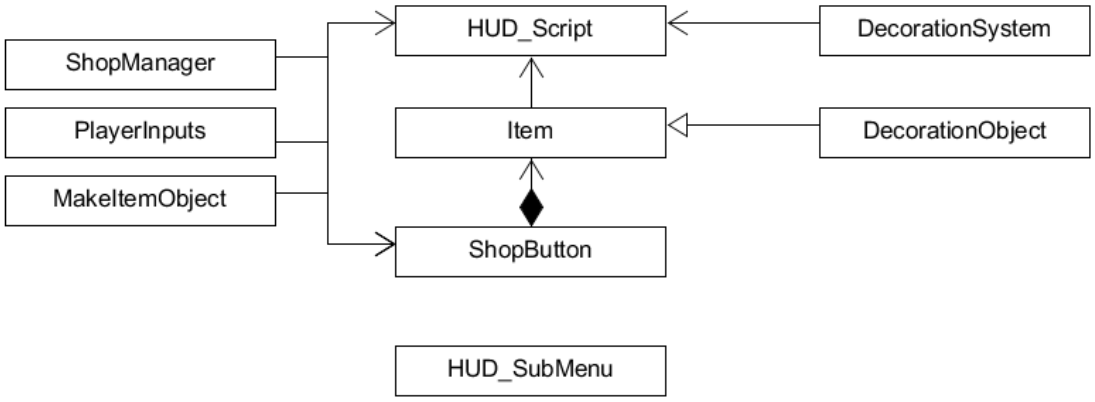
Una part de les funcions creades són assignades a un botó situat ja en escena, però la majoria són adjudicades a través de codi ja que els botons són creats en temps de joc.

Taula 6-5 Classes del sistema de botiga i UI

Scripts implicats	
Nom	**Breu descripció**
Item	És la classe base dels objectes de decoració. Desa els paràmetres bàsics i necessaris per distingir un objecte d'un altre.
ShopButton	Script que crida el *ShopManager* quan vol crear un botó nou. Conté el codi necessari per fer funcionar un botó de forma correcta.
HUD_Script	*HUD_Script* és el component d'un element de l'escena, en aquest programa s'hi troben les funcions publiques que s'associen a cada botó a través de l'escena.
HUD_SubMenu	*Script* associat a varis elements de l'escena, al mateix inspector se l'hi assignen els botons desitjats. La funció d'aquest script és la de canviar la imatge del botó segons el seu estat.

Font: Elaboració pròpia

Figura 6-8 UML simple sistema de botiga i HUD



Font: Elaboració pròpia

6.1.2.6. Sistema d'intel·ligència artificial

La funció principal d'aquest sistema és el control dels agents de la intel·ligència artificial. Per tenir un correcte funcionament té el suport de 3 sistemes més: *Pathfinding* (6.1.2.7), *BehaviourTree* (6.1.2.8) i *SteeringBehaviour* (6.1.2.9). El primer s'encarrega de trobar un camí per on pot desplaçar-se l'agent, el segon defineix el comportament de decisions d'aquest i el tercer del seu comportament cinemàtic.

Taula 6-6 Classes d'intel·ligència artificial

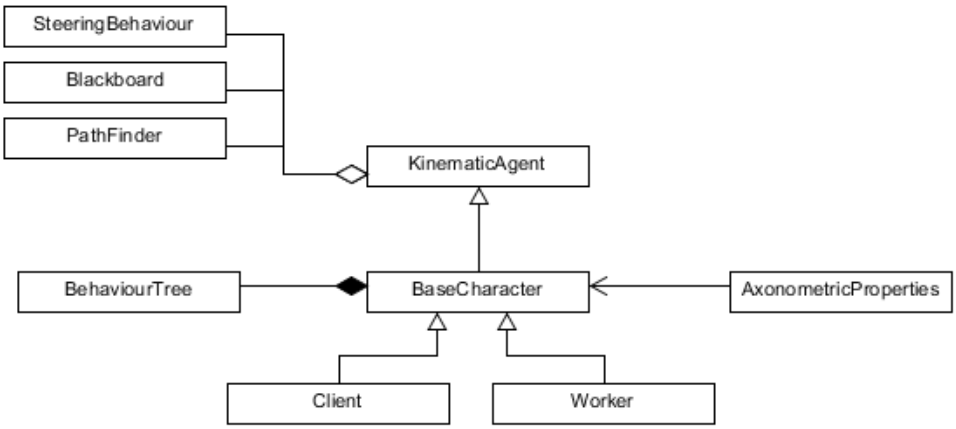
Scripts implicats	
Nom	**Breu descripció**
KinematicAgent	Classe bàsica d'un objecte mòbil dins del programa.
BaseCharacter	Classe bàsica d'un agent mòbil dins del programa, hereta directament de *KinematicAgent*.
Client	Classe filla de *BaseCharacter*, conté la programació especifica pel comportament d'un client.
Worker	Classe filla de *BaseCharacter*, conté la programació especifica pel comportament d'un treballador.
Blackboard	Conté els valors dels paràmetres individuals per cada agent. Cada agent té la seva pròpia *Blackboard*.

Font: Elaboració pròpia

La funció principal de *KinematicAgent* consisteix en actualitzar l'estat del sistema *SteeringBehaviour* i amb ell el moviment de l'agent. També, és el pare de la majoria dels *scripts* del sistema i té la funció de desar les variables que tenen tots en comú.

Tots els agents intel·ligents del projecte provenen del *BaseCharacter*, qui s'encarrega d'actualitzar les animacions d'aquests i a més conté un arbre de comportament que decideix com ha d'actuar.

Figura 6-9 UML simple sistema d'intel·ligència artificial



Font: Elaboració pròpia

6.1.2.7. Sistema de *pathfinding*

Perquè els elements controlats per la intel·ligència artificial tinguin informació del seu entorn i saber per on poden desplaçar-se, s'ha afegit un *pathfinding* A* adaptat al món axonomètric.

Taula 6-7 Classes del sistema de pathfinding

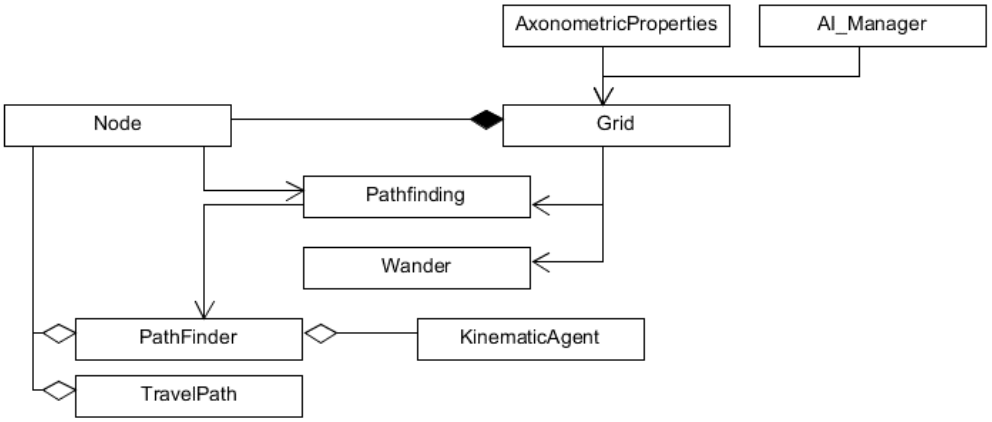
Scripts implicats	
Nom	**Breu descripció**
Grid	Classe encarregada de definir la zona transitable pels agents.
Node	Unitat mínima dins de la zona transitable, conté la informació bàsica pel càlcul de cada cel·la.
Pathfinding	*Script* encarregat de fer els càlculs de cerca de camí.
PathFinder	Component d'un agent encarregat de donar l'ordre de buscar un camí transitable fins la posició desitjada.

Font: Elaboració pròpia

Com s'ha comentat, el sistema de *pathfinding* és un suport al sistema d'intel·ligència artificial (6.1.2.6). El seu funcionament principal consisteix en dividir el món en una graella lògica formada per nodes. Aquests nodes contenen un valor heurístic que indica al programa encarregat de buscar el camí quin cost probablement tindrà recórrer el node.

PathFinder és un component que s'associa a l'agent que es desitgi moure, aquest component és l'encarregat de comunicar l'agent amb el *pathfinding*.

Figura 6-10 UML simple del sistema de *pathfinding*



Font: Elaboració pròpia

6.1.2.8. Sistema d'arbres de comportament

De tots els mètodes per prendre decisions que s'han vist (3.5.5) s'ha escollit utilitzar els arbres de comportament, degut a que els comportaments estudiats als referents Theme Park (Bullfrog Productions, 1994) i Theme Hospital (Bullfrog Productions, 1997) són més fàcils de controlar amb aquest mètode.

Aquest sistema s'encarrega de controlar el flux d'accions d'un arbre de comportament, per construir un arbre l'equip de desenvolupament utilitzarà l'eina d'ajuda especifica (6.1.3.2).

Taula 6-8 Classes del sistema d'arbre de comportament

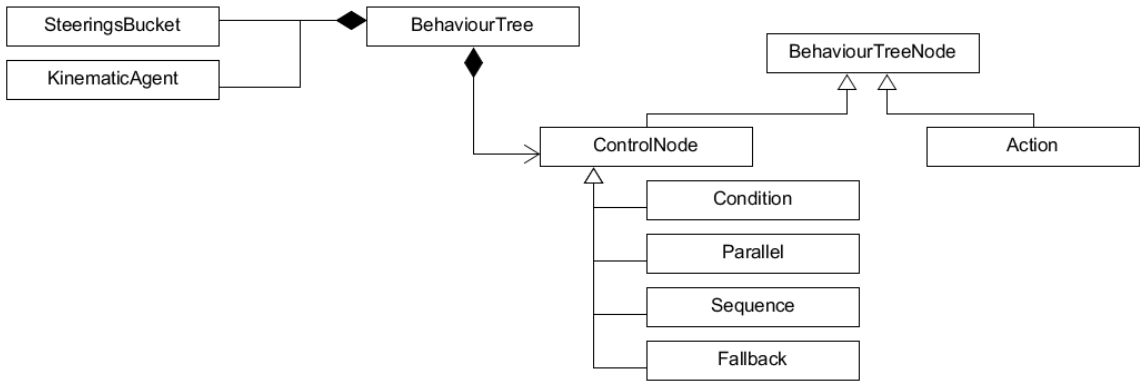
Scripts implicats	
Nom	**Breu descripció**
BehaviourTree	Component associat a un agent, conté el primer node de l'arbre de comportament i una referencia a un *SteeringsBucket* (6.1.2.9).
BehaviourTreeNode	Classe pare dels diferents tipus de node de l'arbre de comportament.
ControlNode	Node pare de les classes de node que poden tenir nodes associats com a fills.
Action	Node que té un *script* de LUA associat a ell, defineix una acció i no una decisió.

Condition	Node que té un *script* de LUA associat a ell, a través d'uns paràmetres retorna una decisió booleana de cert o fals. Aquesta classe és subordinada de la classe *ControlNode*.
Parallel	Node fill de *ControlNode* permet executar tots els fills que té a la vegada. Quan una quantitat de fills té èxit es dona per completada amb èxit.
Sequence	Node fill de *ControlNode* permet executar tots els fills que té seguint un ordre. Quan tots els fills han tingut èxit es dona per completada amb èxit.
Fallback	Node fill de *ControlNode* permet executar tots els fills que té seguint un ordre. Només que un fill tingui èxit es dona per completada amb èxit.

Font: Elaboració pròpia

D'aquest sistema destacar que les *Action* i les *Condition* llegeixen el seu codi de programes escrits amb LUA. Això es fa per facilitar la modificació d'aquests comportaments un cop s'ha realitzat l'executable del joc, ja que no cal tornar-la a fer per aplicar les variacions.

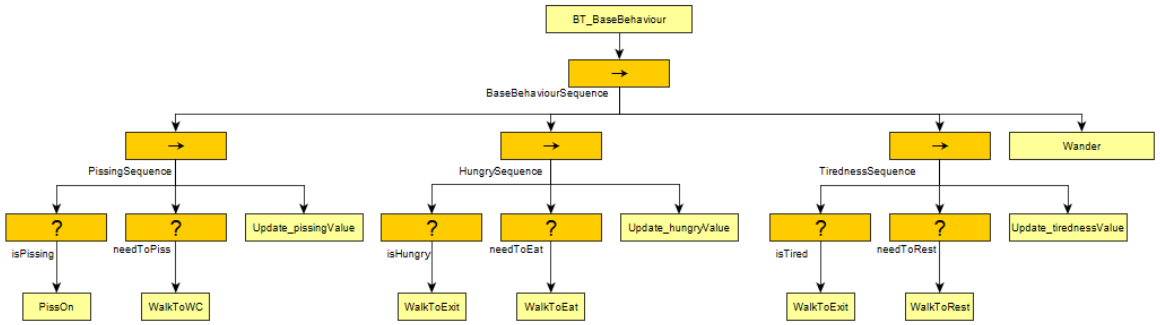
Figura 6-11 UML simple del sistema d'arbre de comportament



Font: Elaboració pròpia

La implementació que s'ha dut a terme, ha estat del comportament base d'un personatge dins del joc, en la qual només es té en compte les seves necessitats bàsiques com són tenir gana, estar cansat i la necessitat d'anar al lavabo. Com es pot veure a l'esquema (Figura 6-12) es dóna prioritat a tenir ganes d'anar al lavabo per sobre de tot, seguides per la gana i el cansament, si té totes les necessitats cobertes el personatge deambula per l'escenari.

Figura 6-12 Arbre de comportament d'un personatge basic

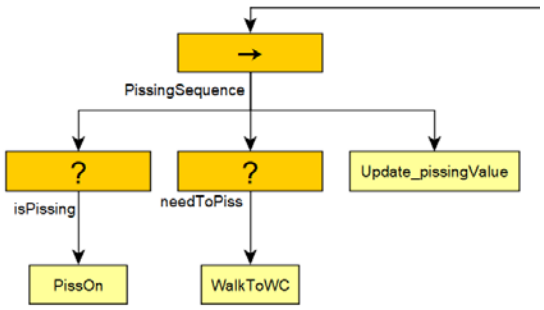


Font: Elaboració pròpia

Com s'ha explicat prèviament les accions i les condicions extreuen el seu codi de fitxers escrits en LUA, en el cas de la seqüència de ganes d'anar al lavabo (Figura 6-13) existeixen dues condicions i tres accions.

Les dues condicions comproven si un valor és més gran que un altre, així que ambdues invoquen el mateix arxiu LUA: *IsGreaterOrEqual*.

Figura 6-13 *Pissing Behaviour*



Font: Elaboració pròpia

Figura 6-14 Exemple condició en LUA:

IsGreaterOrEqual

```
function Tick (completed, pissingValue, maxValue)

            if(pissingValue >= maxValue) then  return
"Failure"
            else return "Success"
            end

end
```

Font: Elaboració pròpia

Cada acció té el seu propi arxiu de LUA associat, ja que no tenen res en comú entre elles. Es pot reutilitzar l'arxiu LUA d'una acció canviant els paràmetres a l'hora de crear-la amb l'eina d'ajuda (6.1.3.2); però no és el cas dins d'aquesta seqüencia, en canvi si es pot veure en la seqüència de cansament.

6.1.2.9. Sistema de comportaments de moviment

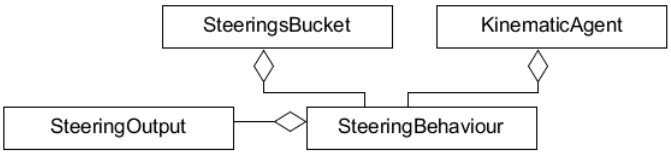
L'objectiu principal d'aquest sistema es gestionar el moviment d'un objecte a través de les formules del moviment rectilini uniformement accelerat [37]. Aquest sistema es basa principalment en les explicacions de Buckland (2005) i Millington i Funge (2009) exposades a l'apartat 3.5.3 del marc teòric.

Taula 6-9 Classes del sistema d'arbre de comportament

Scripts implicats	
Nom	**Breu descripció**
SteeringBehaviour	Component d'un agent. La seva funció és actualitzar el comportament de moviment a traves de les formules de moviment rectilini uniformement accelerat (MCUA).
SteeringOutput	Classe que desa les acceleracions lineal i angular.
SteeringsBucket	Component associat a un agent, s'encarrega d'invocar els comportaments varis específicament per aquest agent.
Comportaments varis	Existeixen diferents *scripts* com *Seek, GoTo, Wander, GoPath.* Cada un d'ells defineix un comportament diferent i modifica les variables necessàries per l'*SteeringBehaviour*.

Font: Elaboració pròpia

Figura 6-15 UML simple del sistema d'*Steerings behaviour*



Font: Elaboració pròpia

[37] Fórmules adjuntades als annexos.

6.1.3. Eines d'ajuda

Les eines d'ajuda serveixen per agilitzar la feina de l'equip. Són parts de codi que amplien les funcionalitats de l'editor de *Unity3D*, van des de personalitzar l'inspector dels objectes fins a la creació de nous objectes personalitzats a partir del propi menú del programa.

6.1.3.1. Nous objectes

Moltes vegades, al comprovar el funcionament del joc s'ha decidit afegir, modificar o eliminar diferents objectes al llarg del desenvolupament. Per fer aquesta tasca més còmoda s'ha optat per fer una eina des de la qual pots crear nous objectes i a l'inspector assignar-li les seves propietats especifiques.

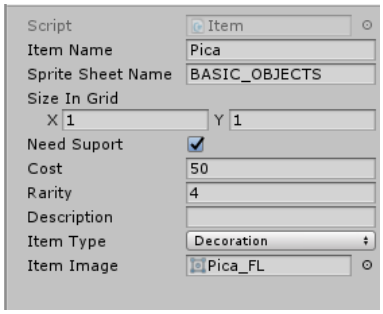
Quan un equip vol crear un nou objecte haurà de fer-ho de la mateixa manera com si creés un *asset* nou: *Assets -> Create*, aleshores trobarà una nova carpeta anomenada *MyObjects* des de la qual podrà crear-lo.

Figura 6-16 Creació de nous objectes

Font: Elaboració pròpia

Són aquests objectes els que el sistema de botiga comprova a l'hora de crear la botiga concreta d'un tipus d'objecte. A l'inspector d'un objecte creat l'equip pot canviar la mida, la imatge, el cost, etc. que es desitgi per l'objecte.

Figura 6-17 Exemple inspector d'un nou objecte



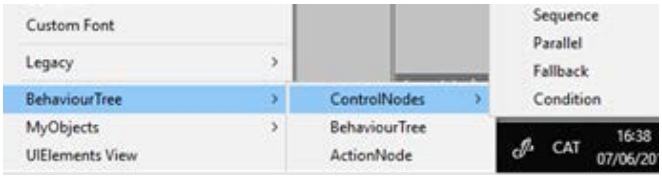
Font: Elaboració pròpia

6.1.3.2. Crear arbres de comportament

Per tal de poder canviar de forma senzilla el comportament d'un agent controlat per IA s'ha optat per crear una ajuda a l'hora de formar els arbres de comportament. Existeixen diferents eines facilitades per la botiga de Unity3D, però el projecte necessitava una adaptada per llegir LUA. Per això s'ha creat una eina senzilla que fos capaç de llegir documents LUA i alhora permetés la construcció de comportaments d'una forma més còmode.

Quan un equip vol crear un element nou per un arbre de comportament haurà de fer-ho de la mateixa manera com si crees un *asset* nou: *Assets -> Create*, aleshores trobarà una nova carpeta anomenada *BehaiourTree* des de la qual podrà crea un node de condició, una acció o un node de control. Si la seva intenció és crear un arbre nou haurà d'utilitzar un node de control (*ControlNode*).

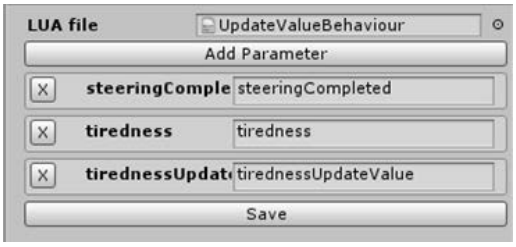
Figura 6-18 Creació elements arbre de comportament



Font: Elaboració pròpia

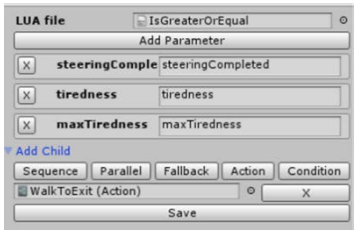
Els nodes d'acció i de condició accepten l'entrada de nous paràmetres, per indicar quins són se'ls ha d'afegir per l'inspector el nom que reben a l'*script*.

Figura 6-19 Exemple *Action*



Font: Elaboració pròpia

Figura 6-20 Exemple *Condition*



Font: Elaboració pròpia

Una condició o node de control pot contenir fills, aquests poden ser qualsevol element de l'arbre de comportament, ja sigui una acció, un node de control o una condició. Aquests fills es poden assignar directament a l'inspector.

Figura 6-21 Exemple *Control Node*

Font: Elaboració pròpia

7. CONCLUSIONS I REFLEXIÓ

Durant la primera part del procés de desenvolupament s'han donat diferències d'opinió entre els equips de treball, a més, de diferents ritmes de treball. Aquests problemes han derivat en certs imprevistos afectant així al pla de treball i al resultat final del projecte.

Al llarg d'aquest capítol s'explicaran les modificacions que s'han hagut d'aplicar a la planificació, l'impacte que han tingut aquestes i quines han estat les decisions més rellevants durant el procés.

Com es pot observar a l'apartat 5.4.1, on estan exposades les taules de planificació de tasques previstes fins el tercer prototip funcional, aquestes taules tenen aproximadament una duració de quinze dies cada una. Seguidament es compararan amb les modificacions que s'han anat aplicant durant el desenvolupament.

Les tasques marcades amb un tic (✓) són les que s'han dut a terme, les que no tenen cap marca són les que encara s'han de fer, les feines amb la lletra P (**P**) estan en procés durant el marge de temps i les marcades amb una creu (✕) i taxades són les taques que s'han desestimat degut als canvis en el disseny.

Taula 7-1 Llegenda de simbologia de les taules comparatives

✓	Tasca realitzada
✕	Tasca eliminada
P	En procès
	Pendents

Font: Elaboració pròpia

7.1. Primera quinzena

Inicialment s'havia acordat crear un sistema de zones com el que implementa el popular joc *Theme Hospital* (Bullfrog Productions, 1997). Amb aquesta mecànica el jugador escull quin tipus de zona vol construir i, segons la zona, té accés a uns objectes o a uns altres.

Per tal de treballar conjuntament amb l'equip de disseny, durant la primera fase s'ha creat un sistema axonomètric i s'ha preparat perquè des de disseny siguin capaços de crear mapes a partir de fitxers de dades. També, durant les dues primeres setmanes de desenvolupament, s'ha observat la necessitat d'una càmera, de l'estructura d'un sistema de guardat i algun element de la interfície d'usuari. Per aquests motius no s'han pogut completar les tasques dirigides a la intel·ligència artificial, ni les dirigides al sistema de construcció.

Taula 7-2 Variació tasques primer prototip

Primer prototip funcional					
Planificació prevista			Tasques re-planificades		
✓	**CONSTRUCCIÓ MÓN ISOMÈTRIC**		✓	**CONSTRUCCIÓ MON AXONOMÈTRIC**	
	✓	Implementació matemàtiques		✓	Mapes a partir d'arxius .CSV
	✓	Graella lògica		✓	Comprovació de la incorporació de personatges en 3D
	✓	Graella visual	✓	**CÀMERA**	
	✓	Mapa provisional		✓	Moviment WASD i fletxes
	SISTEMA CONSTRUCCIÓ			✓	Zoom
		Incorporació de nous elements al mapa		✓	Límits amb el mapa
		Selecció dels elements al mapa		✓	Moviment automàtic ratolí als marges
	INTELIGÈNCIA ARTIFICIAL		P	**SISTEMA DE GUARDAT**	
		Comportaments bàsics		✓	Estructura i funcions bàsiques del sistema
					Classes de guardat
			P	**SISTEMA DE CONSTRUCCIÓ**	
				✓	Selecció de zona
					Construcció de zona
			P	**INTERFÍCIE D'USUARI**	
				✓	Botó de crear zona
					PopUp escollir zona
					PopUp confirmar zona
25 de Gener					

Font: Elaboració pròpia

7.2. Segona quinzena

Durant la segona etapa de desenvolupament s'han reajustat els objectius a realitzar tenint en compte les modificacions aplicades durant la primera quinzena. Les primeres tasques han estat els que van quedar pendents durant la etapa anterior, però amb alteracions degudes als canvis de disseny.

Un cop implementada la construcció de zones s'havia de treballar amb la modificació d'aquestes, però per diversos motius de disseny, aquesta idea s'ha desestimat i s'ha decidit crear les parets de les habitacions d'una forma més propera al joc *The Sims* (Maxis, 2000) així que, es va haver d'adaptar el codi per aconseguir complir amb el disseny del joc. Com s'ha explicat, amb el nou disseny es creen les parets de les habitacions per una banda, els terres per una altra i les portes i finestres per una altra. A més, per cada mecànica s'ha hagut d'afegir una part del *HUD*.

Taula 7-3 Variació tasques segon prototip

Segon prototip funcional	
Planificació prevista	Tasques re-planificades
INTELIGÈNCIA ARTIFICIAL	**SISTEMA DE CONSTRUCCIÓ**
Pathfinding	Incorporació de nous elements al mapa
Blackboard	Selecció dels elements al mapa
Recull de dades	~~Construcció de zones~~
SISTEMA CONSTRUCCIÓ	~~Destrucció de zones~~
Modificar elements del mapa	Adaptar selecció de zona a parets
~~Inventari~~	Construcció de parets (habitacions)
INTERFÍCIE D'USUARI	Destrucció de parets (habitacions)
~~Inventari~~	**SISTEMA DE GUARDAT**
HUD	Classes de guardat
	INTERFÍCIE D'USUARI
	~~PopUp escollir zona~~
	~~PopUp confirmar zona~~
	PopUp menú construcció
	PopUp construir/destruir parets
10 de Febrer	

Font: Elaboració pròpia

7.3. Tercera quinzena

Durant la tercera quinzena de desenvolupament es va seguir optant per deixar apartada la intel·ligència artificial per centrar els esforços a les mecàniques principals. Així doncs aquestes setmanes van servir per perfeccionar el sistema de parets i afegir el sistema de terres, a més, al ja tenir un sistema de construcció base fet ja es va començar a treballar amb el sistema de guardat més profundament.

Tenir en compte que es va invertir part del temps en optimitzar el sistema de previsualització d'ambdues mecàniques (construir parets i construir terres), ja que de la forma en què s'havia programat gastava molts recursos de l'ordinador i no era la més correcta.

Taula 7-4 Variació tasques tercer prototip

Tercer prototip funcional				
Planificació prevista			Tasques re-planificades	
	INTELIGÈNCIA ARTIFICIAL		P	**SISTEMA DE CONSTRUCCIÓ**
	Anàlisi de l'entorn			Incorporació de nous elements al mapa
	Economia del joc			Selecció dels elements al mapa
	Presa de decisions (*Scripting*)		✓	Destrucció de parets (habitacions)
	SISTEMA CONSTRUCCIÓ			Modificar elements del mapa
✕	~~Esdeveniments d'elements~~		✓	Construcció de terres
	LÒGICA DEL JOC		✓	Previsualització de la selecció de parets
	Incorporació de noves característiques		✓	Previsualització de la selecció dels terres
			P	**SISTEMA DE GUARDAT**
			P	Classes de guardat
				INTELIGÈNCIA ARTIFICIAL
				Pathfinding
				Blackboard
				Recull de dades
			✓	**INTERFÍCIE D'USUARI**
			✓	PopUp construir terres
27 de Febrer				

Font: Elaboració pròpia

7.4. Quarta quinzena

A partir del quart prototip no existeix més cap planificació prèvia ja que no es va poder dur a terme sense un disseny previ, així doncs les tasques estan totes creades a partir de les necessitats del disseny i les tasques prèvies que no s'havien acabat o no s'havien fet.

Durant aquest període de temps s'ha continuat treballant amb la optimització del sistema de construcció i previsualització. També s'ha estructurat el controlador del joc com a un element que serveix com a director de tots els *scripts* i s'han afegit les classes de guardat dels sistemes de construcció que s'han implementat fins ara.

Taula 7-5 Variació tasques quart prototip

Quart prototip funcional		
Planificació prevista	Tasques re-planificades	
	P	**SISTEMA DE CONSTRUCCIÓ**
	P	Incorporació de nous elements al mapa
	P	Selecció dels elements al mapa
		Modificar elements del mapa
	✓	Millora i optimització de la previsualització
	✓	Optimització i estructura sistema de construcció
	✓	**SISTEMA DE GUARDAT**
	✓	Classes de guardat
		INTELIGÈNCIA ARTIFICIAL
		Pathfinding
		Blackboard
		Recull de dades
		Anàlisi de l'entorn
		Economia del joc
		Presa de decisions (*Scripting*)
	✓	**LÒGICA DEL JOC**
	✓	Estructura del controlador de joc
18 de Març		

Font: Elaboració pròpia

7.5. Recta final

En arribar a l'últim trimestre de feina i no tenir cap feina de disseny o art prou avançada s'ha hagut d'adaptar el codi a obtenir un prototip funcional de cara a l'última entrega, així doncs s'han realitzat els programes corresponents a la intel·ligència artificial i acabat de polir el joc.

Figura 7-1 Tasques realitzades la 5a quinzena

Sisena quinzena		
Planificació prevista	Tasques re-planificades	
	✓	**SISTEMA DE CONSTRUCCIÓ**
	✓	Arreglar uns errors sistema de construcció i decoració
	✓	Col·locar portes
		INTELIGÈNCIA ARTIFICIAL
		Pathfinding
		Blackboard
		Recull de dades
		Anàlisi de l'entorn
		Economia del joc
		Presa de decisions (*Scripting*)
	P	**SISTEMA DE GUARDAT**
	P	Classes de guardat (noves classes a guardar)
18 d'Abril		

Font: Elaboració pròpia

Figura 7-2 Tasques realitzades la 6a quinzena

Sisena quinzena		
Planificació prevista	Tasques re-planificades	
	✓	**SISTEMA DE CONSTRUCCIÓ**
	✓	Arreglar uns errors sistema de construcció i decoració
		INTELIGÈNCIA ARTIFICIAL
	✓	Comportaments de moviment
	✓	Pathfinding
	✓	Blackboard
		Recull de dades
		Anàlisi de l'entorn
		Economia del joc
	✓	Presa de decisions (*Scripting*)
	✓	Behaviour Tree (eina)
	P	**SISTEMA DE GUARDAT**
	P	Classes de guardat (noves classes a guardar)
3 de Maig		

Font Elaboració pròpia

7.6. Conclusions personals

Al llarg d'aquest projecte m'he adonat que era massa ambiciós, no només per la feina que s'havia de realitzar sinó per la coordinació que exigia amb els altres equips i la inexperiència que teníem tots del tema.

Si ara tornés a començar el projecte segurament l'acotaria més, centrant-me en un sol tema de desenvolupament i no abraçar tot el procés de desenvolupament d'un joc sencer . A més amb els coneixements obtinguts, l'experiència guanyada i un disseny més clar del que enfocaria de forma més ordenada el codi.

8. REFERÈNCIES

8.1. Bibliografia

Adams, E. (2010). *Fundamentals of game design* (2nd ed.). Berkeley, CA: New Riders.

Adams, E., & Rollings, A. (2003). *Andrew Rollings and Ernest Adams on Game Design.* New Riders.

Álvarez Mesa, J., Mazo Muñoz, C., & Moreno Cadavid, J. (2017). Diseño de una Mecánica de Juego para Propiciar Construcciones Colaborativas en Entornos Virtuales. *Actas del V Congreso Internacional de Videojuegos y Educación (CIVE'17).* Medellín, Colombia: Universidad Nacional de Colombia.

Apperley, T. H. (2006, March). Toward a critical approach to video game genres. *SIMULATION & GAMING, 37*, pp. 6-23. doi:10.1177/1046878105282278

Benito García, J. M. (2005). EL MERCADO DEL VIDEOJUEGO: UNAS CIFRAS. *ICONO 14*, 1-11.

Berger, L. (2002). Scripting: Overview and Code Generation. A S. Rabin, *AI Game Programming Wisdom* (p. 505 - 510).

Bernadó, E. (2017). Anàlisi de dades. Mataró: Tecnocampus.

Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., Yergeau, F., & Cowan, J. (1997). Extensible markup language (XML). *World Wide Web Journal 2.4*, 27-66.

Buckland, M. (2005). *Programming Game AI by Example.* Sudbury, MA, EEUU: Wordware Publishing, Inc.

Carlbom, I., & Paciorek, J. (December / 1978). Planar Geometric Projections and Viewing Transformations. *ACM Computing Surveys*, p. 465-502. doi:10.1145/356744.356750

Castronovo, F., Zappe, S., Messner, J., & Leicht, R. (2015). Design of a Construction Simulation Educational Game Through a Cognitive Lens. *122nd ASEE Annual Confercence & Exposition* (p. 26.464.2 - 26.464.15). Seattle, WA: American Society for Engineering Education.

Electronic Arts. (2018). *Origin Store: Populous*. Retrieved from Origin: https://www.origin.com/esp/en-us/store/populous/populous/standard-edition

EpicGames. (January / 2018). *Frequently asked questions (FAQ)*. Recollit de Unreal Engine: https://www.unrealengine.com/en-US/faq

Fagerholt, E., & Lorentzon, M. (2009). *Beyond the HUD: User Interfaces for Increased Player Immersion in FPS Games (Master of Science Thesis).* Sweden: Chalmers University of technology.

Flanagan, D. (2011). *JavaScript: The Definitive Guide.* Sebastopol, CA: O'Reilly Media, Inc.

Games Nostalgia. (2018). *Theme Park.* Retrieved from Games Nostalgia: https://gamesnostalgia.com/en/game/theme-park

Halterman, R. (2017). *Fundamentals of C++ programming (Draft).* Collegedale, TN.

Ierusalimschy, R. (2006). *Programming in Lua, 2nd Edition.* Rio de Janeiro: Lua.org.

Isla, D., & Blumberg, B. (2002). Blackboard Architectures. A S. Rabin, *AI Progamming wisdom* (p. 333 - 344).

Krikke, J. (2000, Juliol/Agost). Axonometry: A Matter of Perspective. *IEEE Computer Graphics and Applications, 20*(4), pp. 7-11. doi:10.1109/38.851742

Llanos , S., & Jørgensen, K. (2011). Do Players Prefer Integrated User. *DiGRA '11 - Proceedings of the 2011 DiGRA International Conference: Think Design Play.* Utrecht, Netherlands: DiGRA/Utrecht School of the Arts.

Madhav, S. (2013). *Game Programming Algorithms and techniques.* Crawfordsville, IN: Addison-Wesley.

Mangiron, C., & Orero, P. (2012). La accesibilidad en videojuegos: retos,oportunidades y propuestas. *Buenas prácticas de accesibilidad en videojuegos*, 22 - 28.

Marks, S., Windsor, J., & Wünsche, B. (2007). Evaluation of Game Engines for Simulated Surgical Training. *GRAPHITE '07* (p. 273-280). Perth, Australia: ACM.

McTee, M. (2014). E-Sports: More Than Just a Fad. *Oklahoma Journal of Law and Technology*, 1-27.

Millington, I., & Funge, J. (2009). *Artificial Intelligence for games* (2nd ed.). Burlington, MA, EEUU: Elsevier.

Nakov, S., & Kolev, V. (2013). *Fundamentals of computer programming with C#.* Sofia.

Peacocke, M., Teather, R., & Carette, J. (2015). Performance of HUDs and Diegetic Displays in FPS Games. *2015 Games Entertainment Media Conference (GEM)* (p. 70-78). Toronto, Canada: IEEE. doi:10.1109/GEM.2015.7377211

Perani, L., Moraes, E., & Sanches, I. (Juliol / 2017). As mecânicas do divertimento:uma análise de affordances em games de simulação de parques de diversão. (UFBA, Ed.) *Metamorfose, 2,* p. 188-207.

Pile Jr, J. (2013). *2D Graphics Programming for Games.* New York: CRC Press.

Poole, S. (2000). *Trigger Happy: Videogames and the Entertainment Revolution.* Arcade Publishing.

Rabin, S. (2002). Implementing a State Machine Language. A S. Rabin, *AI Game Programming wisdom* (p. 314-320).

Ramos, R. (2011). *Tema II. Transformaciones lineales.* Universidad de Oviedo.

Rüppel, U., & Schatz, K. (2011). Designing a BIM-based serious game for fire safety evacuation simulations. *Advanced Engineering Informatics,* 600-611.

Scheck, F. (2010). *Mechanics From Newton's Laws to Deterministic Chaos.* NY: Springer.

Selma García, A. (14 de June de 2017). Desarrollo de aplicaciones para múltiples plataformas y uso de realidad virtual con Unity 3D. *Proyecto Final de Carrera*. Valencia, Valencia, Espanya: Universitat Politècnica de València.

Shafranovich, Y. (2005). Common format and MIME type for comma-separated values (CSV) files.

Sommerville, I. (2011). *Software engineering* (9th ed.). Boston,MA: Pearson.

Tozou, P. (2002). The Perils of Al Scripting. In S. Rabin, *AI Game Programming Wisdom* (pp. 541 - 547).

Unity. (January / 2018). *Editor*. Recollit de Unity3D: https://unity3d.com/es/unity/editor

Unity. (January / 2018). *Unity3D*. Recollit de Unity3D: https://docs.unity3d.com

Vujosevic-Janicic, M., & Tosic, D. (2008). The role of programming paradigms in the first programming courses. *The teaching of mathematics, XI*, pp. 63–83.

Wolf, M. (2009). Chapter 8 Z-axis Development in the Video Game. In B. Perron, & M. J. Wolf, *The video game theory reader 2* (pp. 151 - 168). New York: Routledge.

Wolf, M. J. (2000). *The Medium of the Video Game* (1st ed.). Texas: University of Texas Press.

Xu, X. (2011). *PathFinding.js*. Retrieved from Github: http://qiao.github.io/PathFinding.js/visual/

8.2. Ludografia

Atari Inc. (1980). *Battlezone*. Published by: Atari Inc.

Bullfrog Productions (1989). *Populous*. Published by: Electronic Arts.

Bullfrog Productions (1994). *Theme Park*. Published by: Electronic Arts

Bullfrog Productions (1997). *Theme Hospital*. Published by: Electronic Arts

Bullfrog Productions (1999). *Theme Park World*. Published by: Electronic Arts

Chris Sawyer (1999). *Roller Coaster Tycoon*. Published by: Hasbro Interactive

EA Redwood Shores (2008). *Dead Space*. Published by: Electronic Arts.

Ensemble Studios (1997). *Age of Empires*. Plubished by: Microsoft Game Studios.

Firaxis Games (2001). *Civilization III*. Published by: Infogrames.

Guerilla Games (2009). *Killzone 2*. Published by: Sony Computer Entertainment.

Konami (1992). *Axelay*. Plubished by; Konami.

Maxis (1989). *SimsCity*. Published by: Maxis

Maxis (2000) *The Sims*. Published by: Electronic Arts

Mojang AB (2009). *Minecraft*. Published by: Mojang.

Rockstar North (2008) *Grand Theft Auto IV*. Published by: Rockstar Games.

Valve Software (2008). *Left 4 Dead*. Published by: Electronic Arts.

ANNEXOS

Índex Annexos

Índex de figures

Índex de taules

1. Fórmules

1.1. Cinemàtiques

Els algoritmes de moviment aplicats a la programació del joc segueixen els principis de la cinemàtica, és per això que estan basats en les fórmules de M.R.U, M.R.U.A, M.C.U i M.C.U.A.

Taula 1-1 Fórmules cinemàtiques

M.R.U Moviment rectilini uniforme	v = cte a = 0 $x = x_0 + v_0 \cdot t$
M.R.U.A Moviment rectilini uniformement accelerat	a = cte $x = x_0 + v_0 \cdot t + \frac{1}{2} a \cdot t^2$ $v = v_0 + a \cdot t$
M.C.U Moviment circular uniforme	ω = cte α = 0 $\varphi = \varphi_0 + \omega \cdot t$
M.C.U.A Moviment circular uniformement accelerat	α = cte $\varphi = \varphi_0 + \omega_0 \cdot t + \frac{1}{2} \alpha \cdot t^2$ $\omega = \omega_0 + \alpha \cdot t$

Font: Elaboració pròpia

2. Pseudocodi

2.1. Algoritmes de moviment IA

Les 3 imatges següents exposen el pseudocodi proposat per Millington i Funge (2009) de les estructures per crear el moviment cinemàtic dels agents.

Figura 2-1 Paràmetres de l'estructura base moviment cinemàtic

```
struct Kinematic:
  position    # a 2 or 3D vector
  orientation # a single floating point value
  velocity    # another 2 or 3D vector
  rotation    # a single floating point value
```

Font: (Millington & Funge, 2009, p.46)

Figura 2-2 Funció Update de l'estructura base del moviment cinemàtic

```
struct Kinematic:

  ... Member data as before ...

  def update(steering, time):

    # Update the position and orientation
    position += velocity * time +
                0.5 * steering.linear * time * time
    orientation += rotation * time +
                   0.5 * steering.angular * time * time

    # and the velocity and rotation
    velocity += steering.linear * time
    orientation += steering.angular * time
```

Font: (Millington & Funge, 2009, p. 47)

Figura 2-3 Estructura paràmetres de retorn

```
struct SteeringOutput:
  linear    # a 2 or 3D vector
  angular   # a single floating point value
```

Font: (Millington & Funge, 2009, p.46)

Figura 2-4 Pseudocodi per orientar un objecte

```
def getNewOrientation(currentOrientation, velocity):

  # Make sure we have a velocity
  if velocity.length() > 0:

    # Calculate orientation using an arc tangent of
    # the velocity components.
    return atan2(-static.x, static.z)

  # Otherwise use the current orientation
  else: return currentOrientation
```

(Millington & Funge, 2009, p. 49)

2.2. Exemples de pseudocodi de comportaments d'IA

Figura 2-5 Pseudocodi pel comportament Seek

```
class KinematicSeek:
  # Holds the static data for the character and target
  character
  target

  # Holds the maximum speed the character can travel
  maxSpeed

  def getSteering():

    # Create the structure for output
    steering = new KinematicSteeringOutput()

    # Get the direction to the target
    steering.velocity =
      target.position - character.position

    # The velocity is along this direction, at full speed
    steering.velocity.normalize()
    steering.velocity *= maxSpeed

    # Face in the direction we want to move
    character.orientation =
      getNewOrientation(character.orientation,
                        steering.velocity)

    # Output the steering
    steering.rotation = 0
    return steering
```

Font: (Millington & Funge, 2009, p. 50)

Figura 2-6 Pseudocodi pel comportament Wander

```
class KinematicWander:
  # Holds the static data for the character
  character

  # Holds the maximum speed the character can travel
  maxSpeed

  # Holds the maximum rotation speed we'd like, probably
  # should be smaller than the maximum possible, to allow
  # a leisurely change in direction
  maxRotation

  def getSteering():

    # Create the structure for output
    steering = new KinematicSteeringOutput()

    # Get velocity from the vector form of the orientation
    steering.velocity = maxSpeed *
      character.orientation.asVector()

    # Change our orientation randomly
    steering.rotation = randomBinomial() * maxRotation

    # Output the steering
    return steering
```

Font: (Millington & Funge, 2009, pp. 53-54)

2.3. Algoritme A*

Figura 2-7 Pseudocodi algoritme A*

```
1.  Add the starting node to the open list.
2.  Repeat the following steps:
    a.  Look for the node which has the lowest
        f on the open list. Refer to this node
        as the current node.
    b.  Switch it to the closed list.
    c.  For each reachable node from the current
        node
         i.  If it is on the closed list, ignore
             it.
        ii.  If it isn't on the open list, add it
             to the open list. Make the current
             node the parent of this node. Record
             the f, g, and h value of this node.
       iii.  If it is on the open list already,
             check to see if this is a better
             path. If so, change its parent to the
             current node, and recalculate the f
             and g value.
    d.  Stop when
         i.  Add the target node to the closed
             list.
        ii.  Fail to find the target node, and the
             open list is empty.
3.  Tracing backwards from the target node to the
    starting node. That is your path.
```

Font: (Cui & Shi, 2011, p.125)

2.4. Nodes d'un arbre de comportament

Taula 2-1 Tipus de node d'un Behaviour Tree

Node type	Symbol	Succeeds	Fails	Running
Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	→	If all children succeed	If one child fails	If one child returns Running
Parallel	⇉	If $\geq M$ children succeed	If $> N - M$ children fail	else
Action	text	Upon completion	If impossible to complete	During completion
Condition	text	If true	If false	Never
Decorator	◊	Custom	Custom	Custom

Font: (Colledanchise & Ögren, 2018, p. 9)

Figura 2-8 Node *Sequence* (pseudocodi)

```
1 for i ← 1 to N do
2     childStatus ← Tick (child(i))
3     if childStatus = Running then
4         return Running
5     else if childStatus = Failure then
6         return Failure
7 return Success
```

Font: (Colledanchise & Ögren, 2018, p. 7)

Figura 2-9 Node *Fallback* (pseudocodi)

```
1 for i ← 1 to N do
2     childStatus ← Tick (child(i))
3     if childStatus = Running then
4         return Running
5     else if childStatus = Success then
6         return Success
7 return Failure
```

Font: (Colledanchise & Ögren, 2018, p. 8)

Figura 2-10 Node *Parallel* (pseudocodi)

```
1 for i ← 1 to N do
2     childStatus(i) ← Tick (child(i))
3 if Σ_{i:childStatus(i)=Success} 1 ≥ M then
4     return Success
5 else if Σ_{i:childStatus(i)=Failure} 1 > N − M then
6     return Failure
7 return Running
```

Font: (Colledanchise & Ögren, 2018, p. 8)

Informació extreta de:

Colledanchise, M., & Ögren, P. (2018). *Behaviour Trees in Robotics and AI*

3. Diagrames UML

Ruta: DVD_ANNEXOS\ Diagrama UML.pdf

4. Codi Font

El codi font és dins el projecte de Unity adjuntat als annexos:

Ruta:DVD_ANNEXOS\IsometricTycoon.rar

5. Executable del prototip del joc

Ruta: DVD_ANNEXOS\eTycoon\eTycoon.exe