Treball de Fi de Grau

Grau en Enginyeria Informàtica

Desenvolupament d'aplicacions per mostrar conceptes de programació de videojocs

Memòria del projecte

Autor

Adrià Fors Munar

Especialitat

Computació

Director

Antonio Chica Calaf

Departament

Ciències de la Computació

Defensa

05/07/2019

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

Resum

La cerca de camins o *pathfinding* és una de les moltes disciplines que involucren la programació de la intel·ligència artificial d'un videojoc. Aquest sistema permet que els objectes del joc puguin trobar el camí més curt entre dos punts del món virtual d'aquest evitant els obstacles que puguin haver-hi.

A l'assignatura *Videojocs* (VJ), una assignatura que s'imparteix a la *Facultat d'Informàtica de Barcelona* com a opcional dins el *Grau en Enginyeria Informàtica*, aquest tema forma part dels continguts inclosos a la guia docent. Tanmateix, només es tracta a les classes de teoria com un dels temes relacionats amb la intel·ligència artificial, i no a les sessions de laboratori, on l'alumnat posa en pràctica els coneixements adquirits i programa dos petits videojocs. L'assignatura ofereix als estudiants alguns exemples de codi font sobre els quals treballar i poder començar els projectes, però cap dels actuals tracta la programació d'una eina de cerca de camins per un videojoc.

En aquest projecte s'ha provat d'oferir una solució per aquesta situació, i s'ha desenvolupat una aplicació de mostra d'un sistema de *pathfinding* en una quadrícula, seguint el mateix estil i base de codi dels exemples de què disposa l'assignatura per aconseguir un codi font completament compatible, que es pugui incorporar al laboratori de l'assignatura. L'objectiu principal del projecte ha estat aconseguir un producte que pogués ser utilitzat com a eina de docència en l'assignatura que l'emmarca.

En aquest document es descriu i detalla el procés de desenvolupament d'aquesta aplicació i el codi font de la mateixa. També es detalla com s'ha gestionat el projecte, es contextualitza el tema que tracta i s'enumeren les eines utilitzades en el desenvolupament.

Resumen

La búsqueda de caminos o *pathfinding* es una de las muchas disciplinas involucradas en la programación de la inteligencia artificial de un videojuego. Este sistema permite que los objetos del juego puedan encontrar el camino más corto entre dos puntos del mundo virtual del mismo evitando los obstáculos que pueda haber.

En la asignatura *Videojuegos* (VJ), una asignatura que se imparte en la *Facultat d'Informàtica de Barcelona* (FIB) como opcional en el *Grado en Ingeniería Informática*, este tema forma parte de los contenidos que se incluyen en la guía docente. Sin embargo, solo es tratado en las clases de teoría como uno de los temas relacionados con la inteligencia artificial, y no en las sesiones de laboratorio, donde el alumnado pone en práctica los conocimientos adquiridos y programa dos pequeños videojuegos. La asignatura ofrece a los estudiantes algunos ejemplos de código fuente sobre los cuales trabajar y poder empezar los proyectos, pero ninguno de los actuales trata la programación de una herramienta de búsqueda de caminos para un videojuego.

En este proyecto se ha tratado de ofrecer una solución para esta situación, y se ha desarrollado una aplicación de muestra de un sistema de *pathfinding* en una cuadrícula, siguiendo el mismo estilo y base de código de los ejemplos de la asignatura para obtener un código fuente completamente compatible, que se pueda incorporar al laboratorio de la asignatura. El objetivo principal ha sido obtener un producto que pudiera ser usado como herramienta de docencia en la asignatura marco del proyecto.

En este documento se describe y detalla el proceso de desarrollo de esta aplicación y el código fuente de la misma. También se detalla cómo se ha gestionado el proyecto, se contextualiza el tema que trata y se enumeran las herramientas usadas en el desarrollo.

Abstract

Pathfinding or pathing is one of the many subjects involved in programming the artificial intelligence in a videogame. This method allows for a game object to find the sortest path between two given points in the game world while dodging the obstacles located in it.

Videogames (VJ) is an optional subject taught in the *Bachelor Degree in Informatics Engineering* at the *Facultat d'Informàtica de Barcelona* (FIB) in which pathfinding is part of the contents of the teaching guide. However, this topic in only taught at the theory sessions as one of the many topics associated with artificial intelligence and not in lab sessions, where students put the concepts learned in class into practice by programming two small videogames. The subject gives its students some source code examples to start working in to carry out the projects, but none of those examples is about pathfinding programming.

This project has tried to solve this situation by creating a demo application which shows a working pathfinding tool for grids. The application follows the same coding style as the previously mentioned examples to obtain a source code totally compatible with them, which can be incorporated into the lab sessions of the subject. The main objective has been obtaining a product that could be used as a teaching tool in the framework subject of the project.

This document explains and gives details about the development process and source code of the application, as well as the tools used in it. It also describes the project management and contextualizes the topic the project is about.

1. INTRODUCCIÓ

L'objectiu d'aquest primer apartat introductori es oferir al lector una presa de contacte amb el projecte i amb els temes que aquest tracta. En ell es defineix la temàtica, els objectius i els requeriments del projecte. En el següent apartat, es descriuen les eines que s'han utilitzat en el decurs del projecte, així com els algorismes utilitzats i el desenvolupament i estructura del codi font. En tercer lloc, es mostren els resultats obtinguts i les conclusions extretes un cop finalitzat el projecte. Finalment, es dedica un apartat a la metodologia emprada, l'explicació del decurs del projecte, la planificació temporal i la sostenibilitat del mateix, que inclou un anàlisi pressupostari i un anàlisi sobre el seu possible impacte.

1.1. El Projecte

Aquest projecte és un Treball de Fi de Grau del Grau en Enginyeria Informàtica, desenvolupat a la Facultat d'Informàtica de Barcelona durant el quadrimestre de primavera del 2019. El treball s'ha desenvolupat en el marc de *Videojocs*, una assignatura opcional en el pla d'estudis del grau. Aquesta assignatura introdueix als alumnes els conceptes i tècniques fonamentals de la programació de videojocs, que es posen en pràctica a les sessions de laboratori, on els alumnes dissenyen i desenvolupen dos videojocs utilitzant els coneixements adquirits [1].

Com a punt de partida per aquesta feina, el professorat entrega als alumnes un codi font base que té un seguit de funcionalitats ja implementades. L'objectiu és facilitar i agilitzar la tasca dels projectes, però també mostrar les tècniques i coneixements explicats a teoria en funcionament i servir d'exemple sobre el qual començar a treballar. Partint d'aquesta base, els estudiants poden començar la feina més ràpidament, sense necessitat de dedicar una considerable quantitat de temps a implementar funcionalitats fonamentals. Tenint això en compte, el projecte, de la mateixa manera que els alumnes, utilitza i parteix d'aquesta base de funcionalitats per desenvolupar les seves pròpies, atès que l'àmbit del mateix és l'assignatura a la qual pertanyen. En l'apartat dedicat al desenvolupament es defineix quines són les funcionalitats de partida i quines són les originals del projecte.

Com s'ha dit, aquest codi base cobreix les necessitats més fonamentals de la programació d'un videojoc. Tanmateix, la programació de videojocs engloba diverses disciplines i àrees, com els gràfics, el so, la intel·ligència artificial, la simulació de la física, la xarxa, la gestió de l'execució, etc [2].

Actualment, les eines mencionades cobreixen només una part de les disciplines que s'han citat, concretament les relatives als gràfics (particularment crear la finestra del joc i mostrar-hi la imatge) i la gestió de l'execució. Amb això en ment, el projecte neix amb la intenció d'ampliar la base de funcionalitats proporcionades als estudiants, per tal de poder cobrir altres camps i ampliar el ventall de possibilitats de què l'alumnat disposa en els projectes.

Dit això i a grans trets, l'objectiu principal del TFG ha estat desenvolupar i integrar en una aplicació de mostra un seguit de recursos software que implementin funcionalitats relacionades amb la cerca de camins (pathfinding) en quadrícules, a fi que puguin ser utilitzats en l'assignatura que emmarca el projecte.

1.2. Contextualització

Un cop resumit el nucli del projecte, en aquest apartat es defineix més concretament el marc del mateix, l'assignatura VJ. També es llisten els actors que hi estan implicats i es defineix la disciplina que tracta i l'estat de l'art de la mateixa.

1.2.1. Marc del projecte

Com s'ha mencionat, aquest projecte neix en el marc de VJ. Per tant, és convenient definir de forma més acurada la relació del mateix amb aquesta assignatura. El projecte està estretament relacionat amb les activitats de les sessions de laboratori, i el seu propòsit principal és que els resultats del mateix es puguin fer servir en aquest tipus de classes. La part de laboratori es divideix en dues fases, i cada una d'elles finalitza amb l'entrega d'un projecte en forma de videojoc, el primer en dues dimensions i el segon en tres. Donat que la programació en tres dimensions resulta en general més complexa, l'ús de programes específics per la creació de videojocs està permès en la segona part, però no en la primera. En aquesta, l'alumnat ha de realitzar tot el desenvolupament sense una eina especialitzada, únicament fent servir C++, OpenGL i un seguit de biblioteques específiques, així com el projecte de partida que el professorat li entrega.

Dit això, una de les finalitats del projecte és que pugui ser utilitzat durant la primera part del laboratori, en els projectes en dues dimensions, en els que es requereix que els alumnes facin servir una tecnologia i eines concretes. Per tal que el projecte pugui adequar-se a la situació i sigui compatible, aquest s'ha desenvolupat utilitzant la mateixa tecnologia. La totalitat del codi font, per tant, s'ha escrit en C++, la part gràfica s'ha programat en OpenGL i s'han fet servir les mateixes biblioteques i recursos que es permet utilitzar a classe. D'aquesta manera el codi del projecte es podrà reutilitzar al laboratori i es podrà incorporar als exemples que s'entreguen als alumnes.

1.2.2. Actors implicats

Hi ha quatre actors implicats de forma directa o indirecta en aquest projecte, ja sigui a nivell de desenvolupament o com a beneficiaris o usuaris del resultat. En aquest apartat es detallaran quines són aquestes persones o col·lectius, i quina relació tenen amb el projecte.

- **Desenvolupador**

El projecte consta només d'un desenvolupador, l'alumne autor del projecte i d'aquest document. Ell s'ha encarregat de la part tècnica del projecte (disseny, desenvolupament del codi, proves del mateix…), així com també de la redacció dels documents necessaris, de la recerca d'informació, de la presentació del projecte i de gran part de la gestió associada. Per tant, ha estat el principal responsable d'assolir els objectius del projecte.

- **Director del projecte**

El director del projecte ha estat Antonio Chica Calaf, professor del departament de Ciències de la Computació i coordinador de l'assignatura VJ. La seva tasca principal ha estat guiar, aconsellar i supervisar el desenvolupador en la seva tasca, en la gestió i desenvolupament del projecte i en les dificultats que han sorgit.

- **Professorat de VJ**

Com s'ha manifestat, el resultat del projecte té com a objectiu bàsic que pugui ser utilitzat en l'assignatura. Una de les possibles utilitats és fer-ho servir a les classes perquè el professorat tingui quelcom funcional per poder mostrar i entregar als alumnes. Això pot facilitar les explicacions i afavorir la comprensió dels alumnes dels conceptes exposats.

- **Alumnat de VJ**

Aquest és el col·lectiu que serà principal beneficiari i usuari del resultat del projecte. En primer lloc, l'augment de la base de funcionalitats pels projectes amb la cerca de camins permetrà que els estudiants puguin desenvolupar videojocs de tipus més divers, que necessitin una funcionalitat com aquesta que requeriria molt de temps i esforç si hagués de ser creada des de zero.

En segon lloc, els alumnes podran tenir un exemple sobre com es programa i s'estructura el codi per la cerca de camins i quines tècniques de programació es fan servir. Disposar d'un exemple funcional permetrà a l'alumnat entendre això amb més facilitat.

Per últim, la integració d'aquesta funció en una aplicació que la mostri en funcionament i que els alumnes puguin manipular (canviant certs paràmetres) contribuirà a l'assimilació dels conceptes i permetrà comprendre com afecten aquests paràmetres al comportament i algorismes utilitzats en la cerca de camins, ja que es podrà veure el resultat obtingut amb cada canvi.

1.2.3. Àmbits d'interès

- **Cerca de camins**

En l'àmbit de la computació, es coneix com a cerca de camins (o *pathfinding* de l'anglès) a la cerca o traçat del camí més curt entre dos punts que duu a terme un ordinador. Aquesta cerca es realitza en un graf, que es fa servir per representar l'espai de cerca corresponent a

cada cas. El problema, en teoria de grafs, es coneix com a problema del camí més curt o del camí mínim [3] i consisteix en, donat un node inici i un destí en del graf, trobar un camí que els uneix tal que la suma dels pesos de les arestes per les que passa és mínima. Els pes representa el cost (en temps, distància...) d'anar d'un node a un altre adjacent.

El *pathfinding*, doncs, consisteix en dos problemes principals: el primer, trobar un camí entre els dos nodes indicats, el segon, aconseguir que el camí trobat sigui mínim. El primer problema pot ser resolt amb algorismes com *Breadth-first* i *Depth-first Search* (*BFS* i *DFS* respectivament), que ho aconsegueixen explorant tots els nodes del graf de forma exhaustiva, començant des del node inicial i expandint la cerca pels adjacents fins a trobar-se amb el destí. Aquests algorismes, però, retornen el primer camí que troben i no tenen en compte el cost del mateix. Per tal de resoldre el segon problema, més complicat, se n'ha de fer servir un que tingui en compte el cost de les arestes per les que passa i que, en trobar un camí, sàpiga si pot existir-ne un altre de cost menor. Per aquesta tasca, habitualment es fa servir l'algorisme de Dijkstra [3] o un altre basat en el mateix, com pot ser A*.

En l'àmbit dels videojocs, el *pathfinding* s'utilitza per calcular el camí que haurà de seguir un element del joc per anar d'un punt a un altre dins el món del joc, esquivant els obstacles que hi hagi i/o minimitzant el cost del seu camí. És un camp d'estudi important en aquest context, ja que un videojoc necessita aquest tipus de funcionalitat habitualment. En general, tot videojoc que contingui agents dinàmics que hagin de moure's cap a un destí, com podria ser un vehicle, una persona, un enemic que persegueix el jugador... requeriran d'alguna manera de poder evitar els possibles obstacles del món (com poden ser parets, arbres, la orografia del terreny...) de forma intel·ligent i arribar al seu destí eficientment.

Videojocs com els pertanyents al gènere RTS (*Real-Time Strategy*, estratègia en temps real) serveixen per exemplificar clarament aquesta necessitat. Aquest gènere de videojocs típicament necessita un sistema de *pathfinding* eficaç perquè el jugador pugui controlar els personatges de forma còmoda. En aquests videojocs el jugador controla molts personatges i els mou indicant-los a on han d'anar, generalment clicant amb el ratolí a la zona del mapa on vol que es dirigeixin. Donada aquesta situació, els personatges han de ser capaços de trobar el camí cap a on se'ls ha ordenat moure's sense que el jugador els indiqui quina ruta han de seguir, ja que això resultaria tediós.

Figura 1: *Age of Empires: Definitive Edition*, un videojoc RTS que utilitza *pahtfinding* per moure els personatges esquivant els edificis.

A més a més, també cal que el camí que segueixin sigui, en la majoria de casos, el més curt. Això és degut a dos motius principals. Primer, els personatges s'han de moure de forma coherent perquè el jugador sigui capaç de preveure per on aniran i això li permeti planificar la seva estratègia. Segon, els personatges han d'arribar al destí en el menor temps possible perquè en aquests jocs és important executar les accions amb rapidesa. Aquestes dues condicions es compleixen amb el camí mínim, ja que és el que es recorre en menys temps a la vegada que és el que normalment el jugador espera.

- **Tractament del problema**

Tots els algorismes que s'han mencionat requereixen un graf per poder operar. És a dir, que no poden treballar de forma directa sobre l'escenari del joc, sinó que necessiten un graf per poder realitzar una cerca. Per tant, per poder aplicar un d'aquests algorismes en el *pathfinding* d'un videojoc, serà necessari representar l'escenari del joc amb un graf que aquests algorismes puguin entendre. Caldrà, doncs, dividir l'escenari d'alguna manera en nodes, que estaran connectats entre si per arestes si corresponen a zones adjacents en l'escenari, a les quals se'ls podrà assignar un cost en funció de la distància que representin o del tipus de terreny on se situïn (el moviment per un bosc, per exemple, segurament serà més lent que per carretera i, per tant, ha de tenir un cost major). Hi ha diverses estratègies per afrontar aquest problema, però majoritàriament es poden dividir en dues, en funció de si divideixen el mapa de forma regular (en cel·les de la mateixa forma i mida, per exemple) o no [4].

En el primer cas, es divideix l'escenari del joc en caselles o cel·les (de la mateixa forma i mida) i s'afegeix un node al graf per cada una d'elles. Per cada parella de cel·les adjacents, es crea una aresta que les uneix si i només si les dues caselles corresponen a zones del mapa per les quals un personatge es pot desplaçar. D'aquesta manera, s'obté un graf que

representa totes les zones vàlides (on un es pot desplaçar) del mapa que no permet el moviment a través dels obstacles. Generalment, la forma de les cel·les és una entre dues possibles: quadrada o hexagonal.

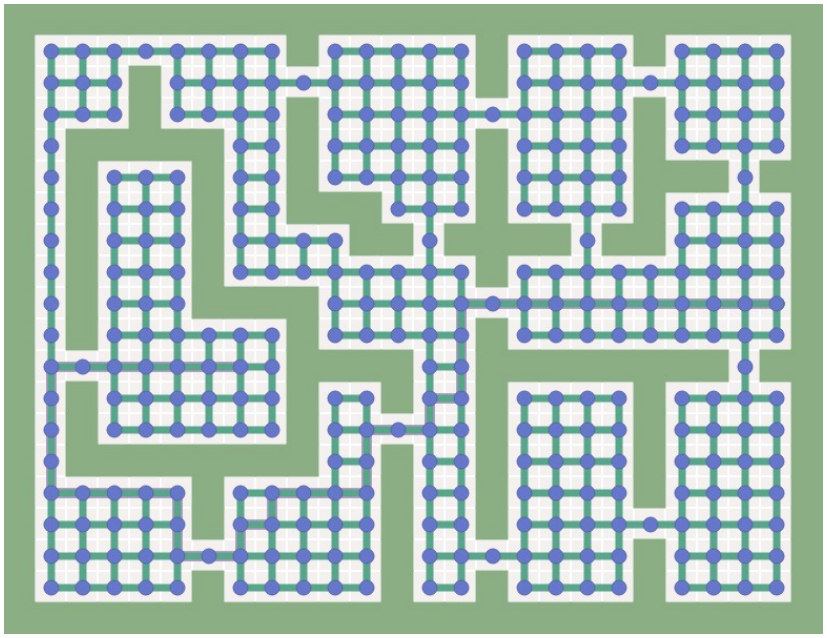

Figura 2: Escenari dividit amb una quadrícula i el graf que la representa.

Aquest tipus de representació té l'avantatge que és simple, ja que es pot representar amb un vector de cel·les que indiqui si s'hi pot caminar o no. Degut a això, el graf resultant és fàcilment modificable, ja que convertir una cel·la vàlida en no vàlida i a l'inrevés consisteix simplement en canviar l'estat de la posició del vector corresponent a la cel·la. A més a més, resulta fàcil de recórrer, ja que un pot assignar coordenades a cada una de les caselles i fer-les servir per consultar el seu estat dins el vector de forma directa, de la mateixa manera que s'accediria a una matriu.

Aquest sistema, però, sol consumir força memòria quan representa un escenari gran, especialment si aquest té àrees grans sense obstacles, ja que per representar una zona per la qual es pot caminar sense problemes utilitzarà moltes cel·les [4] (cosa que resultarà ineficient). A més a més, aquesta sistema representa l'escenari amb una precisió que depèn de la mida de les caselles. Com més petites siguin, més precisa serà la representació dels obstacles i més acurat podrà ser el moviment dels personatges pel mapa, però també consumirà més memòria.

L'altre sistema amb el qual es pot convertir l'escenari en un graf també consisteix en dividir el mateix en zones, però aquest cop de formes i mides diferents, que s'adaptin a la mida i forma de la zona del mapa que s'està representant en concret. Un dels sistemes és fer servir un *waypoint graph*, que consisteix en situar una sèrie de nodes del graf a les zones vàlides de l'escenari, repartits de manera irregular, que depèn de la forma de l'escenari en qüestió. D'aquesta manera s'obté una representació molt aproximada de l'escenari en forma de graf, fàcil d'implementar, que no consumeix gaire memòria, però resulta en un graf que

representa el mapa amb poca precisió. L'alternativa és fer servir *navigation meshes* (malles de "navegació"), tècnica que consisteix en dividir l'escenari en polígons convexos que representin les zones "navegables" de l'escenari, i indicar amb arestes del graf si els polígons són adjacents o no. Aquest sistema permet representar l'escenari de forma molt precisa, més que el sistema de caselles, i té l'avantatge que permet representar vastes zones sense obstacles només amb uns quants punts que representin el/s polígon/s en què s'ha dividit la zona. Tot i així, aquest sistema resulta més complex d'implementar i més difícil de modificar de forma dinàmica, ja que afegir o treure un obstacle de l'escenari comporta haver de canviar la forma dels polígons de la malla i habitualment també crear-ne de nous.

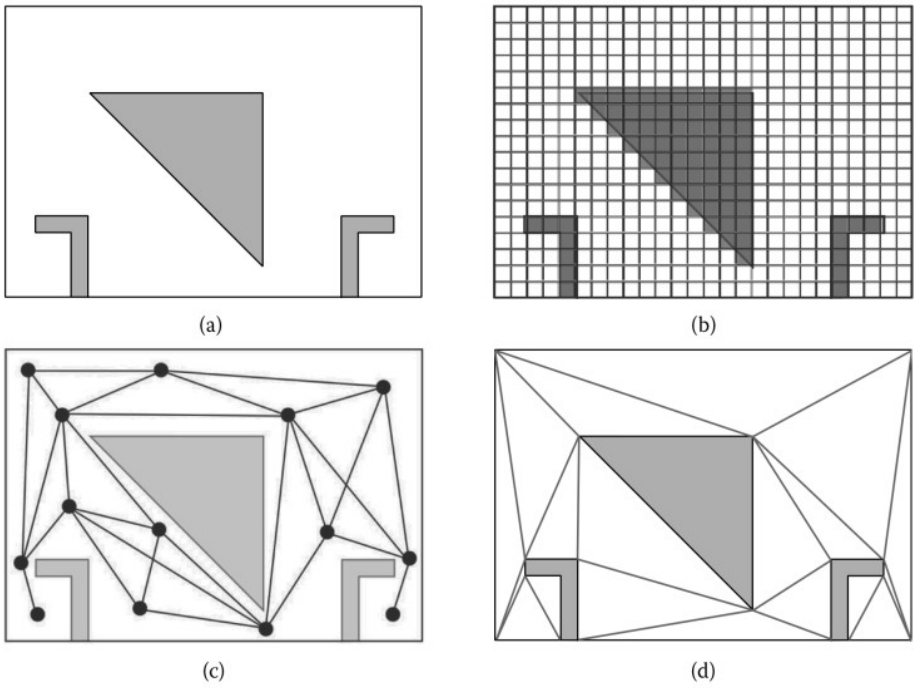

Figura 3: Escenari representat de tres maneres diferents:
(a) Escenari original, (b) quadrícula, (c) *waypoint graph,* (d) *navigation mesh.*

Un cop es disposa de l'escenari en forma de graf, es pot aplicar algun dels algorismes abans citats per realitzar la cerca. Com s'ha dit, l'algorisme que es fa servir típicament per aquest tipus de situacions és l'algorisme de Dijkstra. Aquest algorisme, però, realitza una cerca considerant tots els nodes per igual, sense tenir en compte els que estan més a prop de l'objectiu. Aquest sistema pot resultar convenient si es vol calcular el camí més curt des d'un node a tots els altres del graf, però si només es necessita el camí entre dos nodes concrets, en general és convenient prioritzar la cerca cap als nodes més prometedors, és a dir, els que estan més a prop del destí. Per aquesta situació existeix l'algorisme A* (pronunciat "A estrella"), que intenta dirigir la cerca cap a l'objectiu considerant primer els nodes més propers al mateix mitjançant un heurístic.

Avui en dia l'algorisme A* és l'estàndard dels sistemes de *pathfinding* dels videojocs moderns [5], però pot tenir un cost computacional moderadament alt en algunes situacions, cosa que resulta un problema si no es disposa d'un *hardware* prou potent. En el passat, fer

servir aquest algorisme resultava inviable, i els primers videojocs RTS no es podien permetre utilitzar-lo. Enlloc d'A* o un de similar, feien servir l'algorisme de traçat de contorns, que consisteix en moure's en línia recta cap a l'objectiu i superar els obstacles resseguint (traçant) el seu contorn. Jocs com *Warcraft: Orcs & Humans* (1994), el primer videojoc de la franquícia *Warcraft*, utilitzaven aquest algorisme per moure els personatges, ja que la potència dels ordinadors de l'època no permetia un algorisme més sofisticat.

Figura 4: Warcraft: Orcs & Humans, utilitza traçat de contorns. La línia indica la trajectòria seguida per un personatge. S'observa que recorre el contorn de l'edifici inferior dret.

Aquest algorisme, però, no resultarà mai en el camí òptim tret que no hi hagi obstacles en la ruta o recórrer els contorns sigui efectivament el camí més curt (cosa que habitualment no succeeix). A més, en funció dels obstacles de l'escenari, el camí resultant pot no només no ser el més curt sinó molt més llarg que l'òptim i donar molta més volta. És per això que el traçat de contorns només té sentit en sistemes de potència molt limitada o en escenaris tan simples que la diferència entre el resultat d'aquest i d'un com A* sigui molt petita.

1.2.4. Estat de l'art

En aquest apartat es descriuen algunes de les aplicacions i/o solucions que existeixen actualment sobre *pathfinding* que mostrin la tècnica en funcionament i permetin interactuar-hi o expliquin i/o exemplifiquin la seva implementació.

- **Red Blob Games**

Red Blob Games és una pàgina web on el seu creador, Amit Patel (graduat en ciències de la computació per la universitat de Stanford)[6], es dedica a crear explicacions visuals i

interactives sobre matemàtiques i algorismes, fent servir els videojocs per exemplificar-ho [7]. Algunes de les que ha publicat a la web són sobre *pathfinding*, més particularment sobre l'ús d'A* en el mateix.

A la secció *Introduction to the A* Algorithm* [8] explica com es representen els escenaris d'un joc en un graf i permet veure diverses animacions del funcionament dels algorismes BFS, Dijkstra i A* en quadrícules, però no dona gaire llibertat a l'hora de modificar gaires paràmetres, ja que només permet canviar els punts d'inici i destí i les caselles per les quals es pot passar. També explica el funcionament dels algorismes, A* particularment, i de l'heurístic que aquest fa servir. Per últim, les explicacions inclouen fragments de la implementació en Javascript dels algorismes de cerca.

A la secció *Amit's A* Pages* [9], per altra banda, fa una explicació molt extensa sobre aquest algorisme, els heurístics que pot utilitzar, la seva implementació i algunes variants de l'algorisme, així com també d'altres temes relacionats. Es tracta d'una font força completa sobre el tema. També ofereix alguns enllaços a altres llocs web sobre el mateix tema, i algunes d'elles ofereixen *demos* interactives d'aquests algorismes en funcionament.

- **Pathfinding.js**

PathFinding.js [10] és una aplicació web de *pathfinding* que permet realitzar execucions de diversos algorismes de cerca de camins sobre una graella quadrada. Permet canviar els nodes origen i destí, així com quines caselles es poden travessar i l'algorisme que es vol fer servir per la cerca, d'entre vuit diferents que inclouen BFS, Dijkstra i A*. Pels algorismes que fan servir un heurístic, permet escollir quin es vol utilitzar. També permet determinar si les cel·les estan connectades només amb les dels quatre costats o bé amb també amb les de les diagonals. És una aplicació de mostra de *pathfinding* bastant completa, que permet jugar amb bastants paràmetres i veure el funcionament dels algorismes.

Els dos casos que s'han esmentat són força complets en els seus respectius camps, un en l'explicació dels algorismes i l'altre com a *demo* de funcionament. En certa manera, es complementen l'una a l'altra. Les dues poden ser una bona eina docent (i, de fet, la de *Red Blob Games* es fa servir a teoria de VJ, en parlar d'A*) però no compleixen les característiques concretes que necessita el laboratori de l'assignatura. Per començar, cap d'elles està escrita en C++ ni fa servir OpenGL, que és el que es fa servir pels projectes de VJ. És cert que el codi, encara que estigui en Javascript, pot servir per exemplificar la implementació dels algorismes, però és més convenient que estigui en el llenguatge que es fa servir al laboratori. A més a més, com s'ha mencionat, és imprescindible que el codi sigui completament compatible i integrable amb els exemples i l'altre codi que s'entrega als alumnes (raó per la qual, entre altres, s'ha pres aquest codi com a punt de partida). Això només es pot aconseguir desenvolupant directament el codi com a ampliació de la base de funcionalitats ja disponible, perquè sigui completament compatible i segueixi el mateix estil de programació.

1.3. Abast

En aquest apartat es descriurà tot el que té a veure amb l'abast del projecte. Es definiran els objectius principals del projecte i els requeriments d'aquests objectius.

1.3.1. Objectius

Com s'ha manifestat al principi, l'objectiu principal consisteix en desenvolupar una aplicació de *pathfinding* que es puguin utilitzar al laboratori en l'assignatura VJ. Aquest objectiu general s'ha dividit en dos per ser més concrets.

- El primer objectiu és desenvolupar un codi font que conformi una eina que implementi la funcionalitat de cerca de camins en una graella de cel·les quadrades. A més a més, el codi ha de permetre ser integrat en un dels projectes de l'assignatura amb la fi que aquest pugui utilitzar les seves funcions.
- El segon objectiu consisteix en tenir el codi desenvolupat integrat en una aplicació que el mostri en funcionament. Aquesta aplicació ha de permetre modificar certs paràmetres que tinguin un efecte en el resultat que mostra. Per exemple, alguns dels paràmetres a modificar són el punt d'inici i destí o l'algorisme utilitzat per dur a terme la cerca.

1.3.2. Requeriments

Per tal de considerar que el projecte ha aconseguit el que es proposava i assolit els objectius correctament, s'ha decidit que no només cal complir el que aquests proposen, sinó que també és necessari complir aquest seguit de requeriments:

- El codi desenvolupat ha de ser fàcil d'entendre. En particular, cal evitar programar de forma enrevesada i/o obtusa. El motiu és que els estudiants han de ser capaços de treballar-hi (i per tant entendre'l) amb facilitat.
- En línia amb l'anterior, el codi ha de contenir comentaris que en facilitin la comprensió, així com variables i funcions amb noms que tinguin sentit en el context en què s'estan fent servir i en la funció que tenen.
- Cal que el codi desenvolupat sigui raonablement eficient, però sense que això comprometi la facilitat de comprensió del mateix. Cal recordar que el que es desenvolupi té en part un objectiu didàctic. La llegibilitat i claredat del codi és, per tant, prioritària a l'eficiència.
- El codi desenvolupat ha d'adaptar-se a les eines software que l'assignatura ja té a disposició dels estudiants.
- Les eines de desenvolupament han de les mateixes que les utilitzades als laboratoris de l'assignatura, a fi de poder assegurar la compatibilitat.

2. DESENVOLUPAMENT

Un cop contextualitzat el projecte i la disciplina que tracta, en aquesta secció es tractarà tot allò relatiu al desenvolupament del projecte, és a dir, tot allò relacionat amb les eines utilitzades, els algorismes implementats i la implementació en si.

2.1. Eines utilitzades

En aquest primer apartat es llistaran i definiran les eines que s'han utilitzat en el desenvolupament del projecte i el seu codi font, així com també les diferents biblioteques que s'han fet servir en el mateix.

- **Ordinadors**

La totalitat del codi font del projecte s'ha desenvolupat en dos ordinadors personals amb el sistema operatiu Windows 10. Més concretament, s'ha fet servir un ordinador de sobretaula pel desenvolupament principal i un ordinador portàtil a les reunions de seguiment. S'ha decidit treballar amb Windows perquè és el sistema en què es desenvolupen pràcticament tots els projectes de VJ.

- **Microsoft Visual Studio**

Microsoft Visual Studio [11] és un entorn de desenvolupament integrat (*IDE*) desenvolupat per Microsoft. És un *IDE* àmpliament utilitzat arreu del món i permet el desenvolupament en molts llenguatges de programació, entre els quals figura C++, el llenguatge que s'ha utilitzat en el projecte. Tot el codi font del projecte s'ha desenvolupat fent servir aquest entorn, concretament la versió *Community* del 2017. El motiu de l'elecció de Visual Studio ha estat que es fa servir al laboratori de l'assignatura i que és el que el desenvolupador sap manejar millor (principalment per haver cursat l'assignatura). A més a més, el projecte amb el codi font que s'ha pres com a punt de partida està desenvolupat amb aquesta eina, i fer-la servir ha permès poder posar-s'hi a treballar directament al damunt i assegurar que no hi hagués problemes de compatibilitat.

- **OpenGL**

Open Graphics Library (OpenGL) [12] és una interfície de programació d'aplicacions (*API*) per representar gràfics 2D i 3D en pantalla i desenvolupar aplicacions que en requereixin. Es pot programar en diversos llenguatges (entre ells C++) i és multiplataforma.

OpenGL consisteix bàsicament en un seguit de funcions que permeten interactuar amb la unitat de processament gràfic (*GPU*) per poder dibuixar escenes en dues i tres dimensions. Aquestes escenes es composen amb models formats per punts, línies i/o triangles definits amb les seves coordenades en l'espai tridimensional. Mitjançant *shaders* (un tipus de programa que s'executa a la *GPU*) es poden realitzar diverses operacions sobre aquests models per definir el punt de vista de l'escena, el color que tindran els models en pantalla, entre altres.

OpenGL (concretament la versió 3.30) s'ha fet servir per desenvolupar tota la part gràfica del projecte i per representar tots els elements gràfics en pantalla, com cada una de les caselles de la graella on es realitza el *pathfinding*, el personatge que hi circula i la interfície gràfica de l'aplicació.

- **FreeGLUT**

Free OpenGL Utility Toolkit (FreeGLUT) és una biblioteca *open-source* i portable per programes que utilitzin OpenGL [13]. S'encarrega de les tasques relacionades amb el sistema operatiu, com la creació de finestres i contexts d'OpenGL o la gestió de les entrades de teclat i ratolí. També s'encarrega de la gestió del bucle principal de l'aplicació, que inclou la crida cada cert temps a les funcions per mostrar la imatge, per actualitzar l'estat de l'aplicació tenint en compte els events de teclat i/o ratolí, etc.

Com la resta de biblioteques que es mencionen, el motiu principal d'utilitzar-la ha estat que és la que es fa servir als projectes de VJ, ja que és senzilla i fàcil d'utilitzar i entendre.

- **GLEW**

OpenGL Extension Wrangler Library (GLEW) [14] és una biblioteca multiplataforma de C++ que s'encarrega de detectar quines extensions d'OpenGL estan disponibles pel sistema en qüestió i de carregar-les de forma automàtica.

- **SOIL**

Simple OpenGL Image Loader (SOIL) és una biblioteca multiplataforma molt petita escrita en C que permet carregar imatges (textures) a OpenGL [15]. És compatible amb uns quants dels formats d'imatge més comuns i útils, com PNG o JPEG.

És una biblioteca tan petita que únicament permet carregar imatges a OpenGL o guardar-les com a fitxers. No obstant això, compleix amb totes les funcionalitats que el projecte necessita pel que fa a l'ús d'imatges i ha estat més que suficient per aquest projecte.

- **GLM**

OpenGL Mathematics (GLM) [16] és una biblioteca multiplataforma formada només per capçaleres en C++. Permet definir diversos tipus de dades com vectors o matrius i realitzar operacions amb els mateixos. També posseeix funcions específiques per definir de forma senzilla càmeres (punts de vista d'una escena 3D) en forma de matrius. Segueix el mateix sistema de noms que fa servir GLSL (el llenguatge de programació de *shaders* d'OpenGL) i és totalment compatible amb el mateix.

Se sol utilitzar per realitzar les operacions amb matrius i/o vectors necessàries per fer-les servir a OpenGL. En aquest projecte s'ha fet servir per això i per altres funcionalitats que requerien treballar amb vectors de nombres reals i/o enters.

- **Dear ImGui**

Dear ImGui és una biblioteca per C++ que permet crear interfícies gràfiques d'usuari per OpenGL [17]. És una biblioteca força ràpida, portable i sense dependències externes formada per uns quants fitxers de codi font i capçaleres que es poden incloure i compilar directament. Es tracta d'una biblioteca *open-source* creada per Omar Cornut que es troba disponible al servei d'allotjament de projectes GitHub.

Funciona creant els models que representen la interfície i enviant-los a OpenGL, perquè després l'aplicació els mostri en pantalla quan ho consideri oportú. La biblioteca també s'encarrega de la interacció de l'usuari amb la interfície. Només requereix que l'aplicació li transmeti les entrades de teclat i/o ratolí i li indiqui quines accions ha de dur a terme quan l'usuari interacciona amb cada un dels elements de la interfície.

S'ha fet servir aquesta biblioteca perquè és fàcil d'utilitzar i programar, perquè treballa directament sobre OpenGL i perquè disposa de molts exemples de funcionament amb biblioteques de l'estil de FreeGLUT (inclòs aquest), que han facilitat força la integració de Dear ImGui amb el projecte.

2.2. Algorismes

En aquest projecte s'han hagut d'afrontar un seguit de problemes de computació relacionats amb la cerca de camins, pels quals s'han hagut d'utilitzar alguns dels algorismes que s'han citat en definir el concepte de *pathfinding*. En aquest apartat es llisten i defineixen els principals algorismes utilitzats en el codi font del projecte per desenvolupar la funcionalitat de cerca de camins.

2.2.1. Algorisme de Bresenham

Tot i no ser un algorisme orientat al *pathfinding* ni al recorregut de grafs, l'algorisme de Bresenham s'ha utilitzat degut a la necessitat d'implementar l'algorisme de traçat de contorns en una graella quadriculada. Més endavant s'expliquen els motius que han portat a utilitzar-lo.

L'algorisme de Bresenham és un algorisme per dibuixar línies rectes en graelles quadriculades de dues dimensions, partint de les coordenades d'inici i fi de la línia que es vol dibuixar [18]. Donades aquestes coordenades, l'algorisme determina quines caselles o punts dins la graella s'han de pintar per generar una imatge que aproximi la d'una línia recta que uneixi els dos punts. Se sol utilitzar per dibuixar línies en imatges de mapa de bits, per exemple en un monitor d'ordinador, la imatge del qual es pot considerar formada per una graella de píxels.

L'avantatge d'aquest algorisme respecte d'altres de traçat de línies és que únicament necessita utilitzar enters. A més a més, només requereix realitzar sumes, restes i

desplaçaments de bits (per multiplicar per 2) sobre aquests nombres, operacions que en la majoria d'arquitectures de computador resulten molt ràpides d'executar. És per això que l'algorisme és molt ràpid i simple, i degut a aquestes dues qualitats sol implementar-se directament sobre el *hardware* de les *GPU*s actuals.

Aquest algorisme s'ha utilitzat juntament amb el de traçat de contorns per aconseguir que les línies que traçava aquest algorisme fossin rectes. El traçat de contorns consisteix en desplaçar-se en línia recta cap a l'objectiu i esquivar els obstacles un cop es col·lisiona amb ells. En una graella quadriculada, però, descriure una trajectòria que aproximi la d'una recta no és una qüestió trivial, per això existeixen algorismes com el de Bresenham. És per això que la part de desplaçar-se en línia recta cap a l'objectiu s'ha implementat fent servir aquest algorisme, marcant com a punt d'inici la posició actual del personatge (el que recorre els camins trobats) i com a fi el punt destí de la cerca de camins.

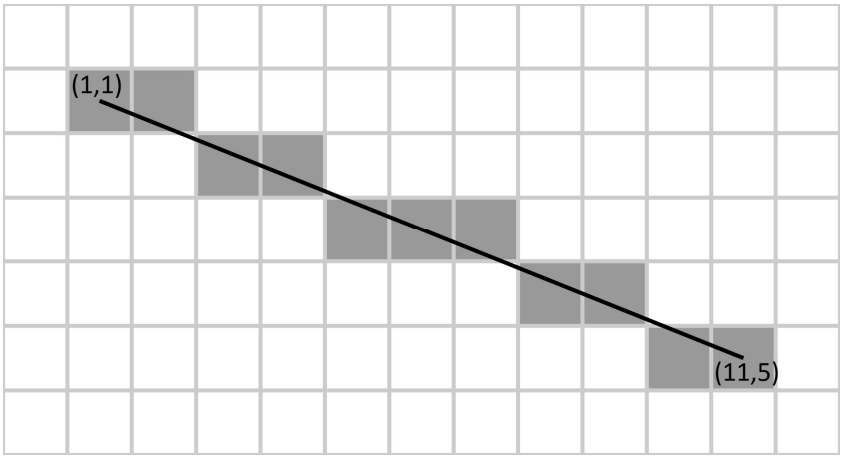

Figura 5: Línia resultant de l'algorisme de Bresenham, del punt (1,1) al (11,5).

L'algorisme funciona per traçar rectes en les quals l'increment en la coordenada *x* del punt inicial al final és positiu i superior a l'increment de la coordenada y (que també ha de ser positiu). Les rectes que no compleixin aquestes característiques primer s'han de tractar, intercanviant el punt inicial pel final i/o intercanviant la coordenada *x* per la *y*, tractaments que s'han de desfer a l'hora de dibuixar els punts.

El funcionament de l'algorisme es basa en un bucle que va incrementant el valor de *x* en 1 (partint de la *x* del punt inicial) i que, per cada iteració, calcula si el valor de *y* (partint de la inicial) ha d'augmentar o no en aquella iteració en concret. Això ho fa fent un seguiment de l'error en el pendent de la recta representada des de l'últim increment de *y*.

2.2.2. Algorisme de traçat de contorns

L'algorisme de traçat de contorns és un algorisme de cerca de camins simple i de cost computacional baix que permet trobar un camí entre dos punts analitzant únicament els punts adjacents al camí que segueix. El seu funcionament és simple i es compon de dues etapes que es van alternant fins a arribar a l'objectiu. La primera etapa consisteix en

desplaçar-se en línia recta des de la posició inicial cap a l'objectiu. Aquest moviment es manté fins que es col·lisiona amb algun element de l'espai, o el que és el mateix, s'intenta desplaçar a una posició marcada com a no vàlida. Un cop es produeix la col·lisió, es canvia a la segona etapa, que consisteix en recórrer el contorn de l'obstacle amb el que s'ha col·lisionat fins a superar-lo. Un cop superat, es torna altre cop a la primera etapa, a moure's en línia recta fins a col·lisionar amb un obstacle. Si existeix algun camí des del punt inicial fins a l'objectiu, seguint aquests passos eventualment l'algorisme troba un d'aquests possibles camins.

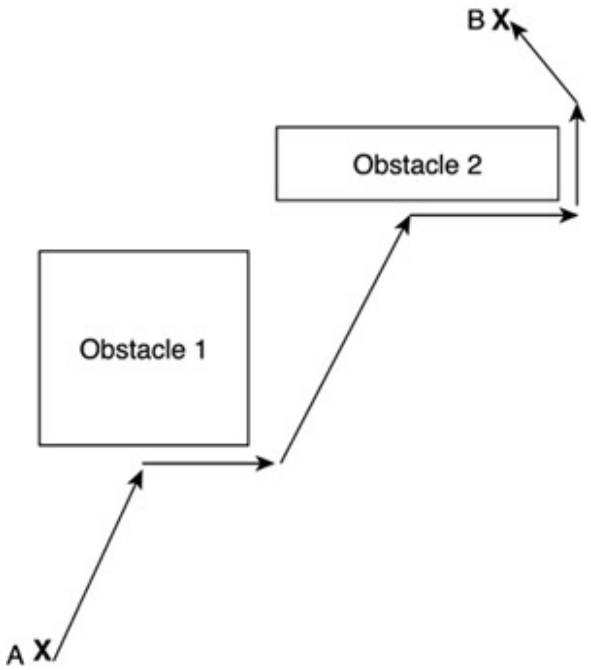

Figura 6: Exemple de trajectòria resultant del traçat de contorns

D'entre tots els camins existents que uneixin el punt inicial i l'objectiu, l'algorisme de traçat de contorns molt rarament aconsegueix trobar el més curt i, de fet, el més habitual és que trobi un camí força més llarg que el mínim. El motiu és que únicament considera els punts adjacents al camí que ha trobat fins ara i no torna enrere en cap moment. Això fa que acostumi a donar molta volta recorrent el contorn dels obstacles i que en encaminar-se cap a una direcció dolenta amb, per exemple, obstacles molt grans resseguint el contorn dels quals un s'allunya de l'objectiu, no sigui capaç de tornar enrere i anar per una altra direcció. Per implementar aquest algorisme en el projecte, doncs, ha calgut ser capaç de resoldre els següents tres problemes:

(1) Donat un punt d'inici i un de fi, traçar una línia recta que uneixi aquests dos punts.

En una graella quadriculada, aquest problema es pot resoldre amb l'algorisme de Bresenham, com s'ha explicat abans, per aproximar el recorregut d'una recta.

(2) Detectar quan es produeix una col·lisió amb un element de l'entorn i posar-se a resseguir el contorn de l'element amb què s'ha col·lisionat.

La detecció de la col·lisió es pot aconseguir comprovant, en cada iteració de l'algorisme de Bresenham, si el punt trobat que aproxima la recta és vàlid o no. En cas que no ho sigui, cal començar el traçat del contorn. El traçat es pot aconseguir escollint un dels dos sentits de gir, horari o antihorari, i desplaçar-se en aquella direcció mantenint sempre una casella no vàlida (que formarà part del contorn de l'obstacle) a la dreta (si el moviment és horari) o a l'esquerra (si és antihorari).

(3) Mentre s'està resseguint el contorn, detectar en quin moment es pot considerar que l'obstacle s'ha superat.

Aquesta part és la que suposa una problemàtica major, ja que no hi ha manera de saber, comprovant només les caselles adjacents, si s'ha superat l'obstacle completament o no. Una primera aproximació pot ser calcular, abans de continuar traçant el contorn de l'objecte, si la cel·la més propera a l'objectiu és vàlida o no, i en cas que ho sigui, aturar el traçat del contorn i prosseguir amb el moviment en línia recta. Aquesta aproximació pot funcionar en alguns casos, però pot donar lloc a cicles infinits en els quals en detectar una casella lliure i moure's en línia recta, es torni al punt on s'ha col·lisionat al començament, donant lloc a repetir els mateixos moviments.

Es poden evitar algunes d'aquestes situacions comprovant si la cel·la més propera a l'objectiu és la cel·la de la que procedim, per evitar tornar enrere pel mateix camí pel que s'ha vingut. Tot i així, hi ha situacions, particularment amb obstacles còncaus, en els quals és inevitable caure en un cicle d'aquest estil.

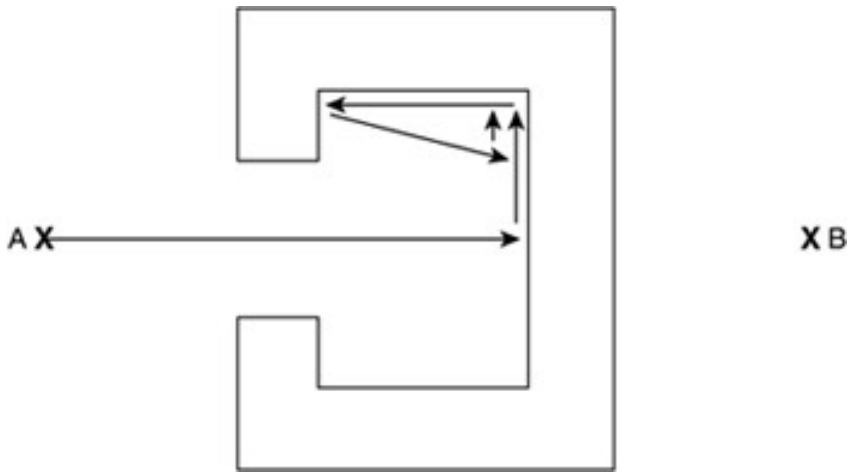

Figura 7: Trajectòria amb traçat de contorns que cau en un cicle

La manera que s'ha trobat per evitar aquest problema consisteix en considerar si la cel·la més propera a l'objectiu és vàlida únicament si aquesta es troba més a prop de l'objectiu que la cel·la en la qual s'ha començat a traçar el contorn. D'aquesta manera s'assegura que, passi el que passi, el traçat del contorn continua fins que s'aconsegueix reduir la distància que queda per arribar a l'objectiu. Així, entre cada canvi entre les etapes de línia recta i traçat del contorn sempre es redueix la distància restant. Fent això, si existeix un camí cap a l'objectiu, l'algorisme l'acabarà trobant eventualment. En cas que no existeixi, però,

l'algorisme es quedarà donant voltes a un obstacle infranquejable de forma indefinida. Per evitar aquesta situació, l'algorisme acabarà si el camí trobat fins el moment supera una certa mida (si s'ha recorregut un nombre de cel·les superior a cert nombre), a partir de la qual es considerarà que no existeix cap camí fins a l'objectiu.

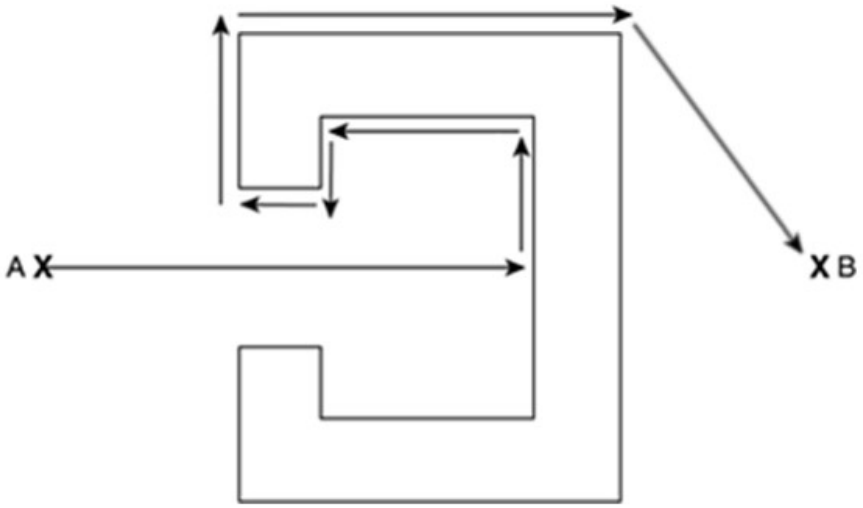

Figura 8: Trajectòria amb traçat de contorns que evita el cicle

2.2.3. A*

L'algorisme A* (pronunciat "A estrella") és un algorisme de recorregut de grafs que (en funció de l'heurístic que utilitza) permet trobar el camí mínim que uneix dos nodes en un graf. Es considera com una variant o optimització de l'algorisme de Dijkstra aplicat a trobar el camí entre dos nodes del graf, ja que dirigeix la cerca cap a l'objectiu.

L'algorisme de Dijkstra, realitza una cerca per tot el graf considerant tots els nodes per igual (pel que fa a arribar a l'objectiu) fins que eventualment topa amb el node buscat. Aquest procés es pot veure, de manera metafòrica, com una "inundació" o com l'avenç d'un front d'ona a través dels nodes del graf, partint des del punt inicial fins a trobar-se amb el destí, avançant més o menys de pressa entre els nodes en funció del cost de les arestes que els uneixen (més de pressa com més baix sigui aquest cost, sense arribar a ser negatiu).

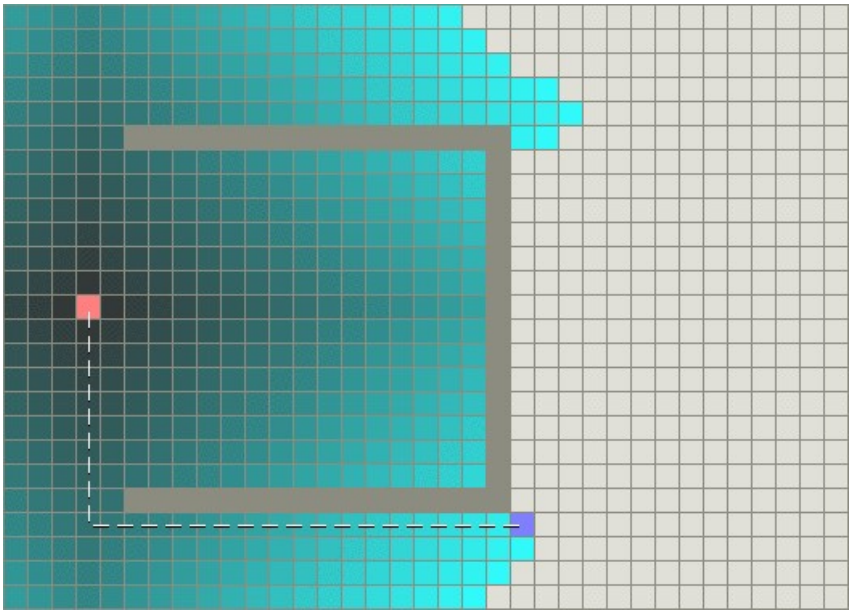

Figura 9: Execució de l'algorisme de Dijkstra. Els nodes que no són beix són els nodes explorats.

La idea darrere l'algorisme A* és fer servir aquest mateix procediment de l'algorisme de Dijkstra però dirigir l'exploració de nodes cap als més prometedors, de manera que, per cada node, tindrà en compte el cost del camí trobat fins al moment i l'estimació del cost del camí restant per arribar a l'objectiu. Més formalment, A* prioritzarà la cerca pels camins que minimitzin la següent funció d'avaluació:

$$f(n) = g(n) + h(n)$$

en la qual, per un node concret n, $g(n)$ és el cost del camí mínim des del node inicial fins a n i $h(n)$ és el valor de la funció heurística, que estima el cost del camí restant que queda fins al destí. De forma més planera, el raonament darrere d'això és que, donat un node qualsevol, el cost del camí mínim de l'origen al destí passant pel mateix serà el cost del camí mínim trobat fins al moment més el cost del camí que queda per arribar al destí. Amb això en ment, l'algorisme prioritza la cerca a través dels nodes pels quals el cost aparent del camí mínim que els travessa és menor.

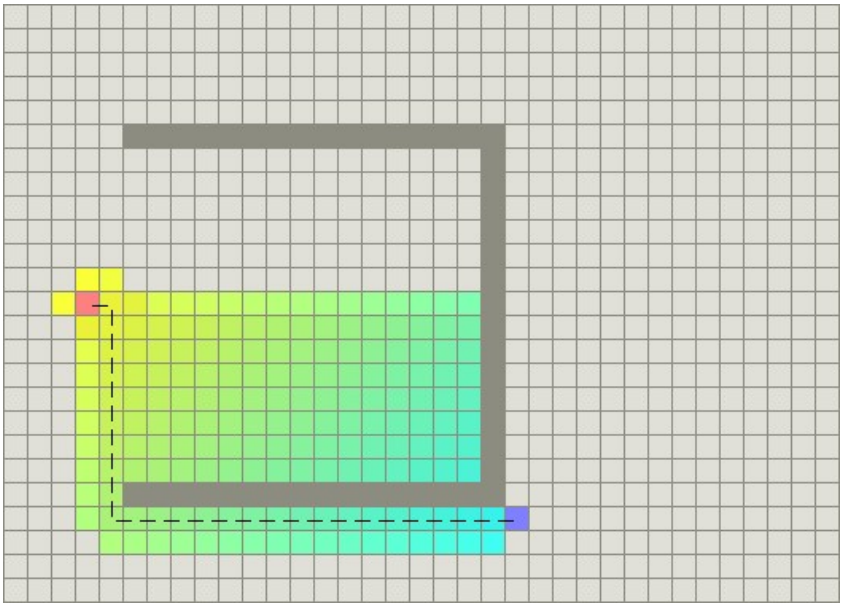

Figura 10: Execució d'A*. Comparat amb Dijkstra, s'aprecia el menor nombre de nodes explorats.

L'algorisme, doncs, necessitarà alguna manera de saber quin/s node/s tenen el valor mínim de f cada vegada que vulgui explorar un node nou. Per tal d'accelerar aquest procés se sol fer servir una estructura de dades anomenada cua de prioritat. Aquest tipus de cua té la propietat que el seu primer element és sempre el més gran o el més petit (especificat en el moment de crear-la) de tots els que conté, a canvi que les insercions i extraccions d'elements de la cua tinguin cost logarítmic (sobre la mida de la cua). El que es fa, doncs, és anar inserint a la cua els nodes amb el seu valor de f a mesura que l'algorisme els va descobrint. La cua s'encarrega de que cada vegada que es necessita explorar un node nou es pugui saber quin té el valor de f més petit.

Vist això, és clar que el sistema que faci servir la funció heurística per estimar el cost restant tindrà molta importància en el resultat de la cerca, ja que determinarà quins nodes s'exploren i quins no. A més, si l'heurístic és admissible (mai sobreestima el cost del camí restant) A* sempre trobarà el camí mínim, si aquest existeix. L'heurístic, però, no només influeix en el resultat de la cerca, sinó també en el cost de la mateixa. Com més petit sigui el valor de la funció heurística, més nodes considerarà A* en la cerca. De fet, amb una funció heurística que sempre retorni 0, A* es comportarà igual que Dijkstra, ja que passarà a considerar únicament $g(n)$ a l'hora de decidir per quins nodes fa l'exploració, que és justament el que fa aquest últim.

En l'aplicació desenvolupada en el projecte es pot escollir quin heurístic es vol utilitzar en la cerca, d'entre uns quants de diferents que, en funció de la configuració actual de la connectivitat de la graella (si es permet el moviment en diagonal o no), poden ser admissibles o no. Els heurístics que hi ha disponibles són tres, cada un d'ells corresponent a una distància diferent.

El primer heurístic disponible és l'anomenada distància *Manhattan*, que és la suma de la distància amb l'objectiu en els dos eixos de coordenades, *x* i *y*. Aquest heurístic és

admissible només quan la connectivitat de la quadrícula és 4 (no es permet el moviment en diagonal), ja que en aquestes circumstàncies el camí mínim entre una posició i una altra passa per recórrer tota la distància en *x* i tota la distància en *y*. Quan es permet el moviment diagonal aquest heurístic deixa de ser admissible.

El segon heurístic disponible és la distància *Chebyshev*, que és el valor màxim entre les distàncies amb l'objectiu en els dos eixos de coordenades. Aquest heurístic sempre és admissible, però sol subestimar el cost del camí restant, particularment quan no es permeten les diagonals.

El tercer heurístic és la distància euclídea, que es calcula com l'arrel quadrada de la suma dels quadrats de les distàncies amb l'objectiu en els dos eixos de coordenades. Aquest heurístic, com l'anterior, sempre és admissible. Igual que *Chebyshev*, sol subestimar el cost del camí restant, però en menor mesura.

L'aplicació també inclou un quart heurístic, que sempre retorna 0, que permet fer funcionar A* igual que Dijkstra per poder comparar resultats.

2.3. Implementació

El codi font d'aquest projecte s'ha desenvolupat seguint el paradigma de la programació orientada a objectes. Per tant, el codi s'ha dividit en classes, cada una amb la seva tasca associada i relació amb les altres, dividida en dos fitxers, capçalera i implementació. Com s'ha comentat anteriorment, el projecte no ha partit de zero, sinó que ha utilitzat com a base el codi font disponible de l'assignatura VJ. En aquesta secció es defineix quin ha estat el punt de partida del projecte, l'estructura del codi i de les classes del projecte i la implementació de cada una d'elles.

2.3.1. Punt de partida

El projecte que s'ha fet servir com a punt d'inici pel desenvolupament contenia vuit classes i el fitxer *main* que iniciava l'execució. El projecte contenia l'estructura bàsica de programació d'un videojoc (ja que està pensat com a exemple per programar-ne un), estructura que s'ha mantingut en el desenvolupament del projecte.

De les vuit classes de partida, només s'han mantingut intactes les relacionades amb el funcionament més bàsic d'OpenGL, com és definir *shaders* i textures i enviar-los a OpenGL o mostrar *sprites* en pantalla. Concretament, les classes que s'han mantingut intactes són:

- ***Shader*:** Permet carregar un *shader* a OpenGL partint d'un fitxer font o d'un *string* i obtenir el seu identificador per poder-lo utilitzar.
- ***ShaderProgram*:** Utilitza la classe *Shader* per unir un *vertex* i un *fragment shader* (els dos tipus de *shader* necessaris) i enllaçar els atributs d'entrada i sortida dels mateixos amb els noms amb què se'ls fa referència al codi dels *shaders*.

- ***Texture*:** Permet carregar textures (imatges) des d'un fitxer de disc (.png, .jpeg...), enviar-les a OpenGL i obtenir l'identificador associat per poder-les utilitzar.
- ***Sprite*:** Permet crear el model d'un quadrat fent servir dos triangles, enviar-lo a OpenGL i mostrar-lo en pantalla aplicant-li una textura a damunt. A efectes pràctics permet mostrar una imatge (la textura) amb una certa mida i en una posició concreta de la pantalla. També permet animar la imatge, però aquesta característica no s'ha fet servir en el projecte.

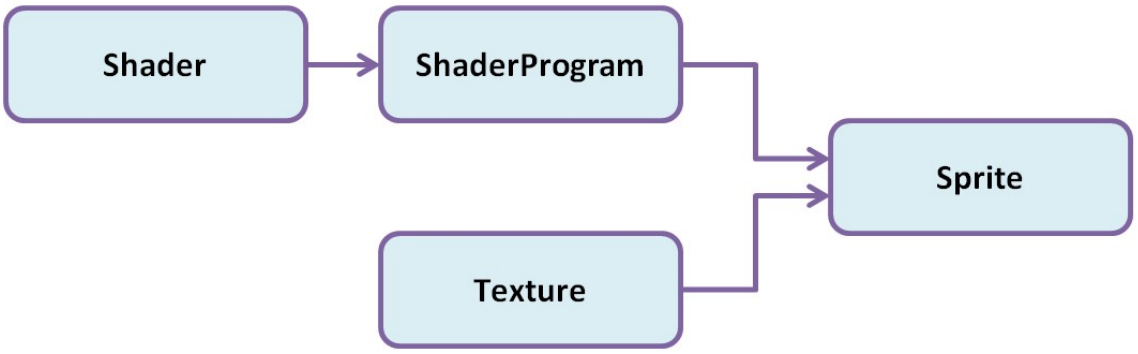

Figura 11: Esquema de les classes no modificades.

La resta de classes inicials, *Game*, *Scene*, *Player*, *Tilemap* i el fitxer *main* s'han modificat gairebé totalment. Es descriurà què fan i què s'ha mantingut del codi original en els següents apartats.

2.3.2. Estructura

El codi font de l'aplicació disposa de 14 classes en total, 4 de les quals són les mencionades a l'apartat anterior. La resta són o bé completament noves o bé només conserven alguns elements (pocs) dels que hi havia a la classe de partida. Aquestes deu classes són:

- ***Game*:** Serveix per representar la totalitat de l'aplicació. Actua com a contenidor de l'escena (un objecte de classe *Scene*) i també guarda les entrades de teclat i ratolí, que la resta de classes poden consultar a través d'ella. Actua també com a enllaç entre la interfície gràfica i l'escena principal, traslladant a la mateixa els canvis que es produeixin a la interfície. El nom de la classe és un remanent del nom original que tenia, el qual s'ha decidit mantenir.
- ***Scene*:** Conté la majoria d'elements principals de l'aplicació, com els *shaders* que s'utilitzen (*ShaderProgram*), la quadrícula on es realitza la cerca (*GridMap*), la representació visual de la mateixa (*TileMap*), el sistema de *pathfinding* (*PathFinder*), el sistema per mostrar els camins trobats (*Path*) i el personatge que els recorre (*Player*). S'encarrega d'inicialitzar tots aquests elements, actualitzar el seu estat quan

és necessari i utilitzar-los quan són necessaris. També gestiona tot allò relacionat amb la part gràfica de l'aplicació, excepte la interfície gràfica.

- ***Player*:** Conté un objecte de la classe *Sprite* que representa el personatge que es desplaça per la quadrícula. S'encarrega de realitzar el moviment del personatge modificant la posició on es mostra en pantalla de forma gradual. Així aconsegueix que el desplaçament del personatge entre les cel·les sigui suau. Com amb *Game*, el nom de la classe és un remanent de l'original, ja que aquesta era la classe que representava el personatge del jugador en el projecte de partida.
- ***TileMap*:** Representa la part visual de la quadrícula. S'encarrega de crear el model de la mateixa, enviar-lo a OpenGL i mostrar cada una de les caselles amb una imatge diferent en funció de si són vàlides (el personatge hi pot anar) o no. Té visibilitat de la totalitat de la cuadrícula per poder actualitzar el que es mostra si hi ha algun canvi.
- ***GridMap*:** Representa la part "física" de la quadrícula. Conté l'estat de totes les cel·les de la mateixa (si són vàlides o no) i disposa de funcions perquè la resta de classes puguin consultar-la o modificar-la.
- ***PathFinder*:** Serveix per realitzar una cerca de camins (un *pathfinding*) sobre la quadrícula de *GridMap*, de la qual té visibilitat. Conté tots els atributs necessaris per realitzar la cerca, com el punt inicial i el final o l'algorisme a utilitzar. Un cop realitza la cerca, guarda el camí que ha trobat perquè una altra classe (*Path*) el mostri en pantalla. Té dues subclasses, cada una de les quals permet realitzar la cerca de camins amb un algorisme diferent definint cada una d'elles el mètode virtual *findPath*.
  - ***ContourTracing*:** Implementa l'algorisme de traçat de contorns i conté tots els atributs que necessita, com la mida màxima del camí permès o el sentit (horari o antihorari) amb què se segueix el contorn dels obstacles.
  - ***A_Star*:** Implementa l'algorisme A*. Conté tots els atributs i estructures de dades que aquest necessita, com la cua de prioritat o les estructures necessàries per recordar quins nodes s'han visitat i quins no.
- ***Path*:** S'encarrega de mostrar els camins trobats. Conté una referència al camí contingut a *PathFinder* i l'utilitza per construir el model del camí i enviar-lo a OpenGL. Representa els camins amb quadrats de diferents colors que se superposen a les caselles de la quadrícula que formen part del camí.
- ***Parameters*:** S'encarrega de llegir els paràmetres d'inici de l'aplicació d'un fitxer de disc. Aquests paràmetres poden ser la mida inicial de la finestra, la mida mínima de la mateixa, la mida de la quadrícula, la ruta en el sistema de fitxers de les imatges que es mostren a les caselles, la velocitat de moviment del personatge, etc. Un cop llegits aquests paràmetres els emmagatzema i la resta de classes poden consultar-los a través d'ella, de forma similar a la classe *Game*.

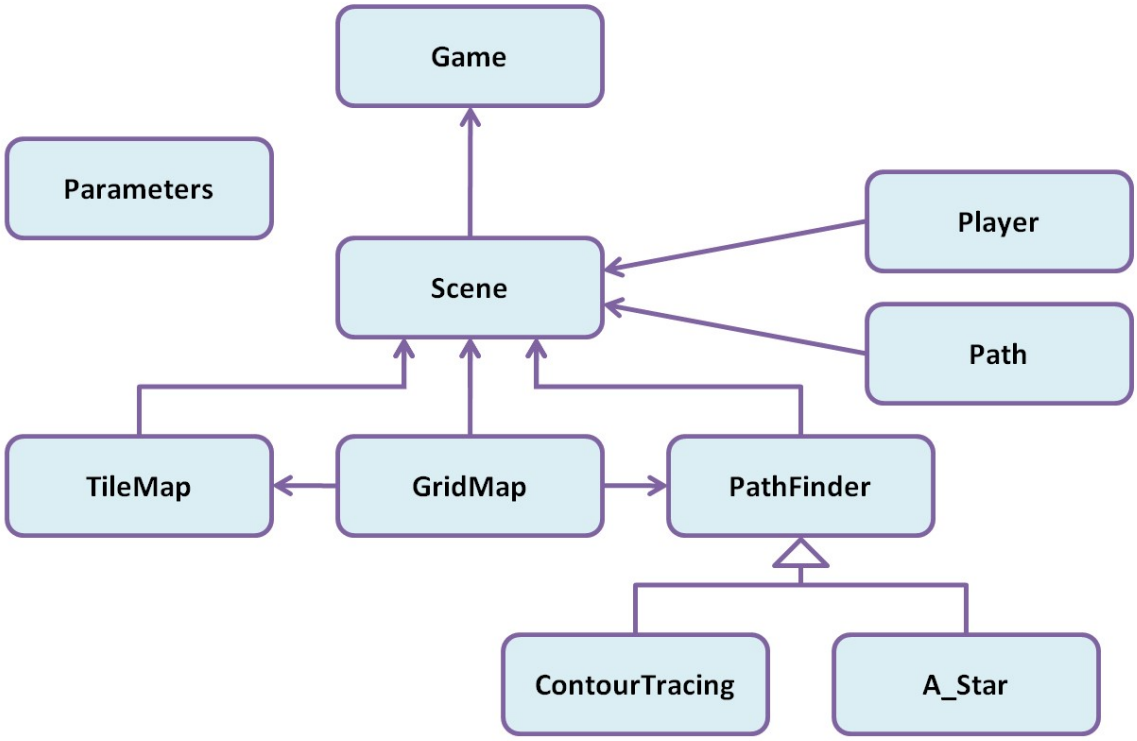

Figura 12: Esquema de classes de l'aplicació desenvolupades pel projecte.

A més d'aquestes 14 classes, el projecte també disposa d'un fitxer *main*, que inicia l'aplicació. Aquest fitxer conté totes les funcions que capturen (i guarden a *Game*) els events de teclat i ratolí i s'encarrega de comunicar a FreeGLUT quina funció ha de cridar quan es produeix cada un d'aquests events. També conté (i comunica a FreeGLUT quines són) les funcions que visualitzen l'aplicació en pantalla, gestionen les mesures a prendre si canvia la mida de la finestra i controlen el bucle d'execució de la mateixa. A més d'això, també s'encarrega d'inicialitzar la interfície gràfica i les instàncies de la classe *Game* i *Parameters*.

El funcionament de l'aplicació, a nivell d'execució, es basa en un bucle infinit controlat per FreeGLUT que va executant periòdicament la funció *idleCallback*, que actualitza l'estat de l'aplicació, i la funció *drawCallBack*, que visualitza l'estat actual de l'aplicació en pantalla. Aquestes dues funcions es criden, en aquest ordre, 60 vegades per segon.

La funció *idleCallback* simplement crida el mètode *update* de *Game*, que al seu torn crida al mètode *update* de *Scene*. Dins de *Scene* és quan es llegeix els events d'entrada i de la interfície gràfica que hi ha hagut i es realitzen les accions necessàries (executar la cerca de camins, donar un camí al personatge perquè el segueixi, modificar alguna casella de la quadrícula...) tenint en compte aquests events. Just després d'això, *Scene* actualitza l'estat de *Player* (que modifica la seva posició si està recorrent un camí) i, si hi ha hagut canvis en la quadrícula, actualitza *TileMap* perquè aquest modifiqui el seu model i s'adeqüi a la quadrícula actual.

La funció *drawCallback* segueix el mateix principi que l'anterior. Primer crida el mètode *render* de *Game*, el qual crida la funció *render* de *Scene*. És aquesta classe la que s'encarrega de cridar el *render* de *TileMap*, *Path* (que només es visualitza si hi ha algun camí a mostrar) i *Player*, sel·leccionant en cada cas el *ShaderProgram* necessari per cada un d'ells i enviant al *shader* tots els paràmetres necessaris. L'ordre en el que es realitza el *render* de cada una d'aquestes classes és exactament l'ordre en què s'han mencionat, ja que és necessari que el camí es dibuixi damunt la quadrícula (perquè aquesta no el tapi) i que el personatge es dibuixi damunt el camí (perquè representa que s'hi està movent pel damunt). Després d'això, la funció *render* de *Game* retorna, i es prossegueix amb la interfície gràfica, que es defineix dins de la funció *UI_function*. Aquesta funció, a més de definir la interfície, també comprova els canvis que hi hagin hagut en l'estat de la mateixa i els transmet a *Game*. La biblioteca utilitzada per la interfície gràfica, Dear ImGui, s'encarrega de crear els models que la representen i enviar-los a OpenGL a mesura que aquesta es va definint. L'últim que es fa és visualitzar la interfície en pantalla.

El bucle infinit que va executant aquestes dues funcions s'inicia en el moment en què *main* crida la funció *glutMainLoop*. En aquest moment el control passa a mans de FreeGLUT, que s'encarrega de cridar les dues funcions mencionades i de capturar els events de teclat i ratolí quan es produeixen.

2.3.3. Classes

En aquest últim apartat es descriurà amb més detall la funció i les tasques que realitza cada classe del projecte. Per cada classe, es descriuran els atributs més rellevants que posseeix i la funció de cada un d'ells, així com també les tasques que duu a terme la classe i alguns dels mètodes que fa servir per realitzar-les.

- **Game**

La classe *Game* representa l'aplicació i conté l'escena i tots els atributs necessaris per guardar l'estat de les entrades de teclat i ratolí de l'usuari. Conté 2 *arrays* de booleans que representen quines tecles estan premudes, 4 booleans que indiquen si els botons dret/esquerre del ratolí estan premuts (o just s'acaben de prémer), i 2 variables enteres que indiquen la posició del cursor del ratolí en coordenades relatives a la finestra.

Aquesta classe s'encarrega de les següents tasques:

- **Retornar la seva pròpia instància**

  Aquesta classe posseeix una funció estàtica anomenada *instance* que conté la declaració (també estàtica) d'una instància de *Game*, que s'instancia el primer cop que es crida la funció. La funció retorna aquesta instància (única en tota l'aplicació) de *Game* cada cop que és cridada, i permet que les altres classes puguin accedir a *Game* sense necessitat de disposar directament de la instància.

- **Emmagatzemar events de teclat i ratolí**

  Amb funcions com *keyReleased*, *mouseMove* o *mousePress*, que són cridades per FreeGLUT cada vegada que es produeix un event extern, es modifiquen els atributs que representen l'estat del teclat o del ratolí.

- **Retornar l'estat del teclat i el ratolí**

  Amb funcions com *getKey*, *clickL*, *getMouseXY*, *Game* s'encarrega de retornar l'estat dels atributs que representen el teclat i el ratolí, per saber si una tecla està premuda o no, si algun dels botons del ratolí ha estat premut i quina posició ocupa el ratolí dins la finestra. Aquestes funcions són cridades per *Scene* a través de la funció *instance* de *Game*.

- **Modificar *Scene***

  Quan es produeix un un canvi d'estat (l'usuari interacciona amb algun element) en la interfície gràfica, aquest s'ha de comunicar a *Scene*, que modificarà els seus paràmetres com pertoqui per reflectir els canvis produïts. Aquesta tasca es duu a terme des de la funció dedicada a la interfície gràfica del fitxer *main*, que detecta els canvis en la *UI* i els comunica a *Game* (a través de la funció *instance)*, que els transmet a *Scene.*

- ***Scene***

Aquesta classe representa l'escena que conté tots els elements principals de l'aplicació. Conté dues instàncies de *ShaderProgram*, una per visualitzar la quadrícula i el personatge, l'altra per visualitzar els camins trobats. També una instància de *GridMap* (la quadrícula), dues de *TileMap* (ja que la quadrícula disposa de dues vistes), dues de *PathFinder* (una de cada subclasse, per cada un dels dos algorismes), una de *Path* i una de *Player*. També conté altres atributs de control (com la mida de la finestra i de la interfície gràfica o quin algorisme s'ha d'utilitzar per les cerques).

Les funcionalitats d'aquesta classe són les següents:

- **Inicialitzar els elements principals de l'aplicació**

Scene és la classe responsable de cridar la funció creadora de totes les classes de les quals posseeix una instància, així com també de passar a aquests objectes tots els paràmetres que necessiten, com per exemple apuntadors a *GridMap* (a les classes que necessitin accedir-hi). També s'encarrega de carregar i compilar els *shaders* per després afegir-los a les instàncies de *ShaderProgram* adients.

- **Gestionar el control de l'usuari sobre l'aplicació**

És necessari comprovar, en cada iteració del bucle principal de l'aplicació, si l'usuari ha interaccionat d'alguna manera amb l'aplicació i actuar en conseqüència. Això es fa dins la

funció *update*. Aquest mètode s'encarrega d'ordenar al personatge (de classe *Player*) que es mogui seguint un camí quan l'usuari ho desitja, de que *PathFinder* executi una cerca quan és necessari i de canviar l'estat de les caselles on l'usuari fa clic.

- **Assegurar-se que els elements visuals es mostren correctament**

L'aplicació s'inicia amb una mida de finestra predefinida, però l'usuari és lliure de modificar-la com desitgi. Si no es realitza un seguiment de la mida actual de la finestra abans de mostrar la imatge, aquesta podria sortir deformada ja que, per defecte, la imatge que es visualitza s'estira o contrau per ocupar la totalitat de la finestra. Cada vegada que l'usuari canvia la mida de la finestra, *Game* notifica a *Scene* de quina és la mida actual. La funció *updateProjection* de *Scene* té en compte aquesta dada i modifica la matriu de projecció (l'atribut *projection* de *Scene*, que determina què es visualitza en pantalla i què no) perquè la quadrícula ocupi el màxim espai possible en pantalla sense deformar-se. Per fer això, primer compara la relació d'aspecte (la relació entre l'amplada i l'alçada) de la finestra amb la de la quadrícula i afegeix espais buits als costats o a la part superior, deixant sempre un espai de mida fixa per visualitzar la interfície gràfica.

- **Calcular en quina casella s'ha fet clic**

La posició del ratolí dins la finestra de l'aplicació es mesura en píxels, amb el punt (0,0) situat a la cantonada superior esquerra de la finestra. Cal disposar d'un sistema que permeti determinar damunt de quina casella està situat el ratolí, partint de la posició en píxels que ocupa dins la finestra. Podríem dir que cal convertir les coordenades de la finestra en coordenades dins la quadrícula. Aquesta tasca la duu a terme la funció *screenToTile*, que té en compte la mida de la quadrícula i dels espais buits al voltant d'aquesta per determinar quin percentatge d'una casella ocupa un sol píxel. Només li cal multiplicar aquest valor per la posició en píxels del ratolí (tenint en compte l'espai buit i el que ocupa la interfície) per saber a quina casella de la quadrícula correspon aquesta posició, arrodonint cap avall el resultat obtingut.

- ***Player***

Aquesta és la classe que representa el personatge que es mou damunt la quadrícula. Disposa d'una instància de *Sprite* que conté el model (un quadrat sobre el qual es mostra una textura) que s'ha de mostrar en pantalla. També disposa d'una textura (una instància de *Texture*) que conté la imatge que es fa servir per representar-lo) i un apuntador al *ShaderProgram* que s'ha d'utilitzar per mostrar-lo en pantalla. També disposa d'atributs que s'utilitzen per controlar el moviment per la quadrícula, com per exemple quantes iteracions del bucle principal ha de tardar per moure's d'una casella a una altra, si està seguint un camí o no i quin és el camí a seguir (això últim a través d'un apuntador al vector de dins la classe *PathFinder*).

Aquesta classe, apart de les funcions bàsiques com visualitzar-se en pantalla, només és responsable d'aquesta tasca:

- **Moure el personatge per un camí**

A través de la funció *travel*, *Scene* ordena a *Player* que es desplaci per la quadrícula seguint el camí que se li ha indicat. Un cop rebuda aquesta ordre, *Player* s'encarrega d'anar modificant la posició on es mostra el personatge de forma gradual perquè aquest recorri el camí a una velocitat determinada. En rebre l'ordre, calcula quantes iteracions del bucle principal necessitarà per desplaçar-se del centre d'una casella al centre d'una altra adjacent, tenint en compte la freqüència a la que s'executa el bucle principal i la velocitat amb la que s'ha de moure. També calcula en quina direcció haurà de realitzar aquest moviment consultant quina és la casella següent dins el camí indicat.

Un cop disposa d'aquestes dades, cada vegada que es crida la funció *update*, desplaça el personatge una certa distància en la direcció adient partint de la posició actual. Va comptant el nombre de vegades que s'ha cridat a *update*, i desplaça el personatge una mica més en funció d'aquest nombre. Quan s'executa *update* tantes vegades com ha calculat que li calen per arribar a la següent casella, situa el personatge en aquesta i calcula de nou les iteracions necessàries per arribar a la següent (amb el mètode *updateFPT*) i la direcció en què es troba la següent casella. És necessari recalcular el nombre d'iteracions per moure's entre caselles perquè la distància entre aquestes pot no ser sempre la mateixa, ja que la distància entre caselles que estan en diagonal és més gran que la de les que estan una al costat de l'altra. En el moment en què detecta que ha arribat a l'última casella del camí, atura el moviment i queda disponible per rebre noves ordres.

- ***TileMap***

La classe *TileMap*, el nom de la qual significa, més o menys, mapa de "rajoles" o "tessel·les" (*tiles*) representa la vista de la classe *GridMap*, és a dir de la quadrícula de l'aplicació. Disposa d'un objecte de *Texture* que conté les imatges que es mostren a cada casella, un apuntador a *GridMap* que utilitza per consultar l'estat de la quadrícula i un apuntador al *ShaderProgram* que fa servir per mostrar la quadrícula en pantalla. També conté informació sobre la mida de la quadrícula i la mida de la imatge d'una casella dins la textura.

Igual que *Player*, s'encarrega d'una sola tasca bàsica, a part de la de visualitzar-se en pantalla. Aquesta tasca és:

- **Construir el model de la quadrícula segons l'estat de la mateixa**

La classe *TileMap* s'encarrega d'accedir a *GridMap* i consultar l'estat de cada una de les caselles que conté. Després fa servir aquest estat per, quan construeix el model que envia a OpenGL, indicar quina imatge s'ha de mostrar en cada una de les caselles, en funció de si són vàlides o no. Això ho fa amb la funció *prepareArrays*, que crea un vector de *floats* en el que va afegint les coordenades dels quadrats (cada un format per dos triangles) que

representen cada una de les caselles, amb informació sobre quin fragment de la imatge de la textura s'ha de mostrar per cada una.

- ***GridMap***

GridMap és la representació "física" de la quadrícula de l'aplicació i únicament conté un vector que guarda l'estat de cada casella, un atribut que indica la mida de la quadrícula (quantes caselles té d'amplada i d'alçada) i una variable booleana que indica si la quadrícula s'ha modificat, variable que *Scene* consulta per saber si ha d'ordenar a *TileMap* que s'actualitzi.

Les tasques que realitza aquesta classe són les següents:

- **Emmagatzemar l'estat de les caselles**

La classe posseeix un vector de variables anomenat *grid*. Cada posició del vector representa una casella de la quadrícula amb una variable *TileType*, una enumeració que permet tres possibles valors: *FLOOR* per les caselles que es poden travessar, *WALL* per les que no i *EMPTY* per les indefinides.

- **Modificar l'estat de les caselles**

L'usuari pot modificar lliurement cada una de les caselles de la quadrícula, alternant el seu estat entre *WALL* i *FLOOR*. La funció *set* permet canviar el *TileType* d'una casella i la funció *swapTileType* canvia la casella indicada de *WALL* a *FLOOR* i viceversa cada vegada que es crida.

- **Retornar l'estat d'una casella**

Per realitzar les cerques és necessari saber per quines caselles es pot passar i per quines no. Amb la funció *get* es pot obtenir el *TileType* d'una casella i amb el mètode *isWalkable* es pot saber si una casella en concret és vàlida o no.

- ***PathFinder***

La funcionalitat principal d'aquesta classe és servir com a classe pare per les dues subclasses que implementen els algorismes de traçat de contorns i A*. Conté tots els atributs que ambdues subclasses necessiten, com un apuntador a *GridMap* per consultar la quadrícula o dades sobre quina és la casella inicial i la destí sobre la qual executar la cerca, quina casella s'està considerant en un moment donat, quin dels dos algorismes s'està fent servir, quin heurístic utilitzar (per A*) o quin és el camí trobat fins el moment.

Les tasques de què s'encarrega són:

- **Declarar els mètodes virtuals**

A la capçalera de la classe, es declaren dos mètodes públics virtuals, *findPath* i *init*. El dos mètodes seran implementats per la subclasse amb l'algorisme de cerca corresponent i les inicialitzacions necessàries de la classe.

- **Calcular la distància entre un punt i l'objectiu**

Per l'algorisme de traçat de contorns és necessari calcular la distància que queda amb l'objectiu en el moment en què es comença a recórrer el contorn d'un obstacle i també cada vegada que s'analitza si l'obstacle s'ha superat o no. D'això se n'encarrega la funció *distToEnd*, que calcula la distància amb l'objectiu en funció de la connectivitat de la quadrícula (ja que la distància és menor si es permet el moviment en diagonal).

- ***ContourTracing***

Aquesta és una de les dues subclasses de *PathFinder*. S'encarrega d'implementar la cerca de camins amb l'algorisme de traçat de contorns. Disposa de tots els atributs de la classe *PathFinder* i n'afegeix alguns de propis. Implementa els mètodes virtuals *findPath* i *init*, que fa servir per realitzar la cerca de camins i inicialitzar alguns atributs.

Les funcions que realitza pel traçat de contorns són:

- **Desplaçar-se en línia recta cap a l'objectiu**

Aquesta funció la realitza amb l'algorisme de Bresenham, que s'ha explicat anteriorment. La funció *moveStraight* s'encarrega d'això. Primer analitza les posicions de la casella origen i de la objectiu per configurar l'entrada de l'algorisme de forma correcta. Després realitza l'execució de l'algorisme, de la manera que s'ha explicat en l'apartat dedicat al mateix, i cada vegada que troba un nou punt de la recta, l'afegeix al camí trobat. En cas de trobar-se amb una casella no vàlida, s'atura en el punt on estava i retorna.

- **Recórrer el contorn d'un obstacle**

Aquesta funció la duu a terme el mètode *traceContour*, que va recorrent les caselles adjacents al contorn de l'obstacle amb un sentit de gir determinat. Abans de començar el recorregut, però, emmagatzema la distància que el separa de l'objectiu. Llavors, mentre recorre el contorn, va comprovant si es compleixen dues condicions: primera, la casella adjacent més propera a l'objectiu està disponible i segona, la distància que la separa de l'objectiu és menor que la que s'ha guardat al principi. Si es compleixen aquestes dues condicions, considera que l'obstacle s'ha superat i la funció retorna.

- **Alternar el traçat de contorn i el moviment en línia recta**

El funcionament de l'algorisme es basa en anar alternant les dues funcions anteriors: anar recte i esquivar els obstacles. Dins la funció *findPath* aquestes dues dues funcions es van

executant de forma alternada dins un bucle que va comprovant si s'ha arribat a l'objectiu i si la mida del camí trobat no supera la màxima permesa.

- **Calcular quina és la casella adjacent més propera a l'objectiu**

Una de les condicions que s'han de complir per considerar que l'obstacle s'ha superat és que la casella adjacent més propera a l'objectiu sigui buida. Per tant, cal alguna manera de calcular, d'entre les caselles adjacents, quina està més aprop de l'objectiu. D'això se n'encarrega el mètode *nextDir*, que donada una casella, calcula quina de les 4 o 8 adjacents està més a prop del destí.

- ***A_Star***

L'altra subclasse de *PathFinder*, que implementa l'algorisme A*. De la mateixa manera que l'anterior, també disposa de tots els atributs de *PathFinder* i n'afegeix de propis. Disposa d'una cua de prioritat (*tileQueue*) a la que es poden inserir caselles amb una variable de tipus *float* associada que sempre manté en la primera posició la casella amb la variable *float* més petita.

També disposa d'uns quants vectors que emmagatzemen informació addicional sobre cada casella del mapa. Té un vector de booleans anomenat *pending* que li permet saber en tot moment si una casella és present a la cua (cosa que vol dir que està pendent d'analitzar) i un altre anomenat *visited* que li indica si una casella ja ha estat visitada. Disposa també d'un vector d'enters (*from*) que, per cada casella, indica des de quina direcció (des de quina casella, per tant) prové el camí mínim que va des de l'inici fins a la casella en concret. Aquesta informació la fa servir per, un cop acabada la cerca, anar seguint aquestes direccions des de la casella destí per poder reconstruir el camí mínim per arribar-hi des de la casella origen. Per últim, disposa d'un vector de variables *float*, anomenat *gCost*, que per cada casella indica el cost del camí mínim trobat fins al moment. Aquesta informació la fa servir per saber en tot moment la llargada del camí mínim que passa per una casella per, en descobrir-la, sumar aquest valor amb el de l'heurístic i inserir-lo a la cua aparellat amb la casella a la que correspon.

Les funcions que realitza per implementar A* són:

- **Marcar una casella com a pendent**

Cada vegada que A* descobreix una casella, ha de calcular el cost del camí mínim per arribar-hi, afegir-li el valor de l'heurístic i inserir aquest valor amb la casella associada a la cua de prioritat. Això es realitza amb la funció *setPending*, que afegeix una casella amb un valor determinat a la cua i marca la casella com a pendent en el vector *pending*.

- **Obtenir la casella amb el valor *f* més baix**

Aquesta acció s'ha de realitzar cada vegada que s'ha de decidir quina és la següent casella que s'explora. La funció *getCheapestTile* extreu un element de la cua de prioritat i obté la

casella associada al mateix. Llavors, marca la casella com a visitada en el vector *visited* i la marca com a no pendent a *pending*.

- **Calcular el valor de l'heurístic d'una casella**

Aquesta tasca és simple. Donada una casella, ha de ser capaç de calcular el valor de l'heurístic de la mateixa. A la funció *calcHeuristic* comprova quin és l'heurístic sel·leccionat en el moment i el calcula tenint això en compte.

- **Executar la cerca**

D'aquesta tasca se n'encarrega la funció *findPath*. La funció esborra el camí trobat anteriorment i afegeix la casella inicial a la cua. Després, executa la cerca d'A* partint de la casella inicial i continua mentre queden elements a la cua de prioritat o fins que troba el destí.

- **Reconstruir el camí mínim**

Un cop la cerca ha conclòs, el resultat és un seguit de caselles que apunten a la direcció d'on prové el camí mínim que les travessa. És necessari, doncs, començar des de la casella destí i anar seguint aquestes direccions fins arribar a la casella inicial. Això es realitza amb la funció *appendRemaining* que, donada una casella (la final) recorre de forma recursiva les caselles d'on prové el camí i s'atura quan es troba amb la casella inicial, moment en el qual reconstrueix el camí afegint cada casella al vector que el representa.

- ***Path***

Aquesta classe representa la vista del camí que s'emmagatzema a *PathFinder*. Disposa d'un apuntador al *ShaderProgram* que utilitza per visualitzar-se i d'un altre apuntador al vector guardat a *PathFinder*.

Funciona de manera similar a *TileMap*, és a dir, s'encarrega d'una sola tasca, a part de la de visualitzar-se en pantalla:

- **Construir el model del camí segons l'estat d'aquest**

Path consulta el vector de *PathFinder* a través de l'apuntador i fa servir aquesta informació per construir el model del camí. La funció *prepareArrays* crea un model format per quadrats que va situant damunt de les caselles corresponents a la quadrícula. En funció de si la casella és la inicial, la final, o entre aquestes dues, li assigna un color diferent (groc, vermell i verd, respectivament).

- ***Parameters***

La funció principal d'aquesta classe és servir de contenidor per emmagatzemar certs paràmetres de l'aplicació. Conté diversos atributs que serveixen per representar cada un d'aquests paràmetres.

Les tasques que duu a terme són:

- **Retornar la seva pròpia instància**

Exactament igual que Game, aquesta classe posseeix una funció estàtica que conté la declaració (també estàtica) d'una instància de la classe. La funció retorna aquesta instància cop que és cridada. Així la resta de classes poden accedir a ella i als paràmetres de l'aplicació sense necessitar la instància directament.

- **Llegir els paràmetres del disc**

Els paràmetres inicials de l'aplicació es troben al fitxer *parameters.txt*. La funció *init* s'encarrega de llegir aquest fitxer línia per línia. Aquest fitxer està compost de diverses línies amb el nom del paràmetre i el valor. Mentre va llegint el fitxer, assigna el valor del paràmetre a l'atribut corresponent en funció del nom que troba.

3. RESULTATS

En aquest apartat es descriuen els resultats obtinguts en finalitzar el projecte, començant pel resultat principal d'aquest, l'aplicació desenvolupada. També s'inclou un breu resum de les diverses funcionalitats que s'ha implementat i un apartat de conclusions extretes un cop acabat tot el desenvolupament.

3.1. Aplicació de *pathfinding*

En aquesta secció es descriu com és l'aplicació resultant del projecte, el seu funcionament (com es controla) i la interfície gràfica de què disposa.

L'aplicació resultant es compon d'una única finestra on es situa la quadrícula de cerca i la interfície gràfica, que permet modificar certs paràmetres de l'aplicació. A la part dreta de la finestra se situa la quadrícula, amb caselles de fons blanc separades per línies verticals i horitzontals de color gris clar. Dins aquesta quadrícula se situa el personatge, que comença a la casella superior esquerra.

A l'esquerra de la quadrícula se situa la interfície gràfica, dins una finestra flotant anomenada *"Options"*. Dins aquesta finestra es poden editar alguns dels paràmetres interns de l'aplicació. Primer de tot hi ha un botó anomenat "*Reset walls*" que esborra totes les "parets" de la quadrícula (fa que totes estiguin lliures). El primer element editable és un botó d'opció (o botó de ràdio), anomenat *"Tilemap view"* que permet escollir quina vista es prefereix per veure la quadrícula, entre una de simple, amb caselles blanques o grises, en funció de si són vàlides o no, i una que és visualment més atractiva, on es fan servir imatges que representen un terra i una paret. El segon element editable, també un botó d'opció (*"Grid connectivity"*) permet canviar la connectivitat entre caselles entre 4 i 8, essent aquesta última opció la que permet el moviment en diagonal. El següent botó (*"Algorithm"*) permet escollir quin algorisme es fa servir per realitzar les cerques, entre els dos disponibles. Quan s'escolleix A* apareix un element extra a la interfície (*"A* heuristic"*) que permet escollir quin heurístic es vol utilitzar d'entre 4 possibles: distància *Manhattan*, distància *Chebyshev*, distància euclídea i Dijkstra, que fa servir un heurístic que sempre dóna 0, cosa que fa que A* es comporti com Dijkstra. Per últim, si es té sel·leccionat A* i es realitza una cerca, a sota de l'element que permet escollir l'heurístic apareixen dos textos que indiquen la llargada del camí trobat per A* i la quantitat de nodes que ha visitat (explorat) en l'execució.

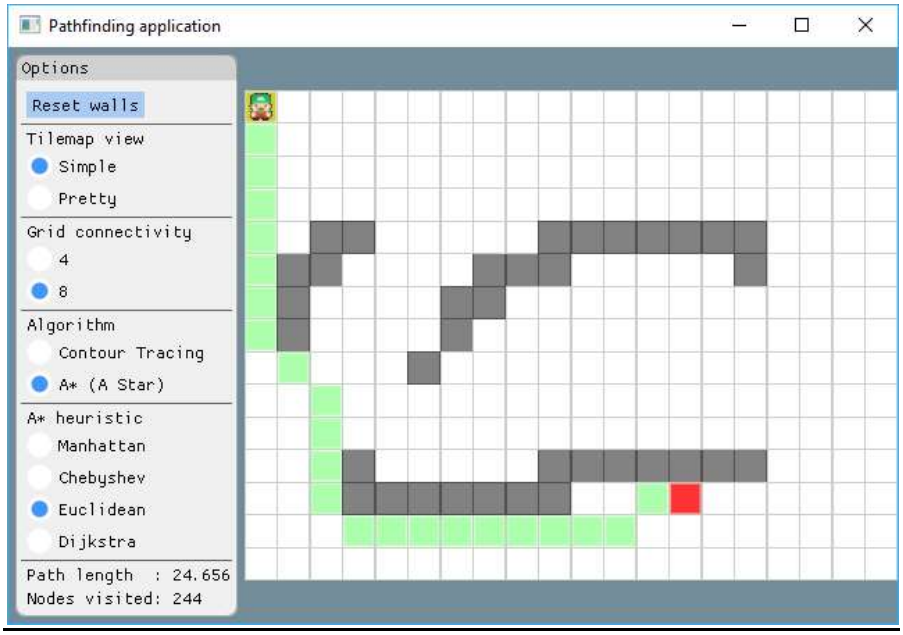

Figura 13: Aplicació amb la interfície.

Si es canvia la mida de la finestra de l'aplicació, la interfície no canvia de mida ni de posició, però la quadrícula s'adapta al canvi realitzat per ocupar el màxim espai possible en el tros de finestra restant però sense deformar-se, de manera que les caselles mantinguin sempre la forma quadrada. Això ho aconsegueix afegint espais buits als costats o a la part superior/inferior de la quadrícula, quan la mida de l'espai de la finestra que no ocupa la interfície té una relació d'aspecte diferent de la de la quadrícula.

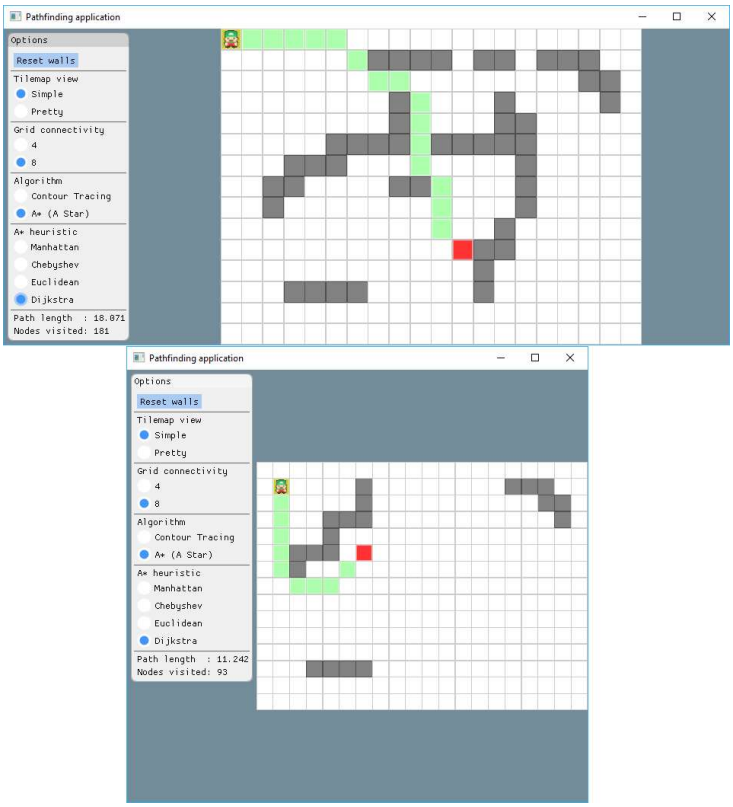

Figura 14: Espais buits per evitar la deformació de la quadrícula

En cas que l'usuari canviï la mida de la finestra i aquesta quedi massa petita, l'aplicació canvia la mida de forma automàtica a la mida mínima determinada. D'aquesta manera s'assegura que per molts canvis que es facin en la finestra, tota la interfície gràfica sigui sempre visible i la quadrícula no es vegi massa petita.

El control de l'aplicació es realitza enterament amb el ratolí, clicant amb els botons esquerre o dret i/o arrossegant el punter per la pantalla. Per defecte, totes les caselles de la quadrícula s'inicialitzen com a vàlides (lliures), però això l'usuari les pot editar. Si es fa clic esquerre damunt una casella vàlida (on no estigui situat el personatge), aquesta passa a quedar bloquejada i, si es manté el botó premut i s'arrossega el punter per la pantalla, totes les caselles que el punter passi per sobre també passaran a estar bloquejades. D'aquesta manera es pot editar el mapa de forma més còmoda i ràpida. El mateix procés es duu a terme en clicar en una casella bloquejada. En fer-ho, aquesta passa a ser vàlida (queda lliure) i si s'arrossega el punter amb el botó encara premut totes les caselles bloquejades per on passi passaran a quedar lliures.

Per tal de realitzar una cerca de camins només cal fer clic amb el botó dret a damunt d'una casella lliure de la quadrícula (on no estigui situat el personatge). En fer-ho, es mostrarà damunt la quadrícula el camí (si existeix) que seguirà el personatge per arribar al seu destí, però aquest romandrà quiet. La casella inicial es mostrarà en groc, la final en vermell i les intermitges en verd.

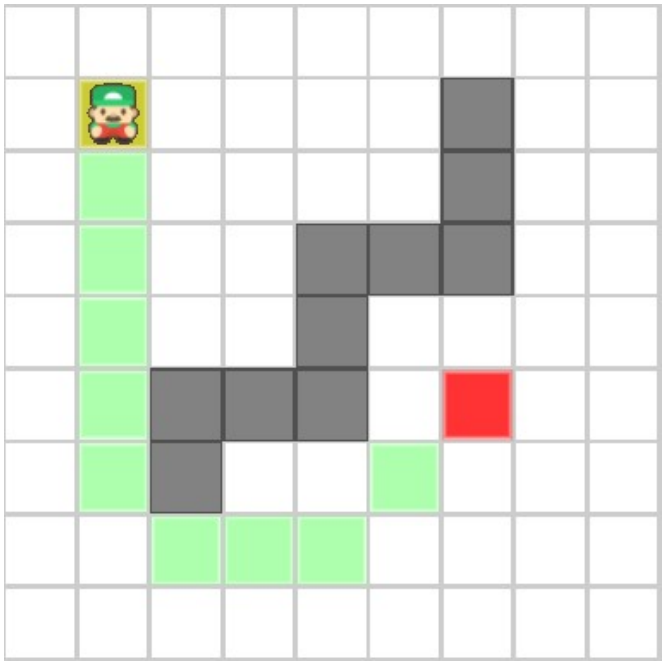

Figura 15: Camí visualitzat damunt la quadrícula

Perquè el personatge s'hi desplaci, cal tornar a fer clic dret a la mateixa casella (la vermella) i llavors el personatge s'hi desplaçarà. Mentre el personatge s'està movent, la modificació de caselles queda desactivada, per evitar que les pertanyents al camí es puguin marcar com a bloquejades, i tampoc es pot realitzar una nova cerca fins que el personatge no s'atura. No

obstant això, si només s'ha fet clic una vegada al destí, és a dir, el camí s'està mostrant però el personatge no ha començat a moure's, encara es poden editar caselles, i el camí es va adaptant dinàmicament als canvis realitzats. El mateix passa si es canvien paràmetres de la interfície quan s'està veient el camí però el personatge està quiet. Si, per exemple, es canvia l'heurístic d'A*, el camí es recalcula i es mostra el nou (mantenint sempre els mateixos punts d'inici i de fi). Per últim, si alguna d'aquestes modificacions fa que deixi d'existir un camí, deixa de mostrar-se'n cap i es pot tornar a escollir una casella per realitzar una nova cerca.

3.2. Funcionalitats de l'aplicació

Les diferents funcionalitats de què disposa aquesta aplicació són les següents:

- Mostrar una quadrícula en pantalla.
- Mostrar un personatge que es desplaci per les caselles.
- Permetre editar les caselles entre lliures i bloquejades.
- Realitzar cerques de camins entre la posició del personatge i la sel·leccionada.
- Permetre modificar certs paràmetres de les cerques amb una interfície gràfica.
- Mostrar el camí trobat abans que el personatge el recorri.
- Adaptar-se als canvis de mida de la finestra.

3.3. Conclusions

Un cop finalitzat el projecte, val la pena realitzar una reflexió dels resultats del mateix, els usos futurs que pot tenir i si pot servir com a base per desenvolupar altres projectes.

3.3.1. Reflexió

S'ha obtingut una aplicació amb les funcionalitats necessàries per servir com a eina de docència a l'assignatura VJ (que era la motivació principal), per introduir l'alumnat als conceptes sobre la cerca de camins, particularment aplicada en l'àmbit dels videojocs. Es volia una solució particular, que fes servir els algorismes concrets de què es parla a l'assignatura i permetés als estudiants veure'ls no només de forma teòrica. A més a més, s'ha desenvolupat una base de codi font amb un seguit de funcionalitats que podran ser utilitzades per ampliar els continguts i els projectes de les classes de laboratori que, a més a més, és compatible amb el que ja existia anteriorment.

Vist això, doncs, podem dir que s'ha arribat a obtenir un producte que podrà ser utilitzat com a eina de docència en l'assignatura que emmarcava el projecte.

3.3.2. Ús futur

Quan el projecte es doni per acabat definitivament, després de la defensa, la totalitat del mateix serà entregada al director del projecte, Antonio Chica, coordinador de VJ, perquè en pugui fer ús lliure en futures edicions de l'assignatura.

L'autor del projecte també en conservarà tot el material, ja que pot servir com a mostra de les seves habilitats en la matèria, cosa que sol ser útil pel que fa al món laboral.

3.3.3. Ampliacions

Aquest projecte podria servir com a base per a futures ampliacions relacionades amb el mateix tema. Com s'ha comentat a la contextualització, existeixen altres tipus de grafs que es fan servir per representar espais de cerca, com són els *waypoint graphs* o les "malles de navegació" (*navigation meshes*). També existeixen sistemes que no fan servir una quadrícula, sinó una graella de cel·les hexagonals. L'adaptació del projecte perquè fes servir qualsevol d'aquests elements podria ser un bon projecte que ampliés les funcionalitats originals d'aquest. Una altra possibilitat podria ser modificar el projecte actual per poder veure una animació de com els algorismes van explorant les caselles, per fer-se una idea més detallada del procediment que han seguit. Per últim, afegir altres algorismes de cerca també podria ser una extensió del projecte, que permetria observar les diferències en el comportament de cada un.

4. GESTIÓ DEL PROJECTE

4.1. Metodologia

En aquest apartat es descriurà la metodologia de treball que s'ha utilitzat en aquest projecte i les mesures que s'han anat prenent per minimitzar els possibles riscos.

4.1.1. Mètode i estil

La metodologia que s'ha utilitzat ha estat d'estil àgil, ja que el projecte ha disposat d'un espai de temps força curt pel desenvolupament. No obstant això, el projecte no s'ha cenyit de forma estricta a una metodologia àgil en concret, ja moltes d'elles estan pensades pel treball en equip (tot i que certs conceptes són igualment vàlids per una sola persona), i aquest projecte només disposa d'un desenvolupador.

Una de les bondats de les metodologies àgils és que proporcionen flexibilitat en el desenvolupament i la possibilitat d'obtenir resultats de forma primerenca. Això permet poder estar provant el projecte constantment, cosa molt convenient per tal de detectar possibles errors de programació. A més a més, obtenir resultats aviat permet que els problemes que poden sorgir durant el desenvolupament es detectin ben aviat i així es puguin minimitzar els riscos.

El que s'ha descrit a l'anterior paràgraf és el que s'ha anat aplicant a mesura que avançava el desenvolupament. Efectivament, ha permès detectar els errors de programació ben de pressa, cosa que ha facilitat molt trobar-hi una solució.

4.1.2. Seguiment del projecte

Des del començament del treball fins el final, a fi de poder detectar ràpidament els problemes que han anat sorgint, s'han anat convocant reunions de seguiment entre l'alumne i el director del projecte, que han estat molt útils per dirigir i supervisar el desenvolupament del projecte.

En les primeres reunions de seguiment es van definir els aspectes bàsics del que seria el projecte i quins serien els objectius i requeriments del mateix. En aquestes primeres reunions el director es va encarregar de dirigir l'alumne sobre com començar el desenvolupament i sobre quines eren les primeres tasques que valia la pena plantejar-se. En cada una de les reunions posteriors es va seguir un format similar. Aquest format consistia en, primer, supervisar i analitzar el que hi havia fet fins el moment i el que s'havia fet des de l'última reunió. Després d'això, alumne i director decidien quina era la següent tasca que l'alumne s'havia de plantejar per la següent reunió o debatien sobre els problemes que haguessin pogut sorgir en la tasca actual i què es podia fer per solucionar-los. Aquestes

reunions s'han anat celebrant cada una o dues setmanes, en funció del temps que requerís la tasca actual o dels dubtes que pogués tenir l'alumne.

4.2. Planificació

En aquest apartat es descriu la planificació del desenvolupament del projecte. La durada d'aquest ha estat de 4 mesos i mig, en el període entre el 18 de febrer (inici de GEP) i el 5 de juliol (data de lectura) que és el que s'havia previst al començament.

A continuació es descriuen les diferents tasques en què s'ha dividit el projecte en l'ordre en què s'han dut a terme. Totes les tasques d'implementació inclouen les proves al codi implementat, que s'han anat realitzant a mesura que es desenvolupava. Al final s'inclou un diagrama de Gantt en el que es pot observar la durada que ha tingut cada tasca i en quin moment del desenvolupament s'han realitzat.

4.2.1. Fase inicial

En aquesta primera fase s'han dut a terme les següents tasques:

- **Gestió de projectes (GEP)**

Aquesta primera tasca es compon de tota la feina duta a terme a l'assignatura de GEP, que va començar el 18 de febrer i va finalitzar el 27 de març amb la presentació oral. Durant aquesta assignatura s'ha fet la definició de l'abast i la contextualització del projecte, així com la planificació inicial i la gestió econòmica i de sostenibilitat del projecte. Aquesta fase conclou amb una presentació oral on s'exposa tot el treball realitzat durant l'assignatura.

- **Preparació de l'entorn de treball**

L'objectiu d'aquesta tasca és ben simple: deixar els ordinadors en què es desenvoluparà el projecte a punt per poder començar la implementació. Ha consistit, doncs, en instal·lar totes les eines necessàries i comprovar que funcionen correctament. Una d'aquestes eines és Microsoft Visual Studio, que s'ha instal·lat amb tots els complements necessaris per desenvolupar el projecte en C++. També ha inclòs obtenir el projecte de partida de l'asisgnatura VJ i comprovar que funcionava correctament.

4.2.2. La quadrícula de cerca

En aquesta fase, que ha consistit en el desenvolupament de la quadrícula on es realitzen les cerques, s'han realitzat les tasques següents:

- **Adaptació de *TileMap***

Aquesta classe ja estava disponible en el projecte de partida, però no feia la tasca que es requeria en el projecte, és clar. Aquesta tasca ha consistit en adaptar aquesta classe fins a

aconseguir que permetés visualitzar una quadrícula amb dues imatges diferents per cada tipus de casella. De la classe inicial només han quedat les funcions bàsiques (per carregar textures i mostrar la quadrícula en pantalla). En aquesta tasca també s'han buscat les imatges que havien de servir per identificar les caselles vàlides de les que no ho són.

- **Implementació de *GridMap***

A diferència de *TileMap*, *GridMap* no existia en el projecte de partida. Aquesta tasca ha consistit en la implementació d'aquesta classe des de zero.

- **Unió de TileMap i GridMap**

TileMap i *GridMap* són dues classes que representen conceptes diferents de la quadrícula, una la part visual i l'altra la part "física". Aquesta tasca ha consistit crear el sistema que permet que, en modificar *GridMap* d'alguna manera, el canvi es vegi reflectit en la imatge que mostra *TileMap*.

4.2.3. El personatge

En aquesta fase ha involucrat tot allò relatiu al personatge que es desplaça per la quadrícula. La fase ha estat composta per les següents tasques:

- **Adaptació de *Player***

Igual que *TileMap*, la classe *Player* ja era present en el projecte de partida. En aquesta tasca s'ha modificat la classe perquè mostrés el personatge a damunt de la quadrícula i perquè permetés canviar la casella on es situava. En aquesta tasca, de moment, el control ha estat a base de les fletxes de direcció del teclat. De la classe inicial només han quedat les funcions de càrrega de textures i de visualització en pantalla. L'últim que s'ha fet en aquesta tasca és buscar una imatge que servís per representar el personatge.

- **Desplaçament per la quadrícula**

Després de poder veure el personatge i canviar-ne la posició, el següent pas ha consistit en fer que aquest es pogués moure entre les caselles de forma suau i seguint una ruta establerta. Aquesta tasca ha consistit en ampliar les funcionalitats de *Player* per permetre les transicions contínues entre caselles a una certa velocitat i perquè el moviment es realitzés segons la llista de caselles emmagatzemada en un vector.

- **Implementació de *Parameters***

Després d'implementar el moviment del personatge, ha estat necessari disposar d'un sistema per modificar la velocitat amb què es movia i el nombre de vegades per segon que s'executa el bucle principal de l'aplicació. És per això que en aquesta tasca s'ha implementat la classe *Parameters*.

4.2.4. Modificació de la quadrícula

Un cop es s'ha pogut veure la quadrícula en pantalla, el següent pas ha consistit en permetre que l'usuari pogués editar les caselles. Les tasques d'aquesta fase han estat:

- **Events de ratolí**

El primer pas ha estat registrar els clics de ratolí i la posició on aquest està situat. En aquesta tasca s'han implementat les funcions al fitxer *main* que capturen aquests events i els mètodes de *Game* que els emmagatzemen i permeten fer consultes.

- **Adaptació a la mida de la finestra**

Si l'usuari canvia la mida de la finestra, la mida amb què es mostra la quadrícula ha de canviar. La mida amb què es mostra la quadrícula es essencial per saber en quin punt de la mateixa s'ha fet clic. Per tant, abans d'interpretar els clics del ratolí cal implementar el sistema que canviï la mida del que es veu a la finestra en canviar la mida d'aquesta, per després tenir-ho en compte. Aquesta tasca ha consistit en desenvolupar aquest sistema.

- **Modificació de caselles**

Un cop implementada la tasca anterior, aquesta ha consistit desenvolupar el sistema que interpreta les posicions on es clica i les "tradueix" a caselles. També s'ha implementat el que permet que, en fer un clic a una casella, l'estat d'aquesta canviï.

4.2.5. Visualització de camins

Aquesta fase ha consistit en poder veure camins a damunt de la quadrícula. Ha estat formada únicament per aquesta tasca:

- **Implementació de *Path***

Aquesta tasca ha consistit en implementar aquesta classe, desenvolupar els *shaders* necessaris per visualitzar el camí correctament i permetre que el model visualitzat es creés a partir de les caselles d'un vector (el camí).

4.2.6. Cerca de camins

En aquesta ha incorporat tot allò relacionat amb el *pathfinding* en si. En ella s'han desenvolupat tots els sistemes que duen a terme la cerca. Les tasques han estat les següents:

- **Implementació de *PathFinder***

Aquesta tasca ha tingut un objectiu simple: desenvolupar la classe *PathFinder* amb els atributs necessaris i les funcions virtuals que les subclasses han d'implementar.

- **Implementació de Bresenham**

En aquesta tasca s'ha desenvolupat el moviment en línia recta del traçat de contorns, fent servir per això l'algorisme de Bresenham. A més de la implementació, ha inclòs la recerca de la informació necessària per implementar l'algorisme.

- **Implementació del traçat de contorns**

Aquesta tasca ha consistit en implementar mètode que permet recórrer el contorn d'un objecte i, un cop implementat, unir-lo amb el moviment en línia recta de Bresenham per implementar la classe *ContourTracing* totalment.

- **Implementació de funcions auxiliars d'A***

Abans de posar-se amb A* pròpiament dit, ha fet falta implementar les funcions que utilitza (per calcular l'heurístic, descobrir caselles...) i afegir els atributs necessaris a *A_Star*. La feina en aquesta tasca ha consistit en això

- **Implementació d'A***

Finalment, aquesta tasca ha consistit en la implementació d'A* i, per tant, la implementació completa de la classe *A_Star*.

4.2.7. La interfície gràfica

Un cop desenvolupades totes les funcions bàsiques del projecte, en aquesta tasca s'ha desenvolupat la interfície gràfica de l'aplicació. Aquestes han estat les tasques:

- **Presa de contacte amb Dear ImGui**

Abans de res, el primer pas ha estat informar-se sobre el funcionament d'aquesta biblioteca per ser capaç d'utilitzar-la. Aquesta tasca ha consistit en instal·lar la biblioteca en un projecte de prova i aconseguir fer-la funcionar en el mateix. Per fer-ho s'han tingut en compte els exemples d'utilització i implementació de què, per sort, Dear ImGui disposa.

- **Disseny i implementació de la interfície**

Un cop clar el funcionament d'aquesta biblioteca, aquesta tasca ha consistit en dissenyar la interfície gràfica, implementar-la i desenvolupar el sistema que la comunica amb l'aplicació.

4.2.8. Fase final

Un cop acabat el desenvolupament de l'aplicació, en aquesta última fase s'ha preparat l'entrega i la presentació final. S'han realitzat les següents tasques:

- **Redacció de la memòria**

Aquesta tasca ha consistit en la redacció per part de l'alumne del document recopilatori del desenvolupament del projecte.

- **Preparació del codi font**

Juntament amb la memòria, cal entregar el codi font desenvolupat. El codi font està inclòs dins un projecte, que cal deixar "presentable" abans de poder-lo entregar. Bàsicament cal extreure del projecte tots els arxius innecessaris, especialment els que que ocupen molt espai que són creats per l'*IDE*.

- **Preparació de la presentació**

Aquesta tasca consisteix en crear el document necessari per servir com a suport en la lectura del projecte també en preparar la mateixa.

- **Lectura**

L'última tasca de tot el treball, prevista per realitzar-se el dia 5 de juliol, consisteix en realitzar la lectura i defensa final del TFG davant el tribunal assignat.

4.2.9. Diagrama de Gantt

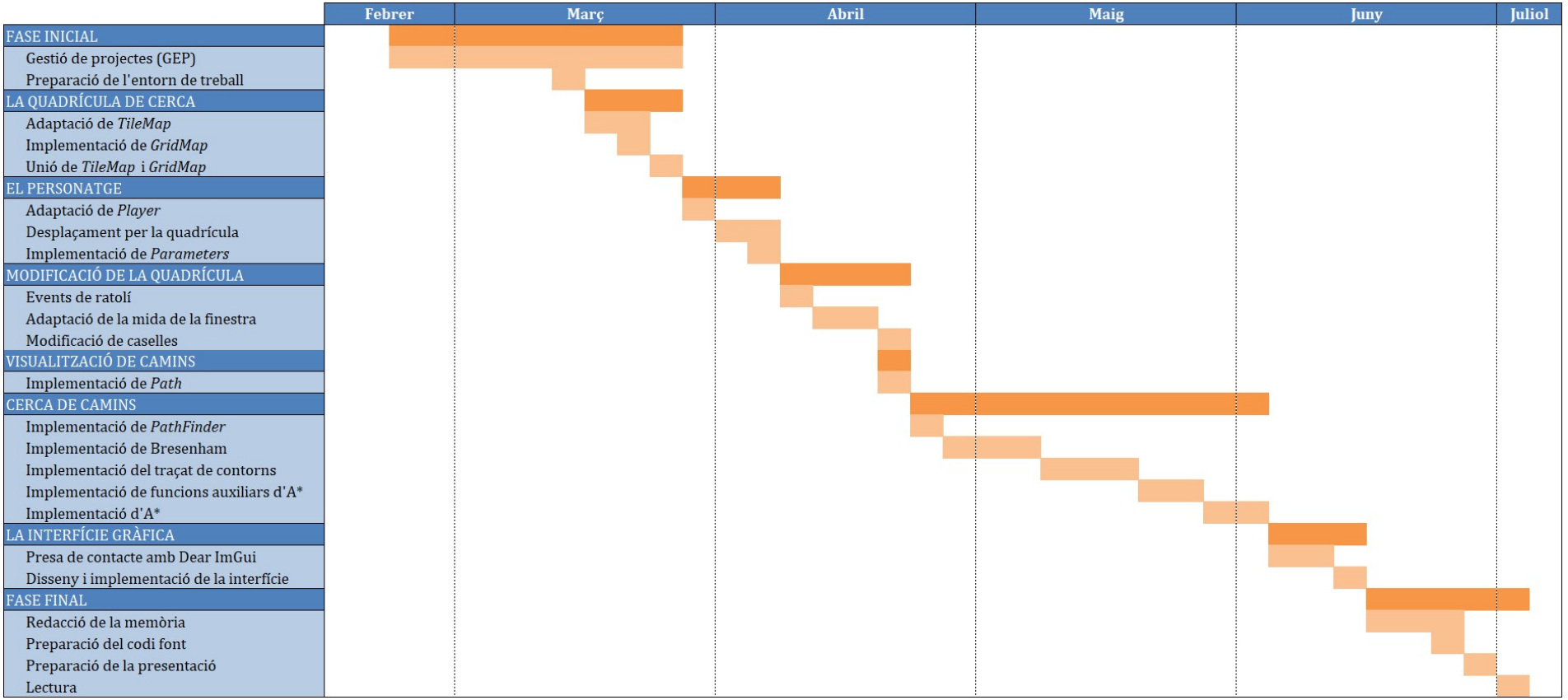

Figura 16: Diagrama de Gantt del desenvolupament del projecte.

4.3. Sostenibilitat del projecte

En aquest apartat s'avalua la sostenibilitat del projecte. Primer, es realitza una estimació del pressupost que tindria el projecte en cas que s'hagués desenvolupat com a projecte de negoci amb un finançament. Després, es realitza un anàlisi del possible impacte econòmic, ambiental i social del projecte segons les tres dimensions de la matriu de sostenibilitat. La dimensió econòmica de la mateixa es tractarà fent servir el pressupost estimat. Per cada una de les dimensions de la matriu, es reflexionarà sobre el projecte posat en producció, la vida útil i els riscos associats al mateix.

4.3.1. Pressupost

Per realitzar una estimació del pressupost es considerarà el cost dels recursos de *hardware* i *software* utilitzats en el projecte, així com també el dels recursos humans i les despeses generals. El càlcul de les amortitzacions dels recursos *hardware* i *software* es farà considerant un temps d'ús de 4,5 mesos sobre 4 anys d'amortització en el cas del *hardware* i sobre 3 anys pel *software*. En tots els apartats del pressupost els impostos que apliquin van inclosos en els costos considerats. Es considerarà que els costos atribuïbles directament a una activitat són només els costos de recursos humans, ja que els costos en *software* són nuls (Tots els programes utilitzats, tret del sistema operatiu, són gratuïts) i els de *hardware* no són directament atribuïbles a tasques concretes (ja que en totes elles s'han fet servir tots els recursos de *hardware*).

- **Recursos**

Per poder realitzar una estimació acurada, és necessari identificar tots els recursos que s'han utilitzat en aquest projecte. A continuació es detallen tots aquests recursos, juntament amb el seu cost o amortització associada. Concretament es detallen els recursos associats a les eines utilitzades (*hardware* i *software*) i als recursos humans utilitzats en el desenvolupament del projecte.

- **Hardware i Software**

Hardware	Preu	Vida útil (anys)	Amortització
Ordinador de sobretaula	1.500 €	4	140,63 €
Ordinador portàtil	900 €	4	84,38 €
Monitor	150 €	4	14,06 €
Ratolí	20 €	4	1,88 €
Teclat	15 €	4	1,41 €

Software			
Windows 10 Home	2 x 145 €	3	0 € (preinstal·lat)
Microsoft Visual Studio (*Community*)	0 €	3	0 €
OpenGL	0 €	3	0 €
GLEW	0 €	3	0 €
FreeGLUT	0 €	3	0 €
GLM	0 €	3	0 €
SOIL	0 €	3	0 €
Dear ImGui	0 €	3	0 €
Total	2.585 €		**242,34 €**

- **Recursos humans**

Tot i que el projecte l'ha desenvolupat enterament una sola persona, el desenvolupador i autor del TFG, cal tenir en compte que ha hagut de fer la funció dels diversos rols que serien necessaris pel projecte. Cada un d'aquests rols té un cost associat per hora diferent i, per tant, és necessari saber el nombre d'hores que s'han dedicat per cada rol per determinar el cost associat als recursos humans. En aquest projecte s'han considerat 3 rols diferents: gestor de projecte, dissenyador de software i programador. Els costos per hora s'han extret de l'estudi de remuneració del 2018 de la pàgina Page Personnel [19], de la part de tecnologia [20]. Per calcular el cost per hora s'ha dividit la remuneració anual entre 1800, que es considera la quantitat d'hores que es treballen en un any en un contracte estàndard de 8 hores diàries. Tenint això en compte, els costos per hora estimats per cada rol han estat de 35 € pel gestor de projecte, 25 € pel dissenyador de *software* i 20 € pel programador.

Dit això, a la següent taula es pot veure quina ha estat la durada aproximada, en hores, de cada una de les tasques del projecte, que després es distribuïran per cada rol.

Tasca	**Durada (h)**
Gestió de projectes (GEP)	75
Preparació de l'entorn de treball	10
Adaptació de *TileMap*	25
Implementació de *GridMap*	15

Unió de *TileMap* i *GridMap*	10
Adaptació de *Player*	15
Desplaçament per la quadrícula	25
Implementació de *Parameters*	5
Events de ratolí	15
Adaptació de la mida de la finestra	30
Modificació de caselles	10
Implementació de *Path*	15
Implementació de *PathFinder*	10
Implementació de Bresenham	35
Implementació del traçat de contorns	45
Implementació de funcions auxiliars d'A*	20
Implementació d'A*	30
Presa de contacte amb Dear ImGui	25
Disseny i implementació de la interfície	15
Redacció de la memòria	35
Preparació del codi font	5
Preparació de la presentació	15
Lectura	0
Total	**485**

La distribució de les hores de dedicació de cada rol en cada tasca es pot veure a la següent taula.

Tasca	Gestor	Dissenyador	Programador
Gestió de projectes (GEP)	75	0	0
Preparació de l'entorn de treball	0	0	10
Adaptació de *TileMap*	0	5	20
Implementació de *GridMap*	0	5	10
Unió de *TileMap* i *GridMap*	0	0	10
Adaptació de *Player*	0	5	10

Desplaçament per la quadrícula	0	5	20
Implementació de *Parameters*	0	0	5
Events de ratolí	0	0	15
Adaptació de la mida de la finestra	0	0	30
Modificació de caselles	0	0	10
Implementació de *Path*	0	5	10
Implementació de *PathFinder*	0	5	5
Implementació de Bresenham	0	15	20
Implementació del traçat de contorns	0	20	25
Implementació de funcions auxiliars d'A*	0	5	15
Implementació d'A*	0	10	20
Presa de contacte amb Dear ImGui	0	10	15
Disseny i implementació de la interfície	0	10	5
Redacció de la memòria	35	0	0
Preparació del codi font	0	0	5
Preparació de la presentació	15	0	0
Lectura	0	0	0
Total	**125**	**100**	**260**

Finalment, en aquesta taula es pot apreciar el cost que tindria cada un dels rols, tenint en compte la quantitat d'hores de dedicació que li pertoquen:

Rol	Cost / h	Dedicació	Cost del rol
Gestor de projecte	35 €	125	4.375 €
Dissenyador software	25 €	100	2.500 €
Programador	20 €	260	5.200 €
Total			**12.075 €**

- **Costos indirectes**

En aquest projecte, els costos indirectes a considerar han estat 3. Com que els ordinadors tenen cert consum elèctric i requereixen accés a Internet, s'han considerat aquestes dues despeses com a cost indirecte. La justificació de l'estimació del consum elèctric es realitza més endavant, a la dimensió ambiental de la matriu de sostenibilitat.

A més a més, també s'ha utilitzat "material d'oficina" (paper i eines per escriure) durant el projecte, per fer els dissenys i esquemes necessaris, sobretot en el disseny i implementació. Aquest últim cost és difícil d'estimar, però s'ha suposat un cost de 15 €, suficient per uns quants bolígrafs, llapis i paper.

Producte	Quantitat	Preu	Cost
Consum elèctric	105 kWh	0,14 €/kWh	14,7 €
Quota d'Internet	4,5 mesos	50 €/mes	225 €
Material d'oficina	-	15 €	15 €
Total			**254,7 €**

- **Imprevistos**

En el desenvolupament de tot projecte, s'ha de considerar la possibilitat que es produeixin imprevistos que puguin augmentar-ne el cost. Els únics imprevistos que podrien tenir un cost significatiu en aquest projecte són dos: primer, els problemes que puguin ser ocasionats per l'avaria d'algun dels recursos *hardware* que es fan servir, i segon, l'increment de cost que comportaria que una tasca del projecte s'allargués més de l'esperat. Cap d'aquests imprevistos s'ha produït realment, però en un pressupost (que es realitza abans del desenvolupament) és necessari incloure'ls.

Pel cas de les avaries, el que podria ocasionar problemes és l'avaria del portàtil, ja que en cas d'avaria del fixe s'hauria pogut continuar el projecte només amb el portàtil. Cal tenir en compte que la probabilitat estimada que això succeeixi és baixa. S'ha considerat un risc del 5 % sobre el preu del portàtil per aquest imprevist.

Pel cas de la durada de les tasques, com amb el cas de l'avaria del portàtil, s'ha considerat que no és una situació gaire probable, donada la metodologia del projecte, i s'ha considerat un 5 % extra sobre el cost en recursos humans per disposar de cert marge extra, apart del que es designa a la previsió de contingències.

Causa d'imprevist	Preu	Probabilitat	Cost
Ordinador portàtil	900 €	5 %	45 €
Recursos humans	12.075 €	5 %	603,75 €
Total			**648,75 €**

- **Pressupost total**

Un cop explicitats tots els costos estimats del projecte, aquests s'han afegit a la taula següent per poder veure la suma total del pressupost. Al cost total s'ha afegit una partida de contingència del 5%, ja que aquest és un projecte de poc risc, en el qual és difícil que hi hagi gaires imprevistos que puguin afectar al pressupost.

Concepte	Cost
Recursos hardware	242,34 €
Recursos software	0 €
Recursos humans	12.075 €
Costos indirectes	254,7 €
Imprevistos	648,75 €
Subtotal	**13.220,79 €**
Contingència (5%)	661,04 €
Total	**13.881,83 €**

4.3.2. Anàlisi de l'impacte

En aquest apartat es realitza l'anàlisi dels impactes del projecte segons les tres dimensions de la matriu de sostenibilitat. Per cada una d'aquestes dimensions, es dóna una resposta a les qüestions que s'hi plantegen.

- **Dimensió ambiental**

En aquesta dimensió de la matriu es tracta el posible impacte sobre el medi ambient que pot tenir el projecte al llarg del seu cicle de vida, que dependrà dels recursos que utilitzi.

- **Projecte posat en producció**

En aquest projecte només s'han fet servir ordinadors personals, que són els únics recursos que tenen un impacte ambiental quantificable, en forma de consum d'electricitat, ja que la

resta de recursos són o bé software o no consumeixen energia. Els ordinadors es faran servir en totes les tasques del projecte i podem estimar que, durant tot el projecte, han consumit uns 105 kWh, que no resulta molt remarcable. Per fer aquest càlcul, considerem una potència consumida 4 h/dia de 200 W i 75 W pel fixe i el portàtil [21], respectivament, i 1,5 W 20 h/dia pels dos ordinadors (consum quan estan apagats [22]). Tenint en compte 138 dies de projecte (18 febrer – 5 juliol)i que el fixe es fa servir el 85% del temps:

$$138 \times (0{,}85 \times (200 \times 4 + 1{,}5 \times 20) + 0{,}15 \times (75 \times 4 + 1{,}5 \times 20)) \simeq 105\ kWh$$

D'altra banda, encara que sigui petit, aquest consum podria haver-se reduït fent servir només l'ordinador portàtil enlloc del fixe, ja que el portàtil consumeix menys. Però el desenvolupament és molt més fàcil i còmode fent servir l'ordinador de sobretaula i, com que l'impacte ambiental és petit, no s'ha considerat aquesta opció, ja que hauria pogut afectar al projecte. També es pot tenir en compte que, com que el projecte reutilitza una base de software ja desenvolupat (de VJ), pot aconseguir reduir la seva durada (i per tant l'energia consumida) sense afectar els objectius i, per tant, permet cert estalvi energètic.

Un cop finalitzat el projecte, no s'han pogut identificar accions que haguessin permès realitzar el projecte reduïnt significativament el nombre de recursos, ja que aquests ja estaven força ajustats de bon principi.

- **Vida útil**

Podem afirmar que, en la seva vida útil, el producte desenvolupat tindrà probablement una petjada ambiental similar a la que tenen les eines software de partida del projecte, ja que aquest ha creat un producte similar partint d'aquesta base. També podem considerar que els recursos que farà servir aquest projecte en la seva vida útil seran els que l'alumnat ja utilitza als projectes de VJ. Per tant, podem considerar que el projecte no ha de comportar, en principi, cap augment en els recursos que ja es fan servir.

Dit això, podem afirmar que el projecte no servirà per reduir ni augmentar en cap manera apreciable l'ús de recursos utilitzats actualment i, per tant, no millorarà ni empitjorarà la petjada ecològica.

- **Riscos**

Durant el desenvolupament del projecte, podria haver-se produït, tot i que amb poca probabilitat, alguna situacion que hauria comportat augmentar la petjada ecològica del projecte, sobretot en forma dels recursos que consumiria. El risc principals que hauria pogut succeir en aquest aspecte hauria estat que el projecte s'hagués allargat durant més temps del que estava previst, cosa que hauria comportat un augment de l'energia consumida per desenvolupar-lo.

- **Dimensió econòmica**

En aquesta dimensió de la matriu es tracta el consum de recursos materials i humans del projecte i el cost econòmic dels mateixos. És per això que s'ha realitzat un pressupost del projecte on es poden veure els recursos i s'identifiquen els costos dels mateixos. També és necessari tenir un mecanisme de control de desviacions del pressupost.

- **Projecte posat en producció**

Un cop realitzat el pressupost, s'ha estimat el cost total de realització del projecte, tenint en compte tots els recursos que ha utilitzat, tant materials com humans. A més a més, s'ha comptat amb un cert marge de contingència i d'imprevistos. Val la pena mencionar que el cost total del projecte seria difícil de reduir per dos motius: el primer que els costos en recursos humans són força ajustats i ocupen la major part del pressupost del projecte i el segon, que tot el software que es fa servir (excepte el SO Windows) és gratuït. Tot i que és cert que es podria estalviar una mica si no es fes servir un ordinador de sobretaula (ja que el portàtil és més barat i consumeix menys energia) o si es contractés una tarifa d'Internet més barata, l'estalvi que es podria assolir seria minúscul, perquè aquests dos conceptes tenen molt poc pes en el pressupost. Tenint això en compte, es pot afirmar que el projecte hauria estat viable si hagués hagut de ser competitiu.

- **Vida útil**

Mentre aquest projecte s'utilitzi, com en el cas de la dimensió ambiental, no està previst que contribueixi en augmentar el cost, en aquest cas econòmic, que actualment té el que ja s'utilitza a l'assignatura VJ. A més a més, no està previst que el projecte requereixi gaires ajustos o reparacions durant la seva vida útil. En cas de donar-se, aquestes situacions tindrien un impacte minúscul.

- **Riscos**

Durant el desenvolupament del projecte també podria haver-se donat alguna situació que n'hauria fet augmentar el cost. La situació mencionada a la part de riscos de la dimensió ambiental (que el projecte s'allargués) també haurien comportat un cert augment en el cost econòmic, ja que l'energia extra utilitzada hauria tingut un cost, i la quota d'internet utilitzada durant aquest temps també.

Un altre possible risc, tot i que molt improbable, que hauria pogut succeir, és que s'hagués produït una avaria dels dos ordinadors utilitzats, no només del portàtil, situació que es té en compte al pressupost. Això només s'hauria pogut solucionar amb una reparació o amb l'adquisició d'un equip nou, i les dues solucions haurien fet augmentar el cost.

- **Dimensió social**

Aquesta dimensió es refereix a l'impacte que el projecte ha tingut i tindrà en les persones que hi ha treballat o que hi estiguin relacionades d'alguna manera (per exemple, qui el faci servir).

- **Projecte posat en producció**

Durant el desenvolupament del projecte, aquest ha tingut una certa repercussió personal sobre la persona que hi treballa, jo mateix, el desenvolupador, autor del TFG i d'aquestes línies. La temàtica del projecte és l'àmbit de la programació de videojocs, emmarcat en l'assignatura VJ. Aquesta assignatura va ser cursada per mi, el desenvolupador, a la qual em vaig apuntar amb entusiasme i va resultar ser molt del meu agrat. Per mi, col·laborar en ampliar una assignatura que m'ha agradat i en la qual he après moltes coses resulta satisfactori i personalment realitzador. A més a més, l'àmbit de la programació de videojocs (i aquests en general, així com la indústria associada) em resulta interessant i atractiu, i és una de les opcions que considero a l'hora d'encarar la vida professional. La realització del TFG ha contribuït a ampliar una mica la base de coneixement de què disposo en la matèria i permet disposar de material que mostri les habilitats personals en el tema, cosa que es valora força a l'hora de buscar feina en la indústria.

- **Vida útil**

El material software de què ja disposa l'assignatura VJ és utilitzat per l'alumnat, que permet facilitar la seva feina i estalviar temps a l'hora d'encarar els projectes de l'assignatura i entendre els conceptes de la mateixa. També el fa servir el professorat per facilitar la seva feina docent, ja que permet disposar d'exemples en els que basar les classes. L'objectiu del projecte és que tingui aquesta mateixa funció, pel que fa a la utilització per part dels alumnes i el professorat. Tanmateix, també té com a objectiu ampliar els àmbits que es cobreixen a les classes de laboratori, ja que es disposarà d'una eina de cerca de camins pels projectes, cosa que no existia a l'assignatura. La idea és que així els projectes de l'assignatura puguin ser més diversos i ambiciosos, ja que disposant de part de la feina necessària feta, es poden acabar projectes més grans en el mateix temps. En aquest sentit, el projecte pot tenir un impacte social positiu en l'alumnat i professorat de VJ, així com en l'assignatura en si, i que no hauria de tenir, en principi, cap impacte negatiu en cap col·lectiu imaginable.

- **Riscos**

Tenint en compte la naturalesa del projecte, resulta difícil imaginar cap situació que pogués comportar cap tipus de perjudici per les persones que l'utilitzin o se'n beneficiïn.

5. REFERÈNCIES

[1] Guia docent de VJ. Facultat d'Informàtica de Barcelona. https://www.fib.upc.edu/ca/estudis/graus/grau-en-enginyeria-informatica/pla-destudis/assignatures/VJ, *accedit el 25/06/2019.*

[2] Moore, M. E., & Novak, J. (2009). *Game development essentials: Game industry career guide*. Delmar Learning.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Single-Source Shortest Paths* a *Introduction to algorithms (pp. 643-671)*. MIT press.

[4] Rabin, S. (2013). *Choosing a Search Space Representation.* A *Game AI Pro: Collected Wisdom of Game AI Professionals (pp. 253-258)*. AK Peters/CRC Press.

[5] Cui, X., & Shi, H. (2011). Direction oriented pathfinding in video games. *International Journal of Artificial Intelligence & Applications*, *2*(4), 1.

[6] Patel, A. *Amit Patel's Home Page*. http://www-cs-students.stanford.edu/~amitp/, *accedit el 25/06/2019.*

[7] Patel, A. *Red Blob Games*. https://www.redblobgames.com/, *accedit el 25/06/2019.*

[8] Patel, A. *Introduction to the A* Algorithm*. https://www.redblobgames.com/pathfinding/a-star/introduction.html, *accedit el 25/06/2019.*

[9] Patel, A. *Amit's A* Pages*. http://theory.stanford.edu/~amitp/GameProgramming/, *accedit el 25/06/2019.*

[10] Xu, X. *PathFinding.js*. https://qiao.github.io/PathFinding.js/visual/, accedit el 25/06/2019.

[11] *Web oficial de Visual Studio*. https://visualstudio.microsoft.com, *accedit el 15/06/2019.*

[12] *Web oficial d'OpenGL.* https://www.opengl.org/, *accedit el 15/06/2019.*

[13] *Web oficial de FreeGLUT*. http://freeglut.sourceforge.net*, accedit el 15/06/2019.*

[14] *Web oficial de GLEW*. http://glew.sourceforge.net*, accedit el 15/06/2019.*

[15] *Web oficial de SOIL*. https://www.lonesock.net/soil.html*, accedit el 15/06/2019.*

[16] *Web oficial de GLM*. https://glm.g-truc.net, *accedit el 15/06/2019.*

[17] *Repositori de Dear ImGui.* https://github.com/ocornut/imgui, *accedit el 15/06/2019.*

[18] Theoharis, Papaioannou, Platis i Patrikalakis (2008). *Graphics & Visualization: Principles and Algorithms.* A K Peters, Ltd.

[19] *Estudis de remuneració de Page Personnel*. https://www.pagepersonnel.es/prensa-estudios/estudios/estudios-de-remuneraci%C3%B3n*, accedit el 25/06/2019.*

[20] *Estudi de remuneració de Page Personnel, tecnologia*. https://www.pagepersonnel.es/sites/pagepersonnel.es/files/PG_ER_IT_2018.pdf*, accedit el 25/06/2019.*

[21] Energuide. *How much power does a computer use?* https://www.energuide.be/en/questions-answers/how-much-power-does-a-computer-use-and-how-much-co2-does-that-represent/54/, *accedit el 25/06/2019.*

[22] M. Bray (2006). *Review of Computer Energy Consumption and Potential Savings*. https://www.dssw.co.uk/research/computer_energy_consumption.html*, accedit el 25/06/2019.*