Colegiul Național „Ion C. Brătianu”

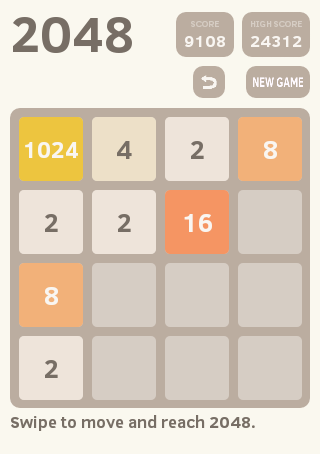
Proiect pentru susținerea examenului de certificare a compețentelor profesionale

- Disciplina Informatică -

|  |  |  |
| --- | --- | --- |
| Profesor coordonator:  Enache Nicoleta | | Elev: Ion David-Gabriel  Clasa a XII-a C  Specializarea:  Matematică-Informatică |
|  |  | |

Mai 2018

Inteligență artificială pentru jocul logic „2048”



Cuprins

Argument

Introducere

Tehnologii utilizate

* Limbajul de programare
* Librăria SDL
* XML

Algoritmii utilizați și arhitectura

* Framework-ul aplicației
  + Partea grafică
  + Partea logică
* Algoritmul Expectimax
  + Euristici
  + Fișierul “config.xml”

Performanțele aplicației

Codul sursă

Concluzii

Argument

Jocul “2048” a fost un mare succes în momentul lansării inițiale și este încă jucat de mii de oameni și astăzi. Este un joc inovator prin simplitatea sa aparentă, dar aceasta dispare pe măsură ce jucatorul capătă mai multă experiență și realizează că totul este mult mai complicat decât pare. Poate un om ar încerca să folosească strategii, precum poziționarea pieselor mari într-un colț, un calculator nu poate gândi astfel, deoarece apar mulți alți factori care intervin și care ar trebui să determine o schimbare în comportament. Astfel, un calculator trebuie învățat să procedeze optim pentru a putea să depășească și un om la acest joc.

Principalul motiv pentru care am ales această temă este dificultatea jocului. Acest joc pare simplu pentru un om, însă este destul de dificil pentru un calculator, fiecare stare putând fi urmată de până la 30 de alte stări după o singură mutare, simularea cărora nefiind la fel de ușor de realizat precum în cazul altor jocuri populare, precum “X și O”. Astfel, este nevoie de o abordare diferită față de programarea dinamică. Se poate folosi o abordare de tip Greedy, așa cum folosește și acest program, însă se pot folosi și algoritmi genetici, bazați pe rețele neuronale. Totuși, este cert că algoritmul utilizat nu este ușor de implementat. Sunt de părere că provocările ne ajută să depășim obstacole și să ajungem la un nivel mai ridicat de cunoștințe. Un proiect ușor de realizat nu aduce acea satisfacție, acea senzație de reușită, spre deosebire de un proiect complex.

Al doilea motiv care m-a determinat să aleg această temă este potențialul de a o exploata și de a învăța de pe seama ei. Acest proiect a presupus utilizarea a numeroase tehnici de programare, precum și îmbinarea lor într-un singur program. Sunt utilizate tehnici și cunoștințe din grafica pe calculator, programarea paralelă, algoritmi bazați pe probabilități, încărcarea fișierelor într-un anumit format și multe altele. Toate aceste domenii oferă soluții la probleme des întâlnite, iar o înțelegere, cel puțin generală, a acestora este esențială în dezvoltarea atât în plan personal, cât și profesional.

În ultimul rând, alegerea acestei teme a fost inițial determinată de pasiunea mea pentru jocuri. Jocurile reprezintă lumi virtuale cu o multitudine de regui în care scopul este maximizarea unui scor. Noțiunea de joc poate fi concretizată pentru a defini simularea: o lume virtuală cu o multitudine de reguli similare cu cele ale lumii reale în care scopul este maximizarea unui scor. Se poate observa că teoria jocurilor poate duce la dezvoltarea diferitelor programe care simulează aspecte ale realității, astfel încât un program poate oferi soluții la probleme reale, calculatorul reușind astfel să-și îndeplinească scopul: de a-l ajuta și depăși pe om.

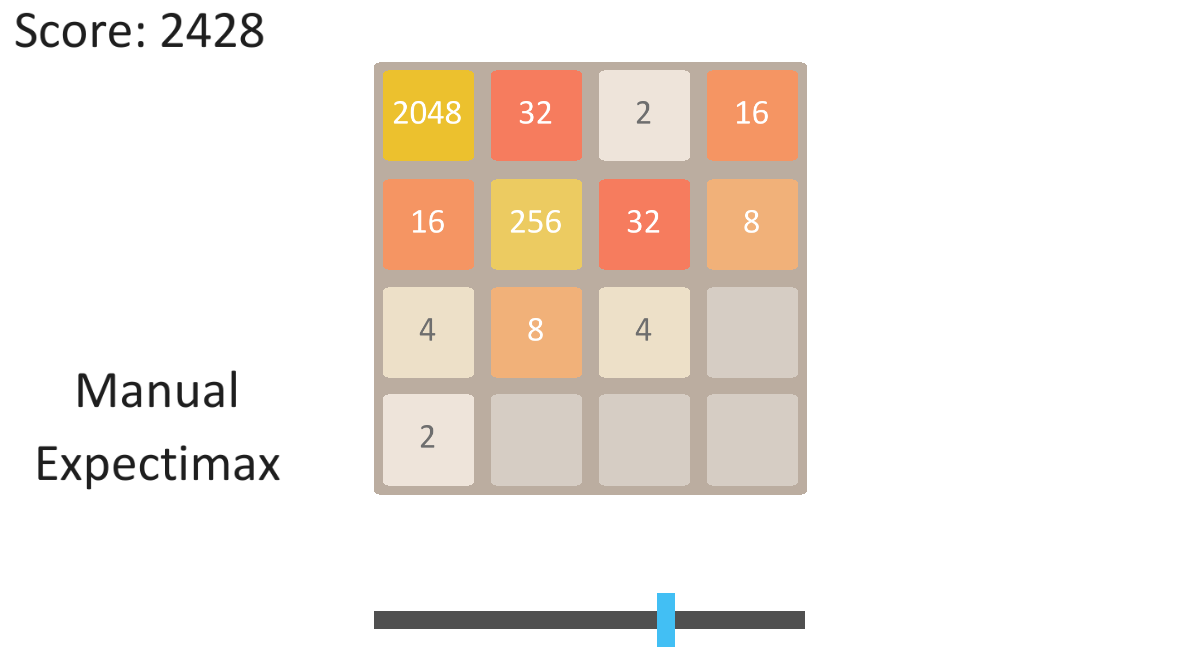
Așadar, din aceste motive consider că tema aleasă este una care îmbină multe domenii importante ale informaticii și care poate fi abordată din mai multe puncte de vedere. Chiar dacă a trebuit să aleg un singur algoritm principal pentru realizarea acestui proiect, există și alte metode mai avansate pentru a ajunge la rezultate cel puțin egale. Nu se poate construi un program perfect care să rezolve puzzle-ul acestui joc, doar programe care aproximează optimul, această tema oferind astfel numeroase oportunități de a rezolva probleme și învăța lucruri noi.

Introducere

Aceasta este o aplicație care încearcă să rezolve puzzle-ul jocului ”2048”. Jocul ”2048” a fost publicat pentru prima dată pe Android în anul 2014 de către digiplex.in și a fost un mare succes încă de la început. Programul acesta nu reușește să câștige jocul în toate cazurile, însă reușește să ajungă la punctaje similare cu cele ale unui om. Pe lângă inteligența artificială, aplicația prezintă și o interfață grafică.

Regulile jocului:

Se dă o tablă de joc de 4 pe 4. Piesele pot fi numai numere puteri ale lui 2, în afară de 1. O mutare constă in deplasarea tuturor pieselor din pozitia inițială către o direcție dată până când a ajuns la capătul tablei sau există o alta piesă în calea sa de valoare diferită. Dacă o piesa întâlnește o alta de aceeași valoare, acestea două se unesc într-o piesă de valoare dublă. După fiecare mutare pe tablă apare o nouă piesă, pe o poziție aleatoar aleasă, de valoare 2 sau 4, după care jucătorul poate muta din nou. Jocul se termină în momentul în care orice mutare aleasă de jucator nu mai poate modifica tabla, adica nu mai există nicio mutare validă. Scopul jocului, precum reiese și din titlul său, este de a ajunge la piesa cu numărul 2048.



Tehnologii utilizate

Limbajul de programare

Programul a fost realizat în C++, un limbaj de programare general, imperativ și orientat pe obiecte. Acesta a apărut în anii 1980 și a fost dezvoltat de Bijarne Stroustrup ca o serie de îmbunătățiri ale limbajului C. Au fost introduse de atunci și până acum noțiunea de clasă, funcțiile virtuale, suprascrierea operatorilor, moștenirea multiplă, șabloanele, exceptiile și multe altele, ultima versiune fiind C++17, cunoscută și sub numele de C++1z, adoptată în iulie 2017. Cu toate acestea, aplicația poate fi compilată folosind standardul C++11. Cu timpul a apărut și librăria standard STL (Standard Template Library), care conține numeroase funcții ajutătoare și structuri de date necesare în majoritatea programelor. Chiar dacă această librărie standard nu poate satisface cerințele multor programe comerciale din cauza eficienței scăzute, această aplicație nu necesită optimizări atât de majore.

Am ales acest limbaj de programare din mai multe motive:

* Este flexibil

Ceea ce C++ poate oferi și alte limbaje precum Java sau C# este manipularea low-level a tot ceea ce se întâmplă în cadrul programului. Nu există un garbage collector (cel puțin nu implicit), deci memoria alocată trebuie ștearsă pentru a evita probleme precum memory leak-urile. Acest lucru poate fi neplăcut pentru unele persoane, deoarece trebuie ținut cont de toate alocările, însă dacă totul este programat corespunzător programul poate beneficia de un mare plus în eficiență. De asemenea, există alegerea de a folosi pointeri, ceea ce este impus în alte limbaje. În limbaje precum Java, o matrice de 4x4 elemente de tip întreg nu poate fi alocată pe stivă și pentru numai 64 de octeți trebuie să folosim memoria heap și alocarea dinamică, lucru care poate fragmenta memoria.

* Este popular

Prin popularitate se înțelege faptul că multe persoane programează în C++, deci majoritatea, dacă nu toate, erorile au fost deja rezolvate, formându-se astfel un limbaj de programare pe cât se poate de stabil. Acest lucru este esențial, deoarece în caz contrar pot apărea probleme care nu depind de noi și care duc la scrierea unui cod greu de înțeles și instabil, care este posibil să nu ruleze corespunzător după rezolvarea problemei inițiale. Un alt rezultat al popularității este abundența librăriilor scrise în C++. Acestea ne pot ajuta, ușurând procesul de dezvoltare și asigurând o funcționare eficientă a programului, precum STL.

Librăria SDL

Pentru a realiza interfața grafică a aplicației a fost folosită librăria SDL 2.0. Simple DirectMedia Layer este o librărie care rulează pe mai multe platforme, care oferă acces de tip low level către dispozitivele audio, tastatură, mouse, joystick și API-uri precum OpenGL și Direct3D. Este folosită de programe de tip video player, emulatoare și jocuri precum cele dezvoltate de Valve și de pe site-ul Humble Bundle. Platformele pe care rulează oficial sunt Windows, Mac OS X, Linux, iOS și Android. SDL este scrisă în C și poate fi compilată cu un compilator pentru C++, putând fi folosită și în alte limbaje, precum C# și Python.

SDL nu este o librărie completă, dar există altele care vin în ajutorul ei și implementează lipsurile sale:

* SDL\_image

Este o librărie care încarcă imagini din memorie sau dintr-un fișier. Librăria SDL poate încărca numai imagini de tip BMP, dar SDL\_image poate încărca imagini de tip BMP, GIF, JPEG, LBM, PCX, PNG, PNM, SVG, TGA, TIFF, WEBP, XCF, XPM și XV. Pentru a încărca o imagine și pentru a o transforma într-o textură gata să fie folosită de SDL sunt necesare urmatoarele linii de cod:

SDL\_Surface\* surface = IMG\_Load(path.c\_str());

texture = SDL\_CreateTextureFromSurface(renderer->window->sdlRenderer, surface);

SDL\_FreeSurface(surface);

* SDL\_ttf

Pentru a putea genera text sub formă de imagini care pot fi desenate pe ecran este necesară folosirea unei librării de tipul SDL\_ttf. Aceasta poate încărca fonturi de tip TrueType și genera imagini ce pot fi folosite în aplicațiile care folosesc SDL. Nu folosește o metodă optimă de a desena textul, generând pentru fiecare string o nouă imagine. O metodă mai bună ar fi generarea unui atlas cu toate caracterele fontului și desenarea fiecărei litere în poziția corespunzătoare fără a mai genera vreo altă imagine. Aplicația generează în timp real numai scorul, deci o generare pe cadru nu este foarte costisitoare. Pentru a genera o astfel de imagine se pot scrie următoarele:

font = TTF\_OpenFont(name.c\_str(), size);

SDL\_Surface\* surface = TTF\_RenderText\_Blended(font, text.c\_str(), color);

texture = SDL\_CreateTextureFromSurface(renderer->window->sdlRenderer, surface);

SDL\_FreeSurface(surface);

* SDL\_gfx

Libraria SDL poate desena linii, dreptunghiuri, imagini etc., însă pentru forme mai complexe este necesară librăria SDL\_gfx. Aceasta poate desena poligoane, cercuri, dreptunghiuri cu colțuri rotunde și multe altele, toate acestea cu o calitate superioară librăriei standard. Spre exemplu, aplicația folosește următorul cod pentru a desena dreptunghiuri cu colțuri rotunde, unde ConvertCoords convertește coordonatele din aplicație în coordonate folosite de fereastră și ConvertDimY convertește distanța.

ConvertCoords(bottomLeft);

ConvertCoords(topRight);

ConvertDimY(radius);

roundedBoxRGBA(window->sdlRenderer, (Sint16) bottomLeft.x, (Sint16) bottomLeft.y, (Sint16) topRight.x, (Sint16) topRight.y, (Sint16)radius, (Uint8)currentColor.r, (Uint8)currentColor.g, (Uint8)currentColor.b, (Uint8)currentColor.a);

Aplicația se folosește de SDL pentru a crea o fereastră, desena pe aceasta, accesa mouse-ul și tastatura și încărca imagini, chiar dacă librăria este capabilă de mult mai multe, precum utilizarea efectelor sonere (caz în care poate fi utilizată librăria SDL\_mixer, care este încărcată de aplicație, dar nefolosită). Există și alte librării care pot realiza aceste lucruri, precum SFML, însă eficiența și ușurința de a folosi această librărie au determinat alegerea finală. Spre exemplu, pentru a inițializa librăria și a crea o fereastră sunt necesare numai următoarele linii de cod, unde w și h sunt dimensiunile dorite:

SDL\_Init(SDL\_INIT\_EVERYTHING);

IMG\_Init(IMG\_INIT\_PNG | IMG\_INIT\_JPG | IMG\_INIT\_TIF);

TTF\_Init();

SDL\_SetHint(SDL\_HINT\_RENDER\_SCALE\_QUALITY, "2");

sdlWindow = SDL\_CreateWindow(title.c\_str(), SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED,

w, h, SDL\_WINDOW\_RESIZABLE);

sdlRenderer = SDL\_CreateRenderer(sdlWindow, -1, SDL\_RENDERER\_ACCELERATED);

Sistemul de coordonate al ferestrei diferă față de cel al jocului, pentru a putea avea o dimensiune variabilă a ferestrei. Daca “w” este lungimea în pixeli a ferestrei, “h” este înălțimea ferestrei în pixeli, iar “ar” este raportul ”w/h”, atunci fereastra are originea în colțul din stânga sus și lungimile w și h către dreapta respectiv jos. Jocul are un sistem de coordonate inspirat din jocurile realizate cu un API de tipul OpenGL sau Direct3D în care originea este în centrul ecranului, înălțimea totală fiind o valoare fixă, anume 20, măsurată de jos în sus, lungimea putând varia în funcție de dimensiunea în pixeli a ferestrei și fiind măsurată de la stânga la dreapta. Acest sistem de coordonate asigură un aspect corect al figurilor geometrice și imaginilor (pătratele au laturile egale atât în cod, cât și vizual), dar și o independență față de dimensiunea în pixeli a ferestrei. Pentru a converti coordonate și dimensiuni din spațiul jocului în spațiul ferestrei sunt folosite următoarele funcții, unde Vector2f este un vector cu 2 componente de tip float și window->left este valoarea corespunzătoare aspectului ferestrei pentru a fi identic cu cel al jocului și pentru păstra înălțimea fixă (spre exemplu, pentru o fereastră de 1280x720, window->left are valoarea 17.7778):

void Renderer::ConvertCoords(Vector2f& v)

{

v.x = ((v.x + window->left) / window->left) / 2.f \* window->w;

v.y = ((-v.y + 10.f) / 10.f) / 2.f \* window->h;

}

void Renderer::ConvertDimensions(Vector2f& v)

{

v.x = (v.x / window->left) / 2.f \* window->w;

v.y = (v.y / 10.f) / 2.f \* window->h;

}

XML

Extensible Markup Language (XML) este un meta-limbaj de marcare care definește un set de reguli pentru a codifica documente într-un format care este ușor de citit, atât de către oameni, cât și de către calculatoare. Obiectivul acestui limbaj este de a accentua simplitatea, generalitatea și uzabilitatea. În ciuda faptului că XML este menit să fie folosit pentru documente, este folosit și pentru reprezentarea diferitelor structuri de date, cum ar fi cele utilizate în serviciile web. Sute de formate de documente au la bază sintaxa XML, precum RSS, Atom, SOAP, SVG și XHTML.

Am ales XML și nu un alt limbaj precum JSON, deoarece este utilizat la scară largă, multe programe folosesc limbajul XML, precum Visual Studio, mediul de programare în care a fost realizat proiectul, Unreal Engine, un motor de jocuri foarte cunoscut, orice browser, întrucât HTML-ul provine din XML și multe altele. De asemenea, XML suportă atribute pentru elemente, având astfel un aspect mai plăcut și fiind mai ușor de înțeles. Am ales să folosesc XML și nu un limbaj binar, deoarece aplicația nu ar beneficia prea mult de citirea mai rapidă, fiind foarte puține date de citit și ar fi mai greu de scris și citit de către un om, eliminând astfel caracteristica de care aveam nevoie în primul rând.

Aplicația folosește limbajul XML pentru a încărca un fișier care specifică modul în care ar trebui să se comporte fără a modifica sursa. Acest lucru este util deoarece algoritmul de bază rămâne același în cazul micilor modificări, doar unele variabile schimbându-și valoarea. Pentru a încărca fișierul XML, este folosit un loader scris, nu o librărie, care este capabil să citească și să scrie fișiere XML relativ simple, adică poate încărca fișiere de tip COLLADA, spre exemplu, dar nu suportă elemente avansate ale limbajului, cum ar fi namespace-uri, entități etc. Acest loader suportă elementele de bază, adică elemente, atribute, prezența comentariilor în fișiere, fiind suficient de puternic pentru majoritatea aplicațiilor, deoarece, până la urmă, tot ce face un fișier XML este să păstreze informații. Am ales să utilizez un limbaj atât de complex pentru stocarea unui număr relativ mic de variabile, deoarece am vrut să nu existe vreo problemă în adăugarea ulterioară de elemente în acest fișier. Un exemplu de fișier “config.xml” este următorul:

<game>

<dpo>

<autorestart>true</autorestart>

<autorestartDuration>0.0</autorestartDuration>

<maxDepth>3</maxDepth>

<maxCheck>4</maxCheck>

<benchmark>true</benchmark>

<penaltyAmount>1.5</penaltyAmount>

<emptyTileBonus>0.075</emptyTileBonus>

<positionalBonus>0.3</positionalBonus>

<ascendingTilesBonus>0.5</ascendingTilesBonus>

<losingPenalty>100000.0</losingPenalty>

</dpo>

</game>

Algoritmii utilizați și arhitectura

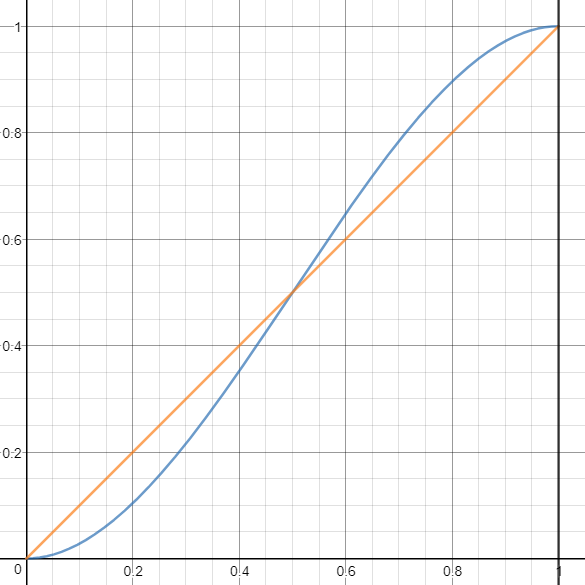
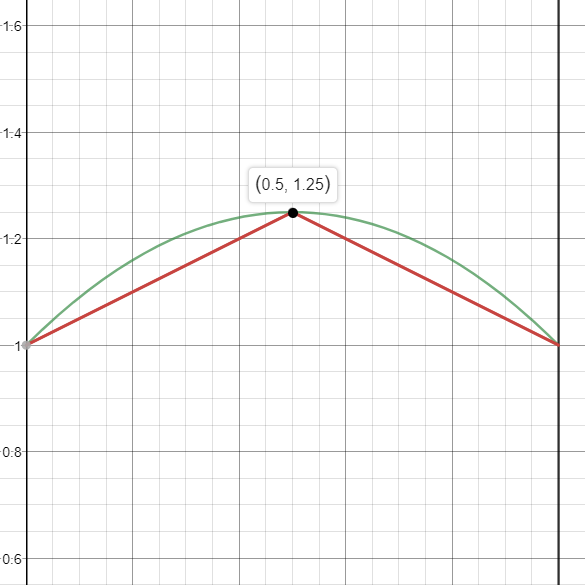
Framework-ul aplicației

Aplicația utilizează un model întâlnit foarte des în jocurile simple. Deoarece scheletul aplicației a fost construit cu gândul de a fi refolosit pentru o gamă largă de aplicații, acel model a fost generalizat. Modelul presupune separarea aplicației în scene. O scenă reprezintă ansamblul obiectelor și personajelor din nivelul curent, numite și entități, cât și lucruri precum camera, care dă perspectiva jucătorul asupra lumii, luminile, soarele sau efectele speciale. Această separare se realizează pentru a putea refolosi entități și în alte niveluri într-un mod convenabil. În aplicație nu există mai multe niveluri, deci tiparul acesta nu este folositor. Prin urmare, singura noțiune referitoare la obiecte pe care am folosit-o este aceea de scenă.

În continuare, pentru a separa partea logică de partea grafică, o scenă conține două funcții principale: “Update” și “Render”. Funcția “Update” este responsabilă de actualizarea părții logice a aplicației. Aici sunt procesate informații legate de perifericele de intrare, este actualizată starea curentă, sunt actualizate butoanele, slider-ul, scorul etc. În funcția “Render” sunt desenate pe ecran starea jocului și interfața grafică. Aceste două funcții sunt apelate o singură dată în fiecare cadru (frame), iar programul este capabil să atingă mii de cadre pe secundă. Imediat dupa apelarea funcției “Render”, fereastra este actualizată și sunt afișate cele desenate. Ca urmare a acestei separări între aceste două aspecte ale unui joc, programul devine mult mai logic și ușor de înțeles. Există și alte tehnici pentru a separa jocul în și mai multe funcții, printre care și ECS (Entity-component-system), care introduce noțiunile de componentă și sistem. O entitate nu mai este programată individual, ci este creată folosind diferite componente, pentru a putea refolosi cod. Sunt eliminate funcțiile anterior menționate și sunt înlocuite de sisteme, în cadrul cărora se afla o listă cu entitățile care conțin anumite componente. Acest model avansat nu este necesar pentru aplicația aceasta, fapt reieșit din simplitatea sa.

Partea grafică

Modelul utilizat este unul construit în jurul paralelizării virtuale. Fiecare entitate ar fi actualizată și funcția “Update” ar presupune simularea activității ei pe o durată foarte mică, precum 16,67 milisecunde. Astfel s-ar crea iluzia mișcării, când de fapt sunt afișate numai imagini succesive asemănatoare una cu cealaltă. Aplicația beneficiază de această consecință prin desenarea tablei de joc și a pieselor. După ce algoritmul se decide asupra unei mutări este generată o listă cu toate modificările aduse tablei de joc, adică toate mișcările pieselor, toate îmbinările și aparițiile de noi piese. Există două funcții ajutătoare: una care desenează mișcările și una care desenează îmbinările și aparițiile. Imediat după o mutare, piesele trebuie mișcate, apoi trebuie afișate și îmbinările alături de apariții. Pentru mișcări și apariții s-a folosit tehnica ”Hermite interpolation”, care definește o funcție polinomială mai plăcută estetic decât funcția liniară. De asemenea, pentru îmbinări s-a folosit tot o funcție polinomială. Aceste funcții au rolul de a atenua schimbările și de a oferi un efect mai lin de tranziție între stări.

În graficul din stânga se afla cu albastru funcția polinomială din tehnica “Hermite interpolation”, care este , iar cu portocaliu funcția liniară. În dreapta se află cu verde funcția , iar cu roșu funcția . Se poate observa că funcțiile polinomiale de grad mai mare ca unu oferă o tranziție mai atenuată față de celelalte.

În implementarea mișcărilor a apărut o problemă ce ține de precizia numerelor reale pe calculator. Aplicația folosește tipul “float” pentru a reprezenta numerele reale, deoarece tipul “double” nu este necesar, fiind procesate mai greu operațiile sale, dar și deoarece precizia sa mai mare nu aduce niciun avantaj. După ce a fost calculată valoarea funcției Hermite, aceasta a fost preluată de o funcție de tranziție între vectori care are rolul de a calcula valoarea unui vector care pornește dintr-un punct și se oprește în altul, fiind parcurs o cantitate de drum dată. Problema apare când punctul de pornire coincide cu punctul de sosire. Piesele manifestă un “tremurat” pe parcursul animației, chiar dacă ele ar trebui să rămână staționare. Din punct de vedere matematic, funcția “lerp” (linearly interpolate) returnează , unde e punctul de plecare, e punctul de sosire și e procentul de drum parcurs. Dacă avem , atunci funcția ar trebui să returneze , dar din cauza preciziei numerelor reale, funcția returnează valori apropiate punctului de sosire. Acest lucru a fost rezolvat prin verificarea în avans a pozițiilor din tabla de joc a pieselor pentru a evita apelarea funcției de tranziție. Aceasta este o problemă minoră, dar care trebuie menționată.

Paleta de culori este identică cu cea a jocului original, cu scopul de a simula într-o măsură cât mai mare atmosfera sa. În partea stângă se află scorul, care nu este calculat ca în aplicația originală, ci fiind suma tuturor pieselor de pe tablă. Am ales această metodă de acordare a punctelor deoarece consider că scorul este mai puțin remarcabil decât piesa cu valoarea cea mai mare, de aceea rapoartele referitoare la performanțele aplicației conțin și piesa maximă existentă la sfârșitul jocului. Tot în stânga se află două butoane: “Manual” și “Expectimax”. Al doilea buton pornește algoritmul care începe să joace, dar dacă utilizatorul dorește să intervină, el poate apăsa primul buton și controla totul din tastatură. Am adăugat această opțiune pentru a putea compara algoritmul cu ființa umană mult mai ușor. În partea de jos se află un slider care ajustează viteza animațiilor.

Partea logică

În funcția “Update” au loc mai multe evenimente. La început sunt actualizate butoanele și slider-ul, apoi are loc actualizarea controller-ului. Controller-ul decide care va fi următoarea mutare în funcție de starea curentă. Un controller manual curent determină următoarea mutare pe baza tastaturii, pe când unul de tip Expectimax folosește algoritmul cu același nume pentru a continua jocul. Controller-ul poate fi gata sau nu să execute mutarea, iar dacă nu este gata se sare peste aceasta. Dacă este gata, mutarea este executată, modificările sunt memorate în câteva liste pentru a reda animațiile și controller-ul este informat de următoarea stare pentru a începe deja să proceseze. Pe durata animațiilor nu se poate efectua nicio mutare.

În alineatele anterioare am menționat o tehnică de a paraleliza virtual anumite sarcini. Acest lucru este posibil și în cazul procesării mutărilor, însă duce la animații greșite sau timp ridicat de procesare. Dacă se execută o mutare pe cadru, animațiile nu vor mai fi fluide, iar dacă se execută întâi animațiile și apoi se procesează mutarea, timpul de așteptare crește. În ambele cazuri butoanele și slider-ul devin mai puțin sensibile la comenzi și astfel răspund mai greu. Pentru a rezolva aceste probleme am recurs la programarea paralelă reală. Astfel, imediat după ce controller-ul execută mutarea, el începe imediat să proceseze noua mutare, animațiile rulând în paralel. După ce animațiile s-au terminat, la fiecare cadru, controller-ul este întrebat dacă poate sau nu să execute o nouă mutare. Dacă nu, se trece la următorul cadru și dacă da, atunci se execută noua mutare și procesul se repetă.

Există opțiunea de a înregistra rezultatele fiecărui joc și de a se calcula o medie a scorurilor și de a se realiza o analiză pe baza piesei cu valoare maximă din fiecare meci. Aceste informații sunt afișate în consolă, dar și într-un fișier separat, generat cu un nume unic, astfel încât toate fișierele anterioare să-și păstreze conținutul pentru a putea compara rezultatele. Există, de asemenea, opțiunea de a reporni jocul automat după încheierea sa. În mod normal, jocul se poate reporni oricând prin apăsarea tastei R, iar la sfârșitul unui meci va apărea un mesaj și jocul va aștepta decizia utilizatorului. Funcția de repornire automată poate fi pornită din fișierul XML “config.xml”, iar durata dintre momentul terminării jocului și momentul repornirii poate fi modificată în același loc. Pentru a realiza un studiu pe baza performanțelor aplicației am activat cele două opțiuni enumerate anterior și am setat durata de repornire ca fiind zero.

O parte esențială a aplicației a fost implementarea simulării jocului propriu-zis. Am realizat acest lucru prin stocarea tablei de joc într-o matrice de 4x4 elemente de tip întreg care memorează exponentul valorii corespunzătoare piesei minus unu sau -1 dacă piesa nu există (spre exemplu, lui 2 îi corespunde 0, lui 16 îi corespunde 3 etc.) și am creat o clasă care se ocupă de simularea diferitelor evenimente care apar pe parcursul unui joc. Aceste funcții sunt folosite în principal de controller, care poate apela de zeci de mii de ori pe secundă funcții pentru a determina viitoarele stări, dar sunt folosite și de partea grafică pentru a executa mutarea controller-ului, deoarece acesta nu oferă și starea curentă pentru a nu apărea erori provocate de lipsă de sincronizare sau piese puse în poziții greșite.

Algoritmul Expectimax

Arborele Expectiminimax este o variantă specializată a arborelui minimax folosit în sisteme de inteligență artificială pentru jocuri strict competitive de doi jucători în care rezultatul depinde de o combinație între abilitățile jocătorului și elemente aleatoare. Pe lângă nodurile de tip “min” și ”max” existente în arborii minimax, arborele Expectiminimax introduce noduri de tip “chance”, care au valoarea medie a evenimentelor aleatoare care urmează.

Arborele Expectimax este o variantă specializată a arborelui Expectiminimax, care elimină cel de-al doilea jucător. Astfel, există numai noduri de tip “max” și “chance”. Arborele Expectimax este bipartit, deoarece un nod de tip “max” nu poate fi legat decât de noduri de tip “chance”, iar nodurile de tip “chance” nu pot fi legate decât de noduri de tip “max”. Nodurile “chance” sunt generate de către joc, pe când cele “max” sunt rezultatul mutărilor voluntare. Fiecărui nod îi este atribuită o stare, iar dacă acest nod este terminal îi este calculată valoarea folosind o funcție specială. Fiecărei muchii din arbore îi este atribuită o valoare, care este probabilitatea ca starea acelui nod să succeadă starea nodului tată. Pentru a calcula valoarea unui nod de tip ”max” calculăm suma ponderată a valorilor fiilor săi. Pentru a calcula valoarea unui nod de tip ”chance” aflăm maximul nodurilor vecine și reținem mutarea pe care o putem efectua pentru a ajunge la acea stare. Căutarea pornește dintr-un nod “chance”, căruia îi este atribuită starea curentă, iar apoi este generat arborele până la o anumită adâncime.

La baza arborelui Expectimax stă faptul că nodurile “chance” nu sunt controlate de un jucător, valoarea acestora nefiind influențată de factori cu un scop dat, precis. Acestea pot fi considerate noduri “min”, folosindu-se arborele minimax, dar această metodă nu aduce rezultate atât de bune, arborele minimax fiind optimizat pentru alegeri mai optime. Deoarece întregul arbore nu poate fi generat rapid, este impusă o adâncime maximă. Această adâncime maximă este cel mai optim 4, 6 sau 8. Aceste numere sunt pare deoarece funcția de generare generează câte două niveluri deodată. O adâncime maximă de 4 duce la o generare rapidă, dar și la punctaje mai mici, iar o adâncime maximă de 8 duce la o generare foarte lentă, dar și la un punctaj mai mare, întrucât o explorare mai amănunțită a arborelui duce la descoperirea din timp a situațiilor limită, existând astfel posibilitatea de a le evita. Deoarece adâncimile maxime sunt numere pare, adâncimile maxime menționate în continuare vor fi 2, 3 sau 4 și vor corespunde valorilor 4, 6, respectiv 8.

Spre deosebire de alți algoritmi, Expectimax se bazează pe probabilități, deci rezultatul final nu este asigurat. Utilizarea acestui algoritm nu garantează un punctaj maxim, acesta utilizatând mai degrabă o abordare de tip Greedy. În ciuda acestui fapt, se pot obține rezultate destul de bune printr-o serie de euristici favorabil alese. Tot comportamentul aplicației depinde de acea funcție specială care calculează scorul unei stări, deci algoritmul de bază este nefolositor de unul singur. Prin aceste euristici îi impunem controller-ului să folosească strategii similare cu cele pe care un om le-ar folosi. Găsirea acestor euristici este diferită pentru fiecare joc, însă intuiția și experiența ne pot ușura munca.

Algoritmul în pseudocod este următorul:

Algoritmul are ca parametrii de intrare nodul curent și adâncimea arborelui curent, iar ca parametru de ieșire scorul maxim care poate fi obținut din acel nod fără a depăși adâncimea maximă. De asemenea returnează mutarea sugerată pentru ajunge la scorul maxim. Lista “scoruri” inițializată pe linita 5 conține atât scorul maxim obținut în urma unei mutări, dar și probabilitatea de a ajunge la starea corespunzătoare scorului. Structura repetitivă de pe linia 6 se referă la iterarea prin toate pozițiile în care poate apărea o piesă, dar și valoarea sa. Programul adevărat folosește doua structuri repetitive pentru a realiza acest lucru: una pentru poziția în care poate apărea piesa (de la 0 la 15) și alta pentru valoarea piesei (2 sau 4). Acest lucru a fost evitat în pseudocod pentru claritate. Se poate observa condiția de oprire a funcției recursive pe linia 9. Cu toate că subalgoritmul Expectimax returnează mutarea corespunzătoare scorului maxim, această valoare este ignorată pe linia 10, deoarece dacă se va ajunge la această stare în viitor, ea va fi procesată din nou folosind o precizie mai mare. Algoritmul “gândește” cu mai multe mutări înainte, dar nu memorează decât mutarea imediat de după starea inițială. Funcția “CalculeazaScor” este acea funcție specială care va fi prezentată în următoarele pagini. La final scorul este adăugat în lista cu scoruri, din care este calculată media ponderată după formula . Este păstrat scorul maxim alături de mutarea corespunzătoare, care este la sfârșit returnat.

Euristici

Algoritmul Expectimax este foarte flexibil deoarece depinde extrem de mult de acea funcție specială utilizată în calcularea scorului. În această aplicație, scorul returnat de funcție nu este și cel afișat, algoritmul nereușind să obțină rezultate foarte bune cu acel scor, în schimb este utilizată o funcție bazată pe mai multe euristici menite să determine algoritmul să joace asemenea unui om. Prin euristici se înțelege o modalitate de a atribui o valoare numerică pe baza unui aspect al stării curente. Acest algoritm implică studierea mai atentă a jocului și găsirea unei strategii de câștig, spre deosebire de alți algoritmi mai avansați, precum Q-Learning, care găsesc de unii singuri strategiile necesare maximizării scorului. . Programul folosește în total 6 euristici diferite pentru a calcula punctajul unei stări:

* Cea mai simplă euristică este cea conform căreia este introdusă o penalizare pentru stările în care nicio mutare nu mai e validă. Dacă nodul corespunzător stării evaluate este terminal, scorul asociat acestuia este .
* Următoarea euristică se basează pe monotonia pieselor din colțul din stânga sus către jos și dreapta. Dacă o piesă este imediat urmată de o piesă cu valoarea piesei inițiale minus unu se adaugă la scor valoarea 3 la puterea exponentului valorii piesei inițiale și înmulțită cu 4-i, unde i este linia sau coloana curentă. Nu se păstrează baza 2 pentru a diferenția între două piese cu valori identice față de o piesă cu valoare dublă, fiind mai avantajos să existe o piesă cu valoare mai mare decât două piese cu valori mai mici ce pot fi unite. Avem astfel inegalitatea . Codul responsabil de calcularea scorului conform acestei euristici pe linii (cel pentru coloane fiind similar) este:

int asc = 0;

for (int i = 0; i < 4; i++)

{

int last = 40;

for (int j = 0; j < 4; j++)

{

if (state.tiles[i][j] == -1) // spațiu gol

continue;

if (state.tiles[i][j] == last || state.tiles[i][j] == last – 1

|| last == 40)

{

if (last != 40)

asc += Pow<int>(3, last + 1) \* (4 - i);

last = state.tiles[i][j];

}

else break;

}

}

* Cea de-a treia euristică analizează cât de amestecate sunt piesele. Pentru fiecare piesă sunt verificați vecinii săi și se adaugă la penalizare 3 la puterea diferenței în modul a exponenților celor două piese. Pentru piese apropiate, spre exemplu 2048 și 1024 sau 8 și 16, penalizarea este de doar 3, dar pentru piese precum 1024 și 16 penalizarea devine 729, care este destul de mare, fiind astfel descutajată această poziționare. Chiar dacă această euristică nu este inspirată dintr-o strategie utilizată de un om, se comportă foarte bine în practică. Codul care calculează această penalizare este:

int penalty = 0;

for (int i = 0; i < 4; i++)

for (int j = 0; j < 3; j++)

penalty += Pow(3, abs(state.tiles[i][j] - state.tiles[i][j + 1]));

for (int j = 0; j < 4; j++)

for (int i = 0; i < 3; i++)

penalty += Pow(3, abs(state.tiles[i][j] - state.tiles[i + 1][j]));

* Cea de-a patra euristică se bazează pe o strategie des întâlnită la persoane, anume păstrarea pieselor cu valori mari într-un colț, anume cel din stânga sus. Valoarea atribuită fiecărei piese se calculează în modul următor: 3 la puterea exponentului valorii piesei curente înmulțit cu valoarea luată din matricea de mai jos, de pe poziția corespunzătoare poziției piesei.
* Cea de-a cincea euristică aplicată se referă la numărul de poziții neocupate. Pentru fiecare poziție neocupată de adaugă o unitate.
* Ultima euristică calculează scorul stării curente ca fiind , unde p este valoarea unei piese. Din nou, s-a folosit baza 3 pentru a putea diferenția între o piesă cu valoarea mai mare și două piese cu valori mai mici, fiind folosită inegalitatea menționată anterior. Ultimele trei euristici au fost combinate în cod pentru a parcurge starea o singură dată, obținându-se astfel codul:

int emptyTiles = 0;

int score = 0;

int posValue = 0;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

if (state.tiles[i][j] == -1)

emptyTiles++;

else

{

int tileVal = Pow<int>(3, state.tiles[i][j] + 1);

score += tileVal;

int diagValue = 7 - i - j;

posValue += diagValue \* tileVal;

}

}

}

Pentru a combina aceste scoruri nu este suficientă o simplă sumă, funcția putând fi puternic influențată de o singură euistică, ignorându-le pe celelalte. Astfel, pentru fiecare valoare există un factor cu care este înmulțită, iar valorile finale fiind adunate. Euristica a cincea contribuie la scorul final prin adăugarea unui procent din ultima euristică. Calculul final este valabil dacă s-a trecut cu succes de prima euristică. Codul final al funcției este:

return score + float(score) \* emptyTiles \* emptyTileBonus + float(asc) \* ascendingTilesBonus + float(posValue) \* positionalBonus - float(penalty) \* penaltyAmount;

Fișierul “config.xml”

Procesul de compilare a codului sursă al aplicației durează câteva secunde, ceea ce nu este incomod dacă sunt făcute modificări majore, însă pentru a modifica o singură variabilă acest proces este destul de lung. De asemenea, încărcarea unor variabile dintr-un fișier face posibilă modificarea programului, într-o anumită măsură, fără a fi nevoie de codul sursă.

Variabilele din fișierul “config.xml” sunt:

* autorestart: Specifică dacă după ce nu mai există vreo mutare validă ar trebui să pornească un alt joc în mod automat. Poate lua valorile „true” sau „false”.
* autorestartDuration: Dacă variabila „autorestart” este adevărată, această variabilă specifică timpul în secunde dintre momentul în care jocul este pierdut și începerea unui alt joc.
* maxDepth: Specifică adăncimea maximă la care poate ajunge arborele generat de algoritmul Expectimax. O valoare de 3 este recomandată, oferind un echilibru între timpul necesar procesării și performanțele din joc. O valoare de 2 determină obținerea unor scoruri mai mici, însă algoritmul procesează un joc întreg în doar câteva secunde. O valoare de 4 este mult prea mare și aplicația poate procesa o singură stare pentru câteva secunde.
* maxCheck: Această variabilă are un dublu rol: dacă variabila “maxDepth” este 3 sau mai puțin, maxCheck este numărul de piese maxim pentru care de aplică o căutare de adâncime mai mare, iar dacă este 4 sau mai mult, reprezintă numărul maxim de piese luate în considerare. În primul caz, această tehnică are ca efect utilizarea unei procesări mai calitative, având astfel o metodă dinamică de a crește calitatea căutării. O valoare negativă determină dezactivarea acestei tehnici. În cel de-al doilea caz, unele piese vor fi omise de căutare, astfel încât timpul necesar procesării nu este foarte mare. Piesele procesate sunt alese în mod aleator.
* losingPenalty: Este una dintre variabilele care influențează în mod direct algoritmul, stabilind care este penalizarea primită de o stare în care nu mai există nicio mutare validă. Această variabilă ar trebui să tindă către infinit, fiind negată în cod. Valoarea implicită a variabilei este de 100000.0.
* ascendingTilesBonus: Afectează în ce măsură cea de-a doua euristică influențează scorul, având o valoare implicită de 0.5.
* penaltyAmount: Afectează în ce măsură cea de-a treia euristică influențează scorul, având o valoare implicită de 1.5.
* positionalBonus: Afectează în ce măsură cea de-a patra euristică influențează scorul, având o valoare implicită de 0.3.
* emptyTileBonus: Afectează în ce măsură cea de-a cincea euristică influențează scorul, având o valoare implicită de 0.075. Important de reținut este faptul că ultima euristică utilizată nu poate fi modificată, scorul putând fi influențat doar de cele cinci variabile menționate anterior. Valorile implicite ale acestor variabile sunt găsite pe cale experimentală și pot fi îmbunătățite pentru a ajunge la rezultate mai bune.
* benchmark: Este o variabilă de tip logic. Dacă este adevărată, atât în consolă, cât și într-un fișier vor fi scrise statistici despre jocurile desfășurate. Vor fi contorizate numărul, scorul, pisa de valoare maximă și numărul de mutări ale jocului curent, scorul mediu și procentul de jocuri în care s-a ajuns la o anumită piesă maximă. Modul recomandat de a folosi această variabilă este de a da valorile “true” și 0 variabilelor “autorestart” și ”autorestartDuration”. Astfel, după terminarea unui joc se va începe altul în mod automat și instantaneu, iar utilizatorul poate lăsa programul să ruleze fără a fi nevoie de intervenție, salvându-se la final date despre modul în care se comportă programul. Numele fișierului nu poate fi schimbat, însă programul nu va scrie peste un fișier deja existent. Numele fișierului este “DPOTest.txt”, iar dacă deja există un fișier cu același nume, se va atașa sufixul “(nr)”, unde “nr” este numărul de încercări eșuate de a scrie într-un fișier (Dacă există fișierul “DPOTest.txt”, se încearcă scrierea în “DPOTest (1).txt”, iar dacă și acesta există se încearcă în “DPOTest (2).txt” etc.). Un fragment dintr-un astfel de fișier este:

...

Game #91: Score: 3260, Max tile: 2048, Moves: 1510

Average score: 3536.901099

512: 1.099%

1024: 19.780%

2048: 67.033%

4096: 12.088%

Game #92: Score: 3168, Max tile: 2048, Moves: 1448

Average score: 3532.891304

512: 1.087%

1024: 19.565%

2048: 67.391%

4096: 11.957%

...

Acesta este doar un fragment dintr-un fișier complet în care au fost jucate 100 de meciuri. Din această secventă reiese faptul ca jocul cu numărul 91 s-a terminat cu scorul 3260, piesa cu valoare maximă a fost 2048, iar numărul de mutări de la început și până la sfârșitul jocului a fost de 1510. Următoarele 5 linii oferă informații legate și de jocurile precedente. A doua linie indică media tuturor scorurilor obținute, iar următoarele linii indică procentajul de jocuri în care o piesă a fost maximă. În 1.099% din jocuri piesa maximă a fost 512, în 19.78% din jocuri a fost 1024 etc. Analog se poate analiza și cea de-a doua parte a fragmentului.

Utilizarea unui astfel de mod de a analiza performanța algoritmului este esențială pentru a afla care sunt rezultatele reale ale aplicației. Un singur joc nu este suficient pentru a decide eficacitatea programului, iar urmărirea manuală a punctajelor este mult prea dificilă.

Performanțele aplicației

Așadar, cum se comportă aplicația în practică? Pentru a realiza această statistică, fișierul ”config.xml” a cuprins următoarele linii:

<penaltyAmount>1.5</penaltyAmount>

<emptyTileBonus>0.075</emptyTileBonus>

<positionalBonus>0.3</positionalBonus> <ascendingTilesBonus>0.5</ascendingTilesBonus> <losingPenalty>100000.0</losingPenalty>

Următorul tabel ilustrează comportamentul aplicației în funcție de celelalte două variabile ce țin de algoritm (“maxDepth” și “maxCheck”):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | maxDepth=2  maxCheck=-1 | maxDepth=3  maxCheck=-1 | maxDepth=3  maxCheck=3 | maxDepth=3  maxCheck=4 | maxDepth=4  maxCheck=4 |
| Scor mediu | 2477.060 | 3533.640 | 4176.333 | 4099.133 | 3703.600 |
| Piesă maximă 512 | 10.000% | 1.000% | 0.000% | 0.000% | 0.000% |
| Piesă maximă 1024 | 49.000% | 19.000% | 6.667% | 3.333% | 26.667% |
| Piesă maximă 2048 | 41.000% | 69.000% | 76.667% | 76.667% | 53.333% |
| Piesă maximă 4096 | 0.000% | 11.000% | 16.667% | 20.000% | 20.000% |
| Piesă atinsă 512 | 100.000% | 100.000% | 100.000% | 100.000% | 100.000% |
| Piesă atinsă 1024 | 90.000% | 99.000% | 100.000% | 100.000% | 100.000% |
| Piesă atinsă 2048 | 41.000% | 80.000% | 93.333% | 96.667% | 73.333% |
| Piesă atinsă 4096 | 0.000% | 11.000% | 16.667% | 20.000% | 20.000% |
| Timp pe mutare (medie) | 15.8 ms | 55 ms | 122 ms | 323 ms | 407 ms |
| Timp pe joc (medie) | 3.6 s | 73 s  1.46 min | 264 s  4.4 min | 588 s  9.8 min | 684 s  11.4 min |
| Mutări pe secundă (medie) | 63 | 18 | 8.11 | 3.10 | 2.45 |
| Număr de mutări pe joc (medie) | 1134.15 | 1578.53 | 2141.33 | 1858.73 | 1677.46 |
| Număr de teste | 100 | 100 | 30 | 30 | 15 |

Prin piesă maximă se înțelege piesa cu valoarea maximă la care a ajuns programul la sfârșitul unui joc, iar prin piesă atinsă se înțelege că la un moment dat acea piesă a fost prezentă pe tabla de joc. Un joc este câștigat în momentul în care s-a ajuns la piesa cu numărul 2048, din această cauză linia a 9-a este evidențiată. Chiar dacă timpul mediu pe mutare indică o valoare precisă, pentru cazurile indicate de coloanele 4 și 5, timpul oscilează între a avea timpul mediu de pe coloana 3 (sunt suficiente spații libere și nu este nevoia aplicării unei precizii mari) și a avea timpi mai mari, care pot depăși chiar și o secundă, respectiv două pe mutare. Acest lucru este cauzat de creșterea preciziei în momente cruciale ale jocului. Pentru a obține informațiile din tabelul de mai sus s-au introdus în fișierul de intrare datele corespunzătoare, după care au fost simulate un anumit număr de meciuri. Din cauza timpului relativ mare de procesare, au fost simulate mai puține meciuri pentru setările indicate de ultimele 3 coloane.

Codul sursă

Structura codului sursă este următoarea:

src

| main.cpp – Fișierul care conține funcția main a programului

| pch.cpp – Fișier necesar pentru utilizarea fișierelor de tip header precompilate

| pch.h – Precompiled header. Fișier inclus în toate celelalte fișiere .cpp. Include librăriile de bază.

|

+---game

| | game.cpp

| | game.h – Clasă responsabilă de management-ul aplicației

| | resources.cpp

| | resources.h – Încarcă fontul implicit utilizat de butoane și label

| | scene.cpp

| | scene.h – Clasă responsabilă de management-ul unei singure scene/nivel

| | uiUtils.h – Conține obiecte ajutătoare care țin de interfață, precum Button, Label și Slider

| |

| \---dpo

| dpoController.h – Clasă ce definește un controller generic

| dpoGame.cpp

| dpoGame.h – Obiect respondabil de simularea jocului

| dpoHumanController.h – Controller uman. Mutările sunt controlate de utilizator

| dpoScene.cpp

| dpoScene.h – Scena jocului propriu-zis. Responsabilă de interfața grafică

| dpoTreeAI.cpp

| dpoTreeAI.h – Controller-ul care folosește algoritmul Expectimax

|

+---graphics

| font.cpp

| font.h – Definește un font încărcat dintr-un fișier

| input.cpp

| input.h – Preia informațiile transmise de utilizator prin: mouse, tastatură, butonul de ieșire etc.

| renderer.cpp

| renderer.h – Un wrapper al funcțiilor din SDL pentru a utiliza obiectele definite în acest proiect

| texture.cpp

| texture.h – Definește o textură care provine dintr-o imagine încărcată dintr-un fișier sau generată

| timer.cpp

| timer.h – Temporizator utilizat pentru a număra numărul de cadre pe secundă

| window.cpp

| window.h – Un wrapper al ferestrei din SDL

|

+---math

| math.h – Include toate fișierele din directorul acesta

| matrix3.h – Matrice de 3x3. Nefolosită în acest proiect

| matrix4.h – Matrice de 4x4. Nefolosită în acest proiect

| quaternion.h – Quaternion. Nefolosit în acest proiect

| simpleMath.h – Clasă de matematică. Wrapper al STL-ului cu câteva funcții noi

| vector2.h – Vector cu 2 elemente

| vector3.h – Vector cu 3 elemente

| vector4.h – Vector cu 4 elemente

|

\---utils

files.cpp

files.h – Funcții ajutătoare pentru citire și scriere de fișiere

string.cpp

string.h – Funcții ajutătoare pentru lucrul cu șiruri de caractere

xml.cpp

xml.h – XML loader și writer

**src/main.cpp:**

#include "pch.h"

#include "graphics\window.h"

#include "math\math.h"

#include "graphics\renderer.h"

#include "graphics\font.h"

#include "game\game.h"

#include "game\dpo\dpoScene.h"

#include "game\resources.h"

class MyGame : public Game

{

public:

MyGame() :

Game(1280, 720, "2048")

{

Resources::Load();

scene = new DPOScene(this);

}

virtual ~MyGame()

{

Resources::Unload();

delete scene;

}

virtual void Update()

{

t += window->timer->GetDeltaTime();

if (t >= 1)

{

t--;

LOG\_VERB("FPS: %d\n", fps);

fps = 0;

}

fps++;

Game::Update();

}

virtual void Render()

{

window->renderer->SetColor(Vector4i(255, 255, 255, 255));

window->renderer->ClearScreen();

Game::Render();

}

float t = 0;

int fps = 0;

};

int main(int argc, char\*\* args)

{

MyGame game;

game.Start();

return 0;

}

**src/pch.h:**

#pragma once

#include <cstring>

#include <cstdlib>

#include <cmath>

#ifdef PRODUCTION

#define LOG\_INFO(...)

#define LOG\_VERB(...)

#elif defined NDEBUG

#include <cstdio>

#define LOG\_INFO(...) printf(\_\_VA\_ARGS\_\_)

#define LOG\_VERB(...)

#else

#include <cstdio>

#define LOG\_INFO(...) printf(\_\_VA\_ARGS\_\_)

#define LOG\_VERB(...) printf(\_\_VA\_ARGS\_\_)

#endif

#include <algorithm>

#include <vector>

#include <queue>

#include <string>

#include <bitset>

using std::pair;

using std::sort;

using std::vector;

using std::queue;

using std::priority\_queue;

using std::string;

typedef int8\_t int8;

typedef int16\_t int16;

typedef int32\_t int32;

typedef int64\_t int64;

typedef uint8\_t uint8;

typedef uint16\_t uint16;

typedef uint32\_t uint32;

typedef uint64\_t uint64;

typedef long double lfloat;

typedef uint8 byte;

typedef unsigned int uint;

typedef long long llong;

typedef unsigned long long ullong;

**src/pch.cpp:**

#include "pch.h"

**src/math/vector2.h:**

#pragma once

#include "pch.h"

#include "simpleMath.h"

template <class T, class FP>

struct Vector2

{

union

{

struct { T x, y; };

struct { T r, g; };

struct { T s, t; };

struct { T data[2]; };

};

static Vector2<T, FP> Up() { return Vector2<T, FP>((T)0, (T)1); }

static Vector2<T, FP> Right() { return Vector2<T, FP>((T)1, (T)0); }

static Vector2<T, FP> Zero() { return Vector2<T, FP>((T)0, (T)0); }

static Vector2<T, FP> One() { return Vector2<T, FP>((T)1, (T)1); }

Vector2() : x(0), y(0) {}

Vector2(const T& a) : x(a), y(a) {}

Vector2(const T& xx, const T& yy) : x(xx), y(yy) {}

Vector2(const Vector2<T, FP>& v) = default;

FP Dot(const Vector2<T, FP>& a) const { return (FP)x \* (FP)a.x + (FP)y \* (FP)a.y; }

FP Length() const { return Sqrt<T>((FP)x \* (FP)x + (FP)y \* (FP)y); }

FP LengthSquared() const { return (FP)x \* (FP)x + (FP)y \* (FP)y; }

FP Distance(const Vector2<T, FP>& a) const { return Sqrt<T>( ((FP)x - (FP)a.x) \* ((FP)x - (FP)a.x) + ((FP)y - (FP)a.y) \* ((FP)y - (FP)a.y)); }

FP DistanceSquared(const Vector2<T, FP>& a) const { return ((FP)x - (FP)a.x) \* ((FP)x - (FP)a.x) + ((FP)y - (FP)a.y) \* ((FP)y - (FP)a.y); }

Vector2<T, FP> Normalized() const { FP l = Length(); return Vector2<T, FP>((T)((FP)x / l), (T)((FP)y / l)); }

Vector2<T, FP> Added(const T& a) const { return Vector2<T, FP>(x + a, y + a); }

Vector2<T, FP> Subtracted(const T& a) const { return Vector2<T, FP>(x - a, y - a); }

Vector2<T, FP> Multiplied(const T& a) const { return Vector2<T, FP>(x \* a, y \* a); }

Vector2<T, FP> Divided(const T& a) const { return Vector2<T, FP>(x / a, y / a); }

Vector2<T, FP> Added(const Vector2<T, FP>& a) const { return Vector2<T, FP>(x + a.x, y + a.y); }

Vector2<T, FP> Subtracted(const Vector2<T, FP>& a) const { return Vector2<T, FP>(x - a.x, y - a.y); }

Vector2<T, FP> Multiplied(const Vector2<T, FP>& a) const { return Vector2<T, FP>(x \* a.x, y \* a.y); }

Vector2<T, FP> Divided(const Vector2<T, FP>& a) const { return Vector2<T, FP>(x / a.x, y / a.y); }

bool Equals(const Vector2<T, FP>& a) const { return x == a.x && y == a.y; }

Vector2<T, FP> Lerp(const Vector2<T, FP>& a, FP v) const { return Multiplied(FP(1) - v) + a \* (v); }

Vector2<T, FP> Reflect(const Vector2<T, FP>& normal) const { return Subtracted((FP)2 \* Dot(normal) \* normal); }

Vector2<T, FP> Refract(const Vector2<T, FP>& normal, const FP& eta) const

{

T ndoti = Dot(normal);

T k = (FP)1 - eta \* eta \* ((FP)1 - ndoti \* ndoti);

if (k < (FP)0)

return Zero();

return Multiplied(eta) - (eta \* ndoti + Sqrt<T>(k)) \* normal;

}

Vector2<T, FP>& Normalize() { FP l = Length(); x /= l; y /= l; return \*this; }

Vector2<T, FP>& Add(const T& a) { x += a; y += a; return \*this; }

Vector2<T, FP>& Subtract(const T& a) { x -= a; y -= a; return \*this; }

Vector2<T, FP>& Multiply(const T& a) { x \*= a; y \*= a; return \*this; }

Vector2<T, FP>& Divide(const T& a) { x /= a; y /= a; return \*this; }

Vector2<T, FP>& Add(const Vector2<T, FP>& a) { x += a.x; y += a.y; return \*this; }

Vector2<T, FP>& Subtract(const Vector2<T, FP>& a) { x -= a.x; y -= a.y; return \*this; }

Vector2<T, FP>& Multiply(const Vector2<T, FP>& a) { x \*= a.x; y \*= a.y; return \*this; }

Vector2<T, FP>& Divide(const Vector2<T, FP>& a) { x /= a.x; y /= a.y; return \*this; }

Vector2<T, FP> operator+(const Vector2<T, FP>& a) const { return Added(a); }

Vector2<T, FP> operator-(const Vector2<T, FP>& a) const { return Subtracted(a); }

Vector2<T, FP> operator\*(const Vector2<T, FP>& a) const { return Multiplied(a); }

Vector2<T, FP> operator/(const Vector2<T, FP>& a) const { return Divided(a); }

Vector2<T, FP> operator+(const T& a) const { return Added(a); }

Vector2<T, FP> operator-(const T& a) const { return Subtracted(a); }

Vector2<T, FP> operator\*(const T& a) const { return Multiplied(a); }

Vector2<T, FP> operator/(const T& a) const { return Divided(a); }

Vector2<T, FP> operator-() const { return Vector2<T, FP>(-x, -y); }

friend Vector2<T, FP> operator+(const T& a, const Vector2<T, FP>& t) { return t.Added(a); }

friend Vector2<T, FP> operator-(const T& a, const Vector2<T, FP>& t) { return t.Subtracted(a); }

friend Vector2<T, FP> operator\*(const T& a, const Vector2<T, FP>& t) { return t.Multiplied(a); }

friend Vector2<T, FP> operator/(const T& a, const Vector2<T, FP>& t) { return Vector2<T, FP>(a).Divided(t); }

Vector2<T, FP>& operator+=(const Vector2<T, FP>& a) { return Add(a); }

Vector2<T, FP>& operator-=(const Vector2<T, FP>& a) { return Subtract(a); }

Vector2<T, FP>& operator\*=(const Vector2<T, FP>& a) { return Multiply(a); }

Vector2<T, FP>& operator/=(const Vector2<T, FP>& a) { return Divide(a); }

Vector2<T, FP>& operator+=(const T& a) { return Add(a); }

Vector2<T, FP>& operator-=(const T& a) { return Subtract(a); }

Vector2<T, FP>& operator\*=(const T& a) { return Multiply(a); }

Vector2<T, FP>& operator/=(const T& a) { return Divide(a); }

bool operator==(const Vector2<T, FP>& a) const { return Equals(a); }

bool operator!=(const Vector2<T, FP>& a) const { return !Equals(a); }

T& operator[](const uint32& index) { return data[index]; }

const T& operator[](const uint32& index) const { return data[index]; }

};

template <typename T, typename FP>

static FP Dot(const Vector2<T, FP>& a, Vector2<T, FP>& b) { return a.Dot(b); }

template <typename T, typename FP>

static Vector2<T, FP> Normalize(const Vector2<T, FP>& a) { return a.Normalized(); }

template <typename T, typename FP>

static Vector2<T, FP> Lerp(const Vector2<T, FP>& a, const Vector2<T, FP>& b, const FP& c) { return a.Lerp(b, c); }

template <typename T, typename FP>

static Vector2<T, FP> Reflect(const Vector2<T, FP>& ray, const Vector2<T, FP>& normal) { return ray.Reflect(normal); }

template <typename T, typename FP>

static Vector2<T, FP> Refract(const Vector2<T, FP>& ray, const Vector2<T, FP>& normal, const FP& eta) { return ray.Refract(normal, eta); }

**// mai există și vector3 și vector4, dar singura diferență e numărul de componente**

**Fragment din src/math/simpleMath.h:**

template <typename T>

static T Clamp(const T& a, const T& b, const T& c) { return a < b ? b : ((a > c) ? c : a); }

template <typename T>

static T Clamp01(const T& a) { return Clamp<T>(a, 0, 1); }

template <typename T>

static T Pow(const T& a, const int32& b)

{

T x = a, y = 1;

int32 n = b;

if (n < 0)

{

x = T(1.0) / x;

n = -n;

}

if (n == 0) return 1;

while (n > 1)

{

if (n % 2 == 0)

{

x \*= x;

n /= 2;

}

else

{

y \*= x;

x \*= x;

n = (n - 1) / 2;

}

}

return x \* y;

}

**src/math/math.h:**

#pragma once

#include "pch.h"

#include "simpleMath.h"

#include "vector2.h"

#include "vector3.h"

#include "vector4.h"

typedef Vector2<float, float> Vector2f;

typedef Vector2<int32, float> Vector2i;

typedef Vector2<double, double> Vector2d;

typedef Vector2<lfloat, lfloat> Vector2l;

typedef Vector3<float, float> Vector3f;

typedef Vector3<int32, float> Vector3i;

typedef Vector3<double, double> Vector3d;

typedef Vector3<lfloat, lfloat> Vector3l;

typedef Vector4<float, float> Vector4f;

typedef Vector4<int32, float> Vector4i;

typedef Vector4<double, double> Vector4d;

typedef Vector4<lfloat, lfloat> Vector4l;

**src/utils/string.h:**

#pragma once

#include "pch.h"

class String

{

public:

static std::vector<std::string> Split(const std::string& str, const std::string& separators);

// Doesn't care if there are multiple consecutive separators

static std::vector<std::string> SplitMultiple(const std::string& str, const std::string& separators);

static std::string GetDirectory(const std::string& path);

static std::string GetFilename(const std::string& path);

static std::string ChangeDirectory(const std::string& currentDirectory, const std::string& directory);

// Returns -1 when not inside, otherwise returns the position

static int IsInside(const std::string& str, char ch);

static std::string ToLower(const std::string& str);

static std::string ToUpper(const std::string& str);

template <class T>

static std::string ToString(const T& v) {

return std::to\_string(v);

}

template <class T>

static std::string ToStringFloat(const T& v, int digits) {

std::string res = std::to\_string(v);

size\_t i = res.find('.');

if (i + 1 + digits > res.size() || i == std::string::npos)

return res;

return res.substr(0, i + 1 + digits);

}

template <class T>

static T ValueOf(const std::string& str) { LOG\_VERB("Cannot convert to string!\n"); }

template <>

static int ValueOf(const std::string& str) {

return atoi(str.c\_str());

}

template <>

static int64 ValueOf(const std::string& str) {

return atoll(str.c\_str());

}

template <>

static double ValueOf(const std::string& str) {

return atof(str.c\_str());

}

template <>

static float ValueOf(const std::string& str) {

return (float)atof(str.c\_str());

}

template <>

static std::string ValueOf(const std::string& str) {

return str;

}

template <>

static bool ValueOf(const std::string& str) {

std::string lower = ToLower(str);

if (lower == "false" || str == "0") return false;

if (lower == "true" || str == "1") return true;

return false;

}

};

**src/utils/string.cpp:**

#include "pch.h"

#include "string.h"

std::vector<std::string> String::Split(const std::string& str, const std::string& separators)

{

std::vector<std::string> res;

int beg = 0;

for (uint i = 0; i < str.size(); i++)

{

bool isSeparator = IsInside(separators, str[i]) != -1;

if (isSeparator)

{

res.push\_back(str.substr(beg, i - beg));

beg = i;

beg++;

}

}

res.push\_back(str.substr(beg));

return res;

}

int String::IsInside(const std::string& str, char ch)

{

for (uint i = 0; i < str.size(); i++)

if (str[i] == ch) return i;

return -1;

}

std::string String::ToLower(const std::string& str)

{

std::string res = str;

for (int i = 0; i < res.size(); i++)

res[i] = tolower(res[i]);

return res;

}

std::string String::ToUpper(const std::string& str)

{

std::string res = str;

for (int i = 0; i < res.size(); i++)

res[i] = toupper(res[i]);

return res;

}

std::vector<std::string> String::SplitMultiple(const std::string& str, const std::string& separators)

{

std::vector<std::string> res;

auto beg = str.end();

auto it = str.begin();

for (it = str.begin(); it != str.end(); it++)

{

bool isSeparator = std::find(separators.begin(), separators.end(), \*it) != separators.end();

if (beg == str.end() && !isSeparator)

beg = it;

else if (isSeparator && beg != str.end())

{

res.push\_back(str.substr(beg - str.begin(), it - beg));

beg = str.end();

}

}

if (beg != str.end())

res.push\_back(str.substr(beg - str.begin()));

return res;

}

std::string String::GetDirectory(const std::string& path)

{

for (int i = path.size(); i >= 0; i--)

{

if (path[i] == '\\' || path[i] == '/')

return path.substr(0, i + 1);

}

return "";

}

std::string String::GetFilename(const std::string& path)

{

for (int i = path.size(); i >= 0; i--)

{

if (path[i] == '\\' || path[i] == '/')

return path.substr(i + 1);

}

return path;

}

std::string String::ChangeDirectory(const std::string& currentDirectory, const std::string& directory)

{

if (currentDirectory.empty() || currentDirectory == "\\")

return directory;

return currentDirectory + '\\' + directory;

}

**src/utils/files.h:**

#pragma once

#include "pch.h"

class Files

{

public:

static char\* ReadFileBinary(const std::string& path, uint64& len);

static std::string ReadFileText(const std::string& path, size\_t bufferSize = 4096);

static void WriteTextFile(const std::string& path, const std::string& text);

};

**src/utils/files.cpp:**

#include "pch.h"

#include "files.h"

char\* Files::ReadFileBinary(const std::string& path, uint64& len)

{

FILE\* file = fopen(path.c\_str(), "rb");

\_fseeki64(file, 0, SEEK\_END);

len = \_ftelli64(file);

\_fseeki64(file, 0, SEEK\_SET);

char\* result = new char[size\_t(len) + 1];

fread(result, 1, size\_t(len), file);

result[len] = 0;

fclose(file);

return result;

}

std::string Files::ReadFileText(const std::string& path, size\_t bufferSize)

{

FILE\* file = fopen(path.c\_str(), "r");

std::string result;

char\* buffer = (char\*) alloca(bufferSize);

char\* line;

while ((line = fgets(buffer, bufferSize, file)) != NULL)

result += line;

fclose(file);

return result;

}

void Files::WriteTextFile(const std::string& path, const std::string& text)

{

FILE\* file = fopen(path.c\_str(), "w");

fprintf(file, "%s", text.c\_str());

fclose(file);

}

**src/utils/xml.h:**

#pragma once

#include "pch.h"

#include "string.h"

class XMLAttribute

{

public:

std::string name;

std::string value;

};

class XMLNode

{

public:

XMLNode();

~XMLNode();

const XMLAttribute\* GetAttribute(const std::string& name) const;

XMLAttribute\* GetAttribute(const std::string& name);

const XMLNode\* GetNode(const std::string& name, uint count = 0) const;

XMLNode\* GetNode(const std::string& name, uint count = 0);

const XMLNode\* GetNodeWithAttribute(const std::string& nodeName, const std::string& attributeName, const std::string& attributeValue) const;

XMLNode\* GetNodeWithAttribute(const std::string& nodeName, const std::string& attributeName, const std::string& attributeValue);

std::vector<XMLAttribute> attributes;

std::vector<XMLNode\*> children;

std::string content;

std::string name;

};

class XMLFile

{

public:

XMLFile();

~XMLFile();

// Finds the node that follows the path provided

// The root node's name is ignored, as an example,

// "graphics/fxaa" is a correct path for the file

// <settings>

// <graphics>

// <fxaa>1</fxaa>

// ...

//

// If the node exists, it gets assigned that value

// Returns true if the value was changed

template <typename T>

bool SetVariable(const std::string& path, T& value)

{

bool found;

XMLNode\* current = root;

std::vector<std::string> parts = String::Split(path, "/\\");

for(uint i = (parts[0] == "" || parts[0] == root->name) ? 1 : 0; i < parts.size(); i++)

{

found = false;

for (uint j = 0; j < current->children.size(); j++)

{

if (current->children[j]->name == parts[i])

{

current = current->children[j];

found = true;

break;

}

}

if (!found)

return false;

}

value = String::ValueOf<T>(current->content);

return true;

}

XMLNode\* prolog;

XMLNode\* root;

private:

};

class XMLFileLoader

{

public:

static XMLFile\* LoadFile(const std::string& path);

static XMLFile\* LoadFileFromMemory(const std::string& text);

private:

static bool HasNode();

static XMLNode\* ParseNode();

static bool HasAttribute();

static XMLAttribute ParseAttribute();

static bool HasContent();

static std::string ParseContent();

static void JumpSpaces();

static void SkipToFirst(char ch);

static void SkipToFirstSpace();

static void SkipToFirstSpaceOr(char ch);

static void SkipToFirstSpaceOr(char ch1, char ch2, char ch3);

static std::string GetLineAndColumn(const std::string& text);

static XMLNode\* ParseProlog();

static std::string RemoveComments(const std::string& str);

static bool HasProlog();

static const char\* str;

static std::string error;

};

struct XMLOutputOptions

{

bool indent = true;

int spacesToIndent = 2;

};

class XMLFileWriter

{

public:

static std::string WriteToMemory(XMLFile\* xml, XMLOutputOptions options = XMLOutputOptions());

static void WriteToFile(XMLFile\* xml, const std::string& path, XMLOutputOptions options = XMLOutputOptions());

private:

static void WriteNode(XMLNode\* node, uint depth);

static void WriteFormatted(const std::string& text);

static void WriteIndent(uint depth);

static void WriteProlog(XMLNode\* node);

static std::stringstream ss;

static XMLOutputOptions options;

};

**src/utils/xml.cpp:**

#include "pch.h"

#include "xml.h"

#include <sstream>

#include "string.h"

#include "files.h"

#define EOF\_CHECK() if(\*str == 0) {error = "Reached end of file too soon!"; return nullptr;}

#define SAFE\_ADVANCE() str++; EOF\_CHECK()

#define EOF\_CHECK\_ATTRIBUTE() if(\*str == 0) return res

#define SAFE\_ADVANCE\_ATTRIBUTE() str++; EOF\_CHECK\_ATTRIBUTE()

const char\* XMLFileLoader::str = nullptr;

std::string XMLFileLoader::error = "";

std::stringstream XMLFileWriter::ss;

XMLOutputOptions XMLFileWriter::options;

XMLNode::XMLNode()

{

}

XMLNode::~XMLNode()

{

for (uint i = 0; i < children.size(); i++)

if(children[i])

delete children[i];

}

const XMLAttribute\* XMLNode::GetAttribute(const std::string& name) const

{

for (uint i = 0; i < attributes.size(); i++)

if (attributes[i].name == name)

return &attributes[i];

return nullptr;

}

XMLAttribute\* XMLNode::GetAttribute(const std::string& name)

{

for (uint i = 0; i < attributes.size(); i++)

if (attributes[i].name == name)

return &attributes[i];

return nullptr;

}

const XMLNode\* XMLNode::GetNode(const std::string& name, uint count) const

{

for (uint i = 0; i < children.size(); i++)

if (children[i]->name == name)

{

if (count == 0)

return children[i];

count--;

}

return nullptr;

}

XMLNode\* XMLNode::GetNode(const std::string& name, uint count)

{

for (uint i = 0; i < children.size(); i++)

if (children[i]->name == name)

{

if (count == 0)

return children[i];

count--;

}

return nullptr;

}

const XMLNode\* XMLNode::GetNodeWithAttribute(const std::string& nodeName, const std::string& attributeName, const std::string& attributeValue) const

{

for(uint i = 0; i < children.size(); i++)

if (children[i]->name == nodeName)

{

const XMLAttribute\* attribute = children[i]->GetAttribute(attributeName);

if (attribute != nullptr)

{

if (attribute->value == attributeValue)

return children[i];

}

}

return nullptr;

}

XMLNode\* XMLNode::GetNodeWithAttribute(const std::string& nodeName, const std::string& attributeName, const std::string& attributeValue)

{

for (uint i = 0; i < children.size(); i++)

if (children[i]->name == nodeName)

{

const XMLAttribute\* attribute = children[i]->GetAttribute(attributeName);

if (attribute != nullptr)

{

if (attribute->value == attributeValue)

return children[i];

}

}

return nullptr;

}

XMLFile::XMLFile()

{

}

XMLFile::~XMLFile()

{

if (root) delete root;

if (prolog) delete prolog;

}

std::string XMLFileLoader::GetLineAndColumn(const std::string& text)

{

if (\*str == 0) return "";

uint col = 0, line = 1;

uint len = str - text.data();

for (uint i = 0; i < len; i++)

{

if (text[i] == '\n')

{

line++;

col = 0;

}

else col++;

}

return "Line: " + String::ToString(line) + "; Col: " + String::ToString(col);

}

XMLFile\* XMLFileLoader::LoadFile(const std::string& path)

{

return LoadFileFromMemory(Files::ReadFileText(path));

}

XMLFile\* XMLFileLoader::LoadFileFromMemory(const std::string& text)

{

std::string preparedText = RemoveComments(text);

str = preparedText.data();

error = "";

XMLFile\* file = new XMLFile();

if (HasProlog())

file->prolog = ParseProlog();

else file->prolog = nullptr;

file->root = ParseNode();

if (error != "")

{

LOG\_INFO("Could not parse the XML file!\n");

LOG\_INFO("Error: %s\n", error.c\_str());

LOG\_INFO("%s\n", GetLineAndColumn(preparedText).c\_str());

return nullptr;

}

return file;

}

void XMLFileLoader::SkipToFirst(char ch)

{

while (\*str != ch && \*str != 0)

str++;

}

void XMLFileLoader::SkipToFirstSpace()

{

while (\*str != ' ' && \*str != '\n' && \*str != '\t' && \*str != '\r' && \*str != 0)

str++;

}

void XMLFileLoader::SkipToFirstSpaceOr(char ch)

{

while (\*str != ' ' && \*str != '\n' && \*str != '\t' && \*str != '\r' && \*str != ch && \*str != 0)

str++;

}

void XMLFileLoader::SkipToFirstSpaceOr(char ch1, char ch2, char ch3)

{

while (\*str != ' ' && \*str != '\n' && \*str != '\t' && \*str != '\r' && \*str != ch1 && \*str != ch2 && \*str != ch3 && \*str != 0)

str++;

}

bool XMLFileLoader::HasNode()

{

const char\* old = str;

if (\*str != '<') return false;

str++;

JumpSpaces();

if (\*str == '/' || \*str == '\\')

{

str = old;

return false;

}

str = old;

return true;

}

bool XMLFileLoader::HasProlog()

{

const char\* old = str;

if (\*str != '<') return false;

str++;

JumpSpaces();

if (\*str == '?')

{

str = old;

return true;

}

str = old;

return false;

}

XMLNode\* XMLFileLoader::ParseProlog()

{

XMLNode\* res = new XMLNode();

JumpSpaces();

EOF\_CHECK();

if (\*str != '<')

{

error = std::string("Unexpected token '") + \*str + "' found! Expected '<'!";

return nullptr;

}

SAFE\_ADVANCE();

if (\*str != '?')

{

error = std::string("Unexpected token '") + \*str + "' found! Expected '?'!";

return nullptr;

}

SAFE\_ADVANCE();

const char\* nameBegin = str;

SkipToFirstSpaceOr('>', '?', '?');

EOF\_CHECK();

res->name = std::string(nameBegin, str);

if (res->name != "xml")

{

error = "The prolog's name must be 'xml', not '" + res->name + "'!";

return nullptr;

}

while (HasAttribute())

{

res->attributes.push\_back(ParseAttribute());

EOF\_CHECK();

}

EOF\_CHECK();

if (\*str != '?')

{

error = std::string("Unexpected token '") + \*str + "' found! Expected '?'!";

return nullptr;

}

SAFE\_ADVANCE();

JumpSpaces();

EOF\_CHECK();

if (\*str != '>')

{

error = std::string("Unexpected token '") + \*str + "' found! Expected '>'!";

return nullptr;

}

SAFE\_ADVANCE();

return res;

}

XMLNode\* XMLFileLoader::ParseNode()

{

XMLNode\* res = new XMLNode();

JumpSpaces();

EOF\_CHECK();

if (\*str != '<')

{

error = std::string("Unexpected token '") + \*str + "' found! Expected '<'!";

return nullptr;

}

SAFE\_ADVANCE();

const char\* nameBegin = str;

SkipToFirstSpaceOr('>', '/', '\\');

EOF\_CHECK();

res->name = std::string(nameBegin, str);

while (HasAttribute())

{

res->attributes.push\_back(ParseAttribute());

EOF\_CHECK();

}

EOF\_CHECK();

if (\*str == '/' || \*str == '\\')

{

SAFE\_ADVANCE();

JumpSpaces();

EOF\_CHECK();

if (\*str != '>')

{

error = std::string("Unexpected token '") + \*str + "' found! Expected '>'!";

return nullptr;

}

SAFE\_ADVANCE();

return res;

}

if (\*str != '>')

{

error = std::string("Unexpected token '") + \*str + "' found! Expected '>'!";

return nullptr;

}

SAFE\_ADVANCE();

JumpSpaces();

EOF\_CHECK();

if (HasNode())

{

while (HasNode())

{

XMLNode\* node = ParseNode();

if (node == nullptr)

return nullptr;

res->children.push\_back(node);

JumpSpaces();

EOF\_CHECK();

}

}

else // has content

{

res->content = ParseContent();

EOF\_CHECK();

if (error != "")

return nullptr;

}

if (\*str != '<')

{

error = std::string("Unexpected token '") + \*str + "' found! Expected '<'!";

return nullptr;

}

SAFE\_ADVANCE();

JumpSpaces();

EOF\_CHECK();

if (\*str != '/' && \*str != '\\')

{

error = std::string("Unexpected token '") + \*str + "' found! Expected '/'!";

return nullptr;

}

SAFE\_ADVANCE();

JumpSpaces();

EOF\_CHECK();

const char\* endNameBeg = str;

SkipToFirstSpaceOr('>');

EOF\_CHECK();

std::string endname = std::string(endNameBeg, str);

if (endname != res->name)

{

error = "The closing tag ('" + endname + "') does not match the starting tag ('" + res->name + "')!";

return nullptr;

}

JumpSpaces();

EOF\_CHECK();

if (\*str != '>')

{

error = std::string("Unexpected token '") + \*str + "' found! Expected '>'!";

return nullptr;

}

str++;

return res;

}

bool XMLFileLoader::HasAttribute()

{

JumpSpaces();

return isalpha(\*str) || \*str == '\_';

}

XMLAttribute XMLFileLoader::ParseAttribute()

{

XMLAttribute res;

const char\* attNameBeg = str;

SkipToFirst('=');

EOF\_CHECK\_ATTRIBUTE();

res.name = std::string(attNameBeg, str);

SAFE\_ADVANCE\_ATTRIBUTE();

char quote = \*str;

if (quote != '\'' && quote != '\"')

{

error = "Attributes must be surrounded by quotes ot apostrophies!";

return res;

}

SAFE\_ADVANCE\_ATTRIBUTE();

const char\* attValueBeg = str;

SkipToFirst(quote);

EOF\_CHECK\_ATTRIBUTE();

res.value = std::string(attValueBeg, str);

SAFE\_ADVANCE\_ATTRIBUTE();

return res;

}

bool XMLFileLoader::HasContent()

{

return \*str != '<';

}

std::string XMLFileLoader::ParseContent()

{

std::stringstream ss;

while (\*str != '<' && \*str != 0)

{

if (\*str == '&')

{

if (strncmp(str, "&lt;", 4) == 0) ss << '<', str += 4;

else if (strncmp(str, "&gt;", 4) == 0) ss << '>', str += 4;

else if (strncmp(str, "&amp;", 5) == 0) ss << '&', str += 5;

else if (strncmp(str, "&apos;", 6) == 0) ss << '\'', str += 6;

else if (strncmp(str, "&quot;", 6) == 0) ss << '\"', str += 6;

else

{

error = "Unexpected token found! Did you forget the ';' at the end of the entity reference?";

return "";

}

}

else ss << \*str, str++;

}

return ss.str();

}

void XMLFileLoader::JumpSpaces()

{

while (\*str == ' ' || \*str == '\n' || \*str == '/r' || \*str == '\t')

str++;

}

std::string XMLFileLoader::RemoveComments(const std::string& str)

{

std::string res;

size\_t last = 0;

size\_t beg;

while ((beg = str.find("<!--", last)) != std::string::npos)

{

size\_t end = str.find("-->", beg + 4);

res += str.substr(last, beg - last);

if (end == std::string::npos)

return res;

last = end + 3;

}

res += str.substr(last);

return res;

}

std::string XMLFileWriter::WriteToMemory(XMLFile\* xml, XMLOutputOptions options)

{

ss.clear();

XMLFileWriter::options = options;

if (xml->prolog != nullptr)

WriteProlog(xml->prolog);

WriteNode(xml->root, 0);

return ss.str();

}

void XMLFileWriter::WriteToFile(XMLFile\* xml, const std::string& path, XMLOutputOptions options)

{

Files::WriteTextFile(path, WriteToMemory(xml, options));

}

void XMLFileWriter::WriteProlog(XMLNode\* node)

{

ss << "<?" << node->name;

for (const XMLAttribute& at : node->attributes)

{

ss << ' ' << at.name << "=\"";

WriteFormatted(at.value);

ss << '\"';

}

ss << "?>";

if (options.indent)

ss << '\n';

}

void XMLFileWriter::WriteNode(XMLNode\* node, uint depth)

{

if(options.indent)

WriteIndent(depth);

ss << '<' << node->name;

for (const XMLAttribute& at : node->attributes)

{

ss << ' ' << at.name << "=\"";

WriteFormatted(at.value);

ss << '\"';

}

if (node->children.size() == 0 && node->content == "")

ss << "/>";

else

{

ss << '>';

if (node->children.size() != 0)

{

for (XMLNode\* c : node->children)

{

if (options.indent)

ss << '\n';

WriteNode(c, depth + 1);

}

if (options.indent)

{

ss << '\n';

WriteIndent(depth);

}

ss << "</" << node->name << '>';

}

else

{

WriteFormatted(node->content);

ss << "</" << node->name << '>';

}

}

}

void XMLFileWriter::WriteIndent(uint depth)

{

for (uint i = 0; i < depth \* options.spacesToIndent; i++)

ss << ' ';

}

void XMLFileWriter::WriteFormatted(const std::string& text)

{

for (char c : text)

{

switch (c)

{

case '<':

ss << "&lt;"; break;

case '>':

ss << "&gt;"; break;

case '&':

ss << "&amp;"; break;

case '\'':

ss << "&apos;"; break;

case '\"':

ss << "&quot;"; break;

default:

ss << c; break;

}

}

}

**src/graphics/window.cpp:**

#include "pch.h"

#include "window.h"

Window::Window(int w, int h, const string& t)

{

SDL\_Init(SDL\_INIT\_EVERYTHING);

IMG\_Init(IMG\_INIT\_PNG | IMG\_INIT\_JPG | IMG\_INIT\_TIF);

TTF\_Init();

this->w = w;

this->h = h;

title = t;

SDL\_SetHint(SDL\_HINT\_RENDER\_SCALE\_QUALITY, "2");

sdlWindow = SDL\_CreateWindow(title.c\_str(), SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED, w, h, SDL\_WINDOW\_RESIZABLE);

sdlRenderer = SDL\_CreateRenderer(sdlWindow, -1, SDL\_RENDERER\_ACCELERATED);

timer = new Timer();

input = new Input(this);

renderer = new Renderer(this);

OnResize();

maxFPSTime = std::chrono::high\_resolution\_clock::now();

}

Window::~Window()

{

SDL\_DestroyRenderer(sdlRenderer);

SDL\_DestroyWindow(sdlWindow);

delete input;

TTF\_Quit();

IMG\_Quit();

SDL\_Quit();

}

void Window::Update()

{

float interval = 1.f / (float)maxFPS;

maxFPSDuration.zero();

do

{

maxFPSAux = std::chrono::high\_resolution\_clock::now();

maxFPSDuration = maxFPSAux - maxFPSTime;

} while (maxFPSDuration.count() < interval);

maxFPSTime = maxFPSAux;

SDL\_RenderPresent(sdlRenderer);

input->Update();

timer->Update();

}

void Window::OnResize()

{

left = 10.f \* w / h;

}

**src/graphics/renderer.cpp:**

#include "pch.h"

#include "renderer.h"

#include "window.h"

Renderer::Renderer(Window\* w)

{

window = w;

}

Renderer::~Renderer()

{

}

void Renderer::ConvertCoords(Vector2f& v)

{

v.x = ((v.x + window->left) / window->left) / 2.f \* window->w;

v.y = ((-v.y + 10.f) / 10.f) / 2.f \* window->h;

}

void Renderer::ConvertDimensions(Vector2f& v)

{

v.x = (v.x / window->left) / 2.f \* window->w;

v.y = (v.y / 10.f) / 2.f \* window->h;

}

void Renderer::ConvertDimY(float& d)

{

d = (d / 10.f) / 2.f \* window->h;

}

void Renderer::SetColor(Vector4i c)

{

SDL\_SetRenderDrawColor(window->sdlRenderer, c.r, c.g, c.b, c.a);

currentColor = c;

}

void Renderer::ClearScreen()

{

SDL\_RenderClear(window->sdlRenderer);

}

void Renderer::DrawLine(Vector2f a, Vector2f b)

{

ConvertCoords(a);

ConvertCoords(b);

SDL\_RenderDrawLine(window->sdlRenderer, (int)a.x, (int)a.y, (int)b.x, (int)b.y);

}

void Renderer::DrawRect(Vector2f a, Vector2f b)

{

ConvertCoords(a);

ConvertDimensions(b);

SDL\_Rect r = { int(a.x - b.x / 2), int(a.y - b.y / 2), (int)b.x, (int)b.y };

SDL\_RenderDrawRect(window->sdlRenderer, &r);

}

void Renderer::FillRect(Vector2f a, Vector2f b)

{

ConvertCoords(a);

ConvertDimensions(b);

SDL\_Rect r = { int(a.x - b.x / 2), int(a.y - b.y / 2), (int)b.x, (int)b.y };

SDL\_RenderFillRect(window->sdlRenderer, &r);

}

void Renderer::DrawRectBounds(Vector2f bl , Vector2f size)

{

ConvertCoords(bl);

ConvertDimensions(size);

SDL\_Rect r = { (int)bl.x, (int)bl.y, (int)size.x, (int)size.y};

SDL\_RenderDrawRect(window->sdlRenderer, &r);

}

void Renderer::FillRectBounds(Vector2f bl, Vector2f size)

{

ConvertCoords(bl);

ConvertDimensions(size);

SDL\_Rect r = { (int)bl.x, (int)bl.y, (int)size.x, (int)size.y };

SDL\_RenderFillRect(window->sdlRenderer, &r);

}

void Renderer::FillPolygon(Vector2f\* v, int n)

{

aux1.clear();

aux2.clear();

for (int i = 0; i < n; i++)

{

ConvertCoords(v[i]);

aux1.push\_back((Sint16)v[i].x);

aux2.push\_back((Sint16)v[i].y);

}

filledPolygonRGBA(window->sdlRenderer, &aux1[0], &aux2[0], n, currentColor.r, currentColor.g, currentColor.b, currentColor.a);

}

void Renderer::FillCircle(Vector2f pos, float r)

{

ConvertCoords(pos);

ConvertDimY(r);

filledCircleRGBA(window->sdlRenderer, (Sint16)pos.x, (Sint16)pos.y, (Sint16)r, (Uint8)currentColor.r, (Uint8)currentColor.g, (Uint8)currentColor.b, (Uint8)currentColor.a);

}

void Renderer::DrawCircle(Vector2f pos, float r)

{

ConvertCoords(pos);

ConvertDimY(r);

circleRGBA(window->sdlRenderer, (Sint16)pos.x, (Sint16)pos.y, (Sint16)r, (Uint8)currentColor.r, (Uint8)currentColor.g, (Uint8)currentColor.b, (Uint8)currentColor.a);

}

void Renderer::FillRoundRect(Vector2f center, Vector2f size, float radius)

{

FillRoundRectBounds(center - size / 2., center + size / 2., radius);

}

void Renderer::FillRoundRectBounds(Vector2f bl, Vector2f tr, float radius)

{

ConvertCoords(bl);

ConvertCoords(tr);

ConvertDimY(radius);

roundedBoxRGBA(window->sdlRenderer, (Sint16)bl.x, (Sint16)bl.y, (Sint16)tr.x, (Sint16)tr.y, (Sint16)radius, (Uint8)currentColor.r, (Uint8)currentColor.g, (Uint8)currentColor.b, (Uint8)currentColor.a);

}

void Renderer::DrawImage(Texture\* t, Vector2f destPos, Vector2f destWH, float angle, Vector2f srcPos, Vector2f srcWH, byte flip, bool centered)

{

SDL\_Rect dest;

if (destWH.x == 0)

{

float iar = (float)((srcWH.x - srcPos.x) \* t->w) / ((srcWH.y - srcPos.y) \* t->h);

destWH.x = destWH.y \* iar;

}

else if (destWH.y == 0)

{

float iar = (float)((srcWH.x - srcPos.x) \* t->w) / ((srcWH.y - srcPos.y) \* t->h);

destWH.y = destWH.x / iar;

}

ConvertCoords(destPos);

ConvertDimensions(destWH);

if (centered)

{

dest.x = int(destPos.x - destWH.x / 2);

dest.y = int(destPos.y - destWH.y / 2);

}

else

{

dest.x = int(destPos.x);

dest.y = int(destPos.y);

}

dest.w = (int)destWH.x;

dest.h = (int)destWH.y;

srcPos.x \*= t->w; srcPos.y \*= t->h;

srcWH.x \*= t->w; srcWH.y \*= t->h;

SDL\_Rect src = { (int)srcPos.x, (int)srcPos.y, (int)srcWH.x, (int)srcWH.y };

SDL\_RenderCopyEx(window->sdlRenderer, t->texture, &src, &dest, angle, NULL, SDL\_RendererFlip(flip));

}

**src/game/game.h:**

#pragma once

#include "pch.h"

#include "graphics\window.h"

#include "scene.h"

#include "utils\xml.h"

class Game

{

public:

static Game\* Instance;

Game(int w, int h, const string& title);

virtual ~Game();

virtual void Start();

virtual void Stop();

virtual void Loop();

virtual void Update();

virtual void Render();

bool running = false;

Window\* window;

Scene\* scene = nullptr;

Scene\* nextScene = nullptr;

XMLFile\* cfg;

};

**src/game/game.cpp:**

#include "pch.h"

#include "game.h"

Game\* Game::Instance = nullptr;

Game::Game(int w, int h, const string& title)

{

if (Instance != nullptr)

LOG\_INFO("Two games were created!\n");

Instance = this;

cfg = XMLFileLoader::LoadFile("config.xml");

window = new Window(w, h, title);

}

Game::~Game()

{

if (cfg != nullptr)

delete cfg;

delete window;

}

void Game::Start()

{

running = true;

while (running && !window->input->shouldClose)

Loop();

}

void Game::Stop()

{

running = false;

}

void Game::Loop()

{

Update();

Render();

window->Update();

if (nextScene != nullptr)

{

delete scene;

scene = nextScene;

nextScene = nullptr;

}

}

void Game::Update() { if (scene) scene->Update(); }

void Game::Render() { if (scene) scene->Render(); }

**Fragment din src/game/dpo/dpoScene.h:**

#pragma once

#include "../scene.h"

#include "dpoGame.h"

#include "../uiUtils.h"

class DPOScene : public Scene

{

private:

...

Vector2f boardblCorner = Vector2f(-6, -4.5);

float tilePadding = 0.5f;

float tileSize = 2.5f;

float moveDurationStandard = 0.18f;

float spawnDurationStandard = 0.12f;

float moveDuration, spawnDuration;

float animationsDuration;

float speedMultiplier = 1;

float timer = 0;

bool autorestart = true;

float autorestartDuration = 3.f;

float autorestartTimer;

bool lost = false;

bool benchmark = false;

Label losingLabel;

Label numbers[20];

Vector4i colors[20];

DPOGame dpogame;

DPOController\* humanController;

DPOController\* expectimaxController;

DPOController\* controller;

Button humanControllerButton;

Button expectimaxControllerButton;

Label scoreLabel;

Slider speedSlider;

FILE\* benchmarkLogFile = NULL;

int benchmarkMaxTiles[30];

int benchmarkMinTile = 30, benchmarkMaxTile = 0;

long long benchmarkAverageScore = 0;

int benchmarkMinScore = 0x3f3f3f3f, benchmarkMaxScore = 0;

int benchmarkCurrentGame = 1;

int benchmarkNumMoves = 0;

};

**src/game/dpo/dpoScene.cpp:**

#include "pch.h"

#include "dpoScene.h"

#include "dpoHumanController.h"

#include "dpoTreeAI.h"

#include "xlsxwriter.h"

#include "math\math.h"

#include "../uiUtils.h"

DPOScene::DPOScene(Game\* g) : Scene(g)

{

game->window->maxFPS = 9999999;

moveDuration = moveDurationStandard \* 0.66f;

spawnDuration = spawnDurationStandard \* 0.66f;

animationsDuration = moveDuration + spawnDuration;

timer = animationsDuration + 1;

game->cfg->SetVariable<bool>("dpo/autorestart", autorestart);

game->cfg->SetVariable<float>("dpo/autorestartDuration", autorestartDuration);

colors[0] = Vector4i(238, 228, 218, 255);

colors[1] = Vector4i(237, 224, 200, 255);

colors[2] = Vector4i(241, 177, 121, 255);

colors[3] = Vector4i(245, 149, 99, 255);

colors[4] = Vector4i(246, 124, 94, 255);

colors[5] = Vector4i(247, 94, 60, 255);

colors[6] = Vector4i(237, 206, 114, 255);

colors[7] = Vector4i(236, 203, 97, 255);

colors[8] = Vector4i(236, 200, 80, 255);

colors[9] = Vector4i(236, 196, 64, 255);

colors[10] = Vector4i(236, 193, 46, 255);

colors[11] = Vector4i(255, 61, 61, 255);

colors[12] = Vector4i(255, 30, 30, 255);

colors[13] = Vector4i(255, 38, 88, 255);

colors[14] = Vector4i(130, 64, 150, 255);

colors[15] = Vector4i(132, 97, 150, 255);

colors[16] = Vector4i(40, 0, 142, 255);

colors[17] = Vector4i(0, 132, 188, 255);

colors[18] = Vector4i(38, 173, 175, 255);

colors[19] = Vector4i(126, 175, 174, 255);

for (int i = 0; i < 20; i++)

{

numbers[i].InitTexture(renderer);

if(i < 2)

numbers[i].InitTextTexture(std::to\_string(1<<(i+1)).c\_str(), Vector4i(110, 110, 110, 255));

else numbers[i].InitTextTexture(std::to\_string(1 << (i + 1)).c\_str(), Vector4i(255, 255, 255, 255));

numbers[i].SetDimY(tileSize \* 0.4f);

}

losingLabel.InitTexture(renderer);

losingLabel.InitTextTexture("You lost! Press R to restart");

losingLabel.pos.x = 0;

losingLabel.pos.y = -6.f;

losingLabel.SetDimX(4 \* (tilePadding + tileSize));

humanControllerButton.InitAll(renderer);

humanControllerButton.InitTextDef("Manual");

humanControllerButton.pos.x = -12;

humanControllerButton.pos.y = -1.75f;

humanControllerButton.SetDimY(1.5f);

scoreLabel.InitTexture(renderer);

scoreLabel.InitTextTexture("Score: 0");

scoreLabel.pos.x = -16;

scoreLabel.pos.y = 9;

scoreLabel.centered = false;

scoreLabel.SetDimY(1.5f);

expectimaxControllerButton.InitAll(renderer);

expectimaxControllerButton.InitTextDef("Expectimax");

expectimaxControllerButton.pos.x = -12;

expectimaxControllerButton.pos.y = -3.75f;

expectimaxControllerButton.SetDimY(1.5f);

speedSlider.center.x = 0;

speedSlider.center.y = -8.f;

speedSlider.dim.x = 12;

speedSlider.dim.y = 0.5f;

speedSlider.sliderDim.x = 0.5f;

speedSlider.sliderDim.y = 1.5f;

speedSlider.val = 0.66f;

game->cfg->SetVariable<bool>("dpo/benchmark", benchmark);

humanController = new DPOHumanController(this);

expectimaxController = new DPOTreeAI(dpogame);

controller = expectimaxController;

if (benchmark)

{

string filePrefix = "DPOTest";

string fileSuffix = ".txt";

string lastFileName = filePrefix + fileSuffix;

FILE\* file = fopen(lastFileName.c\_str(), "r");

int attempt = 1;

while (file != NULL)

{

fclose(file);

lastFileName = filePrefix + " (" + String::ToString(attempt) + ")" + fileSuffix;

file = fopen(lastFileName.c\_str(), "r");

attempt++;

}

file = fopen(lastFileName.c\_str(), "w");

benchmarkLogFile = file;

memset(benchmarkMaxTiles, 0, sizeof(benchmarkMaxTiles));

}

}

DPOScene::~DPOScene()

{

if (benchmark)

fclose(benchmarkLogFile);

delete humanController;

delete expectimaxController;

}

void DPOScene::Reset()

{

scoreLabel.InitTextTexture("Score: 0");

scoreLabel.SetDimY(1.5f);

timer = animationsDuration + 1;

dpogame = DPOGame();

lost = false;

}

void DPOScene::Update()

{

if (humanControllerButton.Update(game->window))

controller = humanController;

if (expectimaxControllerButton.Update(game->window))

controller = expectimaxController;

speedSlider.Update(game->window->input);

float newSpeedMultiplier = speedSlider.val;

newSpeedMultiplier = Clamp(newSpeedMultiplier, 0.01f, 1.f);

if (newSpeedMultiplier != speedMultiplier)

{

timer \*= newSpeedMultiplier / speedMultiplier;

moveDuration = moveDurationStandard \* newSpeedMultiplier;

spawnDuration = spawnDurationStandard \* newSpeedMultiplier;

animationsDuration = moveDuration + spawnDuration;

speedMultiplier = newSpeedMultiplier;

}

if (game->window->input->GetKeyState(SDL\_SCANCODE\_R) == KEY\_JUST\_PRESSED || autorestartTimer <= 0 && lost)

{

Reset();

return;

}

autorestartTimer -= game->window->timer->GetDeltaTime();

if (timer >= animationsDuration && !lost)

{

int result = dpogame.Update(controller);

if (result > 0) // changes

{

scoreLabel.InitTextTexture(("Score: " + String::ToString(result)).c\_str());

scoreLabel.SetDimY(1.5f);

if(benchmark)

benchmarkNumMoves++;

controller->PostStepPrepare(dpogame.GetState());

timer = 0;

for (uint i = 0; i < dpogame.movements.size(); i++)

{

dpogame.movements[i].oldPositionScreen = ToScreenCoords(dpogame.movements[i].oldPosition);

dpogame.movements[i].newPositionScreen = ToScreenCoords(dpogame.movements[i].newPosition);

}

for (uint i = 0; i < dpogame.spawns.size(); i++)

dpogame.spawns[i].spawnPositionScreen = ToScreenCoords(dpogame.spawns[i].spawnPosition);

for (uint i = 0; i < dpogame.joins.size(); i++)

dpogame.joins[i].joinPositionScreen = ToScreenCoords(dpogame.joins[i].joinPosition);

}

else if (result == -1)

{

if (benchmark)

{

int score = 0;

int maxTile = 0;

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

{

score += Pow<int>(2, 1 + dpogame.GetState().tiles[i][j]);

maxTile = Max(maxTile, dpogame.GetState().tiles[i][j]);

}

LOG\_INFO("Game #%d: Score: %d, Max tile: %d, Moves: %d\n", benchmarkCurrentGame, score, Pow<int>(2, 1 + maxTile), benchmarkNumMoves);

fprintf(benchmarkLogFile, "Game #%d: Score: %d, Max tile: %d, Moves: %d\n", benchmarkCurrentGame, score, Pow<int>(2, 1 + maxTile), benchmarkNumMoves);

benchmarkMaxTiles[maxTile]++;

benchmarkMaxTile = Max(benchmarkMaxTile, maxTile);

benchmarkMinTile = Min(benchmarkMinTile, maxTile);

benchmarkMaxScore = Max(benchmarkMaxScore, score);

benchmarkMinScore = Min(benchmarkMinScore, score);

benchmarkAverageScore += score;

LOG\_INFO("Average score: %f\n", double(benchmarkAverageScore) / benchmarkCurrentGame);

fprintf(benchmarkLogFile, "Average score: %f\n", double(benchmarkAverageScore) / benchmarkCurrentGame);

for (int i = benchmarkMinTile; i <= benchmarkMaxTile; i++)

{

LOG\_INFO("%# 5d: %.3f%%\n", Pow<int>(2, i + 1), double(benchmarkMaxTiles[i]) / benchmarkCurrentGame \* 100.);

fprintf(benchmarkLogFile, "%# 5d: %.3f%%\n", Pow<int>(2, i + 1), double(benchmarkMaxTiles[i]) / benchmarkCurrentGame \* 100.);

}

fflush(benchmarkLogFile);

benchmarkNumMoves = 0;

benchmarkCurrentGame++;

}

lost = true;

if (autorestart)

autorestartTimer = autorestartDuration;

}

}

}

void DPOScene::Render()

{

RenderBoard();

if (timer >= animationsDuration)

{

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

if (dpogame.GetState().tiles[i][j] != -1)

RenderTile(Vector2i(i, j), dpogame.GetState().tiles[i][j]);

}

else

{

if (timer < moveDuration)

{

float amount = 1.f - (moveDuration - timer) / moveDuration;

RenderMovements(Clamp01(amount));

}

else if(timer < animationsDuration)

{

controller->Prepare();

float amount = 1.f - (spawnDuration - timer + moveDuration) / spawnDuration;

RenderMovements(1);

RenderSpawnsAndJoins(Clamp01(amount));

}

timer += game->window->timer->GetDeltaTime();

}

if(lost)

losingLabel.Render(renderer);

humanControllerButton.Render(renderer);

expectimaxControllerButton.Render(renderer);

speedSlider.Render(renderer);

scoreLabel.Render(renderer);

}

void DPOScene::RenderBoard()

{

renderer->SetColor(Vector4i(187, 173, 160, 255));

renderer->FillRoundRectBounds(boardblCorner, boardblCorner + Vector2f(1) \* 4 \* (tilePadding + tileSize), 0.15f);

renderer->SetColor(Vector4i(214, 205, 196, 255));

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

RenderTile(ToScreenCoords({ i, j }));

}

void DPOScene::RenderTile(Vector2f coord, float size)

{

renderer->FillRoundRect(coord, Vector2f(tileSize) \* size, 0.15f);

}

Vector2f DPOScene::ToScreenCoords(Vector2i coords)

{

Vector2f res = boardblCorner;

res.x += (float(coords.y) + 0.5f) \* (tilePadding + tileSize);

res.y += (float(3 - coords.x) + 0.5f) \* (tilePadding + tileSize);

return res;

}

void DPOScene::RenderTile(Vector2i coord, int value, float size)

{

RenderTile(ToScreenCoords(coord), value, size);

}

void DPOScene::RenderTile(Vector2f coord, int value, float size)

{

renderer->SetColor(colors[value]);

RenderTile(coord, size);

numbers[value].pos = coord;

Vector2f oldSize = numbers[value].dim;

numbers[value].dim \*= size;

numbers[value].Render(renderer);

numbers[value].dim = oldSize;

}

void DPOScene::RenderMovements(float amount)

{

// 0 -> 1

float hermite = amount \* amount \* (3 - 2 \* amount);

for (uint i = 0; i < dpogame.movements.size(); i++)

{

const DPOTileMovement& m = dpogame.movements[i];

Vector2f position = m.oldPositionScreen.Lerp(m.newPositionScreen, hermite);

if (m.oldPosition == m.newPosition)

position = m.oldPositionScreen;

RenderTile(position, m.value);

}

}

void DPOScene::RenderSpawnsAndJoins(float amount)

{

// 0 -> 1

float hermite = amount \* amount \* (3 - 2 \* amount);

// 1 -> 1.25 -> 1

float val = amount - 0.5f;

float sinval = -val \* val + 1.25f;

for (uint i = 0; i < dpogame.spawns.size(); i++)

{

const DPOTileSpawn& s = dpogame.spawns[i];

RenderTile(s.spawnPositionScreen, s.value, hermite);

}

for (uint i = 0; i < dpogame.joins.size(); i++)

{

const DPOTileJoin& j = dpogame.joins[i];

RenderTile(j.joinPositionScreen, j.value, sinval);

}

}

**src/game/dpo/dpoController.h:**

#pragma once

struct DPOGameState;

class DPOController

{

public:

virtual ~DPOController() {}

// called once after a successful step with the future state

// mainly used to have the ai thread work even while the animations are running

virtual void PostStepPrepare(const DPOGameState& state) {}

// called repeteadly during spawn animation

// mainly used for human input

virtual void Prepare() {}

// is the step function ready to return for this state?

virtual bool Ready(const DPOGameState&) = 0;

// returns the direction for the current state

// 0 = up, 1 = right, 2 = down, 3 = left

virtual int Step(const DPOGameState&) = 0;

// returns the name of the algorithm

virtual string GetName() const = 0;

};

**src/game/dpo/dpoHumanController.h:**

#pragma once

#include "dpoController.h"

#include "../game.h"

#include "../scene.h"

class DPOHumanController : public DPOController

{

public:

DPOHumanController(Scene\* scene) : scene(scene)

{

direction = -1;

}

// true if direction has changed

bool GetInput()

{

Input\* input = scene->game->window->input;

if (input->GetKeyState(SDL\_SCANCODE\_UP) == KEY\_JUST\_PRESSED || input->GetKeyState(SDL\_SCANCODE\_W) == KEY\_JUST\_PRESSED)

direction = 0;

else if (input->GetKeyState(SDL\_SCANCODE\_RIGHT) == KEY\_JUST\_PRESSED || input->GetKeyState(SDL\_SCANCODE\_D) == KEY\_JUST\_PRESSED)

direction = 1;

else if (input->GetKeyState(SDL\_SCANCODE\_DOWN) == KEY\_JUST\_PRESSED || input->GetKeyState(SDL\_SCANCODE\_S) == KEY\_JUST\_PRESSED)

direction = 2;

else if (input->GetKeyState(SDL\_SCANCODE\_LEFT) == KEY\_JUST\_PRESSED || input->GetKeyState(SDL\_SCANCODE\_A) == KEY\_JUST\_PRESSED)

direction = 3;

if (direction == -1)

return false;

return true;

}

void Prepare() override

{

GetInput();

}

bool Ready(const DPOGameState& state) override

{

return GetInput();

}

// 0 = up, 1 = right, 2 = down, 3 = left

int Step(const DPOGameState& state) override

{

int old = direction;

direction = -1;

return old;

}

string GetName() const override

{

return "Human";

}

private:

int direction;

Scene\* scene;

};

**src/game/dpo/dpoGame.h:**

#pragma once

#include "pch.h"

#include "dpoController.h"

#include "math\math.h"

#include <random>

struct DPOGameState

{

// -1 means empty, 0 means 2, 1 means 4, 2 means 8 etc.

int tiles[4][4];

};

struct DPOTileMovement

{

DPOTileMovement(int value, Vector2i oldPos, Vector2i newPos) :

value(value),

oldPosition(oldPos),

newPosition(newPos) {}

int value;

Vector2i oldPosition;

Vector2i newPosition;

Vector2f oldPositionScreen;

Vector2f newPositionScreen;

};

struct DPOTileSpawn

{

DPOTileSpawn(int value, Vector2i spawnPosition) :

value(value),

spawnPosition(spawnPosition) {}

int value;

Vector2i spawnPosition;

Vector2f spawnPositionScreen;

};

struct DPOTileJoin

{

DPOTileJoin(int value, Vector2i joinPosition) :

value(value),

joinPosition(joinPosition) {}

int value;

Vector2i joinPosition;

Vector2f joinPositionScreen;

};

class DPOGame

{

public:

DPOGame();

void Reset(DPOGameState& state, bool addTileChanges);

int Update(DPOController\* controller);

// returns the score if there was a change, -1 if lost or 0 if no change was made

int Update(DPOGameState& state, DPOController\* controller, bool addTileChanges);

const DPOGameState& GetState() const;

// sum of all the tiles

int GetScore() const;

// returns -1 if there's no more space

// otherwise returns the generated tile

int GenerateTiles(DPOGameState& state, int direction, bool addTileChanges);

void Move(DPOGameState& state, int direction, bool addTileChanges);

bool StatesDifferent(const DPOGameState& a, const DPOGameState& b) const;

bool CanMove(DPOGameState a);

float chanceToSpawn2 = 0.9f;

std::vector<DPOTileMovement> movements;

std::vector<DPOTileSpawn> spawns;

std::vector<DPOTileJoin> joins;

private:

int score = 0;

std::mt19937 randEngine;

DPOGameState currentState;

};

**src/game/dpo/dpoGame.cpp:**

#include "pch.h"

#include "dpoGame.h"

DPOGame::DPOGame()

{

std::random\_device rd;

randEngine = std::mt19937(rd());

Reset(currentState, false);

}

void DPOGame::Reset(DPOGameState& state, bool addTileChanges)

{

score = 0;

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

state.tiles[i][j] = -1;

std::uniform\_int\_distribution<int> uid(0, 15);

int v[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };

std::shuffle(v, v + 16, randEngine);

std::uniform\_int\_distribution<int> uid2(2, 4);

int numTiles = uid2(randEngine);

std::uniform\_real\_distribution<float> urd(0, 1);

for (int i = 0; i < numTiles; i++)

{

int ti = v[i] % 4;

int tj = v[i] / 4;

int newTile = urd(randEngine) > chanceToSpawn2 ? 1 : 0;

state.tiles[ti][tj] = newTile;

score += 1 << (newTile + 1);

}

}

int DPOGame::Update(DPOController\* controller)

{

return Update(currentState, controller, true);

}

// returns the score if there was a change, -1 if lost or 0 if no change was made

int DPOGame::Update(DPOGameState& state, DPOController\* controller, bool addTileChanges)

{

if (controller->Ready(state))

{

movements.clear();

joins.clear();

spawns.clear();

int direction = controller->Step(state);

DPOGameState prevState = state;

Move(state, direction, addTileChanges);

bool statesDifferent = StatesDifferent(state, prevState);

if (statesDifferent)

{

int newTile = GenerateTiles(state, direction, addTileChanges);

if (newTile == -1)

return -1;

score += (1 << (newTile + 1));

prevState = state;

int i;

for (i = 0; i < 4; i++)

{

Move(prevState, i, false);

if (StatesDifferent(state, prevState))

break;

}

if (i == 4)

return -1;

}

else

{

movements.clear();

joins.clear();

return 0;

}

return score;

}

return 0;

}

bool DPOGame::CanMove(DPOGameState state)

{

for (int i = 0; i < 4; i++)

{

DPOGameState prevState = state;

Move(state, i, false);

bool statesDifferent = StatesDifferent(state, prevState);

if (statesDifferent)

return true;

}

return false;

}

const DPOGameState& DPOGame::GetState() const { return currentState; }

int DPOGame::GetScore() const { return score; }

// returns -1 if there's no more space

// otherwise returns the generated tile

int DPOGame::GenerateTiles(DPOGameState& state, int direction, bool addTileChanges)

{

std::uniform\_real\_distribution<float> urd(0, 1);

int newTile = urd(randEngine) > chanceToSpawn2 ? 1 : 0;

int v[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };

std::shuffle(v, v + 16, randEngine);

for (int i = 0; i < 16; i++)

{

if (state.tiles[v[i] % 4][v[i] / 4] == -1)

{

state.tiles[v[i] % 4][v[i] / 4] = newTile;

if (addTileChanges)

spawns.emplace\_back(newTile, Vector2i(v[i] % 4, v[i] / 4));

return newTile;

}

}

return -1;

}

void DPOGame::Move(DPOGameState& state, int direction, bool addTileChanges)

{

switch (direction)

{

case 0:

for (int i = 0; i < 4; i++)

{

int last, canJoin = 1;

for (last = 0; last < 4 && state.tiles[last][i] == -1; last++);

if (last < 4)

{

std::swap(state.tiles[last][i], state.tiles[0][i]);

if (addTileChanges)

movements.emplace\_back(state.tiles[0][i], Vector2i(last, i), Vector2i(0, i));

}

for (int j = last + 1, last = 0; j < 4; j++)

{

int value = state.tiles[j][i];

if (value == -1) continue;

state.tiles[j][i] = -1;

if (value == state.tiles[last][i] && canJoin)

{

state.tiles[last][i]++;

canJoin = 0;

if (addTileChanges)

joins.emplace\_back(value + 1, Vector2i(last, i));

}

else

{

state.tiles[++last][i] = value;

canJoin = 1;

}

if (addTileChanges)

movements.emplace\_back(value, Vector2i(j, i), Vector2i(last, i));

}

}

break;

case 2:

for (int i = 0; i < 4; i++)

{

int last, canJoin = 1;

for (last = 3; last >= 0 && state.tiles[last][i] == -1; last--);

if (last >= 0)

{

std::swap(state.tiles[last][i], state.tiles[3][i]);

if (addTileChanges)

movements.emplace\_back(state.tiles[3][i], Vector2i(last, i), Vector2i(3, i));

}

for (int j = last - 1, last = 3; j >= 0; j--)

{

int value = state.tiles[j][i];

if (value == -1) continue;

state.tiles[j][i] = -1;

if (value == state.tiles[last][i] && canJoin)

{

state.tiles[last][i]++;

canJoin = 0;

if (addTileChanges)

joins.emplace\_back(value + 1, Vector2i(last, i));

}

else

{

state.tiles[--last][i] = value;

canJoin = 1;

}

if (addTileChanges)

movements.emplace\_back(value, Vector2i(j, i), Vector2i(last, i));

}

}

break;

case 3:

for (int i = 0; i < 4; i++)

{

int last, canJoin = 1;

for (last = 0; last < 4 && state.tiles[i][last] == -1; last++);

if (last < 4)

{

std::swap(state.tiles[i][last], state.tiles[i][0]);

if (addTileChanges)

movements.emplace\_back(state.tiles[i][0], Vector2i(i, last), Vector2i(i, 0));

}

for (int j = last + 1, last = 0; j < 4; j++)

{

int value = state.tiles[i][j];

if (value == -1) continue;

state.tiles[i][j] = -1;

if (value == state.tiles[i][last] && canJoin)

{

state.tiles[i][last]++;

canJoin = 0;

if (addTileChanges)

joins.emplace\_back(value + 1, Vector2i(i, last));

}

else

{

state.tiles[i][++last] = value;

canJoin = 1;

}

if (addTileChanges)

movements.emplace\_back(value, Vector2i(i, j), Vector2i(i, last));

}

}

break;

case 1:

for (int i = 0; i < 4; i++)

{

int last, canJoin = 1;

for (last = 3; last >= 0 && state.tiles[i][last] == -1; last--);

if (last >= 0)

{

std::swap(state.tiles[i][last], state.tiles[i][3]);

if (addTileChanges)

movements.emplace\_back(state.tiles[i][3], Vector2i(i, last), Vector2i(i, 3));

}

for (int j = last - 1, last = 3; j >= 0; j--)

{

int value = state.tiles[i][j];

if (value == -1) continue;

state.tiles[i][j] = -1;

if (value == state.tiles[i][last] && canJoin)

{

state.tiles[i][last]++;

canJoin = 0;

if (addTileChanges)

joins.emplace\_back(value + 1, Vector2i(i, last));

}

else

{

state.tiles[i][--last] = value;

canJoin = 1;

}

if (addTileChanges)

movements.emplace\_back(value, Vector2i(i, j), Vector2i(i, last));

}

}

break;

}

}

bool DPOGame::StatesDifferent(const DPOGameState& a, const DPOGameState& b) const

{

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

if (a.tiles[i][j] != b.tiles[i][j])

return true;

return false;

}

**src/game/dpo/dpoTreeAI.h:**

#pragma once

#include "dpoController.h"

#include "dpoGame.h"

#include <thread>

#include <atomic>

struct DPOTreeNode

{

float score;

DPOGameState state;

};

class DPOTreeAI : public DPOController

{

public:

DPOTreeAI(const DPOGame& game);

~DPOTreeAI();

bool Ready(const DPOGameState& state) override;

void PostStepPrepare(const DPOGameState& state) override;

float CalcScore(const DPOGameState& state);

int Step(const DPOGameState& state);

string GetName() const override;

private:

void ThreadMain();

// returns the direction or -1 if losing or exceeded max depth

void Solve(DPOTreeNode\* node, int depth, float& maxScore, int& desiredDirection);

bool StatesEqual(const DPOGameState& a, const DPOGameState& b);

private:

float penaltyAmount = 1.5f;

float emptyTileBonus = 0.075f;

float positionalBonus = 0.3f;

float ascendingTilesBonus = 0.5f;

float losingPenalty = 100000;

int maxCheck = 6;

int maxDepth = 3;

DPOTreeNode root;

DPOGame game;

std::mt19937 randEngine;

std::thread aiThread;

std::atomic\_bool workAvailable = false, running = false, working = false, resultedDirectionAvailable = false;

int resultedDirection;

DPOGameState work;

};

**src/game/dpo/dpoTreeAI.cpp:**

#include "pch.h"

#include "dpoTreeAI.h"

#include "game/game.h"

#include "graphics/timer.h"

DPOTreeAI::DPOTreeAI(const DPOGame& game) : game(game)

{

std::random\_device rd;

randEngine = std::mt19937(rd());

running = true;

aiThread = std::thread(&DPOTreeAI::ThreadMain, this);

Game::Instance->cfg->SetVariable<int>("dpo/maxDepth", maxDepth);

Game::Instance->cfg->SetVariable<int>("dpo/maxCheck", maxCheck);

Game::Instance->cfg->SetVariable<float>("dpo/penaltyAmount", penaltyAmount);

Game::Instance->cfg->SetVariable<float>("dpo/emptyTileBonus", emptyTileBonus);

Game::Instance->cfg->SetVariable<float>("dpo/positionalBonus", positionalBonus);

Game::Instance->cfg->SetVariable<float>("dpo/ascendingTilesBonus", ascendingTilesBonus);

Game::Instance->cfg->SetVariable<float>("dpo/losingPenalty", losingPenalty);

}

DPOTreeAI::~DPOTreeAI()

{

running = false;

aiThread.join();

}

bool DPOTreeAI::Ready(const DPOGameState& state)

{

if (working)

return false;

if (resultedDirectionAvailable)

return true;

if (!workAvailable)

{

work = state;

workAvailable = true;

}

return false;

}

void DPOTreeAI::PostStepPrepare(const DPOGameState& state)

{

work = state;

workAvailable = true;

}

float DPOTreeAI::CalcScore(const DPOGameState& state)

{

bool lost = !game.CanMove(state);

if (lost)

return -losingPenalty;

int asc = 0;

for (int i = 0; i < 4; i++)

{

int last = 40;

for (int j = 0; j < 4; j++)

{

if (state.tiles[i][j] == -1)

continue;

if (state.tiles[i][j] == last || state.tiles[i][j] == last - 1 || last == 40)

{

if (last != 40)

asc += Pow<int>(3, last + 1) \* (4 - i);

last = state.tiles[i][j];

}

else break;

}

}

for (int j = 0; j < 4; j++)

{

int last = 40;

for (int i = 0; i < 4; i++)

{

if (state.tiles[i][j] == -1)

continue;

if (state.tiles[i][j] == last || state.tiles[i][j] == last - 1 || last == 40)

{

if (last != 40)

asc += Pow<int>(3, last + 1) \* (4 - j);

last = state.tiles[i][j];

}

else break;

}

}

int penalty = 0;

for (int i = 0; i < 4; i++)

for (int j = 0; j < 3; j++)

penalty += Pow(3, abs(state.tiles[i][j] - state.tiles[i][j + 1]));

for (int j = 0; j < 4; j++)

for (int i = 0; i < 3; i++)

penalty += Pow(3, abs(state.tiles[i][j] - state.tiles[i + 1][j]));

int emptyTiles = 0;

int score = 0;

int posValue = 0;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

if (state.tiles[i][j] == -1)

emptyTiles++;

else

{

int tileVal = Pow<int>(3, state.tiles[i][j] + 1);

score += tileVal;

int diagValue = 7 - i - j;

posValue += diagValue \* tileVal;

}

}

}

return score + float(score) \* emptyTiles \* emptyTileBonus + float(asc) \* ascendingTilesBonus + float(posValue) \* positionalBonus - float(penalty) \* penaltyAmount;

}

int DPOTreeAI::Step(const DPOGameState& state)

{

resultedDirectionAvailable = false;

return resultedDirection;

}

string DPOTreeAI::GetName() const

{

return "Expectimax";

}

void DPOTreeAI::ThreadMain()

{

while (running)

{

while (!workAvailable && running);

if (running == false)

break;

working = true;

workAvailable = false;

Timer timer;

root.state = work;

root.score = CalcScore(root.state);

float score;

int direction;

int emptyTiles = 0;

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

if (root.state.tiles[i][j] == -1)

emptyTiles++;

if(emptyTiles <= maxCheck && maxDepth <= 3)

Solve(&root, maxDepth + 1, score, direction);

else Solve(&root, maxDepth, score, direction);

resultedDirection = direction;

resultedDirectionAvailable = true;

timer.Update();

float workDuration = timer.GetDeltaTime();

//LOG\_INFO("AI Took: %f s\n", workDuration);

working = false;

}

}

// returns the direction or -1 if losing or exceeded max depth

void DPOTreeAI::Solve(DPOTreeNode\* node, int depth, float& maxScore, int& desiredDirection)

{

maxScore = -10000;

desiredDirection = -1;

if (depth <= 0) return; // should not be true

for (int i = 0; i < 4; i++)

{

// viable move?

DPOTreeNode newNode;

newNode.state = node->state;

game.Move(newNode.state, i, false);

//newNode.score = CalcScore(newNode.state);

// if the move wasn't viable

if (StatesEqual(newNode.state, node->state))

continue;

// score and weight

vector< pair<float, float> > scores;

// generate all outcomes

int v[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };

std::shuffle(v, v + 16, randEngine);

// skip tiles only if it becomes a problem

int checked = maxDepth >= 4 ? 0 : -100;

for (int j = 0; j < 16 && checked < maxCheck; j++)

{

int ti = v[j] / 4;

int tj = v[j] % 4;

// cannot generate here

if (newNode.state.tiles[ti][tj] != -1)

continue;

checked++;

for (int k = 0; k < 2; k++)

{

DPOTreeNode generatedNode;

generatedNode.state = newNode.state;

generatedNode.state.tiles[ti][tj] = k;

float generatedScore = generatedNode.score = CalcScore(generatedNode.state);

int generatedDir = -1;

if (depth > 1) // the board is now ready, generate a new move

Solve(&generatedNode, depth - 1, generatedScore, generatedDir);

scores.push\_back(std::make\_pair(generatedScore, k == 0 ? game.chanceToSpawn2 : (1 - game.chanceToSpawn2)));

}

}

float newScore = 0;

for (uint j = 0; j < scores.size(); j++)

newScore += scores[j].first \* scores[j].second;

newScore /= float(scores.size()) / 2;

if (newScore >= maxScore)

{

maxScore = newScore;

desiredDirection = i;

}

}

}

bool DPOTreeAI::StatesEqual(const DPOGameState& a, const DPOGameState& b)

{

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

if (a.tiles[i][j] != b.tiles[i][j])

return false;

return true;

Concluzii

Realizarea acestei aplicații s-a dovedit a fi o sarcină mult mai dificilă decât am anticipat inițial, însă reușește să ajungă la rezultate similare cu cele ale unei persoane din punct de vedere al scorului și să-l depășească din punct de vedere al timpului, acesta fiind obiectivul inițial al proiectului.

Există și posibilitatea de a îmbunătăți programul acesta, cel mai ușor mod fiind prin găsirea unor noi euristici sau modificarea variabilelor din fișierul de intrare. De asemenea, partea grafică ar putea fi îmbunătățită prin utilizarea unei librării specializate, precum Qt, sau a unui API precum OpenGL sau DirectX. Algoritmul ar putea fi schimbat total, dar acest lucru este mai dificil, întrucât Expectimax este printre cei mai performanți algoritmi care nu folosesc rețele neuronale. Printre algoritmii care folosesc inteligența artificială în adevăratul sens al cuvântului se numără Q-Learning și NEAT. Procesul de dezvoltare a avut la bază și refolosirea codului în alte proiecte, un exemplu foarte bun al acestei idei fiind utilizarea limbajului XML.

Procesul de dezvoltarea a necesitat cunoștințe legate de grafica pe calculator, teoria jocurilor, probabilități, parsare, calcul paralel real și virtual, matematică vectorială și multe altele. Am învățat să organizez un proiect complex, metode de a modifica setările compilatorului pentru a obține timpi mai mici fără a modifica sursa, cum funcționează sincronizarea threadurilor, elemente avansate din teoria jocurilor, toate acestea fiind lucruri care îmi vor fi folositoare pe viitor.