UNIVERSITATEA DIN BUCUREŞTI
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licență

Trasarea drumurilor optime cu constrângeri, cu aplicație în World of Warcraft

Coordonator științific **Absolvent**

Lect. dr. Marinescu-Ghemeci Ruxandra **Giuşcă Iulian**

Bucureşti, iunie 2017

Abstract

Tehnologia se află într-o continuă evoluție, iar optimizarea proceselor capătă din ce în ce mai mare importanță. Nu este de ajuns ca un algoritm să ofere un rezultat dacă obținerea rezultatului nu este fezabilă, i.e. timp foarte mare de execuție sau necesități de spațiu foarte mari, deoarece timpul şi spațiul sunt resurse limitate. Astfel, a apărut necesitatea folosirii algoritmilor de aproximare, care oferă o soluție apropiată de soluția optimă într-un timp de execuție mult mai scurt şi mai practic. Această lucrare şi-a propus implementarea algoritmului lui Christofides care aproximează traseul optim pentru atingerea obiectivelor de pe o hartă. Pentru ca modelul folosit să fie apropiat de un exemplu real, am adăugat constrângeri de spațiu, ce țin de poziția obiectivelor în raport cu harta pe care se află, şi constrângeri de timp, ce țin de obiectivele înseşi. Din punct de vedere practic, algoritmul este aplicat şi demonstrat pe un set de obiective din jocul World of Warcraft, însă modelul poate fi aplicat asupra oricărui sistem care necesită determinarea unui drum de cost minim. Aplicația dezvoltată poartă numele de Pathfinder şi are rolul de a ajuta jucătorii să îşi eficientizeze timpul de joc prin eliminarea necesității de a verifica în mod exhaustiv fiecare obiectiv de pe harta de joc. Cu doar câteva click-uri, el primeşte informații în legătură cu toate obiectivele active despre care este interesat şi îi este recomandat un traseu de urmat care minimzează distanța totală de parcurgere, oferind jucătorului asigurarea, în acelaşi timp, că nicio misiune nu va expira înainte de a ajunge acesta la ea. Potențialul de câştiguri individuale pe care această aplicație îl oferă constă în faptul că un utilizator poate selecta toate misiunile care oferă ca recompensă moneda de tranzacție a jocului, ce poate fi convertită în bani reali grație noilor funcționalități adăugate de creatorii jocului, Blizzard Entertainment.

Cuprins

Introducere

Motivarea temei

Pentru a-şi folosi timpul în mod eficient, oamenii planifică din timp modul sau ordinea în care evenimentele se vor desfășura, în așa fel încât aceștia să iasă în avantaj. Însă, nu întotdeauna dețin toate informațiile necesare pentru a putea decide. Tehnologia ne ajută din acest punct de vedere datorită capacității de stocare a datelor, date ce pot fi prelucrate pentru a obține un anume rezultat de interes pentru utilizator.

Lucrarea își propune să ofere o soluție pentru o astfel de problemă de decizie şi în continuare va fi prezentat un algoritm care va calcula un traseu optim din punct de vedere al distanței care trece prin toate obiectivele de pe o hartă, cunoscând distanțele dintre obiective şi eventualele constrângeri. Gradul de generalizare al problemei este mare, algoritmul fiind capabil să ofere o soluție pentru orice tip de problemă cu input similar. Aplicația va fi folosită sub forma unui addon în jocul World of Warcraft, generând pentru jucător un drum cât mai scurt care trece prin toate misiunile active de la un moment dat. Aplicația are rolul de a ajuta jucătorul să reducă din timpul alocat verificării integrale a tuturor obiectivelor de pe hartă, propunând un sistem rapid şi intuitiv de folosit.

Date istorice

În literatura de specialitate, subiectul abordat în această lucrare este asemănător problemei comis-voiajorului (PCV). Problema a fost formulată în anul 1930 de către matematicianul vienez Karl Menger şi presupune să se găsească cel mai scurt drum între un set finit de puncte, cunoscându-se distanțele dintre ele [1]. În anul 1956, Merrill Flood scrie în articolul său că nu există o rezolvare computațională acceptabilă pentru această problemă [2], însă în ciuda acestui fapt, problema a rămas una interesantă în tot acest timp din cauza aplicației practice pe care o posedă. În prezent, Clay Mathematics Institute oferă 1,000,000 $ pentru rezolvarea problemei P=NP, care, dacă ar putea fi rezolvată, atunci ar fi demonstrabil şi că problema comis-voiajorului ar putea fi rezolvată în timp polinomial. Cel mai bun rezultat

înregistrat a fost obținut în anul 2007 cu un algoritm de aproximare, obținând-se o soluție cu 0.0474% mai lungă decât traseul optim [3].

Conținutul lucrării

În primul capitol din această lucrare am prezentat argumentele pentru care un algoritm exact nu poate fi aplicat, în prezent, pentru rezolvarea problemei comis-voiajorului, urmând a prezenta o alternativă care aproximează rezultatul pentru a fi cât mai apropiat de optim. Algoritmul folosit este cel propus de Christofides, care oferă o soluție, în cazul nefavorabil, de cel mult $\frac{3}{2}$ din lungimea optimă a traseului existent. Tratarea constrângerii legate timp va fi abordată în acest capitol, la fel ca și constrâgerea de spațiu. Vor fi prezentate și două metode diferite de realizare a cuplajelor perfecte ca alternativă a algoritmului propus de Edmonds.

În capitolul 2 sunt descrise tehnologiile folosite in dezvoltarea aplicației prezentate. Programul software a fost scris în limbajul de scripting Lua, necesar pentru crearea de addon-uri pe care jocul să le poată executa. Va fi descris şi API-ul pus la dispoziție de Blizzard Entertainment şi folosit pentru extragerea informațiilor necesare pentru manipularea datelor şi pentru crearea de frame-uri necesare pentru a afișa informația obținută. Nu în ultimul rând, pentru generarea sau modificarea imaginilor a fost folosit GIMP.

În capitolul 3 vor fi prezentate funcționalitățile noi aduse interfeței jocului de către aplicația software și modul în care aplicația produce un rezultat în urma comenzilor cerute de către utilizator. Va fi prezentat modul în care interfața grafică a fost gândită și creată, de asemenea vor fi explicate afișările adiționale căruia îi sunt oferite utilizatorului, informații ce sunt incomplete folosind interfața de bază a jocului.

În final, vor fi prezentate concluziile, cu perspective și idei de dezvoltare ulterioară ce pot fi aplicate acestui program software, pentru a deveni popular în rândul utilizatorilor, care vin cu o gamă largă de nevoi.

Capitolul 1 – Fundamentele algoritmice

Se dorește a se implementa un algoritm ce are ca scop atingerea tuturor obiectivelor active de pe harta jocului. Obiectivele vor fi caracterizare și de un termen limită, dorindu-se a ajunge la acestea înainte de expirarea timpului alocat. Harta va fi marcată cu zone inaccesibile sau obstacole, iar algoritmul va trebui să țină cont de acestea atunci când va determina drumul optim între obiective.

Formal, algoritmul caută un lanț Hamiltonian între obiectivele date la intrare, de cost cât mai mic și mai apropiat de soluția optimă. O soluție pentru această problemă ar fi generarea tuturor permutărilor de noduri care alcătuiesc graful nostru. În cazul în care permutarea formează un lanț, calculăm suma tuturor muchiilor care formează lanțul și verificam dacă este mai mică decât minimul curent înregistrat. Un algoritm de acest tip vine cu o complexitate de $O(n!)$, iar un astfel de algoritm nu este practic de utilizat în situații reale, și în special în problema pe care această lucrare își propune să o rezolve.

Este considerat că o astfel de problemă este NP – completă [8], făcând parte din categoria problemelor pentru care o abordare polinomială nu este cunoscută. Obținerea lanțului Hamiltonian reprezintă un caz particular al problemei cunoscută în domeniu cu numele de problema comis-voiajorului, pe care o vom folosi ca punct de plecare pentru obținerea rezultatului dorit.

Descrierea problemei comis-voiajorului

Numele problemei provine din analogia cu un vânzător ambulant care pleacă dintr-un oraș și dorește să viziteze un număr de orașe date, fără a le vizita mai mult de o singură dată, iar la sfârșit să se întoarcă din orașul de unde a pornit. Problema presupune găsirea unui traseu pentru care o parcurgere a sa necesită efort minim (fie din punct de vedere al distanței parcurse ori a duratei călătoriei) [33]. Această problemă este considerată a fi NP-dură, urmând a defini și demonstra în continuare conceptele P, NP, NP-completitudine și NP-duritate.

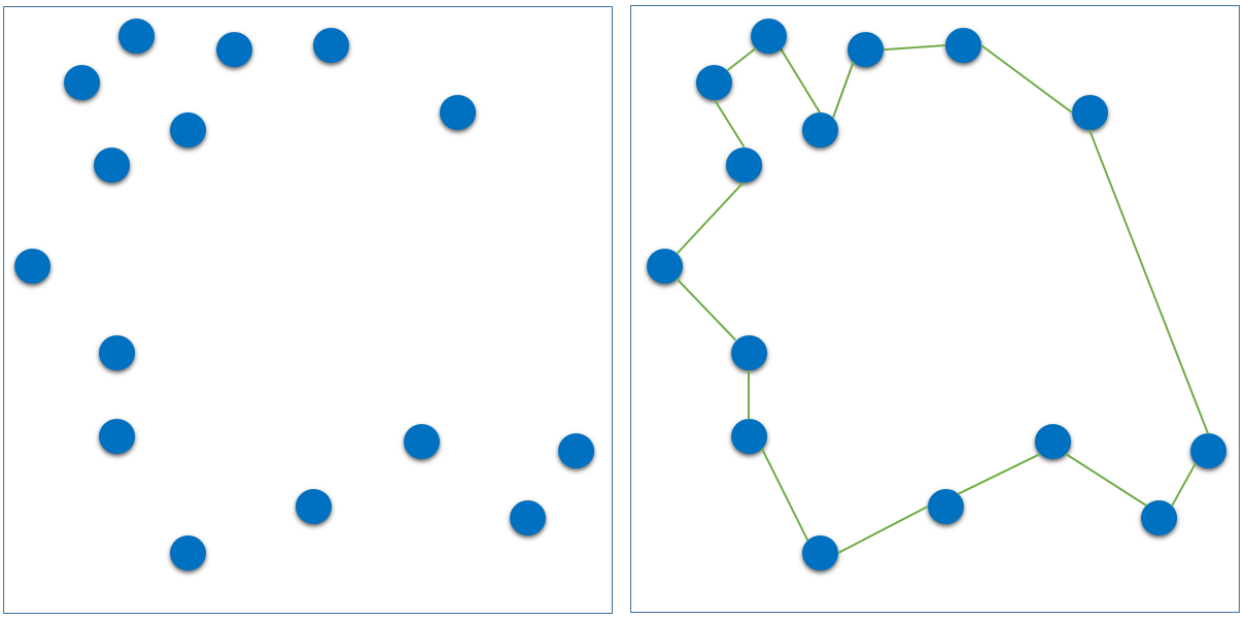


Figura 1: Exemplu de traseu generat pentru problema comis-voiajorului

Clasele de complexitate P și NP

Clasa de complexitate P (polinomială, sau determinist-polinomială) reprezintă un set de probleme de decizie (sau limbaje) L pentru care timpul de rulare pentru obținerea unei decizii, în cazul cel mai nefavorabil, este polinomial. Dat fiind un algoritm A definit pentru o problemă din L și datele de intrare x, atunci timpul de rulare este $p(n)$, unde n reprezintă dimensiunea lui x, iar $p(n)$ este considerată a fi o expresie polinomială. Dacă x aparține limbajului L, atunci algoritmul A oferă ca date de ieșire decizia „da”, altfel decizia va fi „nu” [4].

Clasa de complexitate NP (nedeterminist-polinomială) include clasa de complexitate P, însă algoritmilor problemelor din clasa NP le este permis să efecuteze o operație adițională, și anume operația $choose(b)$, care alege în mod nedeterminist o valoare (spre exemplu, 0 sau 1) iar valoarea este atribuită lui b. Orice algoritm care folosește o astfel de operație choose este considerat a fi un algoritm nederminist. Astfel, putem defini clasa de complexitate NP ca fiind reprezentată de un set de probleme de decizie (sau limbaje) L, pentru care se poate obține o soluție acceptată în mod nedeterminist în timp polinomial. Dat fiind un algoritm A pentru o problemă din L și datele de intrare x, atunci timpul de rulare va fi $p(n)$ polinomial, unde n

reprezintă dimensiunea lui x, iar dacă $x \in L$, atunci există un set de rezultate obținute apelând operația choose, astfel încât algoritmul A oferă ca date de ieșire decizia „da”. Dacă $x \notin L$, atunci orice rezultat s-ar obține apelând operația choose, la ieșire se va obține întotdeauna decizia „nu” [4].

Spunem despre un limbaj L, care definește o problemă de decizie, că este reductibil în timp polinomial la un limbaj M, dacă există o funcție f calculabilă în timp polinomial, și pentru datele de intrare x, le transformă în $y = f(x)$ astfel încât $x \in L \Leftrightarrow y \in M$. Folosim notația $L \xrightarrow{Poly} M$ pentru a indica faptul că limbajul L este reductibil în timp polinomial la M. [4]

Un limbaj M definit de o problemă de decizie este considerat NP-dur dacă orice alt limbaj L din clasa NP este reductibil în timp polinomial la M. Formal, limbajul M este NP-dur dacă $\forall L \in NP,\ L \xrightarrow{Poly} M$. Dacă un limbaj M este NP-dur, și se află și în clasa de complexitate NP, atunci spunem despre M că este un limbaj NP-complet. [4]

Problema comis-voiajorului, privită ca problemă de decizie, este o problemă NP-completă. Ea este definită astfel: date fiind o mulțime de orașe, distanțele dintre ele și o valoare k, să se determine dacă se obține un traseu care trece prin fiecare oraș o singură dată, de cost mai mic sau egal decât valoarea k dată.

Teorema 1:

Problema comis-voiajorului de decizie face parte din categoria problemelor NP-complete.

Pentru demonstrația acestei teoreme, va fi introdusă o lemă.

Lema 1:

Problema Comis-Voiajorului este o problemă NP.

Demonstrație pentru lema 1:

Definim un algoritm nedeterminist care acceptă intrări pentru problema comis-voiajorului. Numerotăm nodurile (obiectivele) de la 1 la n. Apoi, alegem o secvență S de $n + 1$

noduri, și verificăm dacă respectă proprietatea că fiecare nod apare o singură dată în secvență, cu excepția primului și ultimului nod, care sunt egale. Verificăm dacă am obținut un traseu cu un cost mai mic sau egal cu o valoare k, primită ca dată de intrare la început. Un astfel de algoritm rulează în timp polinomial. Dacă șirul S generează un ciclu de cost mai mic sau egal decât k, atunci datele de ieșire ale algoritmului vor fi echivalente cu un răspuns pozitiv, adică algoritmul a găsit un drum optim pentru problema comis-voiajorului. Altfel, returnează un răspuns negativ. De aici rezultă faptul că problema comis-voiajorului este NP [4]. ■

Demonstrație pentru teorema 1:

Pentru ca problema comis-voiajorului să fie NP-completă, ea trebuie să facă parte din clasa de complexitate NP și să fie o problemă NP-dură. Apartenența la clasa de complexitate NP a fost demonstrată cu ajutorul lemei 1.

Pentru a arăta că PCV este NP-dură, vom arăta că ciclul Hamiltonian $\leq_p$ PCV (ciclul hamiltonian se poate reduce la problema comis-voiajorului). Știm despre problema ciclului Hamiltonian că este o problemă NP-completă.

Fie $G = (V, E)$ o instanță al unui ciclu Hamiltonian, iar $G' = (V, E')$ graf complet, unde

$$E' = \{ (i,j) | i,j \in V, i \neq j \} \quad (1)$$

Funcția de cost este definită astfel:

$$t(i,j) = \begin{cases} 0, (i,j) \in E \\ 1, (i,j) \notin E \end{cases} \quad (2).$$

Vrem să arătăm că G conține un ciclu Hamiltonian dacă și numai dacă G' conține un ciclu de cost 0.

⇒

Presupunem existența unui ciclu Hamiltonian h în graful G. Atunci, este evident că fiecare muchie din h va avea costul 0 în G' deoarece G' conține toate muchiile din E. De aici rezultă că graful G' conține ciclul h de cost 0.

⇐

Presupunem că G' conține un ciclu Hamiltonian h' de cost 0. Costul fiecărei muchii din E' poate fi ori 0, ori 1 conform ecuației (2). Cum costul ciclului h' reprezintă suma costurilor tuturor

muchiilor, de aici rezultă că fiecare muchie din h' are costul 0, deci h' conține doar muchii care se găsesc în E.

Astfel, problema comis-voiajorului este NP-completă. ■

Formalizare

Fie graful $G = (V, E)$ oarecare și un cost pentru fiecare muchie aparținând lui E. Problema TSP este aceea de a găsi un ciclu Hamiltonian de cost minim, unde costul este reprezentat de suma tuturor costurilor muchiilor care fac parte din circuit. [9]

Algoritmi de aproximare

Un mod de rezolvare al problemelor NP – complete este folosirea algoritmilor de aproximare [10]. Un algoritm se numește algoritm de α – aproximare dacă rulează în timp polinomial iar rezultatul este întotdeauna cel mult $\alpha\, OPT$ pentru o problemă de minimizare, unde OPT este soluția optimă iar α reprezintă factorul de aproximare . Un astfel de algoritm este cel propus de Christofides, care oferă un factor de aproximare de $\frac{3}{2}$ și care va fi folosit în această lucrare.

Algoritmul lui Christofides

Fie $G = (V, E)$ o instanță a problemei comis-voiajorului și o funcție $d : VxV \rightarrow \mathbb{R}$ ce reprezintă costul unei muchii. G este un graf complet, adică există legături între oricare două noduri aparținând lui V. În plus, $\forall\, u, v, x \in V$, se respectă regula triunghiului, și anume

$$d(u, v) + d(v, x) \geq d(u, x) \quad (3).$$

Altfel spus, pentru oricare ar fi două noduri aparținând lui V, muchia care le leagă este cea mai scurtă distanță dintre ele.

Algoritmul:

1) Se construiește un arbore de cost minim T între vârfurile din V ale grafului complet G.

2) Se consideră toate nodurile din arborele creat la pasul anterior care au gradul impar, și fie H subgraf al lui G format cu aceste noduri. Cu ele se va construi un cuplaj perfect de cost minim, M.

3) Obținem $G' = T \cup M$, graf conex și cu toate nodurile de grad par. Se caută un ciclu Eulerian al grafului G'.

4) Ciclul Eulerian se transformă în lanț Hamiltonian aplicând scurtături. Pentru fiecare nod din ciclu care a fost vizitat anterior, acesta nu va fi considerat pentru alcătuirea lanțului și se va trece la următorul nod.

Se va arăta faptul că algoritmul lui Christofides este corect. Pentru început, se arată faptul că un arbore are întotdeauna un număr par de noduri cu grad impar, condiție necesară pentru realizarea unui cuplaj perfect. Se numește cuplaj într-un graf un set de muchii care nu au noduri comune. Un cuplaj perfect presupune ca toate nodurile din graf să formeze un cuplaj (să facă parte dintr-o muchie). Un cuplaj perfect de cost minim presupune ca suma ponderilor muchiilor ce formează cuplajul să fie minimă.

Se va demonstra apoi că graful format din reuniunea dintre arborele de cost minim și cuplajul perfect de cost minim determină un ciclu Hamiltonian al cărui cost nu depășește $\frac{3}{2}$ din lungimea optimă a problemei comis-voiajorului

Teorema 2:

Fie $T = (V, E)$ arbore oarecare și o funcție $grad: V \to \mathbb{N}$ care calculează gradul fiecărui nod din V. Fie

$$Y = \{y \in V \mid grad(y) = m,\ m\ impar\} \quad (4).$$

Atunci,

$$\sum_{i=1}^{|Y|} y_i = n, n\ par \quad (5),$$

iar

$$|Y| = par \quad (6).$$

Demonstrație:

Pentru orice arbore, suma gradelor nodurilor este egal cu de 2 ori numărul de muchii.

Pentru un arbore T = (V, E) oarecare și o funcție $grad: V \to \mathbb{N}$ avem

$$X = \{x \in V \mid grad(x) = n, n\ par\} \quad (7),$$

și

$$Y = \{y \in V \mid grad(y) = m, m\ impar\} \quad (6).$$

Atunci,

$$\sum_{i=1}^{X} x_i + \sum_{i=1}^{Y} y_i = 2|E| \quad (8),$$

Din ecuația (9) deducem următoarea ecuație:

$$\sum_{i=1}^{Y} y_i = 2|E| - \sum_{i=1}^{X} x_i = k \quad (9).$$

Cum valoarea k obținută în ecuația (9) reprezintă diferența dintre două numere pare, atunci şi numărul k este par, demonstrând astfel ecuaţia (5), şi cum suma unor numere impare are ca rezultat un număr par, putem deduce cardinalitatea mulţimii Y ca fiind un număr par. ■

Fie T arbore de cost minim obținut din graful $G = (V, E)$ instanțiat pentru problema comis-voiajorului. Considerăm $G_1 = (Y, E')$ subgraful complet format din nodurile cu grad impar ale arborelui T. Din teorema 2, demonstrată mai sus, avem $|Y|$ număr par, astfel fiind satisfăcută proprietatea pentru existența unui cuplaj perfect între nodurile grafului G_1. Fie $M = (Y, E_M)$ cuplajul perfect de cost $c(M)$ minim al grafului G_1, cu $E_M \subseteq E'$.

Teorema 3 [5]:

Fie H un ciclu Hamiltonian în graful G, un arbore T de cost minim format din muchiile grafului G, un cuplaj perfect M format din nodurile cu grad impar ale arborelui T și o funcție $c: X \to \mathbb{R}$ o funcție ce definește costul unui graf. Atunci, avem următoarea inegalitate:

$$c(H) \leq c(T) + c(M) \leq \frac{3}{2} c(OPT) \quad (10),$$

unde OPT este valoarea optimă pentru problema comis-voiajorului.

Pentru demonstrarea acestei teoreme, ne vom folosi de următoarea lemă [5]:

Lema 2:

Fie graful $G = (V, E)$ neorientat, și $M = (V, E_M)$ un cuplaj perfect de cost minim. Atunci, avem următoarea inegalitate:

$$c(M) \leq \frac{1}{2} c(OPT) \quad (11).$$

Demonstrație:

Fie $OPT = (x1, x2, \ldots, xn)$ o permutare care generează traseul optim în graful G cu n noduri pentru problema TSP. Considerăm două seturi, M_1 și M_2, astfel:

$$M1 = \{(x1, x2), (x3, x4), \ldots, (xn-1, xn)\} \quad (12),$$

$$M2 = \{(x2, x3), (x4, x5), \ldots, (xn, x1)\} \quad (13).$$

M_1 și M_2 sunt cuplaje în G și

$$c(M_1) + c(M_2) = c(OPT) \quad (14).$$

Putem considera că $c(M_1) \leq c(M_2)$ fără a restrânge generalitatea, de unde se poate deduce că

$$c(M) \leq c(M_1) \leq \frac{1}{2} c(OPT) \quad (15). \blacksquare$$

Demonstrația teoremei 3:

Conform [5],

$$c(T) \leq c(H) \leq c(OPT) \quad (16),$$

unde H este ciclu Hamiltonian G.

Considerăm $G' = (V, E_T \cup E_M)$ un graf parțial cu toate nodurile de grad par al lui $G = (V, E)$, obținut din reuniunea muchiilor arborelui de cost minim $T = (V, E_T)$ şi cele ale cuplajului perfect $M = (Y, E_M)$ format din nodurile de grad impar ale arborelui T. G' conține, astfel, un ciclu Eulerian. Fiindcă un ciclu Eulerian parcurge fiecare muchie din graf o singură dată, $(\forall)\, v_i \in V, i = \overline{1, |V|}$, atunci fiecare nod va fi vizitat cel puțin o dată. Fie $c(G_e)$ costul ciclului Eulerian, mai exact:

$$c(G_e) = c(T) + c(M) \quad (17).$$

Conform lemei anterioare, avem că:

$$c(M) \leq \frac{1}{2} c(OPT) \quad (18).$$

Din relațiile (16), (17) și (18) deducem că

$$c(G_e) < \frac{3}{2}\, c(OPT) \quad (19). \blacksquare$$

Complexitatea algoritmului propus pentru rezolvarea problemei este determinată de modul în care arborele de cost minim și cuplajele perfect de cost minim sunt obținute. Considerând că se aplică cei mai buni algoritmi pentru obținerea rezultatului, o complexitate estimată ar fi $O(n^3)$ care provine de la generarea cuplajului de cost minim, care este cel mai costisitor pas al algoritmului. [5].

Implementarea algoritmului

Algoritmul primește la intrare toate hărțile de interes pentru utilizator care pot conține obiective. Formatul hărților este considerat bidimensional. Iterativ, este parcursă fiecare hartă pentru a extrage obiectivele existente la momentul respectiv. Acestea vor fi reținute într-o structură care va conține mai multe informații despre fiecare misiune în parte. De interes pentru generarea lanțului Hamiltonian vor fi doar coordonatele fiecărui nod ce reprezintă locația pe hartă a acestuia și timpul până la expirarea obiectivului.

După ce au fost reținute toate misiunile, se va genera un graf complet format din toate nodurile. Un nod corespunde unei misiuni. Muchiilor din graful format li se va asocia un cost, care este reprezentat de distanța dintre punctele de pe hartă care formează muchia. Formal, costul reprezintă distanța a două puncte în plan.

Fie O_1 și O_2 două obiective, $(O_1.x,\ O_1.y)$, $(O_2.x,\ O_2.y)$ coordonatele lor pe hartă. Atunci distanța între O_1 și O_2 va fi d, unde

$$d = \sqrt{(O_2.x - O_1.x)^2 + (O_2.y - O_1.y)^2} \quad (20).$$

Tratarea constrângerii de spațiu

Există situații când drumul între două obiective este obstrucționat de un factor ce ține de natura hărții, un exemplu concret fiind Shaladrassil, o regiune din joc în care jucatorii nu au acces. Pe hartă, obstacolele sunt reprezentate de poligoane închise, care marchează granițele zonei inaccesibile. Unui astfel de poligon i se vor reține vârfurile, care vor fi la rândul lor caracterizate de două coordonate x și y. Ordinea parcurgerii vârfurilor este reprezentativă formei poligonului.

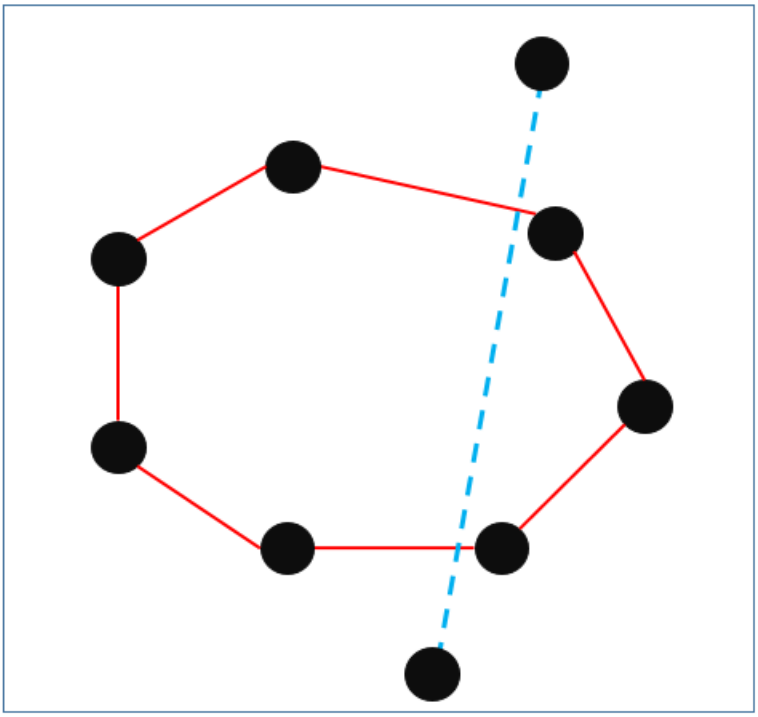


Figura 2: Exemplificare a unei muchii care trece printr-o zonă inaccesibilă

Pentru fiecare muchie M cu capetele $(m1, m2)$ din graful original, verificăm dacă aceasta se intersectează cu poligonul astfel: pentru fiecare muchie C cu capetele $(c1, c2)$ a obstacolului, determinăm ecuațiile dreptelor care trec prin punctele care formează pe M respectiv pe C.

$$\frac{(x - m1.x)}{(m2.x - m1.x)} = \frac{(y - m1.y)}{(m2.y - m1.y)} \quad (21);$$

$$\frac{(x - c1.x)}{(c2.x - c1.x)} = \frac{(y - c1.y)}{(c2.y - c1.y)} \quad (22).$$

Ecuația dreptei care trece prin muchia M:

$$a_1 M.x + b_1 M.y + c_1 = 0 \quad (23).$$

Ecuația dreptei care trece prin muchia C:

$$a_2 C.x + b_2 C.y + c_2 = 0 \quad (24).$$

Înmulțind ecuația (23) cu b_2 și ecuația (24) cu b_1 obținem:

$$a_1 b_2 M.x + b_1 b_2 M.y + c_1 b_2 = 0 \quad (25);$$

$$a_2 b_1 C.x + b_1 b_2 C.y + c_2 b_1 = 0 \quad (26).$$

Din (25) și (26) obținem coordonatele punctului de intersecție a celor două drepte. Fie $P(x, y)$ acest punct, unde

$$x = \frac{(c_2 b_1 - c_1 b_2)}{(a_1 b_2 - a_2 b_1)} \quad (27);$$

$$y = \frac{(-c_1 - a_1 x)}{b_1} \quad (28).$$

Odată obținut punctul de intersecție a celor două drepte, verificăm dacă acesta se găsește pe muchiile inițiale M și C folosind următoarea formulă pentru a determina dacă un punct se află pe un segment:

$$P(x, y) \in [AB] \Leftrightarrow x \in [A.xB.x] \vee y \in [A.yB.y] \quad (29).$$

Pot fi 3 situații relevante pentru algoritm: existența a 0, 1 sau mai multe puncte de intersecție a unei muchii cu un obstacol. Dacă avem 0 puncte de intersecție, atunci muchiei i se va asocia ponderea normală. Un punct de intersecție înseamnă că drumul nu este obstrucționat de constrângere, deci și în acest caz muchia va avea pondere normală. Mai mult de 2 puncte de intersecție înseamnă că ponderea muchiei trebuie recalculată. Numărul punctelor de intersecție poate varia în funcție de forma constrângerii sau de panta dreptei care o intersectează, însă relevante pentru algoritm vor fi primul, respectiv ultimul punct de intersecție, deoarece acestea reprezentă punctul de întâlnire, respectiv de părăsire a obstacolului.

Odată determinate cele două puncte, se va determina din ce muchii care formează poligonul fac parte. Pasul următor este să identificăm cel mai scurt drum între punctul de intrare

și cel de ieșire. Identificarea se va face în $O(n)$, parcurgând constrângerea, unde n reprezintă numărul de muchii ce formează constrângerea. Parcurgerea se face pe marginile sale în sensul acelor de ceasornic, respectiv invers, din punctul de start, până este întâlnit punctul de ieșire. Vom obține două sume, adunând ponderile segmentelor parcurse. Ponderea unui segment reprezintă distanța în plan a celor două puncte care îl determină. Cea mai mică sumă este reprezentativă pentru cel mai scurt drum de urmat pentru a ocoli constrângerea. Ponderea finală a drumului între obiective va fi suma distanțelor de la obiectivul 1 la punctul de intrare în constrângere, valoarea drumului minim care înconjoară constrângerea și distanța de la punctul de ieșire la obiectivul 2.

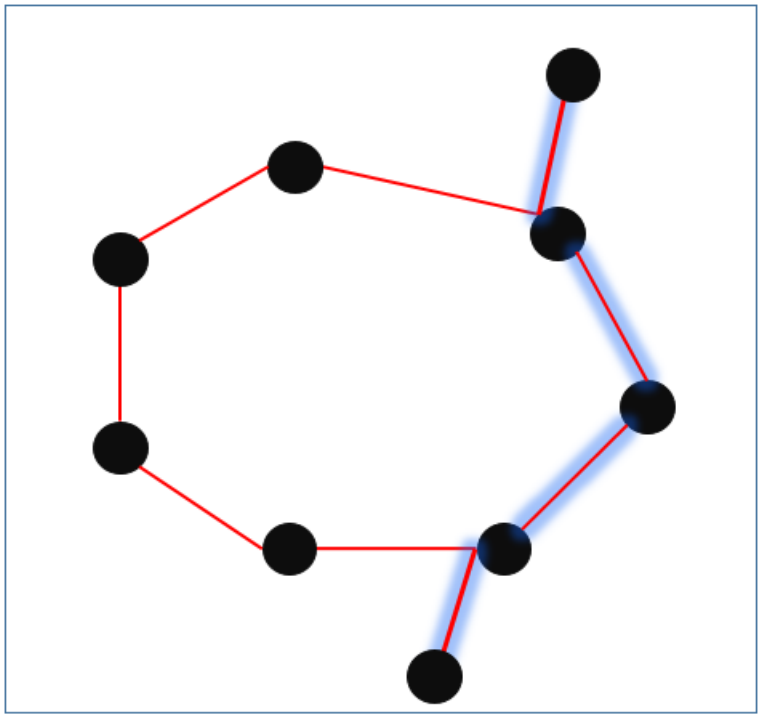


Figura 3: Mod de ocolire a constrângerii pe distanța cea mai scurtă de ocolire a sa

Generarea arborelui de cost minim

Odată format graful complet, se dorește prelucrarea acestuia pentru obținerea grafului de cost minim. Este folosită o strategie de tip greedy ce va fi descrisă în următorul algoritm generic, prin care arborelui îi este adăugată o muchie nouă la fiecare pas. Se pleacă de la premisa că la fiecare pas, A reprezintă un subset al unui arbore de cost minim, T [11].

Algoritmul generic:

Pas 1. $A = \emptyset$

Pas 2. Cât timp A nu formează un arbore de cost minim

Pas 2.1. Găsește (u, v) pereche potrivită pentru A

Pas 2.2. $A = A \cup \{(u, v)\}$

Pas 3. Returnează A

Esența algoritmului o reprezintă găsirea unei perechi de vârfuri (u, v) potrivită pentru A. Existența acestei perechi este asigurată de premisa inițiala, care afirmă că există T arbore de cost minim. Cum A este un subset al lui T, atunci trebuie să existe o pereche de noduri (u, v) care aparțin lui T și nu aparțin lui A. O astfel de pereche este potrivită pentru A.

Teorema 4:

Fie $G = (V, E)$ graf neorientat conex ponderat cu ponderile p. Fie A o submulțime de muchii a lui E, inclusă întru-un arbore de cost minim a grafului G. Fie $(S, V - S)$ o tăietura în graf astfel încât tăietura nu taie și pe A. $(S, V - S)$ conțin perechi de noduri cu proprietatea că dacă un capăt aparține lui S, atunci celălalt capăt aparține lui $V - S$. Fie (u, v) o pereche care formează o muchie de cost minim care aparține lui $(S, V - S)$. Atunci, (u, v) este o pereche potrivită pentru A.

Demonstrație*:*

Fie A o submulțime de muchii a arborelui de cost minim T. Presupunem existenţa unei perechi minime (u, v) care nu aparține lui T. Atunci, construim $T' = A \cup \{(u, v)\}$. Arătăm că perechea (u, v) este o pereche potrivită pentru A.

Cum (u, v) nu este o muchie în T, înseamnă că adăugând această muchie se formează un ciclu pe drumul unic de la nodul *u* către nodul *v*, așa cum se poate vedea în figura 1. Cum (u, v) aparține lui $(S, V - S)$ înseamnă că mai există o muchie (x, y) în T arbore de cost minim aparținând lui $(S, V - S)$, pe drumul unic de la u la v. (x, y) nu taie graful A, lucru stabilit în ipoteza teoremei. Formăm arborele T', eliminând muchia (x, y) și adăugând muchia (u, v). Urmează să arătam că T' este arbore de cost minim, de asemenea.

Cum (u, v) este, prin definiție, o muchie de cost minim din tăietura $(S, V - S)$, și (x, y) aparține lui $(S, V - S)$, înseamnă că $p(u, v) \leq p(x, y)$, p reprezentând ponderile muchiilor în graf. Prin urmare,

$$p(T') = p(T) - p(x, y) + p(u, v) \leq p(T) \qquad (30).$$

Dar T este arbore de cost minim, deci $p(T) \leq p(T')$. Prin urmare, $T = T'$ arbore de cost minim.

Rămâne de arătat că (u, v) este o pereche potrivită pentru A. Avem A submulțime a lui T' și $(x, y) \notin A$. Deci, $A \cup \{(u, v)\}$ este o submulțime a lui T', și cum T' este arbore de cost minim, atunci și muchia determinată de perechea de noduri (u, v) este potrivită pentru A. ■

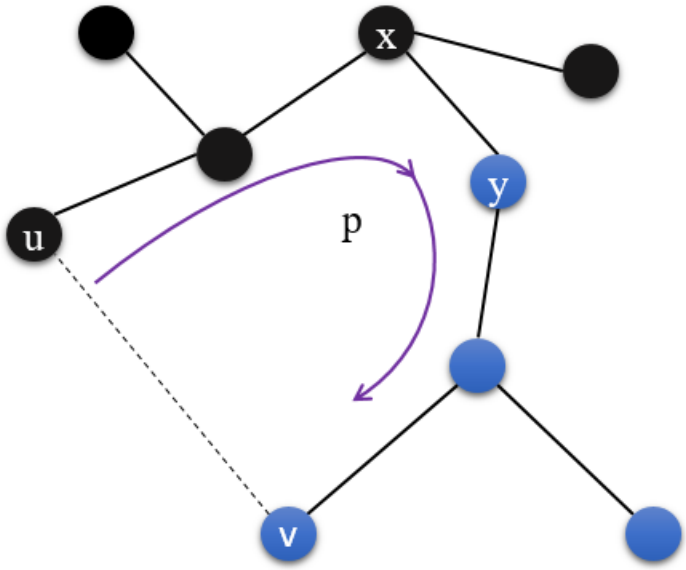


Figura 4: Exemplificarea unei tăieturi (S, V-S)

Pentru implementarea algoritmului generic, lucrarea va prezenta algoritmul lui Kruskal de generare a arborilor de cost minim.

Algoritmul lui Kruskal

Având graful $G = (V, E)$ ponderat cu ponderile p, dorim ca acesta să fie complet pentru a putea aplica algoritmul. În caz contrar, putem considera muchiile care lipsesc ca având lungime infinită, așa cum menționează autorul algoritmului în lucrarea [12].

Algoritmul lui Kruskal[11]:

1. $A = \emptyset$, G=(V, E)
2. Pentru fiecare $v \in V$, formează $|V|$ componente conexe de forma $P_i = \{v_i\}$, unde v_i sunt, inițial, noduri izolate în graful A
3. Sortează muchiile E după ponderea p
4. Pentru fiecare muchie $(u, v) \in E$, în ordine crescătoare după sortare, verifică dacă nodurile u și v fac parte din aceeași componentă conexă. Dacă nu, atunci $A = A \cup (u, v)$ și $P_{uv} = P_u \cup P_v$
5. Returnează A

Complexitatea algoritmului variază în funcție de modul de implementare ales și structurile de date folosite. Inițializarea mulțimii A este efectuată în timp $O(1)$, inițializarea pădurilor se face de asemenea în timp $O(V)$ iar sortarea se efectuează în timp $O(E\ log\ E)$.

Pasul 4 se execută de maxim |E| ori, până când toate componentele conexe vor fi reduse la una singură, care va fi chiar arborele de cost minim. Pentru verificarea apartenenței la o componentă conexă, se ia fiecare componentă în parte și se verifică în mod iterativ. Cum sunt |V| componente conexe și maximul de elemente pe care îl pot conține este |V|, atunci complexitatea acestei operații este $O(V^2)$. Reactualizarea componentelor conexe se execută în timp $O(V)$ pentru fiecare din cele două componente în parte. Reactualizarea componentei A se realizează în timp unitar $O(1)$. De aici rezultă o complexitate de $O(E * V^2)$, care va fi și timpul total de execuție al algoritmului Kruskal.

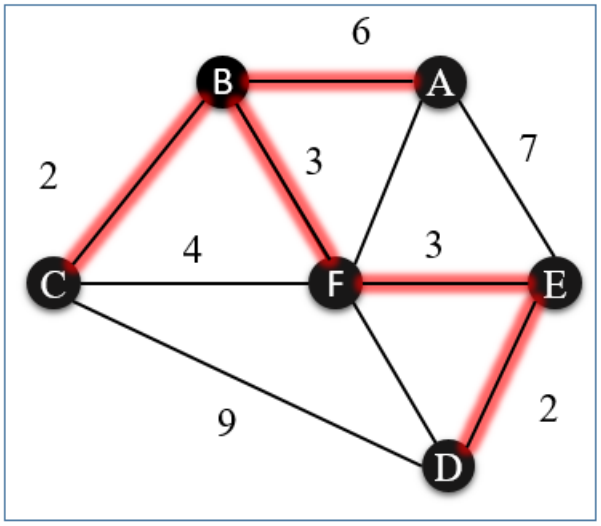


Figura 5: Mod de determinare a arborelui de cost minim folosind algoritmul lui Kruskal. Ordinea selecțiilor muchiilor este BC,DE,BF,EF,BA.

Cuplaj perfect de cost minim

Odată obținut arborele de cost minim, identificăm mulțimea de noduri S care au grad impar, și obținem din S un cuplaj perfect de cost minim [7].

Fie $G = (V, E)$ un graf oarecare ponderat cu ponderile c. Un cuplaj este un subset de muchii $E' \subseteq E$ cu proprietatea că fiecare nod în V are cel mult o muchie incidentă din E'. Dacă toate nodurile din V au exact o muchie incidentă din E', atunci cuplajul se numește perfect. Dacă și suma ponderilor muchiilor care formează cuplajul perfect este minimă, atunci E' se numește cuplaj perfect de cost minim [12].

În 1965, Edmonds a inventat algoritmul Blossom care rezolvă problema determinării unui cuplaj perfect minimal în timp polinomial. Algoritmul are o complexitate timp de $O(n^2\, m)$, unde n reprezintă numărul de noduri și m numărul de muchii din graf [12]. Algoritmul a fost, ulterior, prelucrat și îmbunătățit, menționi notabile fiind Lawler[6] care a obținut o complexitate de $O(n^3)$ sau Gabow, cu o complexitate de $O(nm\ log\ n)$ [13].

În această lucrare au fost implementate două variante ale algoritmului cuplajelor perfecte de cost minim: o variantă aplică metoda Greedy de cuplare, alegând la fiecare pas muchia cu

ponderea cea mai mică, iar a doua variantă reprezintă un mod de cuplare ce ține cont de parcurgerea arborelui de cost minim și realizarea cuplajului între noduri în ordinea descoperirii lor în arbore, urmând mai apoi a compara rezultatele obținute.

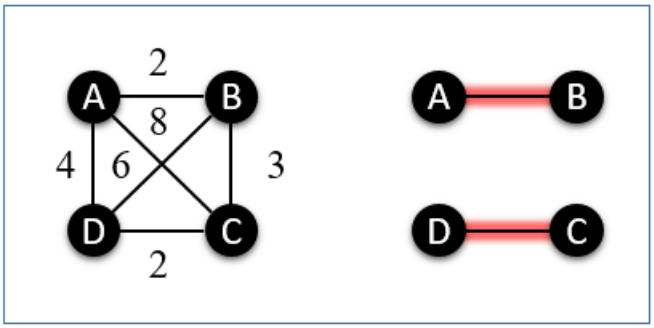


Figura 6: Exemplu de obținere a unui cuplaj perfect de cost minim în graful complet determinat de nodurile A, B, C și D

Varianta Greedy

Fie S o submulțime de vârfuri a grafului $G = (V, E)$ cu proprietatea că S reprezintă nodurile de grad impar a arborelui de cost minim T a grafului G. Se va considera graful complet ponderat $G' = (S, E')$ cu ponderile p' alcătuit din nodurile din mulțimea S. Se vor parcurge muchiile din graful G', după ce au fost sortate, în ordine crescătoare, în căutarea nodurilor care nu sunt cuplate. Un nod cuplat reprezintă un nod care este incident cu o singură muchie din graful M. Inițial, graful M conține doar nodurile mulțimii S, fără muchii între ele. Muchiile se vor adăuga în mod iterativ până când toate nodurile din S vor forma un cuplaj.

Algoritm:

1) $E_M = \emptyset, I = \{0,0,\dots,0\}, |I| = |S|$
2) Sortăm E' în ordine crescătoare după ponderile muchiilor
3) Pentru fiecare $(x, y) \in E'$, verificăm dacă x sau y au format un cuplaj. Dacă da, atunci $E_M = E_M \cup \{(x, y)\}$ și $I = I \cup \{x, y\}$
4) Returnează E_M

Inițializarea mulțimilor I și M se efectuează în timp $O(1)$. Mulțimea I va fi folosită pentru a marca ce noduri au fost cuplate până la momentul curent, valoarea 0 reprezentând faptul că un nod nu a fost cuplat, și 1 indicând faptul că a fost cuplat.

Sortarea necesită timp de execuție $O(E' \log E')$. Pasul 3 se efectuează de $|E'|$ ori, actualizarea lui E_M se efectuează în timp $O(1)$ iar determinarea dacă un nod a fost cuplat sau nu se efectuează în timp $O(1)$, verificând pozițiile x și y din I pentru nodurile de la pasul curent.

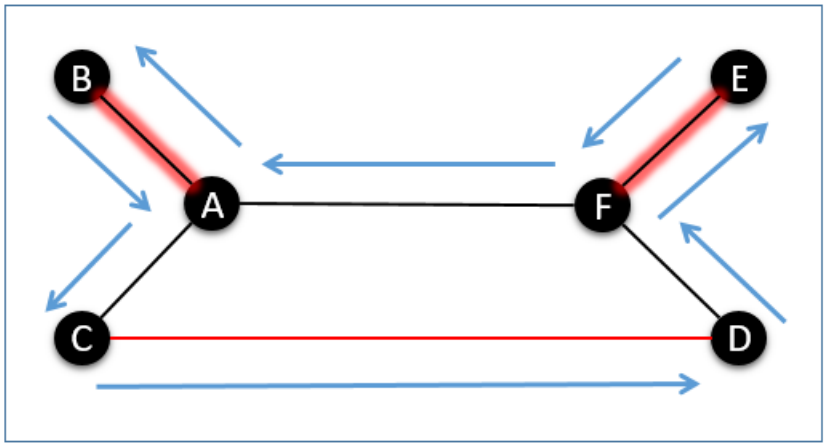


Figura 7: Exemplu de efectuare a cuplajului folosind metoda Greedy

Varianta îmbunătățită de obținere a cuplajelor perfecte

Această metodă de cuplare a nodurilor în scopul obținerii cuplajelor perfect de cost cât mai mic prezintă îmbunătățiri față de metoda Greedy, care nu garantează un rezultat apropiat de soluția optimă. Metoda constă în parcurgerea arborelui de cost minim și cuplând nodurile de grad impar pe măsură ce acestea sunt identificate în arbore.

Se pornește prin determinarea celui mai costisitor drum din punct de vedere al ponderilor muchiilor din arborele de cost minim. O parcurgerea începe la nodul de start și se termină într-o frunză. Drumul de cost maxim determinat va fi referit ca și *drum principal*, restul drumurilor fiind *drumuri secundare*. Fiind drumul cel mai lung, parcurgerea acestuia va avea prioritatea cea mai mică, alegând parcurgerea drumului secundar ori de câte ori este întâlnită o ramificație. Acest aspect este important în parcurgerea arborelui deoarece știm că orice alt drum are cost mai

mic decât cel maxim, deci se minimizează costul întoarcerilor din arbore atunci când se alege un astfel de drum.

Drumul principal se va parcurge iterativ iar pentru fiecare ramificație a drumului în drumuri secundare, se va efectua o parcurgere similară cu o parcurgere Stânga – Dreapta – Rădăcină întâlnită la arborii binari. Pe măsură ce este întâlnit un nod cu un grad impar, acesta va fi reținut într-o structură, pe ultima poziție. Această structură are rolul de a memora ordinea în care nodurile de grad impar au fost descoperite. Pentru realizarea cuplajului perfect, se vor cupla nodurile din structură două câte două în ordinea parcurgerii, spre exemplu nodul de pe poziția 1 cu nodul de pe poziția 2, urmat de nodul de pe poziția 3 cu nodul de pe poziția 4 etc.

Această metodă asigură cuplarea unor noduri apropiate în sensul de parcurgere al arborelui, minimizând astfel parcurgerea arborilor în sensul că nu se vor face cuplaje între drumuri secundare diferite, lucru ce se întâmplă la metoda Greedy. Din cauza acestui fapt, drumurile obținute prin metoda Greedy sunt mai lungi deoarece nu se ține cont ramificarea arborelui din care un nod face parte.

Pentru determinarea celei mai lungi ramificații se aplică un algoritm de tipul căutării în adâncime care necesită timp de compilare $O(E)$, E fiind numărul de muchii a arborelui de cost minim. Apoi, pentru parcurgerea ramurei celei mai lungi, se vor efectua $O(E')$ pași, E' fiind muchiile care alcătuiesc această ramificație. La fiecare pas se pot întâlni un număr arbitrar de ramificații, sau drumuri secundare, a căror parcurgere necesită maxim |E| pași, deci complexitatea totală a parcurgerii devine $O(E' * k * E)$. Formarea cuplajelor necesită $O\left(\frac{S}{2}\right) = O(S)$, unde S reprezintă numărul de noduri de grad impar a arborelui. Cum S este o submulțime a lui V, unde V reprezintă mulțimea tuturor nodurilor din arbore, iar V = E-1, putem aproxima formarea cuplajelor la $O(E)$. În concuzie, complexitatea acestui algoritm este $O(E' * k * E)$.

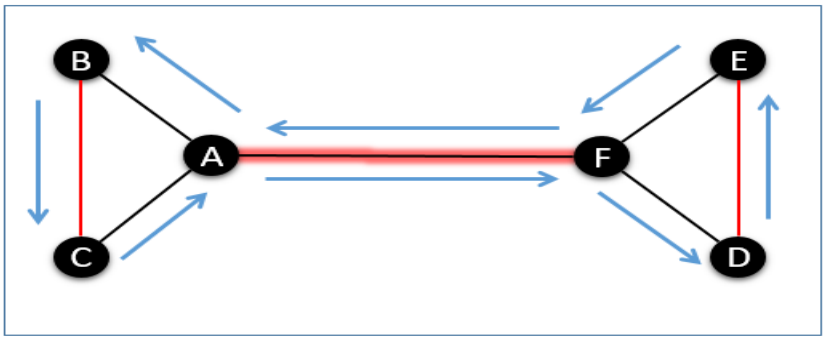


Figura 8:Exemplu de cuplaj între nodurile de grad impar folosind metoda îmbunătățită

În figurile 7 și 8 este exemplificat modul cum algorimii funcționează. Datorită faptului că au fost cuplate muchiile de cost minim în cazul Greedy, la final au rămas necuplate două noduri foarte îndepărtate unul față de celălalt. În schimb, prin a doua metodă detaliată mai sus, a fost evitată generarea unui cuplaj care ar produce salturi foarte mari în graf.

Determinarea ciclului Eulerian

Avem graful $G = (V, E)$ complet, neorientat și ponderat, $T = (V, Et)$ arborele de cost minim a grafului G $și$ $M = (Vm, Em)$ cuplaj perfect de cost minim între nodurile de grad impar ale lui T. Obținem $G' = (V, E')$, unde $E' = Et \cup Em$.

Teorema 5:

Condiția necesară și suficientă ca un graf să fie Eulerian este ca toate vârfurile din graf să fie de grad par.

Demonstrație:

Necesitate [16]: Fie $G = (V, E)$ un graf Eulerian, și fie Z ciclul Eulerian a grafului. Presupunem că punctul de plecare și de sosire este v aparținând lui V. Fiecare nod întâlnit de-a lungul ciclului Eulerian va fi adiacent cu două muchii, una care întâlnește nodul și una care îl părăsește, acest lucru întâmplându-se ori de câte ori un nod este vizitat. Excepție există doar la nodul de start, care în momentul inițial este adiacent doar cu o singură muchie care părăsește nodul, dar acesta va redeveni nod de grad par la final fiindcă i se va adăuga o muchie adiacentă, care închide ciclul Eulerian. O astfel de parcurgere asigură faptul că fiecare nod va avea gradul par.

Suficiența [16]: Fie $G = (V, E)$ un graf conex, cu fiecare vârf având grad par. Presupunem prin reducere la absurd că G nu ar fi graf Eulerian. Cum gradele vârfurilor sunt mai mari sau egale cu 2, atunci G conține un ciclu. Fie Z un astfel de ciclu de lungime maximă. Putem deduce că graful obținut din G prin eliminarea muchiilor lui Z e un graf în care fiecare

nod are gradul par. Fie C o componentă a grafului nou obținut. Cum gradele nodurilor lui C sunt pare, atunci G conține un ciclu Eulerian. Fie v un nod aparținând lui C, cu proprietatea că v aparține și drumului Z în graful G. Fie Z' ciclul Eulerian în graful C. Atunci, $Z \cup Z'$ vor forma un ciclu închis care începe și se termină în punctul v. Astfel, obținem un nou ciclu în G, care depășește lungimea lui Z care s-a presupus a fi de lungime maximă. Presupunerea este, astfel, falsă, iar G este graf Eulerian. ■

Graful G' definit anterior satisface condiția necesară și suficientă pentru ca un graf să fie Eulerian deoarece M este un cuplaj perfect al nodurilor din T cu grad impar. Această lucrare folosește algoritmul căutării în adâncime pentru găsirea ciclului Eulerian în graful G.

Algoritm:

Pas 1: $Visited = \emptyset, Start = v$

Pas 2: Pentru fiecare muchie m adiacentă cu v, marchează noul nod de start ca fiind capătul opus al muchiei, marchează muchia, o șterge din graf și continuă căutarea. Dacă se ajunge la un nod care nu mai conține muchii adiacente, însă nu toate muchiile din graf au fost marcate, se întoarce recursiv la o stare anterioară. Algoritmul continuă până când a fost găsită o soluție, sau au fost epuizate toate posibilitățile fără succes, caz în care întoarce mulțimea vidă.

Pas 3: Returnează ordinea în care au fost parcurse muchiile.

Aplicând scurtături pe ciclul Eulerian astfel obținut, vom obține lanțul Hamiltonian[5]. Considerăm ordinea în care nodurile au fost vizitate în ciclul Eulerian. Datorită faptului că un ciclu Eulerian reprezintă un drum în graf care trece prin toate muchiile, vor exista noduri vizitate de mai multe ori. Pentru formarea lanțului Hamiltonian, vom parcurge ciclul Eulerian și vom considera doar nodurile care au fost vizitate o singură dată. Dacă, de-a lungul parcurgerii ciclului Eulerian, se întâlnește un nod care a fost vizitat anterior, acesta nu va fi inclus în lanț fiindcă el există în lanț deja.

Pentru implementarea algoritmului în aplicație, nodul de start a fost considerat nodul cel mai apropiat de jucător. Algoritmul de căutare în adâncime a fost modificat pentru a trata și constrângerea de timp pe care lucrarea și-a propus să o trateze. În continuare va fi prezentat

modul în care metoda a fost adaptată pentru a ne asigura că toate nodurile sunt atinse înainte de a depăși termenul limită al obiectivelor respective.

Tratarea constrângerii timp

Complexitatea problemei PCV crește odată cu parametrizarea acesteia, adăugând termen limită obiectivelor. Problema devine astfel DTSP (Deadline Travelling Salesman Problem). Ea a fost abordată în lucrări precum [17], autorii estimând o complexitate de timp de $k!$ pentru un algoritm de aproximare cu $\frac{5}{2} * OPT$, unde k reprezintă numărul de termeni limită considerați, ceea ce seamănă cu punctul de plecare al problemei inițiale TSP (fără deadlines) care ar necesita O(|E|!) timp de rulare pentru generarea soluției optime.

Reamintim că fiecare obiectiv este caracterizat, printre altele, de un termen limită, și scopul aplicației este să genereze un traseu care nu depășește, atât cât este posibil, acest termen limită. Fie m timpul mediu petrecut la fiecare obiectiv în parte. Fie n numărul de obiective de pe hartă de interes pentru utilizator. Atunci, $m \times n$ reprezintă timpul maxim pe care utilizatorul îl petrece parcurgând toate obiectivele de pe hartă. Deci, obiectivele care reprezintă o problema sunt acelea cu timpul de expirare mai mic sau egal cu $m \times n$. Pentru fiecare astfel de obiectiv, vom calcula cât de mult putem să îl amânăm, adică determinăm numărul maxim de noduri pe care putem să le parcurgem până când obiectivul depășește termenul impus. Fie O obiectivul cu termen limită $d < m * n$ și fie x numărul maxim de obiective care pot fi atinse înainte ca O sa depășească timpul. Atunci, vom crea un vector de termeni limită unde se va reține pe poziția x obiectivul O. Dacă și obiectivul Q este candidat pentru poziția x în vectorul de termeni limită, atunci se caută o poziție liberă de la $x - 1$ până la 1 în vector. Dacă nu se găsește nici o poziție libera, înseamnă că un traseu între toate obiectivele nu este posibil.

Vectorul de termeni limită este folosit în algoritmul căutării în adâncime de generare a ciclului Eulerian. Fie k pasul curent la care s-a ajuns prin parcurgerea iterativă în adâncime. Verificăm dacă în vectorul de termeni limită, pe poziția k', avem un obiectiv candidat care urmează să expire. Deoarece în graful Eulerian, nodurile sunt traversate de mai multe ori, în k' vom reține numărul de noduri unice parcurse până la momentul k. Acest lucru este important

deoarece nodurile duplicate vor fi eliminate în momentul transformării ciclului Eulerian în lanț Hamiltonian. Dacă un nod urmează să expire, atunci modificăm graful pe care se efectua ciclul Eulerian astfel:

1) Fie x nodul curent și (x, y) o muchie nevizitată, adiacentă cu nodul x. Fie z nodul al cărui timp urmează să expire.
2) Ștergem muchia (x, y). Adăugam muchiile (x, z) și (y, z). Mergem din nodul x în nodul z.

Algoritmul va continua, apoi, determinarea ciclului Eulerian. Graful modificat păstrează condiția necesară și suficientă fiindcă eliminând muchia (x, y) și adăugând (x, z) respectiv (y, z), nodurile își păstrează paritatea. Un exemplu poate fi observat în figura 9, unde se exemplifică un salt efectuat la pasul 3, pornind de la nodul de start S, când nodul etichetat cu K urmează să expire iar parcurgerea ciclului nu a întâlnit nodul. Este șearsă muchia dintre nodul curent și nodul succesor, muchie ce este marcată punctat, și dunt adăugate cele două muchii marcate cu roșu. La pasul 4, determinarea ciclului Eulerian va continua de la nodul etichetat K.

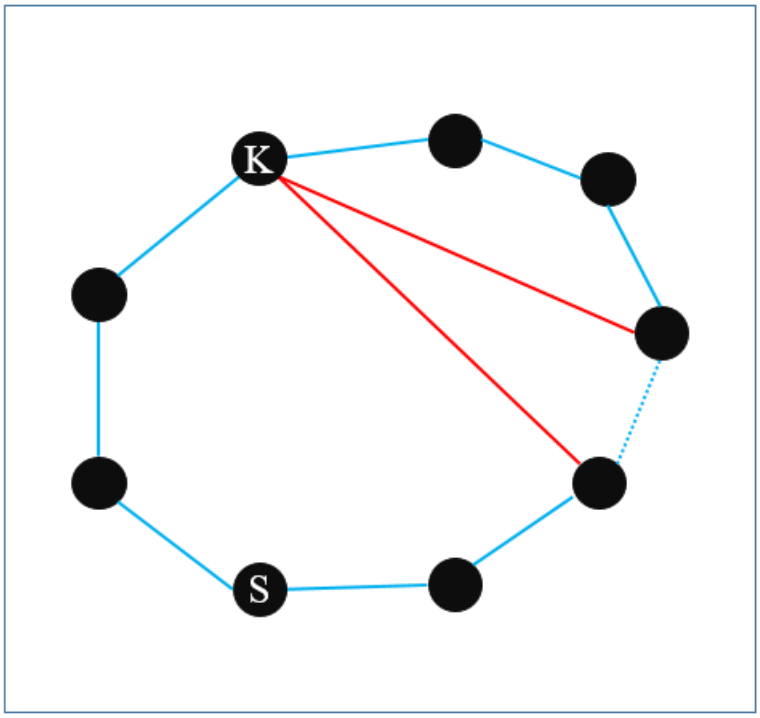


Figura 9: Schițarea pasului 3

Pentru crearea şi popularea vectorului de termeni limită sunt necesare $O(n^2)$ operaţii în cazul cel mai nefavorabil, când toate obiectivele au un termen mai scurt decât $m * n$. Complexităţii algoritmului de căutare în adâncime, care este $O(E + V)$ i se adauga şi numărul de salturi efectuate, deci în cazul cel mai nefavorabil numărul de muchii se dublează. Modificările aduse nu schimbă complexitatea iniţială a algoritmului.

Capitolul 2 – Tehnologii utilizate în implementare

Aplicația prezentată în această lucrare a fost creată pentru a putea fi folosită în cadrul jocului World of Warcraft, și prin urmare au fost folosite tehnici de implementare și tehnologii compatibile, care pot fi executate în interiorul jocului. Aplicația software reprezintă, astfel, un addon care aduce modificări interfeței grafice a jocului și execută o serie de script-uri care definesc modul în care modificările se desfășoară. Pentru a extrage datele de interes pentru crearea acestui addon, am folosit API-ul pus la dispoziție de către Blizzard WoW Client. Limbajul de programare în care această aplicație a fost creată este Lua, un limbaj de scripting. Pentru crearea imaginilor și elementelor care vor fi folosite în interfața grafică, a fost folosit Gimp.

Arhitectura unui addon

Un addon reprezintă o colecție de fișiere, grupate într-un director care va fi pus, spre executare, în directorul jocului World of Warcraft. Aceste fișiere sunt rulate de sistemul de scriptare al jocului și executate în aplicația principală, aducând noi funcționalități și modificări interfeței cu care interacționează jucătorul. Un addon este alcătuit din mai multe elemente: un fișier de tipul .toc, ce reprezintă tabelul de conținut care ajută la identificarea addon-ului și a fișierelor ce trebuie încărcate, fișiere media, cum ar fi imagini sau sunete folosite în aplicație, scripturi Lua care definesc modul în care addon-ul se comportă și fișiere XML pentru definirea elementelor vizuale [18].

Din punct de vedere al funcționalităților unui addon, există mai multe categorii; sunt addon-uri care afișează informații adiționale, cum e cazul și în aplicația descrisă în această lucrare, addon-uri care schimbă modul de afișare al interfeței grafice, pentru a facilita personalizarea jocului în funcție de preferințele utilizatorului, sau pot să ofere funcționalități noi prin adăugarea de noi butoane cu care jucătorul poate interacționa. Înainte de 2007, addon-urile erau capabile și să automatizeze jocul pentru utilizator, manipulând caracterul să se miște în joc

sau folosirea automată a abilităților. Însă Blizzard Entertainment consideră că un joc care se joacă singur nu reprezintă unul din modurile de joc în spiritul în care acesta a fost creat, astfel că acest gen de funcționalități nu mai sunt accesibile dezvoltatorilor de addon-uri. De asemenea, este interzis ca un addon să aducă modificări care sunt împotriva Termenilor de Utilizare World of Warcraft [19].

Addon-urile create de dezvoltator vor fi incluse în fișierul *./World of Warcaft/Interface/Addons* pentru funcționare. În momentul execuției, sistemul caută fișierul .toc, care trebuie să aibă acelasi nume ca și numele directorului care îl conține pentru a putea fi detectat. Acest fișier reprezintă tabelul de conținut al aplicației.

```
## Interface: 70200
## Title: PathFinder
## Notes: Acesta este addon-ul descris in lucrarea de diploma
## DefaultState: enabled
## Author: Iulian Giusca

libs\LibStub\LibStub.lua
libs\HereBeDragons-1.0\HereBeDragons-1.0.lua
libs\HereBeDragons-1.0\HereBeDragons-Pins-1.0.lua

init.lua
main2.lua
main2.xml
removeExtra.lua
constraintFunctions.lua
QuestDistance.lua
edges.lua
minCostTree.lua
parity.lua
minperfectmatch.lua
combineGraphs.lua
matriceAdiacenta.lua
eulerPath.lua
adaugaScurtaturi.lua
timeHandling.lua
drawmap.lua
```

Figura 10: Tabelul de continut

Un tabel de conținut începe prin definirea mai multor tag-uri. Există un număr mare de tag-uri ce pot fi folosite, şi rămâne la alegerea programatorului ce tag-uri va include. În continuare vor fi prezentate tag-urile care au fost folosite în aplicația prezentată.

Tag-ul Interface nu este un tag opțional. Acesta specifică pentru ce versiune a jocului a fost creat addon-ul. Dacă versiunea curentă nu corespunde cu valoarea tag-ului, atunci addon-ul este considerat de joc ca neactualizat, iar jocul nu îl va incărca datorita faptului că s-ar putea să fi apărut modificări în API care pot genera erori. Un addon neactualizat poate fi încărcat dacă se selectează opțiunea „Load Out-of-Date Addons" însă cea mai sigură metodă este ca dezvoltatorul addon-ului respectiv să se asigure că toate funcționalitățile au rămas nealterate, și eventual să rescrie programul relativ la noua versiune a jocului. Valoarea tag-ului se poate afla în mai multe moduri, cea mai rapida fiind extragerea valorii dintr-o funcție oferită de WoW API.

Figura 11: Exemplu de determinare a versiunii jocului

Tag-ul Title reprezintă numele addon-ului, și modul în care acesta va fi găsit în lista de addon-uri. Este recomandat ca numele să aibă cel puțin tangențial legătura cu facilitățile pe care programul le oferă. Am considerat că numele „Pathfinder" este potrivit pentru acest program care recomandă un drum de urmat pentru jucător printre misiunile existente.

Tag-ul Notes va conține o scurtă descriere a addon-ului sau orice altceva ce programatorul vrea să comunice utilizatorilor săi. Valoarea tag-ului poate fi vizualizată în meniul de selectare al addon-urilor.

Figura 12: Exemplificarea tag-ului notes

Tag-ul DefaultState determină dacă un addon este activat automat în momentul instalării sale. În mod contrar, addon-ul ar necesită activare manuală din meniul de addon-uri.

Tag-ul Author conține numele autorilor.

Pe lângă tag-uri, în tabelul de conținut sunt menționate și toate fișierele Lua si XML de care programul are nevoie. Ordinea în care fișierele sunt introduse în tabel este importantă deoarece ele sunt încărcate și executate în ordinea în care apar în tabel, și dacă se rulează un script care folosește o parte de cod ce nu a fost înca încărcată, atunci se va genera eroare.

Limbajul Lua

Lua (Luna, traducere din limba Portugheza [23]) este un limbaj de scripturi, apărut în anul 1993 la PUC-Rio, Universitatea Pontifical-Catolica din Rio de Janeiro, Brazilia. De atunci, Lua a evoluat și a ajuns vast răspândit în programare, și în particular în dezvoltarea de jocuri video, fiind unul din cele mai folosite limbaje în acest domeniu. Singurele structuri de date folosite de acest limbaj sunt tabelele, aceasta fiind și caracteristica principală a limbajului. Chiar dacă nu este o particularitate nouă relativ la celelalte limbaje, în Lua rolul tabelelor este central, oferind implementări simple și eficiente pentru vectori, clase, module, liste etc. [21]. De la început, limbajul a fost construit pentru a fi integrat în software scris în C/C++ și alte limbaje convenționale. Lua nu se ocupă cu programarea de tip low-end, performanța codului sau interfațarea cu alte sisteme deoarece se bazează pe limbajul C să facă toate acestea. În schimb, oferă structuri dinamice, fără dependențe, și ușurința în testarea codului, arii în care limbajul C nu excelează [20]. Pentru Lua, nu există noțiunea de „Main” a unui program deoarece, în esență, Lua este o librărie care furnizează funcții pe care mașina gazdă le execută [23].

Proprietățile acestui limbaj sunt extensibilitatea, simplicitatea, eficienta si portabilitatea. Extensibilitatea limbajului vine din faptul că Lua operează foarte mult cu librării, lucru facilitat de faptul că limbajul admite polimorfism provenit de la input dinamic, delegarea automată a resurselor care simplifică lucrul cu interfețele și nu este nevoie de a decide cum se produce alocarea si dezalocarea memoriei. De asemenea, funcțiile de ordin înalt si funcțiile anonime asigură un grad înalt de parametrizare, oferind versatilitate limbajului [20].

Simplitatea limbajului vine din faptul că folosește puține concepte; acest lucru contribuie la facilitatea folosirii limbajului și la dimensiunile scăzute de memorie pe care acesta le necesită.

Din punct de vedere al eficienței, surse [22] atestă faptul că Lua este unul din cele mai rapide limbaje de scritpare, ceea ce permite dezvoltatorilor să scrie o parte substanțială a aplicației în Lua.

Este și un limbaj portabil, putând fi rulat pe toate platformele existente fiindcă nu folosește compilare condițională pentru a putea rula codul pe diverse mașini, ci este implementat în ANSI C, astfel orice mașina care poate compila C, poate compila și cod scris în Lua.

Fiind un limbaj dinamic, variabilele nu au un tip de date, ci valori. În Lua nu există noțiunea de declarare a tipului. Următorul exemplu va exemplifica cum poate fi folosit Lua ca un limbaj de configurare simplu [23].

$A = 100$

$B = A * 10$

$C = „variabila”$

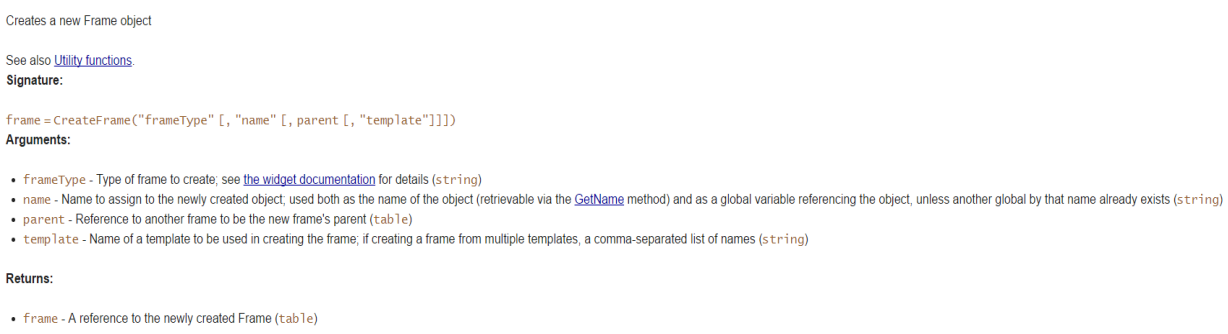
Particularități ale limbajului Lua au fost folosite și în scrierea funcțiilor aplicației prezentate. Programul a fost împărțit în mai multe fișiere, fiecare conținând un set de funcții specific care tratează o anumită problemă. Acest aspect ține mai mult de organizarea codului și de revenirea cu usurință asupra sa în cazul în care este nevoie. Fiecare fișier are un nume intuitiv, și conține un set specific de funcții. Chiar dacă în Lua nu există noțiunea de „main” al unui program, aplicația noastră deține un fișier cu acest nume în scop sugestiv, fișierul conținând un set de instrucțiuni prin se apelează într-o ordine specifică pentru obținerea output-ului dorit.

Pentru trasarea drumurilor pe hartă, ca indicator vizual pentu utilizator, am folosit librării puse la dispoziție de către alți dezvoltatori de addon-uri [24]. Librăria HereBeDragons reprezintă un set de funcții care folosesc API-ul WoW pentru interfața hărților din joc. Printre funcționalitățile oferite se numără calcularea distanțelor dintre obiective, determinarea coordonatelor de pe hartă a jucătorului sau a anumitor elemente din joc și marcarea elementelor vizuale pe hartă, lucru care va fi prezentat în capitolul 3 al acestei lucrări.

API-ul World of Warcraft

WoW API (Application Programming Interface) este reprezentat de o listă foarte lungă de funcții puse la dispoziție în mod gratuit de către compania care a produs jocul, Blizzard Entertainment, pentru a putea fi folosite în addon-uri și macro-uri de utilizatori pentru a modifica sau a interacționa cu interfața jocului. API-ul se modifică constant cu fiecare versiune nouă a jocului și din această cauză este necesară și revizuirea addon-urilor pentru a asigura buna funcționalitate a aplicațiilor. API-ul integral este disponibil în referința [25]. Fiecare funcție este prezentată printr-o scurtă descriere unde se explică ce face acea funcție, signatura funcției și uneori sunt oferite chiar și exemple de cod.

Signatura funcției este reprezentată de numele funcției, argumentele necesare execuției și eventual rezultatul obținut prin rularea acesteia. Fiecare funcție este descrisă în [25] într-o anumită măsură. Funcțiile sunt împărțite pe categorii, pentru a facilita căutarea uneia de care dezvoltatorul are nevoie. În figura 13 este prezentat un exemplu de documentare al unei funcții din API. Pe primul rând apare o scurtă descriereal funcției ce se apelează. Urmează apoi signatura funcției, cu numele acesteia, argumentele necesare pentru apelare și rezultatul returnat. Argumentele opționale se notează în paranteze. În cazul în care lipsesc, câmpurile pe care le determină vor fi considerate „nil”. Mai jos sunt descrise tipul de argumente și rezultatul obținut. O functie poate returna mai multe valori sau tabele, caz în care variabilele din stânga egalului se vor scrie cu virgulă între ele.



Creates a new Frame object

See also Utility functions.

Signature:

```
frame = CreateFrame("frameType" [, "name" [, parent [, "template"]]])
```

Arguments:

- `frameType` - Type of frame to create; see the widget documentation for details (`string`)
- `name` - Name to assign to the newly created object; used both as the name of the object (retrievable via the GetName method) and as a global variable referencing the object, unless another global by that name already exists (`string`)
- `parent` - Reference to another frame to be the new frame's parent (`table`)
- `template` - Name of a template to be used in creating the frame; if creating a frame from multiple templates, a comma-separated list of names (`string`)

Returns:

- `frame` - A reference to the newly created Frame (`table`)

Figura 13: Exemplu de documentare al unei funcții din API

Funcțiile din API folosite în aplicație vor fi descrise în continuare. Am avut nevoie de apelarea funcțiilor pentru a extrage informații despre misiuni, despre hărțile din joc, despre poziția jucătorului în raport cu harta de joc și pentru crearea interfeței grafice pe care jucătorul o utilizează pentru a avea acces la funcționalitățile oferite.

1) C_TaskQuest.GetQuestsForPlayerByMapID(QUEST_MAPS[2],QUEST_MAPS[1])

Această funcție returnează un tabel care conține toate misiunile de pe harta primită la primul argument. Hărțile sunt reprezentate de un ID. Al doilea argument reprezintă ID-ul celei de-a doua hărți, care este mai generală. Prima hartă reprezintă o sub-zonă a unei zone mai largi, reprezentată de a doua hartă. Pentru a extrage toate misiunile existente de pe harta generală, care este un lucru de interes pentru aplicație, a trebuit să apelăm această funcție pentru fiecare sub-zonă existentă, așa cum se poate vedea în figura 14. Această funcție întoarce un tabel de unde vom extrage informații utile: ID-ul misiunii și coordonatele x, y de pe harta generala ce reprezintă locația unde se află misiunea.

Fiecare obiect existent în joc este reprezentat de un ID, care se poate afla de pe resurse web dedicate deținerii acestui tip de informații [26].

```
WORLD_QUESTS_BY_ZONES={
   C_TaskQuest.GetQuestsForPlayerByMapID(QUEST_MAPS[2],QUEST_MAPS[1]), -- quests
   din Dalaran
   C_TaskQuest.GetQuestsForPlayerByMapID(QUEST_MAPS[3],QUEST_MAPS[1]), -- quests
   din Azsuna
   C_TaskQuest.GetQuestsForPlayerByMapID(QUEST_MAPS[4],QUEST_MAPS[1]), -- quests
   din Stormheim
   C_TaskQuest.GetQuestsForPlayerByMapID(QUEST_MAPS[5],QUEST_MAPS[1]), -- quests
   din Val'Sharah
   C_TaskQuest.GetQuestsForPlayerByMapID(QUEST_MAPS[6],QUEST_MAPS[1]), -- quests
   din Broken Shore
   C_TaskQuest.GetQuestsForPlayerByMapID(QUEST_MAPS[7],QUEST_MAPS[1]), -- quests
   din Highmountain
   C_TaskQuest.GetQuestsForPlayerByMapID(QUEST_MAPS[8],QUEST_MAPS[1]), -- quests
   din Suramar
   C_TaskQuest.GetQuestsForPlayerByMapID(QUEST_MAPS[9],QUEST_MAPS[1]), -- quests
   din Eye of Azshara
}
```

Figura 14: Modul de obținere al informatiilor principale despre misiuni

2) nume_quest,faction_id = C_TaskQuest.GetQuestInfoByQuestID(id);

Pentru fiecare misiune în parte extrasă din tabelul obținut apelând funcția descrisă la punctul 1), începem să colectăm informații în legătură cu acestea. Spre exemplu, această funcție va

furniza numele misiunii cu id-ul dat ca parametru și id-ul facțiunii pentru care vom primi puncte de reputație în urma îndeplinirii obiectivelor.

3) *duration = C_TaskQuest.GetQuestTimeLeftMinutes(id);*

Această funcție furnizează timpul în minute până la expirarea misiunii cu id-ul dat ca parametru.

4) *distance = C_TaskQuest.GetDistanceSqToQuest(id);*

Funcția întoarce distanța de la poziția curentă a jucătorului și până la misiunea ce are id-ul dat ca parametru. Este folosit același principiu care a fost descris în capitolul 1.

5) *tagID, tagName, worldQuestType, rarity, isElite, tradeskillLineIndex = GetQuestTagInfo(id)*

Funcția întoarce diferite valori care vor ajuta la filtrarea misiunilor, asupra căreia se va reveni în capitolul 3.

6) *posX, posY = GetPlayerMapPosition("player");*

Funcția întoarce poziția jucătorului sub formă de coordonate x și y pe harta generală.

7) *frame = CreateFrame("Frame", "MyFrame", mainPanel)*

Interfața utilizatorului în joc are la bază frame-urile. Toate elementele vizuale care apar pe ecran, butoane, text sau imagini trebuie să apară în interiorul unui frame. Un frame poate răspunde la evenimente ce se petrec în interiorul jocului, declanșând un anumit rezultat vizual în momentul în care se întâmplă ceva. Utilizatorul poate interacționa cu aceste frame-uri, prin evenimente generate de apăsarea cu mouse-ul în regiunea designată sau evenimente generate de apăsarea unui buton. Astfel, pentru ca un addon să poată oferi interfața grafică, trebuie să existe frame-uri. Funcția descrisă la punctul 6) este funcția de creare a unui frame. Primul argument desemnează tipul care se dorește a fi creat. „Frame” este doar un tip abstract de element vizual, fiindcă există o multitudine de cazuri particulare care derivează din acest punct, cum ar fi butoane, tooltip, scrolling frame, etc. Lista completă se poate verifica în referința [27]. Al doilea argument reprezintă numele widget-ului. Acesta este un argument opțional. Numele este modul după care un frame este identificat, spre exemplu în momentul în care vrem să specificăm părintele unui alt frame, sau dorim să aducem unui frame modificări dintr-o altă parte a codului. Identificarea nu ar putea avea loc dacă argumentul numelui este *null*. Al treilea argument specifică frame-ul părinte. Un frame poate avea un singur părinte și oricât de mulți copii. Există

un frame special, cu numele de *UIParent,* care este părinte pentru toate frame-urile existente [18]. În cazul în care al treilea argument este *null*, atunci frame-ul creat va avea ca părinte UIParent. Acest frame este util deoarece, având un singur frame comun, și interfața grafică va scala corespunzător pe orice tip de ecran. Odată creat frame-ul, acesta poate fi customizat cu funcții specifice, așa cum se poate observa în figura 15.

```
local frame = CreateFrame("Frame", "MyFrame", mainPanel)
frame:SetSize(1024, 512)
frame:SetPoint("RIGHT",-25,0)
frame:SetClampedToScreen(true)
frame:SetHitRectInsets(0,0,0,0)
local texture1 = frame:CreateTexture()
texture1:SetAllPoints()
texture1:SetColorTexture(0,0,0,0.1)
frame.background = texture1
frame:SetScale(0.75)
```

Figura 15: Crearea unui frame si setarea caracteristicilor

8) frame.Hide()

Această funcție face ca frame-ul cu numele „frame” să nu mai fie vizibil. Dacă un frame este ascuns, atunci toți copiii săi vor fi ascunși, de asemenea.

9) myCheckButton1 = CreateFrame("CheckButton", "Check_rep_1", mainPanel, "ChatConfigCheckButtonTemplate");

Funcția creează un buton de tipul CheckButton, cu numele Check_rep_1, părintele mainPanel și care moștenește template-ul oferit de Blizzard. În momentul bifării sau debifării unui astfel de buton, se dorește apelarea unei funcții care aduce o modificare stării curente a jocului. Un exemplu de definire al unei astfel de funcții se poate observa în figura 16. A fost generat un eveniment de tipul „OnClick” și de fiecare dată când butonul este apăsat, utilizatorul va primi un mesaj cu starea butonului. Funcția definită este anonimă, putând fi apelată doar când acest eveniment are loc. Nu intotdeauna, însa, rezultatul trebuie să se producă imediat, o situație fiind aceea când utilizatorul bifează mai multe opțiuni, urmând ca mai apoi să se producă o schimbare.

10) enabled = myCheckButton1:GetChecked();

Funcția GetChecked() verifică starea unui Check Button si retunează o valoare booleană în funcție de starea butonului – true dacă a fost bifat si false dacă nu.

```
myCheckButton1 = CreateFrame("CheckButton", "Check_rep_1", mainPanel, "ChatConfigCheckButtonTemplate");
myCheckButton1:SetPoint("TOPLEFT", 0, 0);
getglobal(myCheckButton1:GetName() .. 'Text'):SetText("Armies of Legionfall");
myCheckButton1.tooltip = "This is where you place MouseOver Text.";
myCheckButton1:SetScript("OnClick",
    function()
        enabled = myCheckButton1:GetChecked();
        if enabled then
            print("Enabled!")
        else
            print("Disabled!")
        end;
    end)
```

Figura 16: Crearea unul CheckBox button cu declararea unei functii anonime

Limbajul XML

XML sau Extensible Markup Language, este folosit pentru a formata și transfera date într-un mod ușor și congruent. Acest limbaj face parte din un subset SGML (Standard Generalized Markup Language) care este simplificat. Într-un mod similar HTML, XML folosește markup tags cu stringuri de text care transmit aplicației informația necesară. XML diferă de HTML prin faptul că cel din urmă descrie cum trebuie afișate datele din o pagină web pe când în XML se descriu însăși datele. Astfel, XML este un limbaj de descriere a datelor ce ne permite organizarea lor în structuri de date. O particularitate foarte avantajoasă pentru acest limbaj constă în faptul că permite crearea propriilor taguri. Acest fapt conferă limbajului o libertate mai mare comparativ cu HTML, însă interpretarea codului realizat depinde în mod strict de claritatea cu care a fost scris de către autor. Pentru a începe un document nou este necesară folosirea <DOCUMENT> </DOCUMENT>.

Realizarea tagurilor proprii poate fi ajutată prin definirea tipurilor de taguri folosite, ordinea în care acestea pot fi folosite precum și ce tipuri de taguri pot conține. Acest lucru poate fi realizat în Document Type Declaration (DTD). Un DTD este opțional însă poate facilita și ușura interpretarea tagurilor noi. Interpretarea documentelor de tip XML poate fi ușurată prin folosirea unui parser, care are rolul de a realiza diviziuni până la structurile logice fundamentale[32]. Majoritatea programelor de tip XML parser sunt scrise în Java. Un document XML este considerat bine formatat dacă acesta conține unul sau mai multe elemente, dacă există un singur element (de regulă <DOCUMENT>) pentru care începutul sau sfârșitul nu se află în interiorul altui element și dacă toate elementele sunt corect definite în interiorul buclelor de care aparțin. O entitate este un termen care reprezintă un anumit tip de date; acestea trebuie să fie predefinite în XML sau DTD. Tipurile de date predefinite în XML sunt: *amp, lt, gt, apos, quot;*

acestea reprezintă*: &, <, >, „ ”*. Pentru analiza unui document cu un program de tip parser nu este necesară existența unui DTD, însă dacă există acesta va fi folosit. Tipurile de caractere folosite în XML sunt descrise de standardul Unicode și de "ISO/IEC 10646" (toate caracterele din ISO 10646 sunt accesibile și în Unicode cu aceleași valori de tip 16-bit). Există și anumite caractere cărora le este dat un nume în XML și anume spațiile albe, la care se face referință cu simbolul *S*. Structura formală a unui document XML și care specifică cum este stocată și organizată informația este markup tagul. Tot textul care nu este marcat este considerat de tip *character data* de către document. Acestea pot fi:

- Comentarii
- Referințe de entitate
- Referințe de caracter
- Instrucțiuni de procesare
- Segmente de tip CDATA
- Elemente de start/end
- Elemente goale
- Declarații de tip DTD

GNU Image Manipulation

GIMP este o aplicație de modificare și generare a imaginilor ce oferă o sumedenie de opțiuni precum: retușarea pozelor, compoziția de imagini și construcția de imagini. Aplicația este distribuită gratuit sub o licență de uz public general [28]. Aplicația a fost dezvoltată de autori ca fiind foarte adaptabilă, permițând folosirea unor plug-ins și extensii. Interfața avansată de scripting permite automatizarea chiar și a celor mai complexe proceduri de manipulare de imagine. Capabilitățile programului includ:

- suită de unelte de modificare precum (pensule, airbrush, clonare)
- Memorie limitată doar de spațiul disponibil de HDD
- Tehnică de anti-aliasing de calitate înaltă
- Posibilitate de lucru cu alpha channels (canale de transparență)

- Posibilitate de stratificare și lucru cu straturi suprapuse (tehnici de compoziție)
- Capabilități de scripting avansat
- Număr mare de *Undo* și *Redo* (limitat doar de spațiul disponibil pe HDD)
- Bază de date procedurală pentru funcțiile GIMP interne
- Suport pentru o gamă largă de tipuri de imagine (GIF, JPEG, PNG, XPM, TIFF, TGA, MPEG, PS, PDF, PCX, BMP ș.a.)
- Unelte de selecție variate

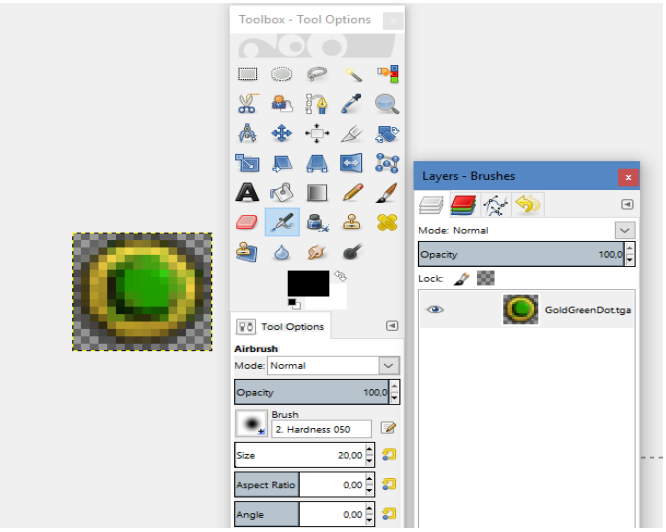


Figura 17: Modificarea unei imagini în GIMP, diferitele instrumente de modificare și selecție fiind prezente în toolbar

Imaginile sunt entitățile fundamentale folosite de GIMP. O imagine este un singur fișier (exemplu: JPEG). În GIMP o imagine există pe un strat numit *layer*, acesta permite suprapunerea mai multor astfel de straturi pentru a conferi imaginii anumite caracteristici dorite. Aplicația permite și manipularea straturilor prin măști de selecție și prin existența mai multor canale de culoare. Un canal este o componentă unitară a culorii unui pixel. Pentru a genera un pixel colorat este necesară existența a mai multor canale de tipul: Red, Green, Blue și Alpha (transparență). În cazul imaginilor de tip Grayscale există canalele: Gray și Alpha. Prin manipularea valorilor acestor canale este posibilă realizarea de imagini cu gama de culori dorite și cu transparența necesară, acest lucru fiind de o importanță mare pentru interfața grafică a aplicațiilor de tip addon din World of Warcraft datorită necesității unui stil singular din punct de vedere estetic (care se potrivește sau complementează restul elementelor grafice din interfața utilizatorului).

Pentru modificarea selectivă a imaginii există un mecanism de selecție cu forme geometrice comune sau de tip free-form, acesta permițând modificarea anumitor părți de interes pentru a genera imaginea cu parametrii doriți. Pentru realizarea unor acțiuni mai complexe este posibilă generarea unor scripturi în diferite limbaje de programare (C, Python, Perl). Pentru modificarea imaginilor există o bară de unelte unde se pot găsi cele mai uzuale funcții. Prima categorie de funcții importante ce se află în toolbar sunt cele de tip tool. Acestea au o varietate de utilizări (selecția unor părți ale imaginii, modificarea dimensiunii, etc). A doua categorie importantă se referă la funcțiile de culoare ce sunt prezente în un număr important de operații (modificarea culorilor intră la această categorie) precum și tranziția sau realizarea unor canale de tip transparență sau grayscale. Ultima categorie importantă pentru cele mai uzuale operații realizate în GIMP este funcția de tip Brush; aceasta permite pictarea imaginilor precum și apelarea la anumite tehnici precum smudging pentru a obține imaginea dorită. Pentru realizarea elementelor grafice necesare aplicației din World of Warcraft a fost necesară folosirea unui anumit tip de format de imagine recunoscut de către joc. Una din operațiile de bază ce pot fi realizate de către GIMP este modificarea extensiei imaginii sau generarea de imagini sub anumite extensii. Extensia recunoscută de World of Warcraft este de tip TGA. Pentru optimizarea cantității de memorie pe care un addon o folosește este posibilă comprimarea imaginilor pe care acesta le incorporează. Comprimarea unor imagini presupune pierderea calității originale, lucru care va scădea automat și memoria pe care imaginea o ocupă. Pierderea de calitate înseamnă adesea un număr scăzut de pixeli precum și alte valori pentru culoare. Un exemplu concret în acest caz este oferit chiar de manualul de utilizare al GIMP, consumul de memorie pentru o imagine la un nivel de calitate de 70 este de 15.2 KB și în cazul unui nivel de calitate de 100 fiind de 72.6 KB respectiv. Se poate concluziona că variația calitate-memorie nu este una liniară și deci este posibilă reducerea calității la un nivel insesizabil pentru utilizator dar cu un aport semnificativ pentru memoria consumată. Găsirea unui nivel optim al calității se face empiric. *[29]*

Capitolul 3 – Utilizarea și interfața aplicației

Fundamentele algoritmice prezentate în capitolul 1 şi tehnologiile prezentate în capitolul 2 au fost îmbinate pentru a crea un produs software cu rolul de a fi apelat în cadrul jocului World of Warcraft pentru a obține un traseu cât mai optim printre obiectivele cunoscute în joc ca şi "World Quests", o varietate de misiuni ce apar la interval neregulat şi care oferă diferite recompense în momentul completării acestora. Drept urmare, această aplicație este destinată jucătorilor de World of Warcraft care doresc să înlăture rutina planificării rutelor şi să îndeplinească toate obiectivele care sunt de interes pentru aceștia. Drept urmare, aplicația şi-a propus să fie simplu şi intuitiv de utilizat, necesitând un input minimal de la utilizator, oferind în același timp toate informațiile de care jucătorul are nevoie într-un mod atractiv şi sugestiv.

Instalarea aplicației

Pentru ca un utilizator să poată folosi aplicația în cadrul jocului, acesta are nevoie să dețină directorul cu fișierele aplicației. Acesta trebuie adăugat în fișierele jocului, mai exact la calea \World of Warcraft\Inerface\Addons. Este important ca toate fișierele menționate în tabelul de conținut să se găsească în acest director, altfel aplicația nu va rula corespunzător.

După ce directorul a fost mutat, jucătorul trebuie să se conecteze cu un cont de utilizator activ şi existent la serverul de joc. După această etapă, este necesar ca jucătorul să se asigure că aplicația este pornită, apăsând pe butonul AddOns din colțul ecranului. Se va deschide o fereastră cu toate addon-urile instalate. Este necesar ca checkbox-ul cu numele PathFinder să fie verificat pentru a putea folosi noile funcționalități pe care programul le oferă.

Figura 18. Interfața pentru pornirea unui Addon din joc

Funcționalitățile aplicației

Rolul unui addon este acela de a oferi jucătorului noi modalități de folosire a interfeței jocului. În continuare vor fi prezentate schimbările pe care programul software instalat le aduce programului principal.

Comenzile slash

O primă modificare e reprezentată de introducerea de noi comenzi de tipul slash. O comandă slash are forma "/nume_comanda", motiv pentru care poartă acest nume, iar funcţionalităţile pe care aceste comenzi le aduc acoperă un spectru larg, de la interacţiunea între jucători până la apelul funcţiilor din API, care in general ajută la crearea comenzilor de tip macro.

```
840  SLASH_PATHFINDER1 = "/pathf"
841  SLASH_PATHFINDER2 = "/pathfinder"
842  SlashCmdList["PATHFINDER"] = function(self, txt)
843      if mainPanel:IsVisible() then
844          mainPanel:Hide();
845      else
846          mainPanel:Show();
847      end
848  end
849
```

Figura 19. Comenzile slash din add-on

World of Warcraft stochează handlerul unei comenzi de tipul slash într-un tabel numit SlashCmdList. Tabelul este indexat cu un șir string arbitrar, unic, desemnat de către autorul addon-ului. Indecșii se folosesc, la rândul lor, pentru a crea variabile de tipul SLASH_INDEXn care vor conține comenzile de tip slash [18]. În cazul nostru, ne-am creat două astfel de variabile globale, SLASH_PATFINDER1 care înregistrează comanda /pathf şi SLASH_PATHFINDER2 care înregistrează comanda /pathfinder. În momentul deschiderii ferestrei de chat şi introducând una din cele două comenzi, se va apela funcția anonimă stocată în tabelul SlashCmdList pe intrarea PATHFINDER, care va deschide sau închide mainPanel, în funcție de starea sa actuală; mainPanel este frame-ul principal al aplicației.

Sistemul de comenzi slash stochează comenzile folosite de către utilizator într-un cache, ceea ce reduce impactul macro-urilor asupra performanței jocului. Dacă se suprascrie o intrare din tabelul SlashCmdList fără a efectua refresh asupra interfeței utilizatorului, care are rolul de a goli cache-ul, atunci cache-ul va trebui actualizat. Spre exemplu, dacă este folosită o comandă slash creată într-un fișier lua, şi dacă se produce o modificare în cod iar mai apoi este apelată comanda, atunci jocul va încerca să folosească funcția originală[18].

Interfața principală a programului

Figura 20. Interfața grafică a aplicației

Interfața creată a fost gândită pentru a necesita un minim de informație de la utilizator, dar în același timp oferind toate informațiile de care un utilizator ar avea nevoie. Mai concret, utilizatorul trebuie să ofere ca input misiunile de interes pentru acesta, ce sunt de mai multe categorii. Ca rezultat, harta jocului va fi marcată sugestiv, indicând un traseu ponderat minimal între misiunile considerate, dar în același timp aplicația oferind jucătorului şi o listă interactivă

cu toate misiunile considerate, cu informații suplimentare pentru fiecare in parte afișate într-un tooltip în momentul unui event de tipul mouse-over pe numele misiunii.

În momentul apelării comenzii slash /pathf sau /pathfinder, se va deschide frame-ul cu numele mainPanel. Acesta este un frame de tip părinte care conține toate celelalte frame-uri. La rândul său, frame-ul mainPanel îl are ca părinte pe UIParent, care este un frame de tip default de care sunt ancorate toate elementele vizuale ce aparțin interfeței și apar unui jucător pe ecran. Dacă la crearea unui frame nu se specifică părintele, atunci acesta va deveni un frame de tip child al lui UIParent.

Există mai multe straturi grafice (graphical layers [19]) care grupează texturile şi font-string-urile ce se afișează. Primul strat este stratul BACKGROUND, care reprezintă stratul cel mai de jos. De regulă, este folosit pentru setarea unei imagini de fundal sau un design pentru frame, şi fiind plasat pe ultimul strat, se va asigura faptul că imaginile nu se vor suprapune peste elementele vizuale de pe straturile de deasupra. Următorul este stratul BORDER, scopul lui fiind adăugarea de borduri sau alte elemente artistice menite să finiseze background-ul aplicaţiei. Al treilea strat este ARTWORK, care, la rândul său, în mod tipic, conţine elemente non-funcţionale decorative care trebuie sa apară deasupra stratului BORDER. Stratul OVERLAY reprezintă stratul cel mai de sus. În acest strat vor fi declarate elementele cu care utilizatorul interacţionează. Există un al cincilea strat, HIGHLIGHT, care este afişat doar în urma evenimentelor de genul mouse-over pe elementele din frame-ul părinte. Este necesar ca părintele să aibă activată captarea evenimentelor generate de mouse. Ca regulă generală, elemente aparţinând aceluiaşi strat sunt generate în mod aleator. Textul va fi întotdeauna generat deasupra unui element grafic, eliminând necesitatea poziţionării acestuia pe un strat superior [18].

Frame-ul mainPanel conţine două astfel de straturi grafice, care împreună formează imaginea de fundal ce se poate observa în figura 20. Al doilea strat conţine 8 imagini de dimensiuni 256x256, care sunt situate pe două linii a câte patru imagini fiecare. O astfel de poziţionare reîntregește imaginea originală. În partea dreaptă a imaginii se poate observa cum aceasta capătă transparență, şi din acest motiv am folosit un strat grafic de tip BACKGROUND care conține o imagine care acoperă toată lungimea frame-ului mainPanel. Din momentul în care imaginea începe să fie transparentă, imaginea de dedesubt va deveni vizibilă, obținând astfel efectul produs. Stratul grafic BACKGROUND are, în plus, rolul de a asigura faptul că nicio porțiune din frame-ul principal nu rămâne transparentă. Dacă nu ar exista, atunci design-ul

aplicației ar fi unul care produce confuzie datorită faptului că jucătorul nu ar putea ști cât de mult se întinde frame-ul pe ecran, iar elementele din partea dreaptă ar părea ancorate direct pe frame-ul UIParent.

Dimensiunea frame-ului principal a fost considerată în așa fel încât să respecte dimensiunile imaginii în stratul grafic BORDER, astfel că i s-a setat dimensiunea 1024 x 512. Cele 8 imagini sunt texturi în cadrul frame-ului principal, ancorate în așa fel încât toate imaginile sunt puse cap la cap pentru a obține imaginea întreagă. În XML, pentru ancorare sunt necesare trei atribute: punctul de ancorare, frame-ul de care elementul se ancorează și punctul din frame-ul relativ față de care un element este ancorat [19]. Sunt nouă puncte care pot fi folosite pentru ancorare: cele patru colțuri(topleft, topright, bottomleft, bottomright), cele patru jumătăți ale laturilor dreptunghiului care alcătuiesc un frame sau o textură (top, bottom, left, right), sau punctul de intersecție al diagonalelor dreptunghiului (center). Punctul va fi plasat în punctul relativePoint, care poate fi tot unul din cele nouă puncte de pe frame-ul părinte specificat în relateiveTo. În plus, o ancoră poate conține și tag-ul Offset, care conține două atribuituri, x și y. Absența tag-ului este echivalentă cu existența acestuia, cu x și y având valorile 0. Valorile asociate atributelor x și y indică numărul de pixeli cu care o imagine se deplasează față de poziția originală. Spre exemplu, pentru x = 30, imaginea va fi deplasată la dreapta cu 30 de pixeli. Pot fi atribuite și valori negative, x = -30 indicând că imaginea va fi deplasată spre stânga.

Imaginile care au fost folosite în realizarea acestui addon fac parte dintr-o colecție prezentă printre fișierele jocului. Am considerat importantă folosirea de elemente grafice existente deoarece utilizatorul e deja familiar cu acestea, aplicația oferind senzația unei extensii naturale a jocului. Colecția este inaccesibilă utilizatorilor din motivul că toate elementele grafice încărcate în momentul execuției jocului sunt prezente în acest fișier. Modificarea acestor fișiere ar produce modificări ale elementelor grafice din interiorul jocului, iar acest nivel de acces este restricționat unui utilizator obișnuit. Se poate, însă, obține o copie a acestei colecții, care are rolul de a permite vizualizarea fișierelor existente care alcătuiesc interfața completă a jocului. Modificări aduse acestei copii nu influențează în niciun fel fișierul original. Pentru obținere, este necesar ca utilizatorul să activeze accesul la consola de comandă din interiorul jocului. Comanda exportInterfaceFiles art va exporta colecția de imagini existentă în fișiere de tip MPQ, care se găsesc în fișierele de instalare [31]. Noul folder creat va putea fi găsit în folderul principal al jocului, cu numele de BlizzardInterfaceCache. Acesta reprezintă o imagine în oglindă a fișierului

Interface, inaccesibil utilizatorilor pentru modificare. Însă, se poate folosi calea directă, inaccesbilă, în momentul scrierii addon-urilor, acest folder oferind doar o modalitate de a vizualiza structura folderului original și fișierele conținute de acesta.

Frame-ul principal are atașat şi un eveniment de tipul OnLoad, care va apela funcţia loadFrame, definită în fişierul lua de tip main. Acest tip de eveniment este aplicabil doar când evenimenul este definit in XML, fiindcă evenimentul e receptat în momentul când frame-ul este creat. Dacă evenimentul este creat cu o funcţie CreateFrame(), atunci un astfel de eveniment va fi receptat înainte ca funcţia să returneze frame-ul, astfel că handler-ul nu mai poate fi accesat ulterior [30].

LoadFrame este o funcţie care, atunci când se apelează, generează restul de elemente ale interfeţei cu care jucătorul interacţionează. Este creat un nou frame, de dimensiuni mai mici decât frame-ul principal, ce va conţine lista tuturor misiunilor după momentul selecţiei. În loc de imagine de fundal, texturii i-a fost atribuit un element de tip RGBA, care oferă culoare, respectiv transparență elementului. Am setat transparență acestui frame pentru a nu acoperi mare parte din design cu un frame de culoare mată. Pentru un număr mare de misiuni, însă, dimensiunile frame-ului nu sunt suficiente pentru a cuprinde toată informaţia, motiv pentru care i-a fost atribut ca element child un frame de tipul ScrollFrame, care la rândul său are ca child pe scrollbar. Valorile setate frame-ului scrollbar prin apelul funcţiei SetMinMaxValues(min, max) reprezintă cât de mult se poate scrola afișând conţinutul corespunzător. Pasul de scrolare este setat apelând funcţia SetValueStep(1) frame-ului de tip slider, funcţie aparţinând API-ului World of Warcraft. În momentul în care utilizatorul scrolează, această valoare se modifică, executând handler-ul “OnValueChanged”, care va apela o funcţie anonimă ce are rolul de a seta conţinutul corespunzător spre vizualizare.

În urma apelului funcţiei sunt generate şi o serie de CheckButtons. Acestea sunt frame-uri speciale care pot avea două stări: activ (enabled), stare reprezentată vizual de un indicator de tip „tick”, şi inactiv (disabled), unde un astfel de indicator lipseşte. CheckButton-urilor le-au fost atribuite şi text, care indică tipul de misiune selectat în urma activării unui astfel de buton. În momentul selecţiei, se va verifica starea fiecărui CheckBox în parte şi se vor adăuga misiunilor selectate doar acelea ce au fost asociate CheckBox-urilor cu starea activă.

Butoanele cu numele asociat Select All şi Deselect All au fost adăugate interfeţei cu scopul de a trivializa selecţia. Prin apăsarea primului buton, jucătorul poate seta toate CheckBox-

urile pe modul enabled, nefiind necesar să fie activate fiecare în mod individual. La fel se procedează şi cu al doilea buton, cu diferenţa că toate CheckBox-urile vor fi setate pe modul inactiv.

Butonul cu textul Generate Path este un buton care, în urma unui eveniment de tipul OnClick, va apela funcţia filterQuests, care reprezintă motorul acestei aplicaţii. În cadrul acestei funcţii se vor obţine toate rezultatele descrise în capitolul 1 al acestei lucrări. Înainte de a detalia conţinutul acestei funcţii, se vor prezenta obiectivele care alcătuiesc graful complet care va duce la generarea drumului de cost minim între acestea.

Obiectivele

Obiectele principale prelucrate de acest program software sunt misiunile cunoscute în joc după numele de World Quests. Acestea sunt misiuni diferite faţă de misiunile normale ale jocului, care se pot completa doar o singură dată şi au, în general, scop narativ. World quests sunt evenimente dinamice, repetitive, care apar distribuite pe toată suprafaţa hărţii de joc.

Aşa cum se poate observa in figura 21 ce reprezintă un tipar pentru toate misiunile, un world quest are mai multe caracteristici: numele misiunii, urmat de facţiunea pentru care se primesc puncte de reputaţie în urma îndeplinirii obiectivelor. Facţiunile sunt, din punct de vedere al narativei jocului, un fel de organizaţii pe care jucătorul le ajută prin soluţionarea problemelor cu care acestea se confruntă. Pe al treilea rând este specificat timpul rămas până la expirarea misiunii. Obiectivele misiunii sunt prezentate pe rândul 4, şi desemnează acţiunile sau evenimentele pe care un jucător trebuie să le execute în joc pentru a completa misiunea cu succes. Dacă obiectivele nu au fost atinse până la expirarea misiunii, atunci aceasta nu se va mai găsi pe harta de joc, iar jucătorul nu va mai putea obţine recompensa pe care în mod normal ar primi-o în urma îndeplinirii obiectivelor. Recompensa apare descrisă pe rândul 6, unde se prezintă obiectul oferit. Şi acesta poate fi de mai multe feluri, fiind caracterizat de un nume, tip de recompensă, constrângeri adiţionale şi în final, unde e cazul, o descriere a obiectului. Jucătorul poate intra în posesia unei recompense doar dacă îndeplineşte la timp obiectivele unui world quest.

Figura 21. Afișarea obiectivelor, recompenselor, timpul rămas precum și a facțiunii unui world quest din joc

Acestea sunt datele pe care le oferă interfaţa originală a jocului. În lipsa unui program care să grupeze aceste world quests, jucătorul ar trebui să verifice fiecare eveniment în parte şi să facă o decizie aproximativă a modului în care va parcurge harta pentru a participa la ele. Acest lucru poate dura destul de mult timp şi, pe parcurs, devine o rutină pentru un jucător care e nevoit odată la câteva ore să verifice harta integral în căutarea misiunilor care oferă recompense sau alte beneficii de care acesta e interesat. Datele oferite de interfaţa jocului sunt şi insuficiente pentru rezultatul pe care programul software caută să îl obţină. Se pot obţine informaţii adiţionale despre world quests apelând API-ul pus la dispoziţie.

În momentul încărcării fişierelor lua, se iniţializează un tabel numit, în mod sugestiv, QUESTS, care va conţine toate informaţiile necesare în restul aplicaţiei. Fiecare linie din tabel va conţine un set de informaţii despre fiecare misiune în parte.

Cu ajutorul funcţiei din API descrisă în subsecţiunea 2.3.1, se obţin obiecte de tip tabel, de unde extragem id-ul unei misiuni, care este unic, şi coordonatele x, y care indică poziţia misiunii pe harta de joc. Cunoscând id-ul, se extrag restul informaţiilor: numele misiunii, id-ul facţiunii reprezentante, distanţa de la poziţia curentă a jucătorului până la misiunea respectivă, raritatea şi tipul de recompensă.

Raritatea este o caracteristică care defineşte importanţa misiunilor, şi din acest punct de vedere pot fi trei tipuri de misiuni: obişnuite, care sunt reprezentate vizual prin faptul că numele lor este scris cu alb, rare, indicate prin culoarea albastră, şi epice, indicate de culoarea mov.

Recompensele pot fi împărţite în trei categorii: o monedă de tranzacţie, echivalentul banilor din lumea reală, un tip de obiect de schimb, similar monedei, dar care este utilizabilă într-un cadru mai restrâns şi mai specific, şi obiecte fizice, cu care se poate interacţiona. Acestea din urmă sunt, la rândul lor, de diferite tipuri, toate cu specificaţii aparte. Versatilitatea tabelelor în limbajul de scripting lua ne ajută să putem stoca în tabelul principal diferite tipuri de recompense. Spre exemplu, pentru moneda de tranzacţie, se va crea un tabel care conţine doar numele recompensei şi cantitatea oferită, pe când pentru un obiect se vor reţine numele, tipul, level-ul sau cantitatea, după caz. Tabelele din lua sunt asemănătoare obiectelor din limbajele de programare ca şi concept.

Odată obţinute toate aceste date despre misiuni, acestea vor fi stocate în tabelul QUESTS, asupra căruia se vor aplica fundamentele algoritmice descrise în capitolul 1 al acestei lucrări.

Figura 22. Exemple ale culorilor misiunilor ce indică raritatea

Funcţia filterQuests

Înainte de a genera un traseu ponderat minim într-un graf, algoritmul are nevoie de nodurile care formează graful respectiv. În momentul apelării, funcţia cunoaşte tabelul QUESTS care conţine toate evenimentele active de la momentul curent. Dintre acestea, se vor selecta doar acele misiuni pentru care CheckButton-ul criteriului reprezentat este activ. Ele se vor adăuga tabelului FilteredQuests, care va fi trimis ca parametru algoritmilor în scopul generării drumului minim.

De exemplu, în figura 23 se prezintă un model de selecţie, unde mai întâi se atribuie unei variabile starea CheckButton-ului. Apoi, se verifică dacă misiunea a mai fost adăugată până în

acel moment. Pe poziţia 3 în tabelul QUESTS se află id-ul facţiunii reprezentate, deci se vor selecta toate misiunile care au valoarea 1090 pe pozitia 3.

```
enabled = myCheckButton5:GetChecked();  --selecting Kirin Tor
if enabled then
    for i=1,table.getn(QUESTS) do
        if (QUESTS[i][3] == 1090) and checkQuest(FilteredQuests,QUESTS[i][1])==0
         then
            FilteredQuests[table.getn(FilteredQuests)+1]=QUESTS[i];
        end
    end
end
```

Figura 23. Modelul de selecţie al stării butonului de verificare

Dupa obţinerea drumului de cost minim, se doreşte afişarea acestuia pe harta de joc pentru ca jucătorul să îl poată vizualiza şi să înceapă parcurgerea sa. Pentru aceste reprezentări vizuale au fost folosite funcţii din librăria HereBeDragons descrisă în secţiunea 2.2.

Figura 24. Traseul generat de aplicaţie în joc

Pentru fiecare nod din graful de evenimente selectate, s-a creat un frame de tipul buton. Texturii sale îi este atribuită imaginea unui cerc verde cu contur galben, aşa cum se poate vedea în figura 24. Punctul va fi situat pe hartă in punctele de coordonate x,y caracteristice fiecărei misiuni în parte. Dimensiunile punctelor au fost setate pentru a se putea distinge uşor faţă de restul, care alcătuiesc muchiile grafului. Pentru generarea căilor dintre misiuni, se păstrează constantă

distanţa dintre punctele care alcătuiesc muchiile drumului de cost minim, setând lungimea drumui ca fiind direct proporţională cu numărul de puncte care formează o muchie din punct de vedere vizual. Orientările muchiilor depind de poziţia unui punct faţă de celălalt, astfel că au fost calculate puncte intermediare care se află pe dreapta determinată cele două puncte, la distanţe egale între ele. Spre exemplu, în figura 25 se arată modul de generare a punctelor ce reprezintă o muchie, în cazul în care coordonatele x şi y a primului punct sunt mai mici decât ale celuilalt. Numărul de puncte care alcătuiesc calea este proporţional cu distanţa celor două puncte în plan. Se calculează h1 şi h2 care asigură o discretizare echidistantă a punctelor in intervalul [x1, x2] repsectiv [y1, y2]. În final, se creează frame-uri pentru toate punctele ce vor reprezenta vizual muchia. Punctele vor fi situate la noile coordonate x, y calculate şi oferite ca argument funcţiei. Dimensiunile punctelor vor fi mai reduse pentru a nu se confunda cu un nod ce reprezintă un eveniment.

```
distance = math.floor(distance);
h1=math.abs(x2-x1)/distance;
h2=math.abs(y2-y1)/distance;


if x1 <= x2 and y1<=y2 then
    if table.getn(intersectii)==0 then
        for i=2,distance do
            B:setPathPoints(QUEST_MAPS[1],0,x1+(i-1)*h1,y1+(i-1)*h2);
```

Figura 25. Modul de generare al unei muchii

Pe lângă reprezentarea drumului pe harta de joc prin elemente grafice, programul afişează ordinea de parcurgere a misiunilor şi în frame-ul principal, şi anume în frame-ul ancorat în partea stângă. Există situaţii când misiunile sunt foarte apropiate, iar vizualizarea drumului devine greu de urmărit, aceasta fiind o situaţie când lista de misiuni poate clarifica ambiguitatea generată de aglomerarea multor elemente vizuale în aceeaşi regiune.

Pentru fiecare misiune în parte, este creat un frame individual. Primul va fi ancorat în partea TOPLEFT a frame-ului părinte, însă celelalte vor fi ancorate de frame-ul de dinaintea sa, astfel obţinându-se o listă de misiuni aşa cum se poate observa în figura 26. La prima vedere, fiecare frame conţine un font-string cu numele misiunii şi culoarea acesteia. Culorile sunt indicator pentru raritatea evenimentelor, lucru acoperit anterior. Însă fiecare frame conţine un handler care captează evenimente de tipul OnEnter.

Figura 26 Listele generate de aplicație în joc cu detalii adiționale

În urma declanşării unui astfel de eveniment va apare un tooltip ce conţine informaţii adiţionale despre fiecare misiune în parte, cum ar fi numărul total de facţiuni cu care va primi reputaţie, timpul rămas exact până la expirarea evenimentului, sau numărul exact de monede pe care jucătorul îl primeşte. Afişajele sunt, la rândul lor, afişate corespunzător. Spre exemplu, o zi rămasă va fi afişată drept 1 day pe când două zile vor fi recunoscute ca fiind mai multe, lucru ce se poate vedea în figura 26.

Interfaţa originală a jocului menţionează doar o singură facţiune în momentul oferirii informaţiilor despre misiuni, însă în realitate ele pot fi mai multe. Completarea unei astfel de misiuni acordă puncte cu toate facţiunile implicate, un exemplu fiind facţiunea Wardens, care oferă misiuni pe toată întinderea hărţii. Însă fiecare regiune din joc aparţine unei facţiuni distincte, şi astfel realizarea unui eveniment oferă bonus atât cu Wardens, cât şi cu facţiunea prezentă în regiune. Aplicaţia prezintă informaţii complete cu privire la facţiunile implicate şi sunt luate corect în calcul în momentul selecţiei.

Timpul rămas până la expirarea evenimentului este generat de o funcție din API, care întoarce numărul de minute rămase. Cunoscând faptul că o oră conține 60 de minute şi o zi conține 24 de ore, timpul poate fi convertit cu exactitate în format zi – ore – minute. Interfața obișnuită nu oferă informații atât de exacte cu privire la timpul rămas deoarece aceasta afișează întotdeauna unitatea cea mai semnificativă. Spre exemplu, 7 ore şi 36 de minute vor fi afișate în joc drept 7

ore, cele 36 de minute fiind omise. Interfața aplicației prezentate oferă informaţia completă, întregind perspectiva utilizatorului asupra evenimentelor active.

O transformare similară există şi în cazul recompensei de tip monetar. În cazul în care o misiune oferă o astfel de recompensă, atunci funcţia GetQuestLogRewardMoney(id), ce provine din API, va returna o valoare mai mare ca 0 ce reprezintă numărul de monezi de cupru pe care un jucător îi va primi. Cunoscând că 100 de monezi de cupru valorează cât o monedă de argint şi că o monedă de argint valorează cât una de aur, atunci se poate afla cu exactitate cât va primi jucătorul ca recompensă pentru completarea obiectivelor. Moneda de schimb reprezintă un element important al jocului, fiind posibilă obţinerea acesteia încă de la primele etape. În primul rând este folosită pentru achiziţionarea tuturor obiectelor folosibile în joc, astfel un scop al jocului fiind obţinerea în mod constant de astfel de monezi de tranzacţie pentru a avea acces la serviciile pe care jocul le oferă. În al doilea rând, există conevrsii bidirecţionale între bani reali – banii virtuali reprezentaţi de această monedă, graţie unor funcţionalităţi recent adăugate de Blizzard Entertainment care permit utilizatorilor să achiziţioneze servicii cu moneda de tranzacţie din joc, ca alternativă pentru banii reali. Din aceste motive, un utilizator doreşte să ştie cu exactitate câte astfel de monezi primeşte în urma completării unui World Quest, pentru a putea estima câştigul total, iar interfaţa creată de aplicaţia prezentată oferă această informaţie.

Concluzii

Lucrarea aceasta exemplifică un mod prin care algoritmi teoretici sunt puși în practică. O parte importantă a procesului este reprezentată de crearea datelor de input, ce poate fi un proces foarte complex pe cazuri concrete. În aplicația prezentată în această lucrare, complexitatea provine din existența unei multitudini de criterii de selecție a obiectivelor care au diferite caracteristici. Au fost acoperite cazurile cele mai generale şi mai comune din punct de vedere al selecţiei, însă sistemul de filtrare se poate imbogăţi prin adăugarea mai multor parametri ce fac o selecţie mai specifică. Rezultatul obţinut este în egală măsură de important, deoarece nu este de ajuns ca un algoritm să se termine cu succes. Output-ul trebuie interpretat şi prelucrat pentru a oferi informaţiile unui utilizator care nu cunoaşte fundamentele algoritmice sau procesele care funcţionează pentru generarea rezultatului. Detalii cu privire la complexitatea algoritmului sau problema pe care acesta încearcă să o rezolve sunt mai puţin importante din punctul lui de vedere, însă output-ul prelucrat trebuie să fie cât mai sugestiv şi mai intuitiv cu putinţă pentru a fi uşor de interpretat şi de către jucător.

Nu trebuie neglijată, însă, importanța algoritmului propriu-zis. Aici sunt definite modalităţile prin care output-ul este prelucrat în scopul obţinerii rezultatului final. O problemă de tipul P = NP este abordată în lucrare, care e rezolvată cu algoritmi de aproximare în lipsa unui algoritm polinomial. Rezultatul obţinut nu este întotdeauna cel optim, însă e întotdeauna apropiat de acesta, aplicând agorimul lui Christofides care garantează generarea unui drum cu factor de $\frac{3}{2}$ din optim. Algoritmului i-au fost aduse modificări pentru a trata situațiile în care algoritmul întâlnește constrângeri spațiale și/sau temporale. Reducerea factorului de aproximare este dificilă având în vedere faptul că lumea jocului World of Warcaft este amplă, vastă şi dinamică. Algoritmul de trasare a drumului optim este aplicat unor evenimente dinamice, care pot expira în timp, fiind înlocuite de altele, astfel condiţii prestabilite de generare a drumurilor, ca de exemplu un grup de misiuni care se execută mereu în aceeaşi ordine, nu pot exista. Output-ul algoritmului este influenţat şi de poziţia jucătorului pe hartă, lucru de asemenea foarte importantă pentru alegerea nodului de start, care poate influenţa în mod dramatic traseul generat.

Aplicând bazele teoretice demonstrate în capitolul 1, folosind tehnologiile menţionate şi descrise în capitolul 2 şi adăugând noi funcţionalităţi interfeţei grafice detaliate în capitolul 3, a fost obţinută versiunea cu numărul 1.0 al programului Pathfinder. Acest program automatizează procesul de selecţie şi de decizie al unui traseu printre evenimentele active, îmbogăţeşte interfaţa grafică originală a jocului oferind informaţii adiţionale importante şi adaugă elemente vizuale noi hărţii de joc, unde va fi marcat drumul generat de algoritm. Informaţiile adiţionale prezentate întregesc cunoştinţele legate de o misiune pentru jucător, care poate calcula cu exactitate care este câştigul total în urma realizării evenimentelor. Exactitatea este necesară, în special în cazul în care recompensa este reprezentată de moneda de tranzacţii a jocului, care poate fi convertită în bani reali datorită funcţionalităţilor jocului.

Această versiune a programului a pus bazele unei posibile dezvoltări ulterioare a acestuia. Spre exemplu, timpul alocat de 5 minute pentru fiecare eveniment în parte poate fi recalculat după fiecare efectuare a traseului. Se calculează timpul mediu de terminare a unui eveniment care depinde de stilul de joc al fiecărui utilizator în parte, iar timpul alocat va putea fi ajustat în funcţie de acest aspect. Timpul este considerat un aspect important pentru decizia traseului deoarece ordinea de parcurgere a misiunilor este generată în aşa fel încât obiectivele să nu expire înainte de a ajunge la ele. Sistemul nu este momentan perfect deoarece, dacă timpul pre-alocat nu corespunde stilului de joc al unui jucător, algoritmul poate exclude misiuni din traseu pe care jucătorul ar fi putut să le efectueze, sau poate exista situaţia când un jucător cu stil de joc mai relaxat nu va ajunge la misiuni în timp util şi acestea vor expira înainte de a fi completate, chiar dacă algoritmul le-a inclus, iniţial, în traseul optim.

Poate fi extinsă interfaţa grafică adăugând câştigurile totale ale misiunilor selectate. Utilizatorul va şti astfel care sunt recompensele totale obţinute în urma traseului de cost minim şi astfel poate compara câştigurile faţă de alte drumuri.

Selecţia poate fi extinsă în sensul că fiecare misiune aleasă poate să fie inclusă sau nu în traseul final, în cazul în care utilizatorul decide că nu vrea să participe la un eveniment anume. De asemenea, poate fi inclusă şi opţiunea de customizare a interfeţei grafice, utilizatorul putând alege o anumită imagine de fundal în stilul celei originale sau celei pre-determinate.

Toate acestea pot fi incluse în versiunea 2.0 Pathfinder, deoarece se consideră faptul că versiunea 1.0 a atins toate obiectivele propuse.

Referinţe bibliografice

[1] Karl Menger, *Ergebnisse eines Kolloquiums* 3, 11-12 (1930).

[2] M. M. Flood, *The Traveling-Salesman Problem*, Oper. Res. 4, 61-75(1956).

[3] TSP World Record http://www.math.uwaterloo.ca/tsp/world/

[4] Goodrich, Michael T.; Tamassia, Roberto , Wiley, *Algorithm Design and Applications*, pp. 480(2015)

[5] Nicos Christofides, *Worst-case analysis of a new heuristic for the travelling salesman problem*, Report 388, Graduate School of Industrial Administration, CMU, Februarie 1976 http://www.dtic.mil/dtic/tr/fulltext/u2/a025602.pdf

[6] Lawler, E., *Combinatorial Optimization* http://www.plouffe.fr/simon/math/CombinatorialOptimization.pdf

[7] J.A. Hoogeveen, *Analysis of Christofides' heuristic: Some paths are more difficult than cycles,* Operations Research Letters 10, 291-295, July 1991

[8] Alan A. Bertossi , *The Edge Hamiltonian Path Problem is NP – Complete* ,Information Procssing Letters, 1981

[9] Arthur M. Hobbs, *Traveling Salesman Problem*

[10] Chaoxu Tong, „*Approximation Algorithms for the Traveling Salesman Problem*", ORIE 6300 Mathematical Programming ,December 2014 https://people.orie.cornell.edu/dpw/orie6300/Recitations/Rec12.pdf

[11] Cormen, Thomas; Charles E Leiserson, Ronald L Rivest, Clifford Stein (2009). *Introduction To Algorithms (Third ed.)*. MIT Press, P 562, 569-570

[12] Kruskal, J. B. (1956). *On the shortest spanning subtree of a graph and the traveling salesman problem*. Proceedings of the American Mathematical Society. 7: 48–50.

[13] Vladimir Kolmogorov *Blossom V: a new implementation of a minimum cost perfect matching algorithm*, https://pdfs.semanticscholar.org/930b/9f9c3bc7acf3277dd2361076d40ff03774b2.pdf

[14] Gabow, H.: *Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs*. PhD thesis, Stanford University (1973)

[15] Christofides, N. ,"*The shortest hamiltonian chain of a graph", Siam Journal on Applied Mathematics*, Vol.19, No.4, pp. 689-696

[16] Eulerian and Hamiltonian graphs, http://compalg.inf.elte.hu/~tony/Oktatas/TDK/FINAL/Chap%203.PDF

[17] H.J. Bockenhauer, J. Hromkovic, J. Kneis, J. Kupke, "*Theory of Computing Systems*", 2007, Volume 41, Issue 3, pp 431-444

[18] J. Whitehead, R. Roe, "*World of Warcraft Programming – A Guide and Reference for Creating WoW Addons, Second Edition*", 2010

[19] World of Warcraft: Terms of Use http://eu.blizzard.com/en-gb/company/legal/

[20] Roberto Ierusalimschy, *Programming in Lua*, 2013

[21] Ierusalimschy, R.; Figueiredo, L. H.; Celes, W. (2007). *"The evolution of Lua"*

[22] *The computer language shootout benchmarks* http://benchmarksgame.alioth.debian.org/

[23] Lua – an extensible extension language http://www.lua.org/spe.html

[24] HereBeDragons Lua library https://www.wowace.com/projects/herebedragons

[25] WoW API updated list http://wowprogramming.com/docs

[26] Game IDs http://wow.gamepedia.com/MapID

[27] Ierarhia obiectelor interfeței grafice http://wowprogramming.com/docs/widgets_hierarchy

[28][GPL] General Public License (GPL). http://www.fsf.org/licensing/licenses/gpl.html .

[29] GIMP User Manual https://docs.gimp.org/2.8/en/

[30] OnLoad Frame Event, http://wowprogramming.com/docs/scripts/OnLoad

[31] Extracting interface files http://wowwiki.wikia.com/wiki/Extracting_interface_files

[32] Steven Holzner, XML Complete, Computing-McGraw-Hill, pp:23-165, 1997

[33] Problema Comis-Voiajorului, http://software.ucv.ro/~cmihaescu/ro/teaching/ACA/docs/PCV.pdf