SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ELEKTROTEHNIČKI FAKULTET

Sveučilišni preddiplomski studij računarstva

DINAMIČKI GRAF

Završni rad

Alen Čamagajevac

Osijek, 2016.

SADRŽAJ

1. Uvod

Obrada i sortiranje podataka su jedne od temeljnih funkcija računala u modernom programiranju. Efikasno manipuliranje podacima omogućava sustavima brže djelovanje i jednostavniju uporabu. Upravo zato programeri koriste razne tehnike za strukturiranje podataka i izvršavanje raznih algoritama nad njima. U ovom radu opisana je struktura podataka graf. Pravljenjem dinamičkog grafa omogućuje se lako dodavanje podataka u navedenu strukturu, te pravljenje poveznica između danih podataka. Ovakav graf pogodan je za korištenje u brojnim simulacijama, matematičkim izračunima i prikazivanju odnosa između pojedinih podatkovnih jedinica. Poznate web-tražilice modeliraju internet kao graf gdje su web stranice čvorovi u grafu a poveznice (linkovi) između stranica, bridovi grafa. Računalne igrice, također, često zahtijevaju rad sa velikom količinom informacija. Upravo zato grafovi su redovito korišteni u njihovoj izradi, a ponekad su i osnova rada programa koji upravlja radom igre (*Game Engine*). Pravljenjem igre fantazijske tematike, čija se radnja odvija u Zaboravljenim Tamnicama, pokušati će se dati primjer korištenja grafa.

U prvom djelu rada opisuje se sama struktura grafa i razlike između pojedinih tipova grafova, te opis osnovnih algoritama koji se izvršavaju nad strukturom. Zatim se prikazuje programska implementacija grafa u C++ programskom jeziku, pisanog u .NET Visual Studio 2015 okruženju. Prikazuje se dinamičko alociranje memorije, oslobađanje zauzete memorije i kreiranje grafa korištenjem matrice susjedstva te listom susjeda. Razrađuju se objekti i strukture koje su potrebne za realizaciju grafa. Nakon toga se opisuje ideja pravljenja računalne igrice. Strukturnim UML dijagramom prikazuju se atributi i funkcije koje klase obavljaju, te međuodnos između pojedinih klasa. Na posljetku se prikazuju rezultati igranja igre i predstavljaju mogućnosti koje igrač ima u svojim avanturama.

1.1 Zadatak završnog rada

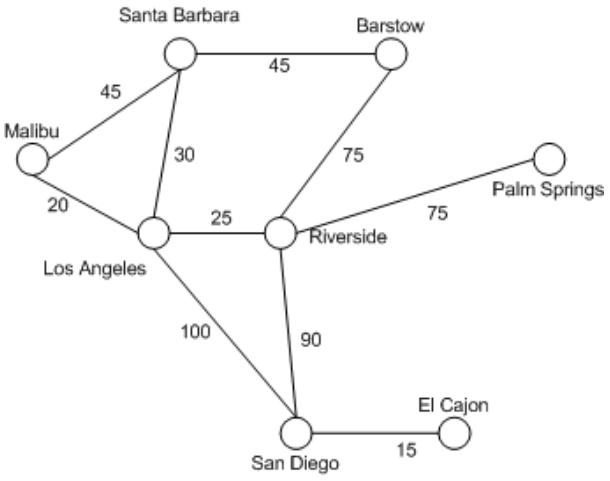
Opisati strukturu podataka graf, te dati ideju kako bi se implementirao dinamički graf koji bi se mogao širiti u smislu broja vrhova i grana. U obliku primjera aplikacije ili računalne igre pokazati kako se takva struktura može praktično izvesti. Posebno opisati kako se u tom slučaju rukuje s memorijom. Poželjno je koristiti objektno orijentirani pristup

2. TEORIJSKI OPIS GRAFOVA

Graf se sastoji od skupa čvorova (*Vertex*) i skupa bridova (*Edge*). Matematički, može se zapisati kao Graf G = (V, E).

Ovdje čvorovi predstavljaju podatak koji želimo spremiti u samu strukturu. Podatak može biti broj, ime, datum ili neki drugi osnovni podatak, no može biti i složeni podatak sastavljen od više osnovnih podataka. Takav podatak mogao bi biti 'zaposlenik' koji bi se sastojao od dva niza znakova koji bi predstavljali ime i prezime, te broja koji bi predstavljao godine. Također, moramo poznavati ulazni i izlazni stupanj čvora. Ulazni stupanj govori koliko bridova ulazi u čvor, a izlazni stupanj koliko bridova izlazi iz čvora.

Grane objašnjavaju relaciju između dva čvora. Same grane mogu imati usmjerenje i vrijednost, pa tako i grafovi mogu biti usmjereni i neusmjereni, te težinski i bestežinski. U usmjerenom grafu ovo znači da ako možemo iz čvora V1 doći u čvor V2 ne znači da možemo iz čvora V2 doći u čvor V1. Težina brida može biti korisno svojstvo u modeliranju stvarnih entiteta. Ako bi čvorovi predstavljali gradove u državi a bridovi ceste koje ih spajaju, težina brida može predstavljati dužinu ceste. Težina brida se najčešće predstavlja kao uređeni par T(W1, W2) gdje je W1 težina brida kada promatramo put od čvora V1 do čvora V2, a W2 težina brida kada promatramo graf od čvora V2 do čvora V1.

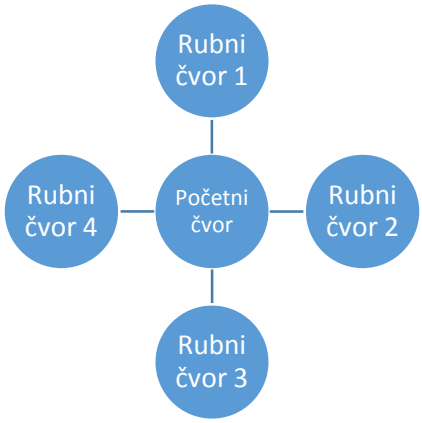


Slika 2.1

Na slici 2.1. vidi se graf u kojem čvorovi predstavljaju gradove a bridovi ceste između gradova. Graf je težinski i neusmjereni. Težine bridova prikazane su brojčano na bridu. Slika također prikazuje i da graf pripada nelinearnoj strukturi podataka jer pojedinačni entitet može biti povezan s više od jednog drugog entiteta. Čvor 'Los Angeles' tako je povezan sa čvorovima 'Santa Barbara', 'Malibu', 'Riverside' i 'San Diego'.

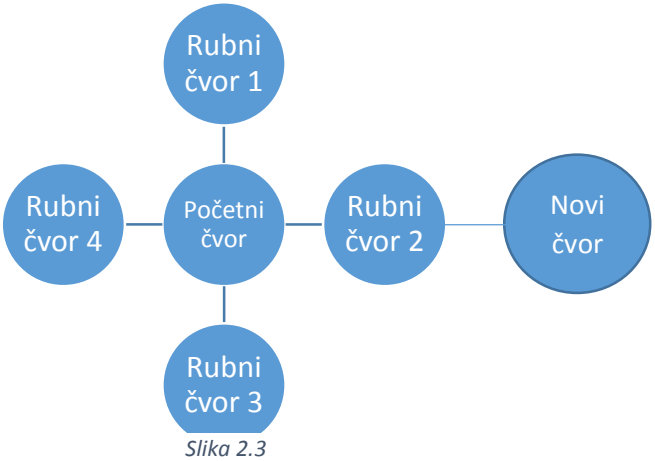
2.1 Opis dinamičkoga grafa u službi igre

U igri svaki čvor unutar grafa predstavlja jedno polje po kojem se igrač može kretati. Kretanje se odvija u četiri smjera; gore, dolje, lijevo i desno. Zato svaki čvor ima ulazni, tj. izlazni stupanj četiri. To znači da svaki čvor može biti povezan s najviše četiri ostala čvora.



Slika 2.2

Na slici 2.2. vidimo da su bridovi između čvorova neusmjereni i bestežinski. Pošto se igrač mora moći kretati u svim smjerovima bolje rješenje je postavljanje jednog neusmjerenog brida nego dva usmjerena. Kod dinamičkoga grafa, na početku same igre, ne inicijalizira se nijedan čvor osim početnoga čvora na kojemu igrač počinje igru. Unošenjem naredbi kretanja igrač se pomiče u određenom smjeru te se tek tada inicijalizira čvor tamo gdje je potreban. Inicijalizacijom čvora pravi se i brid između polja na kojemu se igrač prethodno nalazio i novostvorenog polja na kojemu se igrač trenutno nalazi. Ovo omogućuje kretanje unedogled jer će se svaki puta inicijalizirati novi čvor kada igrač prijeđe rubni čvor. Slika 2.3 prikazuje stanje grafa nakon što igrač unese naredbu za kretanje desno a prethodno se nalazio na rubnom čvoru 2. Treba naglasiti kako se u grafu ne bi kreiralo novo polje kad igrač unese naredbu za



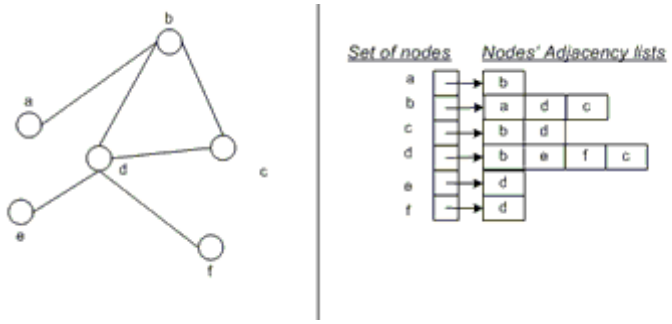
Slika 2.3

kretanje u smjeru gdje polje već postoji. U takvom slučaju igrač bi samo otišao na već postojeće polje.

3. PRIKAZ GRAFA UNUTAR C++ PROGRAMSKOGA JEZIKA

Iako su grafovi jako česta struktura podataka koja se koristi u nizu različitih problema, ne postoji ugrađena struktura grafa unutar .NET c++ sučelja. Razlog tomu je što efikasna implementacija grafa ovisi o velikom broju faktora koji su specifični za problem koji se rješava. Grafovi su zato najčešće modelirani pomoću liste ili matrice susjedstva a u ovome radu koristiti će se lista susjedstva.

Korištenjem liste susjedstva u svaki čvor stavlja se lista svih čvorova s kojima je taj čvor povezan. Ovo omogućava laku navigaciju unutar grafa. Prelaženje s jednog čvora na drugi odvija se u tri koraka. Prvo igrač unosi naredbu za pomicanje u određenom smjeru. Zatim se provjerava lista susjedstva trenutnog čvora. Ako postoji čvor u smjeru gdje se igrač želi pomaknuti, onda odlazimo na to polje. Ako ne postoji inicijalizira se novo polje i stavlja u matricu susjedstva trenutnog polja. Treći korak je da novostvoreno polje povežemo sa prethodnim poljem tako što prethodno polje stavimo u listu susjeda novostvorenog polja. Grafički prikaz ovog sustava prikazan je na slici 3.1.



Slika 3.1

Programski se ovo ostvaruje pomoću klase *GameMap* koja u sebi sadrži sve čvorove koji se trenutno nalaze na karti. Ova klasa sadrži funkcije za dodavanje polja unutar karte, dodavanje bridova između dva čvora, pomicanje igrača u određenom smjeru, traženje predmeta i napadanje protivnika koji se nalaze na trenutnom čvoru.

```
class GameMap
{
private:
        typedef std::map<std::pair<int, int>, tile*> tmap;
        tmap TileMap;
```

```
        Coordinate Coordinates;
public:
        GameMap();
        bool addvertex(const std::string, const int, const int);
        void addedge(const int fromX, const int fromY, const int toX, const int toY, int cost);
        void move(std::string direction)
        void search(Hero&);
        void set_coordinates(int cx, int cy);
        Coordinate get_coordinates();
        tmap get_tiles();
};
```

Svi čvorovi karte objedinjeni su u asocijativnom spremniku *std::map* nazvanog *tmap*. *Std::Map* je spremnik koji na osnovu ključa pronalazi i daje određeni podatak. Ključ u ovom slučaju je par cijelih brojeva koji predstavlja koordinate na karti a podatak je čvor. Sam čvor je struktura nazvana tile definirana na sljedeći način.

```
struct tile {
        typedef std::pair<int, tile*>   AdjacentTile;
        std::vector<AdjacentTile>       adjacent;
        std::string                     name;
        std::string                     description;
        Item*                           item;
        Monster*                        monster;
        tile(std::string);
};
```

Vidimo da svaki čvor sadrži listu čvorova susjeda nazvanu *adjecent*, ime čvora, opis čvora i dva pokazivača koji služe za prikaz protivnika i stvari koje se mogu naći na polju. Ova dva pokazivača se nasumično alociraju stvaranjem polja u konstruktoru strukture *tile*.

```
tile::tile(std::string nme)
{
        name = nme;

        srand(time(NULL));
        int rndItemGen = rand() % 100;

        if (rndItemGen > 70)
        {
                item = new Item;
        }
        else
        {
                item = NULL;
        }
```

```
        srand(time(NULL));
        int rndMonsterGen = rand() % 100;

        if (rndMonsterGen > 50)
        {
                monster = new Monster;
        }
        else
        {
                monster = NULL;
        }
}
```

Dvije varijable *rndItemGen* i *rndMonsterGen* su varijable koje poprimaju nasumičnu vrijednost u rasponu od 0 do 100. Ako *rndItem* ima veću vrijednost od 70 dodaje se *item* na polje, dok *rndMonster* mora imati vrijednost veću od 50 da bi se protivnik stavio na polje.

Dodavanje čvora u *std::map* odvija se pomoću funkcije *GameMap::addvertex(const std::string, const int, const int);*

```
bool GameMap::addvertex(const std::string name, const int Xcord, const int Ycord)
{
        tmap::iterator itr = TileMap.begin();
        itr = TileMap.find(std::make_pair(Xcord, Ycord));
        if (itr == TileMap.end())
        {
                tile *t;
                t = new tile(name);
                TileMap[std::make_pair(Xcord, Ycord)] = t;
                return true;
        }
        std::cout << "\nPolje već postoji!" << std::endl;
        return false;
}
```

U ovoj funkciji prvo pregledavamo listu susjeda trenutnog čvora te tražimo čvor koji ima koordinate na kojima se igrač trenutno nalazi. Ako takvog polja nema dinamički se alocira novo polje naredbama.

```
tile *t;
t = new tile(name);
```

To novo polje stavlja se u spremink *TileMap*. U slučaju da se na igračevim koordinatama našao čvor tada bi ispisali poruku da polje već postoji i vratili *false* vrijednost koja bi indicirala da nismo napravili novo polje na karti.

Funkcija *GameMap::addedge(const int fromX, const int fromY, const int toX, const int toY, int cost)* dodaje brid između dva polja. Funkcija za argumente prima koordinate polja iz kojega dodajemo brid i koordinate polja prema kojemu ide brid.

```
void GameMap::addedge(const int fromX, const int fromY, const int toX, const int toY, int
cost)
{
	tile *f = (TileMap.find(std::make_pair(fromX, fromY))->second);
	tile *t = (TileMap.find(std::make_pair(toX, toY))->second);

	std::pair<int, tile*> edge = std::make_pair(cost, t);
	f->adjacent.push_back(edge);
}
```

Funkcija pronalazi čvorove koji se nalaze na koordinatama *F(fromX,fromY)* i *T(toX, toY).* Zatim pravi novi brid *edge* koji se sastoji od težine brida i čvora prema kojemu brid ide. Zadnja naredba stavlja taj brid u matricu susjedstva trenutnog čvora. Tako se u svakom čvoru nalazi lista svih čvorova koji su susjedni tom čvoru i težina brida koji ih spaja. Argument *cost* koji predstavlja težinu brida postavlja se na 1 u svim slučajevima.

3.1 Pomicanje po grafu

Pomicanje i dinamičko alociranje novih čvorova obavljeno je u funkciji *void GameMap:: move(std::string direction)*; Ova funkcija kao argument prima samo naredbu igrača za smjer. Najprije funkcija pamti koordinate na kojima se igrač trenutno nalazi. Te u ovisnosti o smjeru kretanja modificira koordinate.

```
void GameMap::move(std::string dir)
{
	int oldX, oldY;
	oldX = this->get_coordinates().X;
	oldY = this->get_coordinates().Y;

	if (dir == "go west") {
		set_coordinates(--oldX, oldY);
	}

	else if (dir == "go east") {
		set_coordinates(++oldX, oldY);
	}

	else if (dir == "go north") {
		set_coordinates(oldX, ++oldY);
	}
```

```
else if (dir == "go south") {
        set_coordinates(oldX, --oldY);
}
```

Funkcija zatim provjerava postoji li polje na novim koordinatama. Ako postoji sljedeći dio koda se preskače i ispisuje se samo opis polja na novim koordinatama, tj. polja na kojem se igrač nalazi nakon kretanja. U slučaju da se igrač nalazio na rubnom polju te u smjeru kretanja nema alociranog čvora funkcija pravi novi čvor pozivom funkcije *bool GameMap::addvertex.* Pozivom funkcije *bool GameMap::addvedge* dva puta prave se bridovi između čvora sa novim koordinatama i čvora sa starim koordinatama. Na kraju se ispisuje opis čvora na novim koordinatama, provjerava se postoji li protivnik na novom čvoru i ukoliko postoji ispisuje se i opis protivnika.

```
std::map<std::pair<int, int>, tile*>::iterator itr;
itr = TileMap.find(std::make_pair(Coordinates.X, Coordinates.Y));

if (itr == TileMap.end())
{
        srand(time(NULL));
        int random = rand() % 12 + 1;
        std::string name;

        if (random == 1)      name = "Overgrown ruins";
        else if (random == 2)  name = "Crypt";
        else if (random == 3)  name = "Hazarrdous terrain";
        else if (random == 4)  name = "Cavern";
        else if (random == 5)  name = "Great Hall";
        else if (random == 6)  name = "Temple";
        else if (random == 7)  name = "Underground fortress";

        addvertex(name, Coordinates.X, Coordinates.Y);
        addedge(oldX, oldY, Coordinates.X, Coordinates.Y, 1);
        addedge(Coordinates.X, Coordinates.Y, oldX, oldY, 1);

std::cout << TileMap[std::make_pair(Coordinates.X, Coordinates.Y)]->description <<
std::endl;

if (TileMap[std::make_pair(Coordinates.X, Coordinates.Y)]->monster != NULL)
std::cout << TileMap[std::make_pair(Coordinates.X, Coordinates.Y)]->monster-
>printDescription() << std::endl;
        }
```

Ukoliko nismo trebali praviti novi čvor na novim koordinatama ispisujemo samo opis polja i opis protivnika ako se on nalazi na polju.

```
        else
        {
                std::cout << TileMap[std::make_pair(Coordinates.X, Coordinates.Y)]-
>description << std::endl;
                if (TileMap[std::make_pair(Coordinates.X, Coordinates.Y)]->monster !=
NULL)
                        std::cout << TileMap[std::make_pair(Coordinates.X, Coordinates.Y)]-
>monster->printDescription() << std::endl;
        }
}
```

3.2 Prikaz ostalih klasa potrebnih za realizaciju igre

Uz već spomenute klase *GameMap, Monster* i *Item,* potrebne su nam i klase *Hero* i *Game.* Klasa *Hero* reprezentira igrača unutar igre.

```
class Hero
{
private:
        int Strenght;
        int Agility;
        int Magic;
        int Mana;
        int Health;
        std::list<Item> Inventory;
        Item* armorSlot;
        Item* weaponSlot;
public:
        Hero();
        bool isAlive();
        void printStats();
        void printAttributes();
        void printInventory();
        void takeItem(Item*);
        void equipItem();
};
```

Varijable *Health* i *Mana* označavaju zdravlje igrača i njegovu sposobnost da koristi razne magije. Varijabla *Strenght* označava igračevu snagu, *Agility* označava igračevu sposobnost izbjegavanja udaraca a *Magic* označava jačinu igračevih magija. Igrač ima i listu stvari koje nosi nazvanu *Inventory*. *armorSlot* je pokazivač koji predstavlja štit koji igrač nosi a *weaponSlot* oružje koje igrač drži u ruci.

Funkcija *bool isAlive()* vraća vrijednost koja govori dali je igrač trenutno živ. Funkcije *void printAttributes()* i *void printStats()* ispisuju atribute igrača dok *void printInventory()* ispisuje

sve stvari koje se nalaze u igračevom inventaru. Naredbama *void takeItem(Item*)* i *void equipItem()* igrač skuplja stvari sa polja na kojemu se nalazi i dodaje stvari u svoj inventar.

Klasa Game glavna je klasa unutar igre. Ona objedinjuje sve druge klase i poziva određene funkcije na temelju igračevih naredbi.

```
class game
{
private:
	GameMap Map;
	Hero hero;
public:
	game();
	GameMap& get_map()
	{
		return Map;
	}
	void command(std::string);
};
```

Kao privatne varijable klasa Game sadrži *GameMap* varijablu *Map* i *Hero* varijablu *hero*. Funkcije koje klasa Game sadrži su *GameMap& get_map()* i *void command(std::string)*. Funkcija *command* je glavna funkcija unutar igre jer ona poziva funkcije iz svih ostalih klasa u ovisnosti o igračevim naredbama.

```
void game::command(std::string comm)
{
	if (comm == "attack")
	{
		Map.attack(hero);
	}

	else if (comm == "search")
	{
		Map.search(hero);
	}
	else if (comm == "stats")
	{
		hero.printStats();
	}
	else if (comm == "attributes")
	{
		hero.printAttributes();
	}
	else if (comm == "inventory")
```

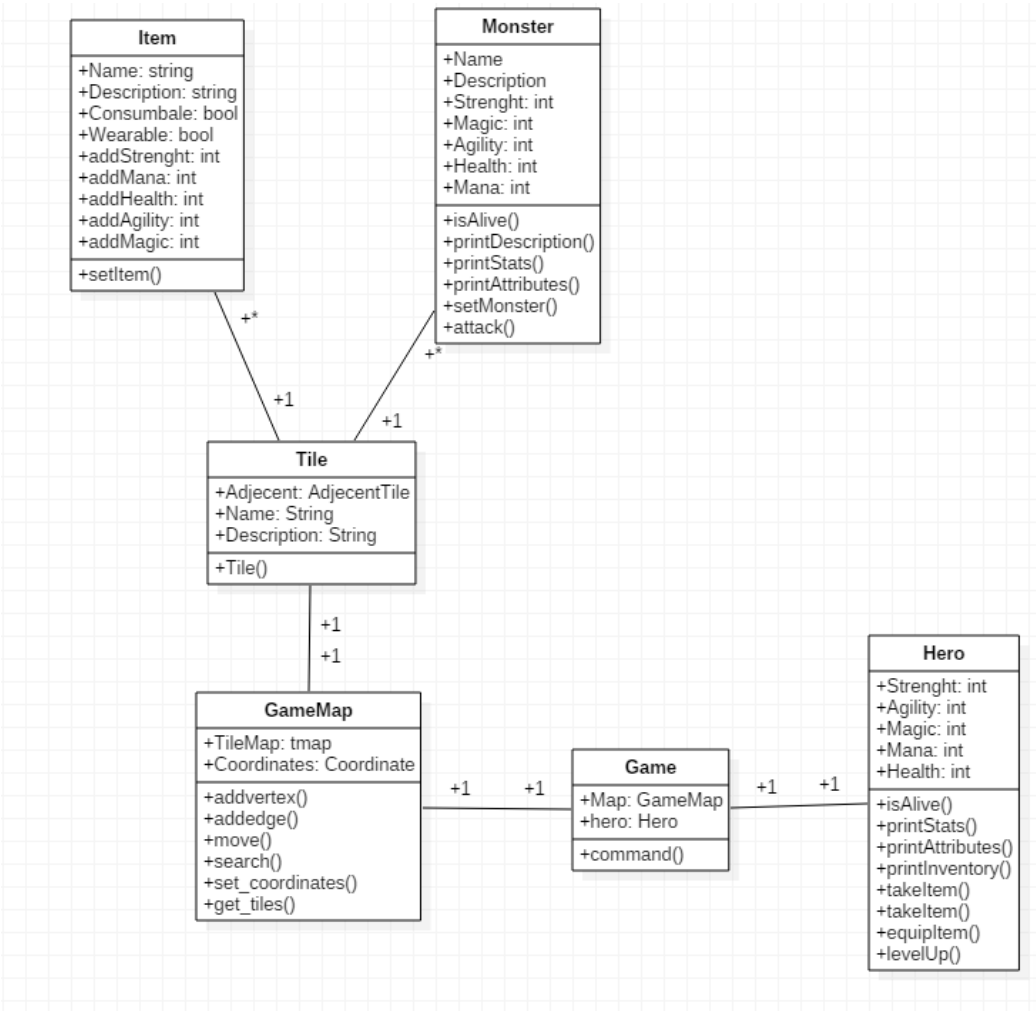
```
        {
                hero.printInventory();}

        else
        {
                Map.move(comm);
        }
}
```

3.3 UML prikaz klasa

Sve klase koje čine igru mogu se prikazati UML dijagramom. Za generiranje dijagrama koristi se *StarUML* aplikacija.

Na slici 3.2 možemo vidjeti da je entitet Game središnji entitet koji nasljeđuje svojstva *Hero* i *GameMap* entiteta. *GameMap* entitet sastoji se od entiteta tile. Tile entitet u sebi sadrži svojstva entiteta *Item* i *Monster*. Ovdje je veza jedan na više jer jedan tile može sadržavati više od jedne stvari i više od jednog protivnika.

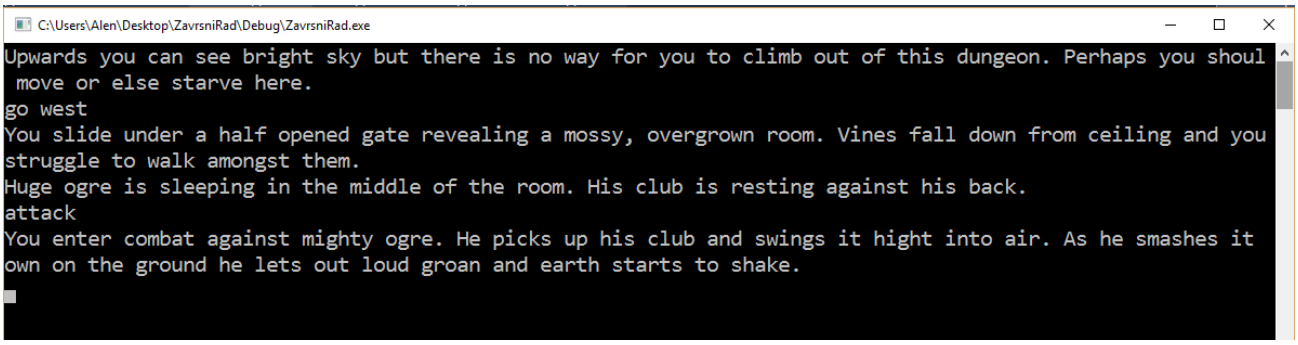


Slika 3.2

4. PRIMJER IGRANJA IGRE

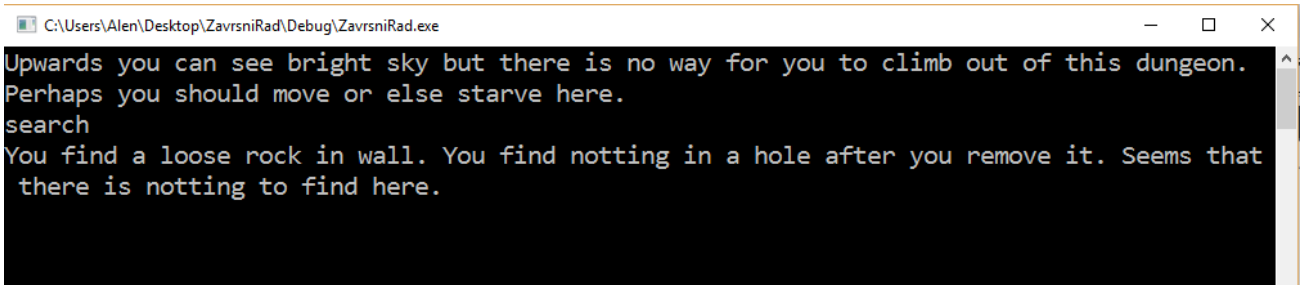
Igrač ima pet funkcijskih naredbi na raspolaganju i četiri naredbe kretanja. Funkcijske naredbe su:

- *Attack* – Igrač napada protivnika koji se nalazi na polju.



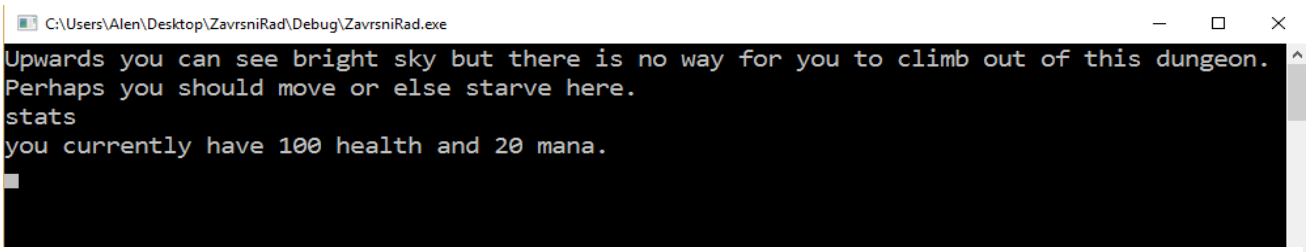
Slika 4.1

- *Search* – Igrač pretražuje polje na kojemu se trenutno nalazi. Može pronaći stvar ili napitak. Pronalaženjem stvari otvara se opcija da se stvar uzme i stavi u inventar. Pronalaženjem napitka igrač može konzumirati napitak i tako vratiti dio izgubljenog života ili mane.



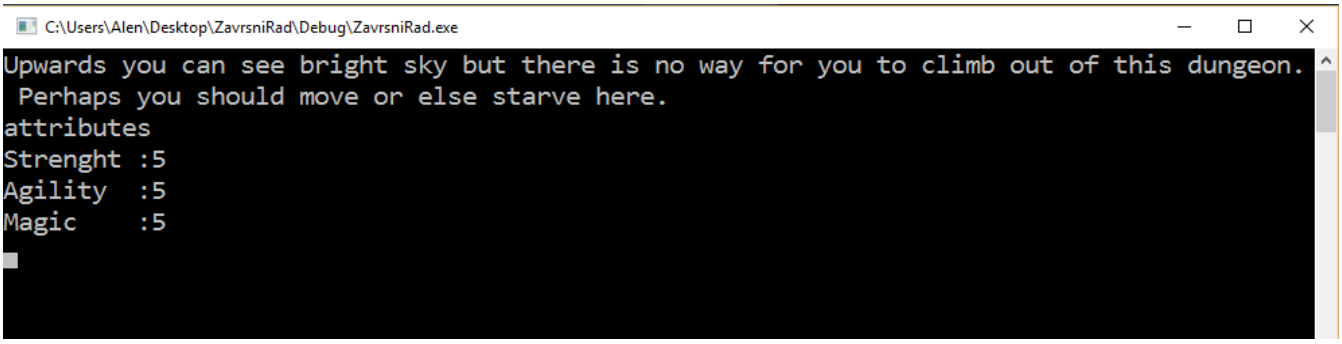
Slika 4.2

- *Stats* – Naredba *stats* ispisuje trenutno stanje igračevog života ili mane. Igračev život i mana smanjuju se unutar bitke a mogu se nadopuniti konzumiranjem različitih napitaka



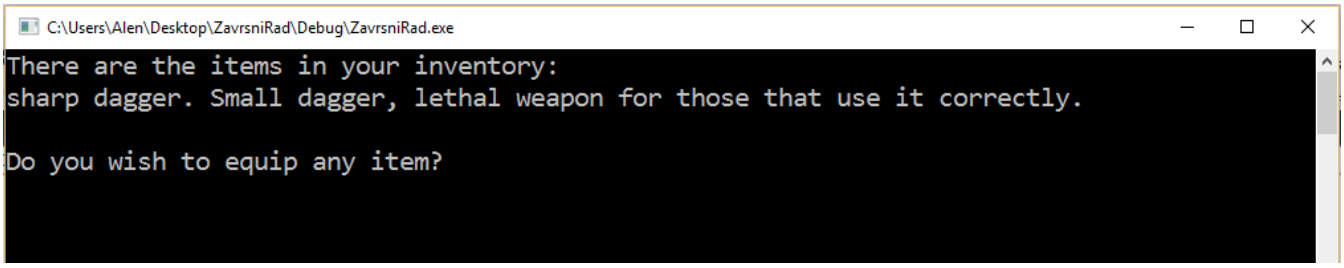
Slika 4.3

- *Attributes* – Naredba *attributes* ispisuje trenutno stanje igračeve snage, vještine i magije.



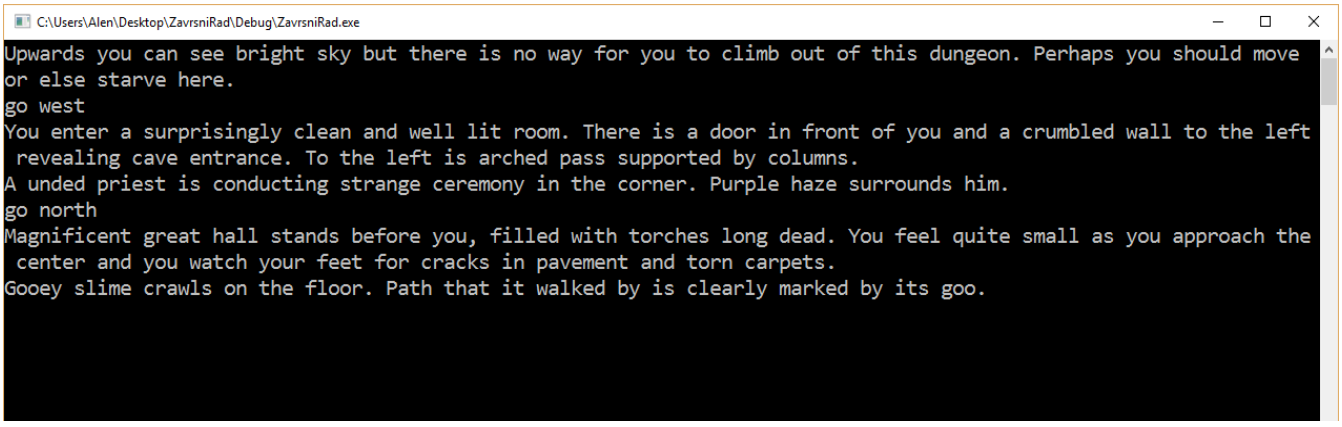
Slika 4.4

- *Inventory* – Ova naredba ispisuje sve stvari koje se trenutno nalaze u igračevom inventaru.



Slika 4.5

Također postoje i četiri naredbe kretanja. Kretanje se izvršava naredbom '*go*' te utipkavanjem smjera u kojem se želi ići. Smjerovi su '*West*', '*East*', '*South*' i '*North*'. Odlaskom u drugi smjer ispisuje se opis polja na koje je igrač došao i omogućuje mu se utipkavanje nove naredbe.



Slika 4.6

Slika 4.6 prikazuje kretanje igrača po tamnicama. Kretanjem igrač nailazi na razne protivnike koje može napasti. Ovaj primjer prikazuje dolazak na polje *'dungeon room*' gdje se nalazi protivnik *'Undead Priest*'. Kretanjem u sjevernom smjeru igrač dolazi na polje *'great hall*' gdje se nalazi protivnik '*Gooey slime'*.

5. ZAKLJUČAK

Upotrebom grafova možemo na lak i efikasan način spremiti puno podataka. Ovaj projekt prikazao je ideju upotrebe podatkovne strukture graf na području kreiranja sustava za pogon igara. Njegova fleksibilnost u implementaciji i adaptivnost pokazale su se kao ključ stvaranja ove igre. Kroz projekt se prikazao opći izgled podatkovne strukture i način na koji se takva struktura može primijeniti u brojnim drugim programskim solucijama.

Daljnjim razvijanjem aplikacije bilo bi moguće implementirati glasovne naredbe gdje bi se igrač mogao kretati i upravljati igrom pomoću glasa. Igra bi odgovorila čitanjem opisnog teksta. Ovakva igra bila bi pogodna za osobe sa invaliditetom. Također, bila bi potrebna i razrada pojedinih sustava igre, te dodavanje nekih svojstava. Sustav nivoa, korištenja magija i zadataka koje igrač mora obavljati dodali bi se u narednim verzijama igre.

LITERATURA

[1] Microsoft, An Extensive Examination of Data Structures Using C# 2.0

https://msdn.microsoft.com/en-us/library/ms379574(v=vs.80).aspx (Stranica posjećena: 06. lipnja 2016.)

[2] Stack overflow, Graph implementation c++

http://stackoverflow.com/questions/5493474/graph-implementation-c (Stranica posjećena: 06. lipnja 2016.)

[3] Stack overflow, Implementation of Adjacent Matrix with 2D vectors c++

http://stackoverflow.com/questions/16440721/implementation-of-adjacent-matrix-with-2d-vectors-c?lq=1 (Stranica posjećena: 06. lipnja 2016.)

[4] Pillars of Eternity wiki, Creatures libary

http://pillarsofeternity.gamepedia.com/Creature (Stranica posjećena: 15. lipnja 2016.)

[5] Pillars of Eternity wiki, Item libary

http://pillarsofeternity.gamepedia.com/Item (Stranica posjećena: 15. lipnja 2016.)

[6] Tutorials Point, c++ Classes and Objects

http://www.tutorialspoint.com/cplusplus/cpp_classes_objects.htm (Stranica posjećena: 02. lipnja 2016.)

[7] Stack overflow, Ideas on implementing graphs in c++

http://stackoverflow.com/questions/3853101/ideas-on-implementing-graph-in-c (Stranica posjećena: 06. lipnja 2016.)

[8] Stack overflow, Making an adjacency list in C++ for a directed graph

http://stackoverflow.com/questions/22120639/making-an-adjacency-list-in-c-for-a-directed-graph (Stranica posjećena: 06. lipnja 2016.)

SAŽETAK

DINAMIČKI GRAF

Cilj ovoga rada bio je izraditi c++ aplikaciju u obliku računalne igre koja će prikazati korištenje dinamičkoga grafa u razvoju aplikacija. Dinamički graf se postavlja kao osnova pogona za pokretanje igre, te se primjenjuje u svim aspektima i sustavima igre. Teorijski dio rada obuhvaća opis grafa kao podatkovne strukture, prikaz dinamičkog grafa, te rukovanje memorijom u takvoj podatkovnoj strukturi. Nadalje prikazuje se sama implementacija grafa unutar c++ programskoga jezika i .NET Visual Studio 2015 programske okoline. Dalje se prikazuje korištenje takovog grafa u izradi igre. Sama igra sastoji se od tekstualnog opisa igračevih avantura, te mogućnosti unošenja različitih naredbi koje kontroliraju daljnji tijek avanture. U zadnjem dijelu rada prikazan je primjer igranja igre.

Ključne riječi: dinamički graf, visual studio 2015, c++, čvor, brid

ABSTRACT

DINAMIC GRAPH

The goal of this project was to make c++ application in form of computer game that will show the use of dinamic graph in application development. Dinamic graph is set as the core of game engine, and it is used in all aspects and game mechanics. The theoretical part of this assignment includes graph description as data structure, display of dinamic graph, and handling of system memory in such data structure. Next, implementation of graph inside of c++ programming language and .NET Visual Studio is shown. Further, the use of this graph in development of game is shown. Game itself consists of text description of players adventures and ability to input different commands to control flow of players adventures. Last part of the project shows example of playing the game.

Key Words: dinamic graph, visual studio 2015, c++, vertex, edge

ŽIVOTOPIS

Alen Čamagajevac rođen je u Osijeku, 1995 godine i trenutno živu u Belišću. 2001 godine kreće u Osnovnu Školu Ivana Kukuljevića u Belišću gdje prolazi sa odličnim uspjehom. Tokom školovanja počeo se zanimati za šah, te odlazi na brojna regionalna natjecanja. Nakon završene osnovne škole upisuje opću gimnaziju u Valpovu. Postaje redovan sudionik Hrvatskog otvorenog natjecanja u informatici. Sa položenom državnom maturom iz informatike, matematike, hrvatskog jezika i engleskog jezika upisuje Preddiplomski sveučilišni studij računarstva na Elektrotehničkom fakultetu u Osijeku 2013. godine.

PRILOZI

CD

- Visual Studio 2015 projekt
- Rad u .docx i .pdf formatu