Izrada strateške igre na poteze u programskom alatu Unity

Jukica, Tomislav

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tomislav Jukica

Izrada strateške igre na poteze u programskom alatu Unity

ZAVRŠNI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tomislav Jukica

Matični broj: 44072/15–R

Studij: Informacijski sustavi

Izrada strateške igre na poteze u programskom alatu Unity

ZAVRŠNI RAD

Mentor/Mentorica:

Dr. sc. Mladen Konecki

Varaždin, srpanj 2019.

Tomislav Jukica

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj rad se sastoji od dva dijela. Praktičnog dijela gdje se izrađuje 3D strateška igra na poteze implementirana u programskom alatu Unity, te pisanog dijela gdje će se opisati sam programski alat Unity i prikazati neke od igara koje su kroz povijest definirale sam žanr. Opisati ćemo osnovne mehanike koje se nalaze u takvim igrama poput kretanja, prepreka i pucanja. Detaljno ćemo objasniti kako funkcionira A* algoritam za pronalaženje najkraćeg puta i ilustrativno prikazati njegov rad. Također ćemo pokazati i dodatan program zvan level editor koji je namijenjen izradi razina za našu igru. Naposljetku ćemo objasniti kako se sama igra izvodi ne ulazeći u sam programski kod.

Ključne riječi: Unity; strateška igra na poteze; programiranje; A* algoritam; pathfinding; C#; računalna igra

Sadržaj

1. Uvod

Strateške igre na poteze su jedne od najstarijih igara koje su implementirane na računalima. Kao prvu takvu igru implementiranu na računalima možemo smatrati Los Almos chess [1], varijantu šaha koja se igrala na 6x6 ploči bez lovaca. Ona je na računalima implementirana davne 1956. godine i tad je, zbog hardverskih ograničenja napravljena ova manja varijanta šaha bez lovaca umjesto cijele igre.

U ovom radu ćemo se baviti drugačijim strateškim igrama na poteze. Često se takvim igrama dodaje naziv *taktička* igra na poteze (TBT – turn based tactical) jer su prve igre takvog tipa obično imale mali tim vojnika koje su koristile vojne taktike za postizanje ciljeva, vrlo slično specijalnim policijskim i vojnim postrojbama koje izlaze na teren u kriznim situacijama. Teško je definirati kad je taj žanr nastao, jer su mnoge igre koristile slične mehanike, no prva takva igra u kojoj možemo prepoznati mehanike modernih TBT igara je vjerojatno *Laser Squad (1988)* Juliana Gollopa, čovjeka poznatog po tome što je stvorio XCOM serijal igara, na čemu se ovaj rad i temelji.

Ovaj žanr igara je 2012. godine doživio preporod izlaskom igre *XCOM: Enemy Unknown*, koja je te godine proglašena igrom godine, što je ohrabrilo mnoge studije da se i sami okušaju u izradi TBT igre, što je rezultiralo raznim varijacijama na ovaj žanr. U ovom radu ćemo pokušati analizirati koje su to mehanike što definiraju ovaj žanr, te što je to taktično u taktičnoj igri na poteze.

2. Metode i tehnike rada

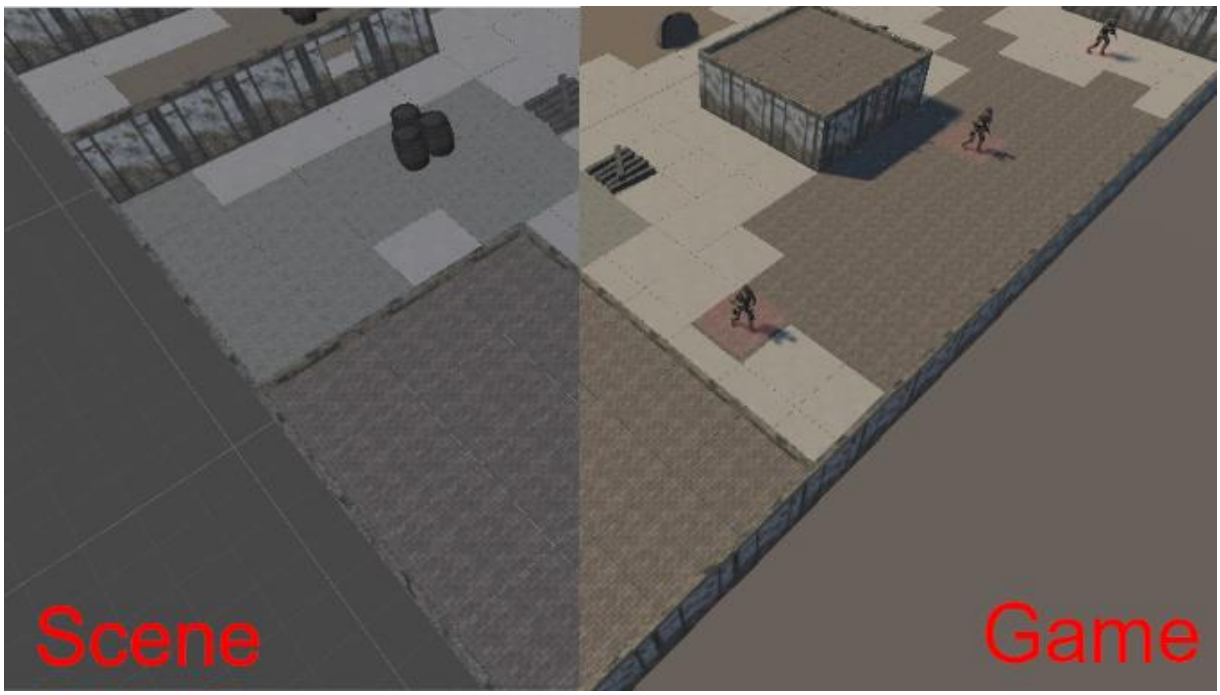
Glavni alat koji se koristio prilikom izrade završnog rada jest programski alat Unity, koji koristi C# programski jezik. Za pisanje programskog koda se koristio Microsoft Visual Studio 2017, sa ekstenzijom za Unity. Za verzioniranje koda se koristio Github. Od ostalih programskih alata koristio se Gimp, program besplatnog koda za obradu fotografija u kojem je izrađeno korisničko sučelje igre.

3. Unity

Unity je jedan od najpopularnijih i najčešće korištenih *game engine-a*, odnosno programa za izradu video igara. Preko 3 milijarde uređaja u svijetu i čak 50% igara za mobilne uređaje koriste Unity [2]. Razlog za njegovu popularnost je njihov poslovni model. Naime, Unity je besplatan za privatne korisnike te funkcionira na način da ste dužni platiti samo ako ste u prethodnoj fiskalnoj godini zaradili više od $100.000. Također, Unity na svojim službenim stranicama nudi pregršt visoko kvalitetnih tutorijala, počevši od onih za početnike koji nikada nisu ništa programirali, do naprednih za ljude koji se aktivno bave razvojem video igara. Upravo zbog toga mnogo početnika koji se žele baviti razvojem video igara odabire Unity.

Prva verzija Unity-ja je izašla 2005. godine kao ekskluzivan program za Mac računala. Danas, Unity podržava više od 25 različitih platformi. Može se koristiti za razvoj 2D i 3D igara, kao i igara virtualne ili proširene realnosti, ali i za izradu različitih simulacija. Njegovoj moći svjedoči i to što ga koriste i druge industrije poput filma, arhitekture, inženjerstva i drugih [3].

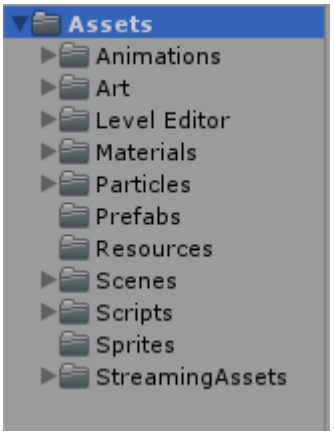
Sučelje Unity-ja je vrlo intuitivno i prilagođeno za početnike. Vrlo je slično ostalim programima za kreiranje poput onih za obradu slika ili videa. Sve se temelji na različitim prozorima koji imaju određene funkcije. Svaki projekt u Unity-ju će imati neke od osnovnih prozora kao što su *scene* i *game* prozori u kojima vidimo trenutnu scenu sa svim objektima, te izgled te scene kako će ju igrač vidjeti.



Slika 1. Usporedba scene i game pogleda [autorski rad]

Nakon toga imamo prozor hijerarhije gdje su prikazani svi objekti koji se trenutno nalaze u sceni. Prozor se zove hijerarhija jer možemo objekte grupirati u različite podgrupe. Recimo da u igri imamo šumu, a jedno stablo je jedan objekt. Sigurno ne želimo vidjeti svako stablo u našem popisu objekata jer bi to bilo poprilično nepregledno. U takvom slučaju možemo stvoriti novi prazni objekt i nazvati ga „šuma“, te staviti sva stabla da budi pod tim objektom. Osim preglednosti, taj način rada nam daje i druge pogodnosti, kao mogućnost da odjednom djelujemo na sve objekte koji se nalaze pod tim objektom.

Nadalje imamo prozor našeg projekta, gdje se nalaze svi mogući resursi koje koristimo u našem projektu. Iznimno je važno ovaj prozor držati organiziranim jer u protivnom bi mogli gubiti jako puno vremena tražeći sliku ili zvuk koji želimo koristiti jer ne znamo gdje smo ju spremili.



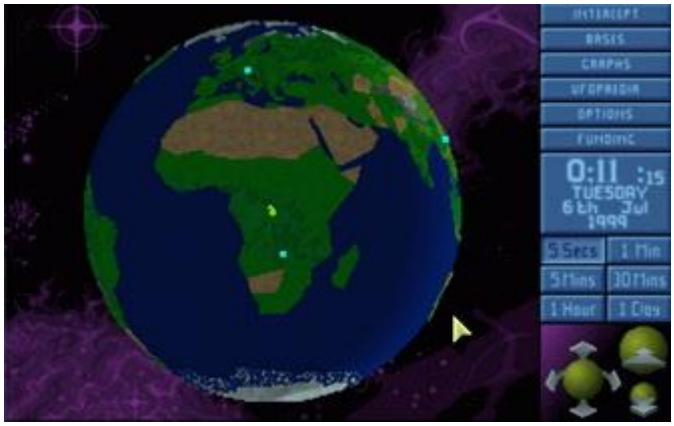
Slika 2. Prikaz projektnog prozora [autorski rad]

Uz sve to, također ćete imati i prozor inspektora. U njemu će te vidjeti sve informacije o određenom objektu, odnosno sve komponente koje su mu pridružene. Komponente su stvari kojima objektima dajemo nekakvu funkcionalnost. Jedna od osnovnih komponenti jest *transform* komponenta, koju moraju imati svi objekti u našoj sceni. Ta komponenta određuje poziciju objekta u našoj sceni koristeći 3D koordinate (0,0,0 označava centar scene). Skripte koje pišemo također spadaju pod komponente, stoga ako smo napisali skriptu za kretanje igrača potrebno ju je dodati tom objektu da bi ju on mogao koristiti.

Zadnja stvar koju je dobro imati otvorenu ukoliko sami pišete programski kod jest prozor konzole. On vam omogućuje da ispisujete različite stvari koje se dogode tijekom izvedbe igre kao i da vidite ako se dogode neke greške. Ispisivanje u konzolu je vrlo korisno u ranim vaza razvoja video igre kada još nemate korisničko sučelje. Ukoliko testirate kod za pucanje iz puške i želite provjeriti jeli meta pogođena jednostavno možete to ispisati u konzolu i onda kad znate da vam kod radi možete se baviti animiranjem te akcije.

4. Turn-based tactics

Pravi začetnik ovog žanra, kultni klasik i inspiracija za ovaj rad jest igra **UFO: Enemy Unknown** (u Americi poznata pod imenom *X-COM: UFO Defense*). Igra je izašla davne 1994. godine za MS-DOS i Amiga računala, a 1995. godine i na Playstationu. Gameplay se sastojao od dva dijela, prvog strateškog gdje smo vidjeli pregled cijele zemlje, gradili baze i reagirali na napade vanzemaljaca, sve u realnom vremenu.



Slika 3. Strateški pogled [4]

Drugi dio, onaj u kojemu će se većina igre događati, te onaj na kojem se ovaj rad temelji je taktički pogled. Nakon što na strateškom pogledu odaberemo misiju, te odaberemo i opremimo vojnike igra se prebacuje u taktički pogled na prikazu manje mape gdje sa svojim vojnicima moramo obaviti neki zadatak.

Slika 4. Taktički pogled [4]

Nekoliko faktora je presudnih za njezin uspjeh. Prvo igra je bila vrlo teška. Jedan krivi potez bi mogao značiti smrt cijele postrojbe. Nadalje, svakog vojnika se moglo uređivati, promijeniti mu ime, odabrati oružja i opremu. Sve to je pridonijelo tomu da se igrač poveže sa svojim vojnicima, čineći sve kako bi ih zaštitio. Smrt vašeg najdražeg vojnika zbog slučajne paljbe je možda frustrirajuća, no upravo zbog te nepredvidivosti tu je igru moguće igrati ponovo i ponovo i uvijek imati drugačije iskustvo.

Pet godina kasnije izlazi još jedan kultni klasik, **Jagged Alliance 2**. Ova igra veći naglasak stavlja na samu postrojbu, gdje je broj vojnika koje možete koristiti ograničen (iako ih ima puno više nego što je potrebno za vrijeme jedne igre) no svaki vojnik je poseban. Igra uzima puno elemenata iz RPG-ova (eng. *role playing games* – igre igranja uloga), pa tako svakom vojniku možete trenirati određena svojstva poput snage, spretnosti, rukovanje eksplozivima ili pak medicinu.

Slika 5. Jagged Alliance 2 [5]

Dulje vrijeme se ništa zamjetno nije događalo na tom žanru, sve do 2012. godine kada izlazi **XCOM: Enemy Unknown**. Za izlazak te igre je zaslužan dizajner Jake Solomon koji je dulji niz godina radio sa Sid Meierom, te je kako on kaže u intervjuu za GameSpot [6] cijelo to vrijeme htio raditi na njegovoj najdražoj video igri iz mladosti, no bilo je potrebno 10 godina prije nego li mu je Sid Meier dao povjerenje da dizajnira svoju igru.

Jake je pred sobom imao vrlo težak zadatak. Trebao je dizajnirati igru koja će se svidjeti fanovima koji su igrali originalni UFO: Enemy Unknown, ali i novim igračima koji nikada nisu čuli za XCOM serijal. Da stvar bude gora, ta igra je morala biti napravljena kako za PC, tako i za tadašnje konzole, koje nisu ni približno jake poput osobnih računala. Odlučio je analizirati

koji su to ključni elementi koji čine originalnu igru i to iskoristiti u novoj igri. Rezultat toga je puno jednostavnija, no ipak prepoznatljiva igra koja je budila iste osjećaje poput originala.

Slika 6. XCOM: Enemy Unknown [7]

Nakon uspjeha XCOM-a, pojavljuje se nekoliko studija koji odlučuju napraviti svoju TBT igru. Jedna od takvih je **Mutan Year Zero: Road To Eden** koja je izašla 2018. godine. Ova igra kombinira *stealth* mehanike gdje se šuljate i pokušavate ukloniti što više neprijatelja bez da vas vide, sa klasičnim bitkama na poteze.

Slika 7. Mutan Year Zero: Road To Eden [8]

5. Izrada igre

U ovom poglavlju ćemo opisati samu izradu igre, fokusirajući se na osnovne mehanike kao što su kretanje, skrivanje, te pucanje. Također ćemo obraditi i jedan dodatni dio, koji nije nužno dio same igre, ali je važan pri izgradi igre, a to je uređivač razina (eng. *level editor*), odnosno program koji će nam pomoći pri izradi scena za našu igru.

1.1. Kretanje

Kretanje u TBT igrama je relativno jednostavno. Prvo što trebamo učiniti je podijeliti našu scenu na mrežu polja, sličnu šahovskoj ploči, gdje je svako polje ploče jedna čvor. To se naravno može zakomplicirati različitim vrstama polja, zidovima, koji tehnički ne pripadaju nijednom polju nego se nalaze između dva polja i katovima gdje moramo računati i vertikalne kretnje.

1.1.1. Mreža

Princip na kojem se bazira ova igra jest taj da na početku igre generiramo scenu sa svim poljima koju zovemo mrežom, a svako polje sadrži potrebne informacije o sebi. Svoje koordinate, nalazili se što na tom polju (poput zida, prepreke ili igrača), postoji li zaštita na tom polju i u kojem je smjeru i slično. Prilikom stvaranja nove scene moramo odrediti veličinu mreže (u poljima). Recimo ako želimo da nam mreža bude velika (10,3,10), dobit ćemo nešto nalik šahovske ploče sa 3 kata, a na svakom katu imamo 100 polja. Sva ta polja bit će zapisana u 3-dimenzionaln array (polje). Zbog potencijalne veličine tog polja, imamo nekoliko funkcija kojima nam je olakšano pretraživanje tog polja poput vraćanje polja na kojem se nalazi 3d vektor ili ono što smo najčešće koristili, pronalazak polja pomoću njegovih koordinata u svijetu.

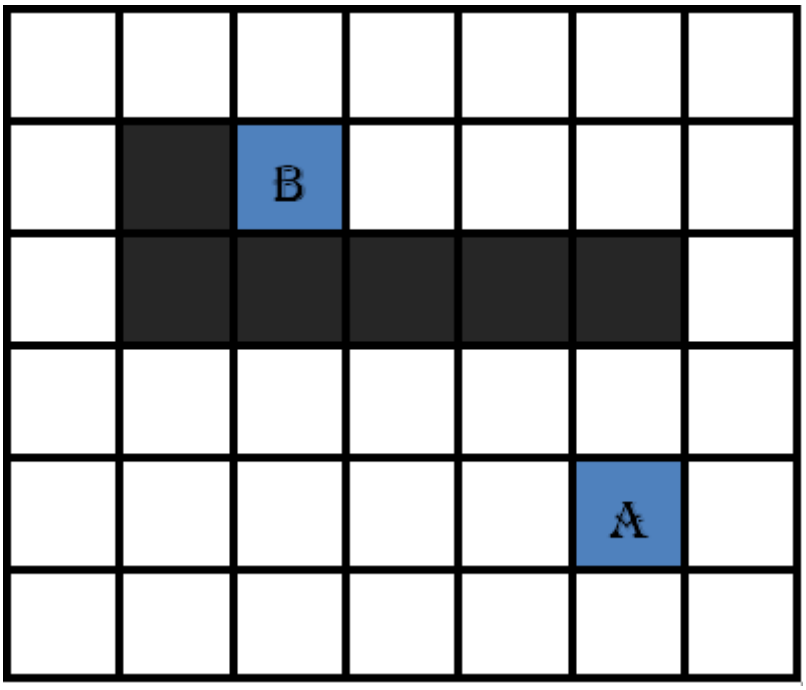
Sada kada imamo mrežu s poljima, moramo znati kako se kretati od jednog polja do drugog. Problematikom pronalaska puta od jedne do druge točke se bave algoritmi pronalaska puta ili *pathfinding* algoritmi. Postoji mnogo poznatih pathfinding algoritama, svaki sa svojim prednostima i manama, no za ovu igru smo odabrali A* algoritam.

1.1.2. A* pathfinding

Ovaj algoritam je nastao 1968. godine na Stanfordu od strane P. Harta, N. Nilssona i B. Raphaela. Na njega možemo gledati kao ekstenziju Dijkstrinog algoritma koji postiže bolje performanse zbog korištenja heuristike. [9]

1.1.2.1. Graf područja

Da bi došli s jednog mjesta na drugo, prvo što činimo jest pojednostavljujemo područje pretrage, odnosno radimo mrežu s poljima. Neovisno koliko je područje veliko, uvijek ga možemo podijeliti na polja jednake veličine. Nakon što smo podijelili područje na polja, označimo polje gdje se mi nalazimo (A), polje gdje želimo doći (B) i polja kroz koja ne možemo proći (obojena crno).



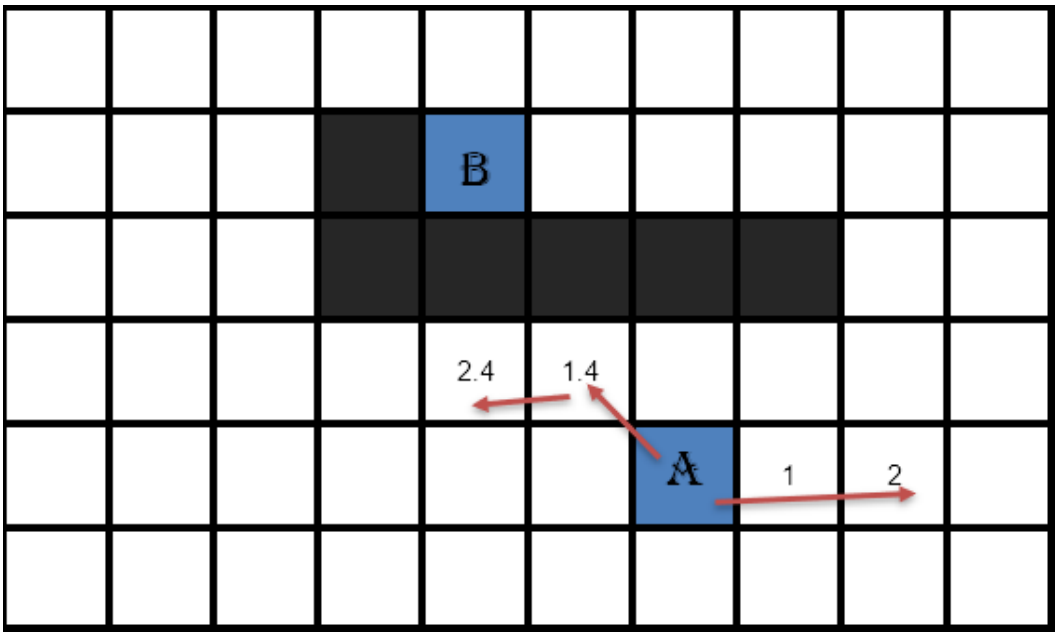
Slika 8. A* početno stanje [autorski rad]

Sada kada imamo pojednostavljeno područje, potrebne su nam dvije liste:

1. Jedna u kojoj ćemo zapisivati sva polja po kojima bi se mogli kretati u pronalasku najkraćeg puta. Zvat ćemo ju **otvorena lista** (eng. *open list*)
2. Jedna u koju ćemo zapisivati polja o kojima smo već razmišljali hoćemo li se kretati. Zvat ćemo ju **zatvorena lista** (eng. *closed list*)

Algoritam započinje tako da prvo u otvorenu listu dodamo početnu poziciju (A) i sva polja kroz koje se možemo kretati, a da graniče s poljem A. Ovdje dolazimo do prve problematike. Trebamo odlučiti koja sva to polja graniče s nekim poljem, odnosno hoćemo li uključiti dijagonalno kretanje. U našem slučaju dijagonalno kretanje je dozvoljeno, no sada dolazimo do toga da ne prijeđemo jednaku udaljenost krećući se horizontalno ili vertikalno i dijagonalno. Da bi si riješio taj problem svako polje će imati dvije cijene koje ćemo zvati **G** i **H**.

- **G** je cijena kretanja od početne pozicije A do trenutnog polja.
- **H** je procijenjena cijena kretanja od trenutnog polja do odredišnog polja B. Slovo H se koristi jer se ovdje misli na heuristiku, proces dolaženja do rješenja putem pokušaja i pogrešaka [10].

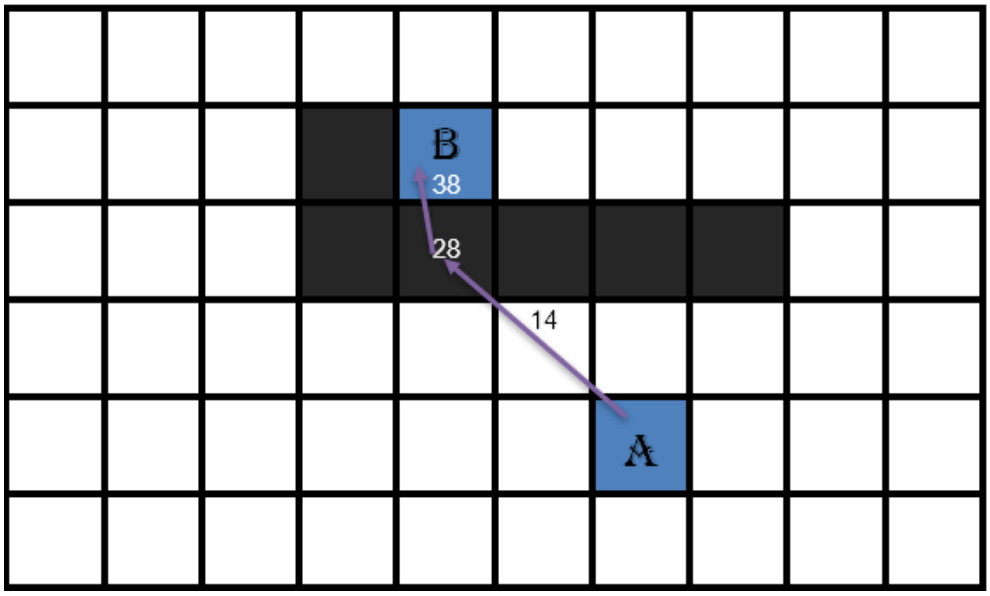


Slika 9. G cijena kretanja [autorski rad]

Na slici 9. su prikazane G cijene kretanja. Kod kretanja horizontalno (ili vertikalno) slučaj je vrlo jednostavan. Cijena za kretanje iz jednog u drugo polje je 1 te sa svakim novim poljem cijena se povećava za 1. Primijetimo da kod kretanja dijagonalno cijena je 1.4. Razlog tomu jest **Pitagorin poučak** koji kaže da je duljina hipotenuze c jednaka:

$$c = \sqrt{a^2 + b^2}$$

Uvrstimo li u tu formulu cijene za kretanje horizontalno i vertikalno dobivamo da je kretanje dijagonalno jednako $\sqrt{2}$ što je približno 1.4. Radi pojednostavljenja računanja uzeti ćemo da je cijena kretanja horizontalno i vertikalno jednaka 10, a cijena kretanja dijagonalno jednaka 14.



Slika 10. Početna najkraći H put [autorski rad]

Kod H cijene stvari se zakompliciraju. Prije smo rekli da se kod H puta radi o procjeni cijene. Razlog tomu jest što mi „ne vidimo“ sva polja te H put traži na početku najkraći mogući put. Pogledamo li sliku 10. vidjeti ćemo početnu pretpostavku za najkraći H put. Kao što vidimo taj put je nemoguć jer bi trebali proći kroz polje na kojem je prepreka. U ovom trenutku možemo započeti sa A* algoritmom [11].

Za početak uvodimo novu cijenu koju nazivamo **F**, a ona je zbroj G i H cijene. Suština A* algoritma se može pojednostavniti u nekoliko koraka koje ćemo prikazati sljedećim pseudo kodom [12].

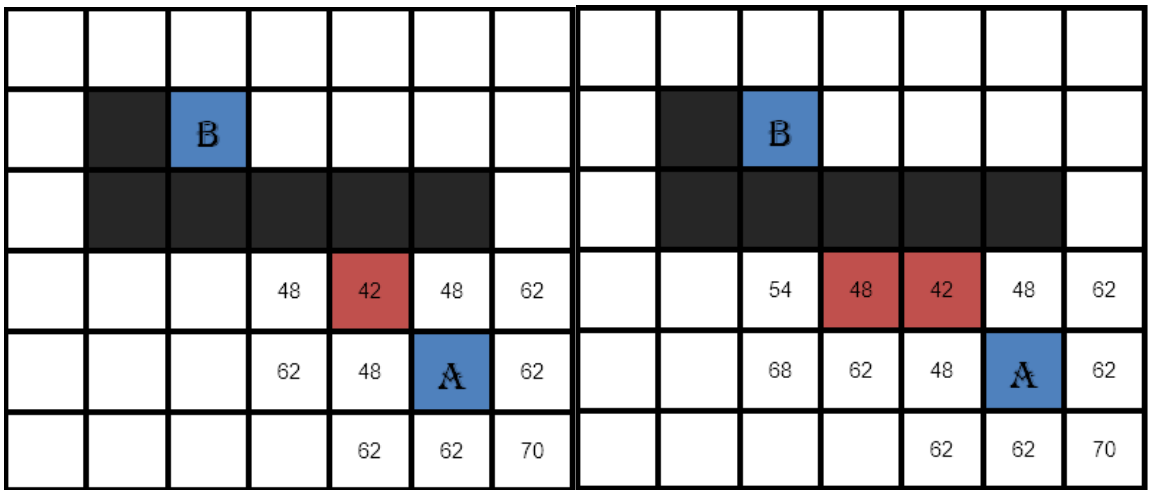
1.1.2.2. Pseudo kod

```
OPEN //otvorena lista
CLOSED //zatvorena lista
add the start node to OPEN

loop
        current = node in OPEN with the lowest f_cost
        remove current from OPEN
        add current to CLOSED

        if current is the target node //pronašli smo put
                return

        foreach neighbour of the current node
                if neighbour is not traversable or neighbour is in CLOSED
                        skip to the next neighbour

                if new path to neighbour is shorter
                  OR neighbour is not in OPEN
                        set f_cost of neighbour
                        set parent of neighbour to current
                        if neighbour is not in OPEN
                                add neighbour to OPEN
```

1.1.2.3. Ilustracija algoritma

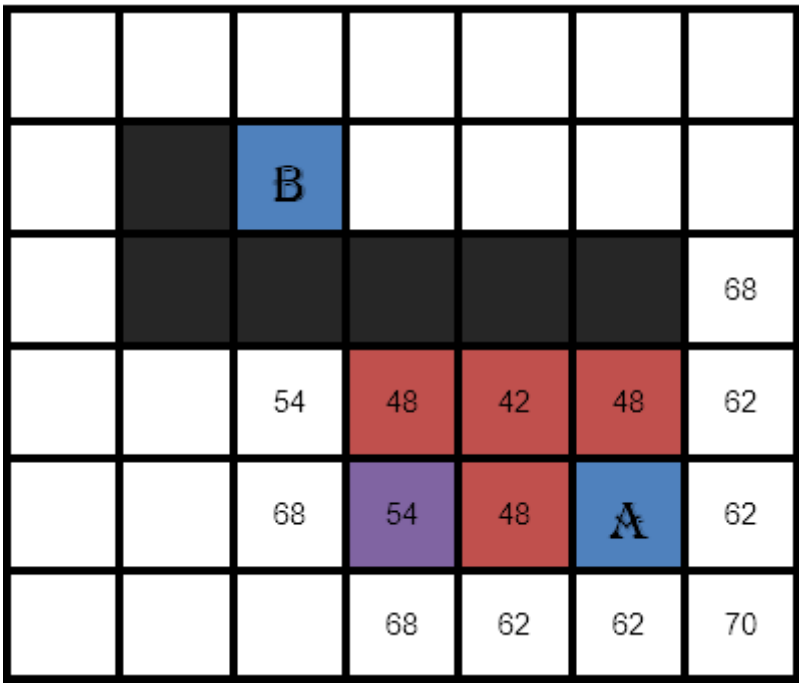
Zbog jednostavnosti prikaza, od sada ćemo prikazivati samo F cijenu polja. Pokušajmo sada pronaći put od točke A do točke B koristeći A* algoritam. Na slici 11. vidimo početno stanje sa F cijenama svih susjeda. U ovom slučaju odabiremo najmanji broj i postavljamo ga kao trenutno polje. Zatim računamo F cijene njegovih susjeda i ponovno odabiremo polje sa najmanjom cijenom. No tu dolazimo do problema. Trenutno imamo 3 polja sa cijenom 48. U

ovakvim slučajevima gledamo H cijenu polja i odabiremo polje s najmanjom. U ovom slučaju to je polje lijevo jer je ono najbliže točki B.



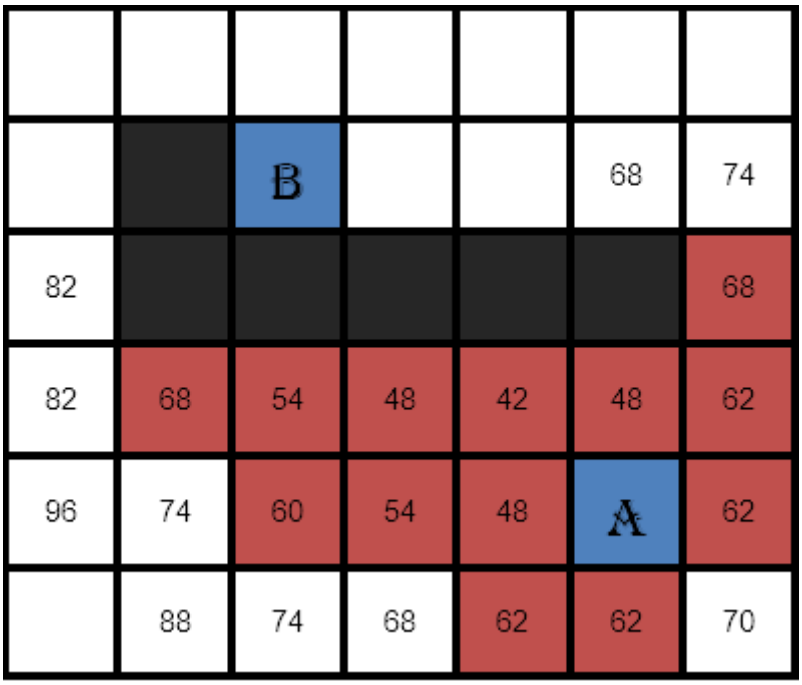
Slika 11. Ilustracija A* algoritma [autorski rad]

Sada smo došli do slučaja gdje imamo dva polja sa istom F i H cijenom. U tom slučaju odabiremo nasumično jedno od dva polja te nastavljamo dalje s algoritmom. Prolaskom kroz oba polja vidimo da nam se jedno polje (označeno ljubičastom bojom) promijenilo. Razlog tomu je jer smo pronašli brži put do njega te su se cijene promijenile.



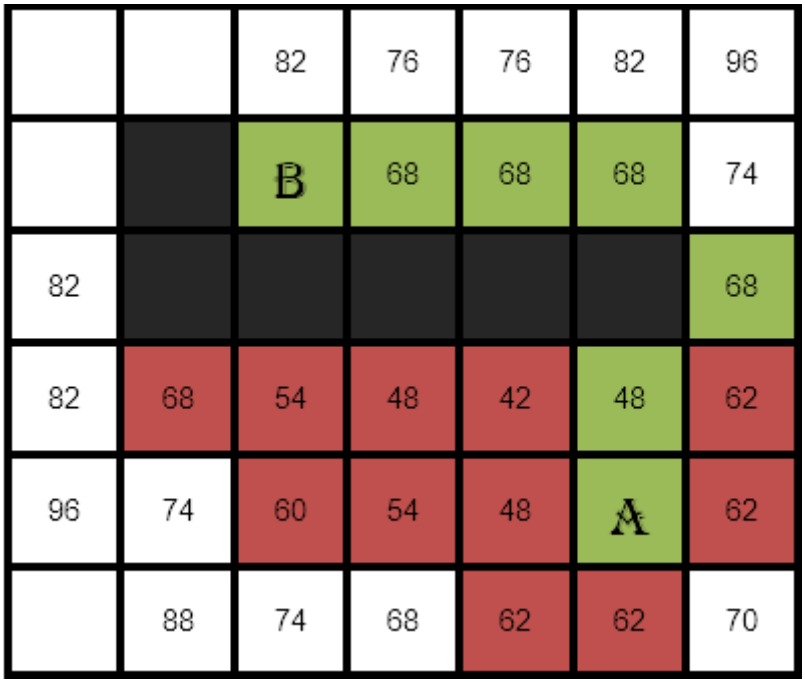
Slika 12. Promjena vrijednosti polja [autorski rad]

Nastavimo li sa algoritmom otvarajući najjeftinija polja doći ćemo do trenutka kada ćemo „zaobići“ prepreku kao na slici 13.



Slika 13. Rad A* algoritma [autorski rad]

Sada se javlja situacija gdje je svako sljedeće polje koje otvorimo ima jednaku F cijenu. Razlog tomu jest taj što se G cijena povećava, a H cijena smanjuje te se na taj način izjednačavaju. U tom trenutku algoritam je riješen te smo pronašli najkraći mogući put.



Slika 14. Rješenje A* algoritma [autorski rad]

1.1.2.4. Implementacija algoritma

Premda je složenost ovog algoritma u najgorem slučaju jednaka Dijkstrinom, u većini slučajeva će se pokazati puno bržim, iako se performanse mogu drastično promijeniti ovisno o tome koji tip heuristike ćemo koristiti (u našem slučaju koristimo ravnu crtu od početne pozicije do cilja) te izgled našeg grafa, odnosno područja pretrage.

Što se tiče implementacije algoritma u kodu, ona je relativno jednostavna. Jedini problem koji imamo jest kako optimizirati taj kod jer u testovima koji smo proveli prilikom izrade algoritma se pokazalo da je algoritmu potrebno 1-2 sekunde da izračuna područje kretanja za pojedinog vojnika (test se izvodio na i5 procesoru 8. generacije) što se možda ne čini previše no u igri u kojoj možete imati i do šest vojnika koji se mogu dva puta po potezu kretati ta brojka se penje i do 20 sekundi. Da bi riješili taj problem kod je potrebno optimizirati. Većina ljudi se odlučuje za optimizaciju pomoću sortiranja hrpom, no pošto je naša igra vrlo jednostavna i mala, za izvođenje algoritma ćemo koristiti *multithreading*.

Na početku postavimo maksimalan broj poslova (niti) koje želimo u jednom trenutku otvoriti, te stvorimo dvije liste, jedna koja će sadržavati trenutne poslove u izvođenju i drugu koja će sadržavati poslove koje imamo na čekanju. Također u kodu koristimo delegate, koji će nam javljati kad je neki posao gotov. Ukoliko izvodimo manje od maksimalnog broja poslova, a postoje poslovi na čekanju, stvaramo novu nit koja će obavljati taj posao. Trenutno jedini posao koji će obavljati jest pronalazak puta.

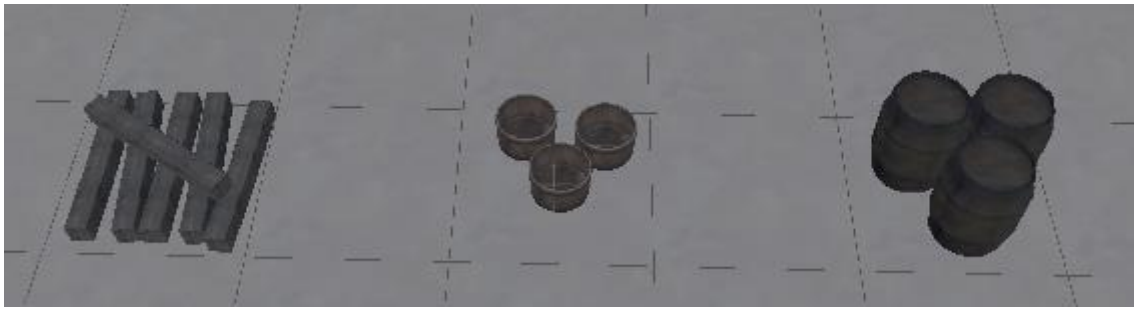
```
public int MaxJobs = 3;
        public delegate void PathfindingJobComplete(List<Node> path);
        private List<Pathfinder> currentJobs;
        private List<Pathfinder> todoJobs;
        void Start()
        {
            currentJobs = new List<Pathfinder>();
            todoJobs = new List<Pathfinder>();
        }
        void Update()
        {
            int i = 0;
            while(i < currentJobs.Count)
            {
                if(currentJobs[i].jobDone)
                {
                    currentJobs[i].NotifyComplete();
                    currentJobs.RemoveAt(i);
                }
                else
                {
                    i++;
                }
            }
            if(todoJobs.Count > 0 && currentJobs.Count < MaxJobs)
            {
                Pathfinder job = todoJobs[0];
                todoJobs.RemoveAt(0);
                currentJobs.Add(job);
                Thread jobThread = new Thread(job.FindPath);
                jobThread.Start();
            }
        }
        public    void    RequestPathfind(Node    start,    Node    target,
PathfindingJobComplete completeCallback)
        {
            Pathfinder    newJob    =    new    Pathfinder(start,    target,
completeCallback);
            todoJobs.Add(newJob);
        }
```

1.2. Pucanje

Druga bitna mehanika u ovoj igri jest odvijanje bitke. U našem slučaju to se odnosi samo na pucanje. Tijek bitke je sljedeći. Imamo dva igrača na mapi koji igraju jedan protiv drugog. Cilj je ubiti sve protivničke vojnike. Taktički dio bitke se sastoji od upotrebe prepreka za zaštitu, korištenje povišenih pozicija za bolje ciljanje, te odabira pravog oružja za svaku situaciju.

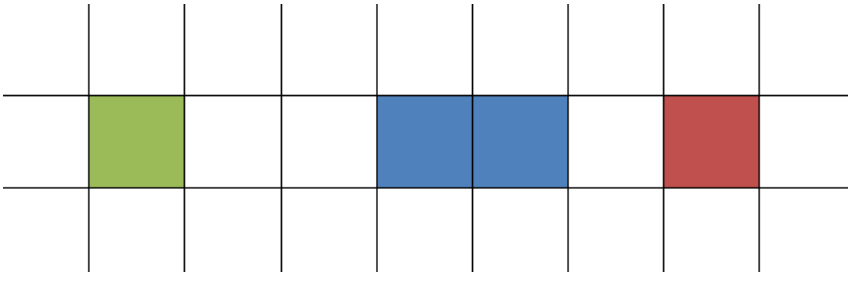
1.2.1. Prepreke

Postoje dvije vrste prepreka koje imamo u igri. Jedna je koja potpuno zaštiti igrača od neprijateljske vatre i kroz koju se ne može pucati, i druga koja nas djelomično zaštiti no iz nje možemo pucati.



Slika 15. Tipovi djelomične zaštite [autorski rad]

Zidovi također spadaju u prepreke iako ne zauzimaju polje. Kod zidova također imamo dva tipa. Puni zid koji djeluje poput potpune zaštite te se kada smo iza njega ne može pucati, te zida na kojem je prozor koji ima funkciju djelomične zaštite te se naravno kroz prozor može pucati. Kod određivanja možemo li i kolika nam je šansa za pogoditi neprijatelja koristimo informacije koje su zapisane u svakom polju. Pogledamo li sliku 16. vidimo da naš vojnik koji je prikazan zelenom bojom je udaljen 6 polja od neprijatelja koji je prikazan crvenom bojom. Ovo bi inače bio siguran pogodak, no između njih se nalaze dvije prepreke označene plavom bojom. U našoj smo igri postavili da svaka prepreka umanjuje šansu za pogotkom za 30%. Kako su između njih dvije prepreke, šansa za pogotkom se umanjuje za 60%.



Slika 16. Ilustracija bitke sa preprekama [autorski rad]

Korištenjem informacija o poljima koji su u suštini 2D i prikazom igre u 3D može dovesti do nekoliko nelogičnosti. Naime, kada gledamo informacije o polju, ono može reći da između nas i neprijatelja nema nikakva prepreka, no kada igrač pogleda vidjeti će da se zid nalazi između. Ovo je pogotovo izraženo ako se jedan vojnik nalazi na povišenoj poziciji i nije na samom rubu već je odmaknut, a drugi vojnik je u blizini zgrade kao u situaciji na slici 17. Premda između njih nema prepreka, svejedno se ne bi smjeli moći pogoditi. Da bi riješili taj problem prilikom gledanja imamo li pogled na neprijatelja (eng. *line of sight*) koristimo **raycast**. Raycast [13] je funkcija koja baca zraku od jedne točke u određenom smjeru i ukoliko ta zraka prođe kroz neki okidač funkcija će vratiti istinu što znači da na tom putu postoji neki objekt. Ova kombinacija 2D provjere preko informacija o poljima i fizičke 3D provjere sa raycastom može dovesti do drugih problema, no na ovoj razini je zadovoljavajuće. Čak i veliki studiji imaju problema s ovim, pa tako u već spomenutom XCOM serijal nerijetko možemo vidjeti kako vojnik puca kroz zid.

Slika 17. Pucanje sa povišenog mjesta [autorski rad]

1.2.2. Oružja

Oružja su bitan aspekt svake TBT igre. Zbog opsega ovog rada, u njemu ćemo simulirati tri osnovna tipa oružja koje možete pronaći u ovakvim igrama. To su oružja za blisku, srednju i daleku borbu. Prilikom računanja šanse za pogodak gledat ćemo nekoliko faktora. Vrstu oružja koju koristimo, udaljenost neprijatelja i nalaze li se prepreke između nas i naše mete.

U oružja za blisku borbu najčešće spadaju pištolji, sačmarice (eng. *Shotgun*) i ukoliko je u igri to implementirano, hladna oružja poput noževa, palica i slično. U našoj igri smo se odlučili za klasičnu sačmaricu. U našoj igri udaljenost ćemo mjeriti u poljima, tako da ćemo i

kada govorimo o oružjima koristiti polja kao veličinu za mjerenje efektivne udaljenosti pojedinog oružja.

Slika 18. Shotgun [14]

Sačmarica je relativno jednostavno oružje za rukovanje pa osnovna šansa za pogodak s njom je 95%. Njezina efektivna udaljenost jest 3 polja, što znači da sa udaljenosti većoj od 3 polja njena šansa za pogodak (eng. *Chance to hit – CtH*) opada. Prikaz šansi za pogodak u ovisnosti o udaljenosti je prikazano u tablici 1.

Tablica 1: Prikaz preciznosti sačmarice

Udaljenost u poljima	**3**	**6**	**9**
CtH	95	30	0

Što se tiče štete koju učinimo, ona ovisi o dijelu tijela kojeg smo pogodili. Tijelo možemo podijeliti na tri glavna dijela, glavu, trup i udove. Tako i u našoj igri imamo tri tipa štete koju možemo nanijeti. Pogodak u glavu će učiniti najviše štete i taj pogodak obično zovemo *critical hit*. Pogodak u trup je standardni pogodak koji čini srednju razinu štete i najlakše je pogoditi. Zadnji tip pogotka, u nogu ili ruku, čini najmanje štete i njime simuliramo kada vojnik skoro pa promaši metu. U ovoj igri to je simulirano poprilično primitivno. Naime kod sačmarice imamo samo dva tipa pogotka u trup i udove. Način na koji provjeravamo koji se pogodak dogodio jest sljedeći. Uzimamo nasumičnu vrijednost između 0 i 100. Ukoliko je ta vrijednost 90 ili više, postigli smo pogodak u glavu, a ako je vrijednost manja od 15 onda smo postigli pogodak u udove. Sve između te dvije vrijednosti je pogodak u trup. Usporedbu štete različitih oružja možemo vidjeti u tablici 4.

Za oružje srednje udaljenosti odabrali smo svjetski poznatu jurišnu pušku AK 47. Ova puška je nešto kompliciranija za rukovanje te je njena osnovna šansa za pogodak 80%. Njena efektivna udaljenost jest 6 polja, no sada postoji i minimalna udaljenost, te ukoliko smo ispod te minimalne udaljenosti gubimo na preciznosti. Detaljnije šanse za pogodak možemo vidjeti u tablici 2.

Tablica 2: Prikaz preciznosti jurišne puške

Udaljenost u poljima	**3**	**6**	**9**	**12**
CtH	50	80	50	20

Posljednje oružje koje nam je dostupno u igri jest snajper. To je oružje specijalizirano isključivo za bitku na velike udaljenosti. Osnovna šansa za pogodak jest 80%, a efektivna udaljenost jest 20 polja, s minimalnom udaljenošću od 10 polja. U tablici 3 su prikazane šanse pogotka sa različitim udaljenostima.

Tablica 3: Prikaz preciznosti snajpera

Udaljenost u poljima	**6**	**8**	**12**
CtH	20	50	80

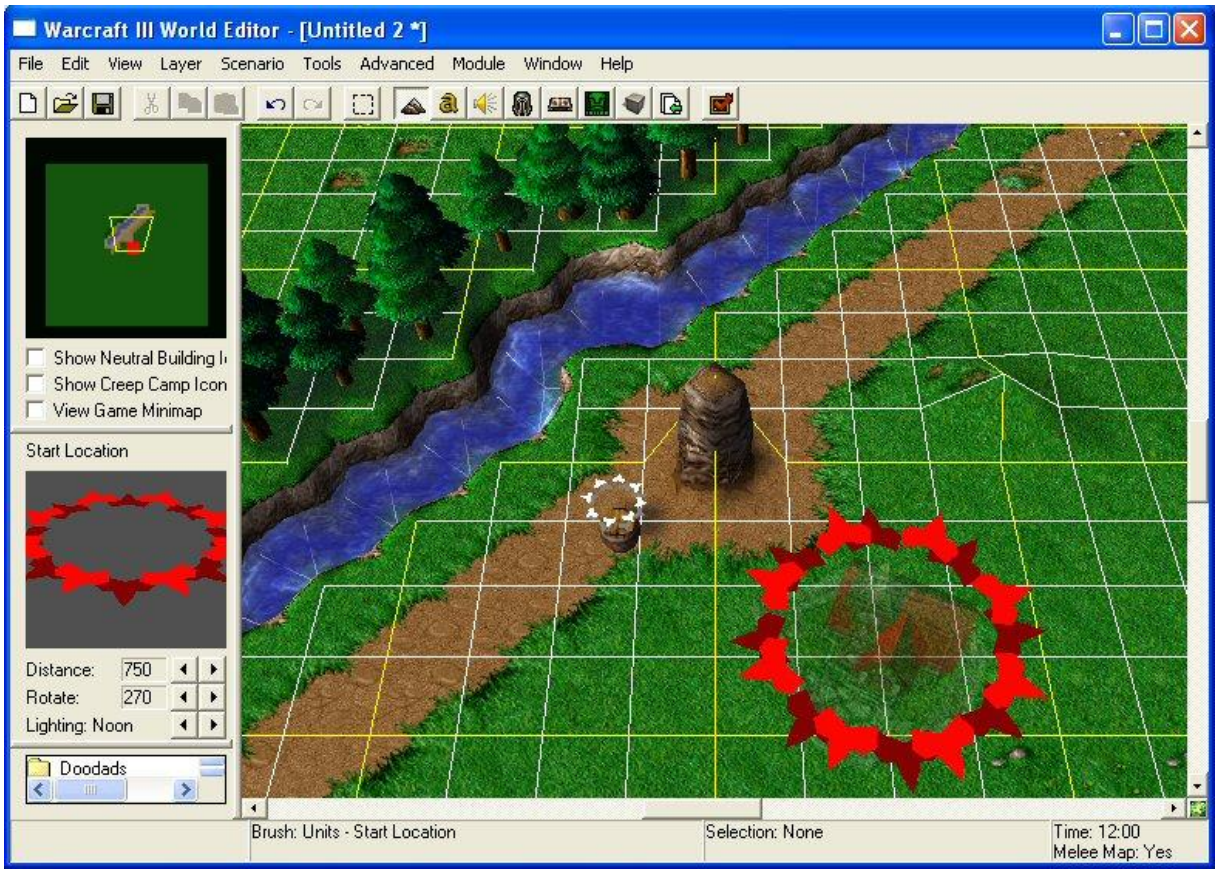
Štete koja pojedina puška učini najlakše je prikazati i usporediti tablično. Kao što smo rekli već postoje tri tipa štete, a šanse za pogodak pojedinog dijela tijela kod svih oružja je ista. Kao što vidimo iz tablice 4. najveću potencijalnu štetu učiniti ćemo sa sačmaricom no njom je najteže rukovati. Najmanju štetu radimo s jurišnom puškom, no ona ima najbolju preciznost u većini slučajeva. Korištenje određene puške u datoj situaciji jest taktički izbor igrača te naučiti koristiti sva oružja u pravim situacijama je cilj uspjeha u TBT igrama.

Tablica 4: Usporedba štete među oružjima

	Udovi	**Trup**	**Glava**
Sačmarica	3	5	5
Jurišna puška	1	3	4
Snajper	3	3	6

1.3. Level editor

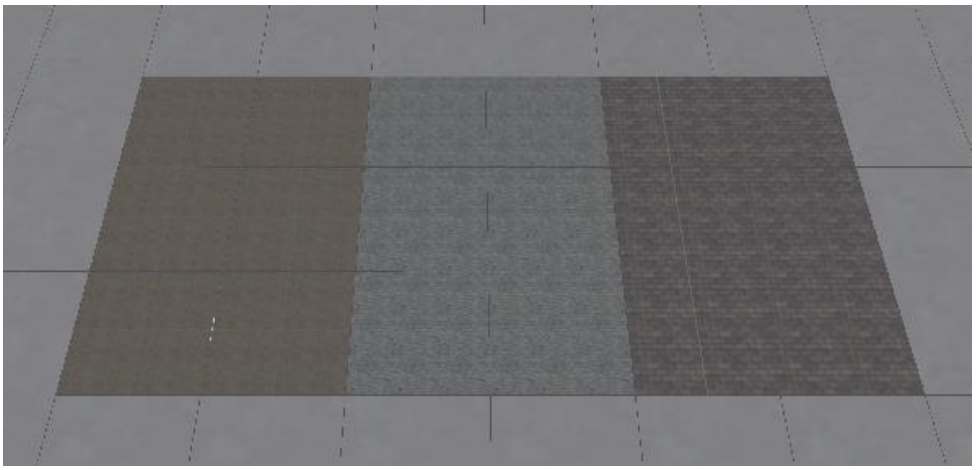
Prilikom izrade video igara programeri često naprave vlastite alate koji će im pomoći prilikom izrade igre. Jedno od najpoznatijih takvih pomagala je uređivač razina (eng. *Level editor*). To je program pomoću kojeg izrađujemo razine u našoj igri. Iako su ti programi prvenstveno namijenjeni developerima, nerijetko se zna dogoditi da se taj program ušminka i izda zajedno sa igrom za koju je rađen kao takozvani *map editor*. Vjerojatno najpoznatiji primjer map editora je onaj od studija Blizzard za igru *WarCraft III*. Naime, zahvaljujući njemu sami igrači su mogli stvarati svoje mape, levele pa čak i igre, čak i ako nisu znali ništa programirati. Najpoznatija igra koja je nastala iz takvog map editora je *Defense of the Ancients* – DotA, igra poznata kao začetnik jednog od najpopularnijih žanrova igara danas. Riječ je naravno o MOBA-a (eng. *Multiplayer online battle arena*) igrama kao što su *DotA 2*, *League of Legends*, *Smite* i *Heroes of Newerth*.



Slika 19. WarCraft III world editor [15]

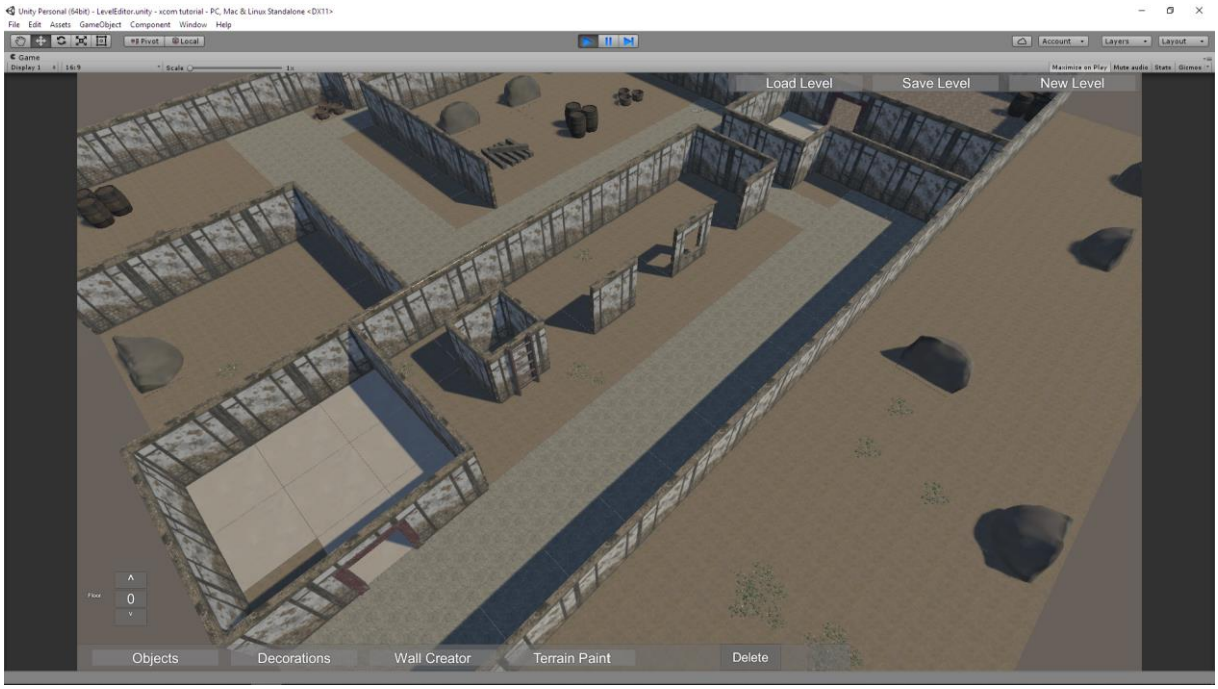
Level editor kojeg ćemo mi koristiti puno je skromnijeg obujma i namijenjen je za implementaciju u različite projekte u Unity-ju. Programski kod se može naći na internetu [16], a mi smo ga samo morali implementirati u naš projekt.

Korištenje level editora je vrlo jednostavno. Prvo i osnovno je odrediti veličinu samog levela. Nakon toga u editoru imamo nekoliko osnovnih opcija za odabir. Prvo i najosnovnije je bojanje terena (eng. *Terrain paint*). Prikaz materijala terena je vrlo bitan jer je teren najočitiji dio samog levela.



Slika 20. Različiti tereni u level editoru [autorski rad]

Sljedeća bitna stavka su zidovi. Njih je relativno jednostavno postaviti, te je moguće izgraditi prostorije kvadratnog i pravokutnog oblika. Kada smo izgradili zidove na njih je moguće postaviti tri različite stvari. Prvo su vrata, koja u suštini funkcioniraju kao rupa u zidu kroz koju se može proći. Zatim imamo prozore koji predstavljaju djelomičan zaklon od neprijatelja, te se kroz njih može prolaziti kao kroz vrata. Zadnja stvar koju možemo stavljati na zidove su ljestve koje nam omogućuju da se penjemo na katove. Nakon podova i zidova još nam samo preostaju posebni predmeti koji će djelovati kao prepreke na mapi.



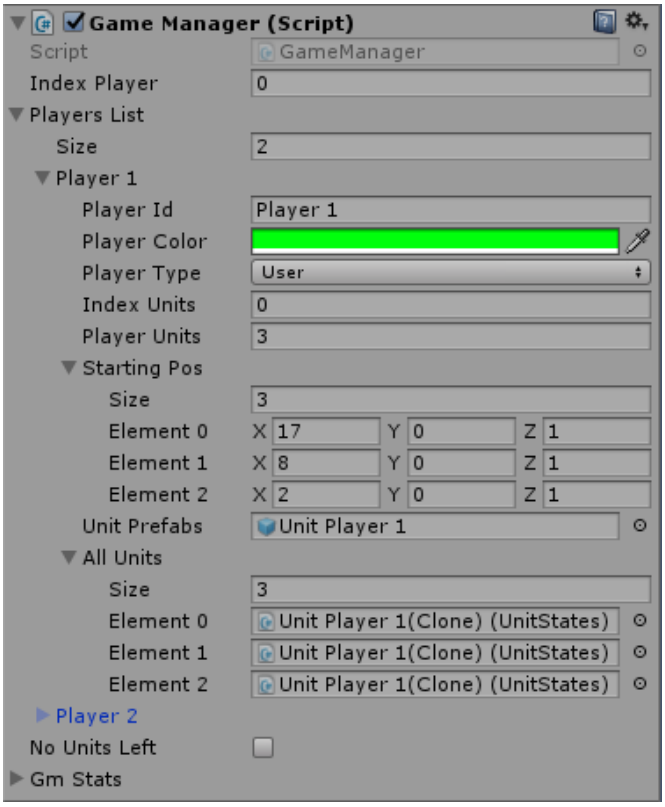
Slika 21. Primjer mape napravljene u level editoru [autorski rad]

1.4. Izvođenje igre

Kada smo u level editoru mapu izradili sve što trebamo učiniti je spremiti ju i možemo ju koristiti u igri. Način na koji spremamo mapu je povezan s načinom na koji je realizirano pokretanje igre. Kada spremimo mapu, mi u stvari zapisujemo informacije o svakom polju, koji materijal koristi, postoji li prepreka na njemu, koji sve smjerovi imaju zid, nalazili se vojnik na njemu itd. Prilikom pokretanja igre mi čitamo te informacije i po njima slažemo mapu za igranje sa željenim postavkama.

Za izvođenje igre koristimo zapravo 4 scene u Unity-ju. Prvu scenu nazivamo *„Menu“* čiji je zadatak samo prikazati različite opcije koje bi mogli imati u igri te pozvati drugu scenu koju zovemo *„Dependencies“*, odnosno ovisnosti. Ta scena na sebi ima objekt nazvan *„SessionMaster“* koji ima zadaću upravljanjem bazom podataka u kojoj se nalaze sve stvari koje možemo staviti na mapu (prepreke, dekoracije, podovi). Nakon što odlučimo pokrenuti igru pozivamo novu scenu pod nazivom *„LevelEditor“* čija je osnovna zadaća izgradnja odabranog levela. Nakon što smo uspješno izgradili level poziva se posljednja scena koju smo nazvali *„InGame“*. Ona je zadužena za izvođenje same igre i na nju ćemo se trenutno fokusirati.

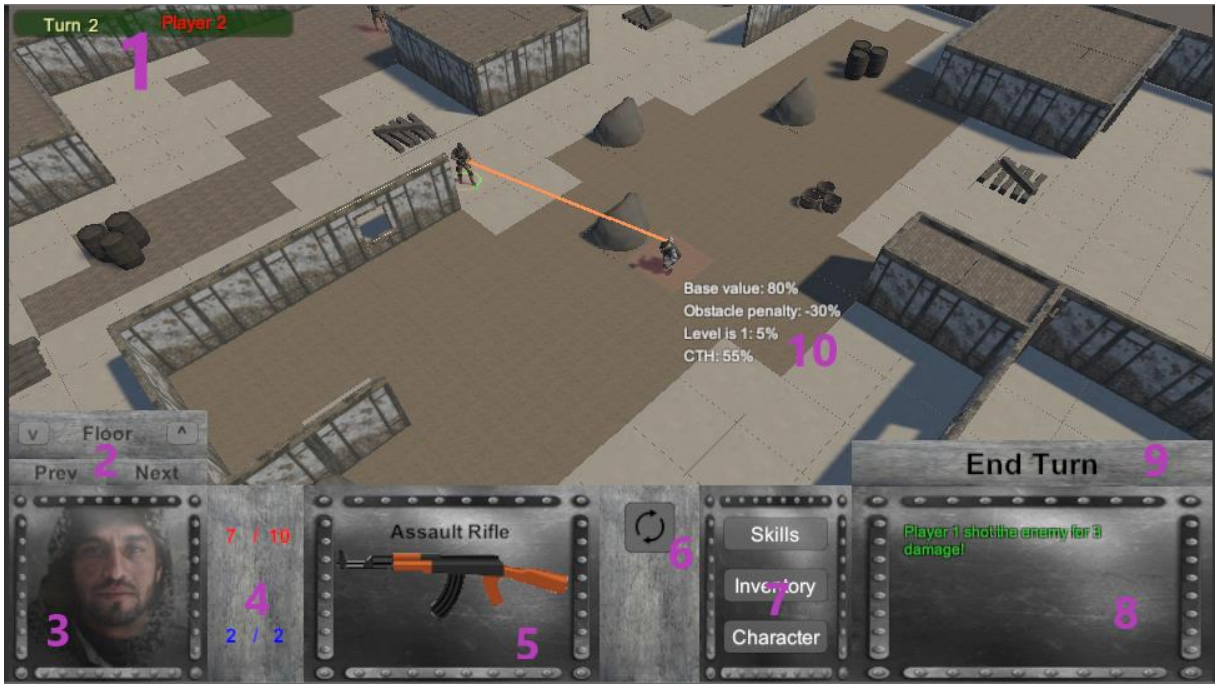
Za uspostavljanje igre podešavamo skriptu koju smo nazvali *„Game Manager“*. U njoj definiramo sve počevši od broja igrača, imena igrača, broj jedinica i njihova početna mjesta te koji prefab će se koristiti za tu jedinicu.



Slika 22. Game Manager skripta [autorski rad]

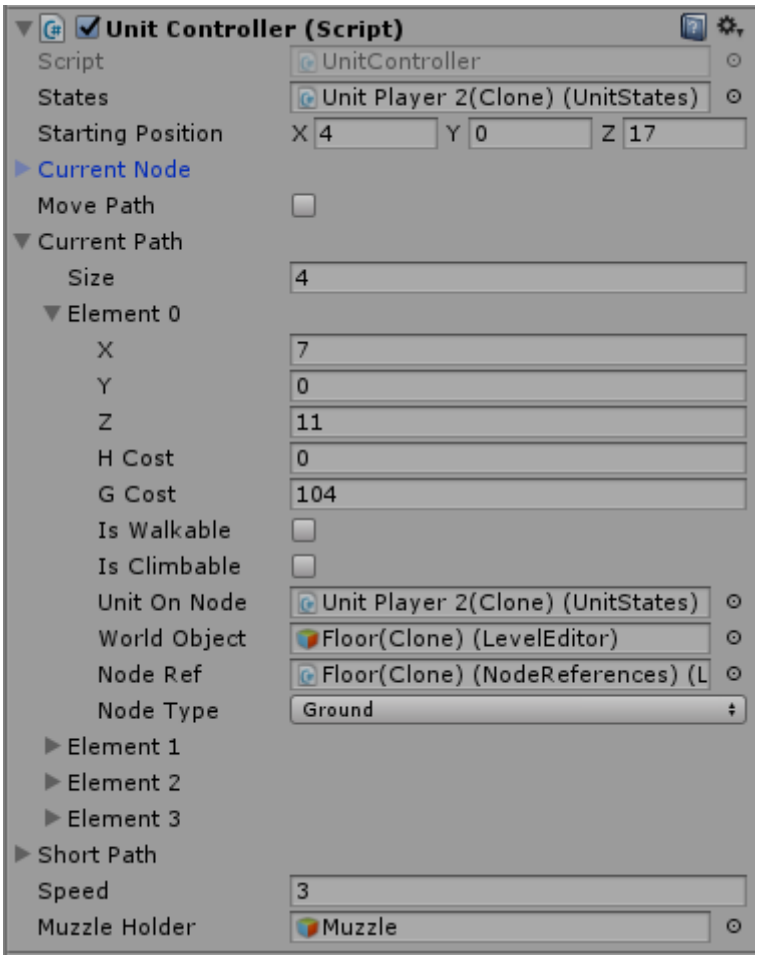
Prije nego što krenemo u samu igru pogledajmo malo korisničko sučelje (eng. *User Interface – UI*) prikazano na slici 23. Imamo nekoliko dijelova korisničkog sučelja pa krenimo redom od lijeva prema desno:

1) Imamo broj koji označava broj poteza, te pored njega ime igrača.
2) Pomoću gornja dva gumba mijenjamo razinu (kat) koji trenutno gledamo, a donja dva gumba nam omogućuju da mijenjamo trenutnog igrača.
3) Portret trenutnog vojnika da bi ih lakše razlikovali.
4) Crveni broj označava zdravlje vojnika, a plavi broj označava broj akcija koje možemo napraviti.
5) Trenutno oružje koje vojnik koristi.
6) Ovaj gumb nam omogućuje da mijenjamo oružje koje vojnik trenutno koristi, a ima ih na raspolaganju.
7) Ovi gumbovi trenutno ne rade ništa te su stavljeni kao *placeholder* za buduće funkcionalnosti koje bi se mogle naći u ovakvoj igri.
8) Ovaj prozor se naziva **combat log**. Pošto igra nema neki sofisticirani UI, jedini način da korisniku damo povratnu informaciju o tome što se događa u igri je preko ovog combat log-a.
9) Gumb kojima završavamo svoj potez.
10) Kada lebdimo mišem iznad protivnikovog igrača kojeg možemo pucati pojaviti će nam se ovakav prozor u kojemu su nam opisane naše šanse za pogodak. Ukupna šansa za pogodak prikazana je zadnjim redom CTH (eng. *Chance to hit*).



Slika 23. Korisničko sučelje [autorski rad]

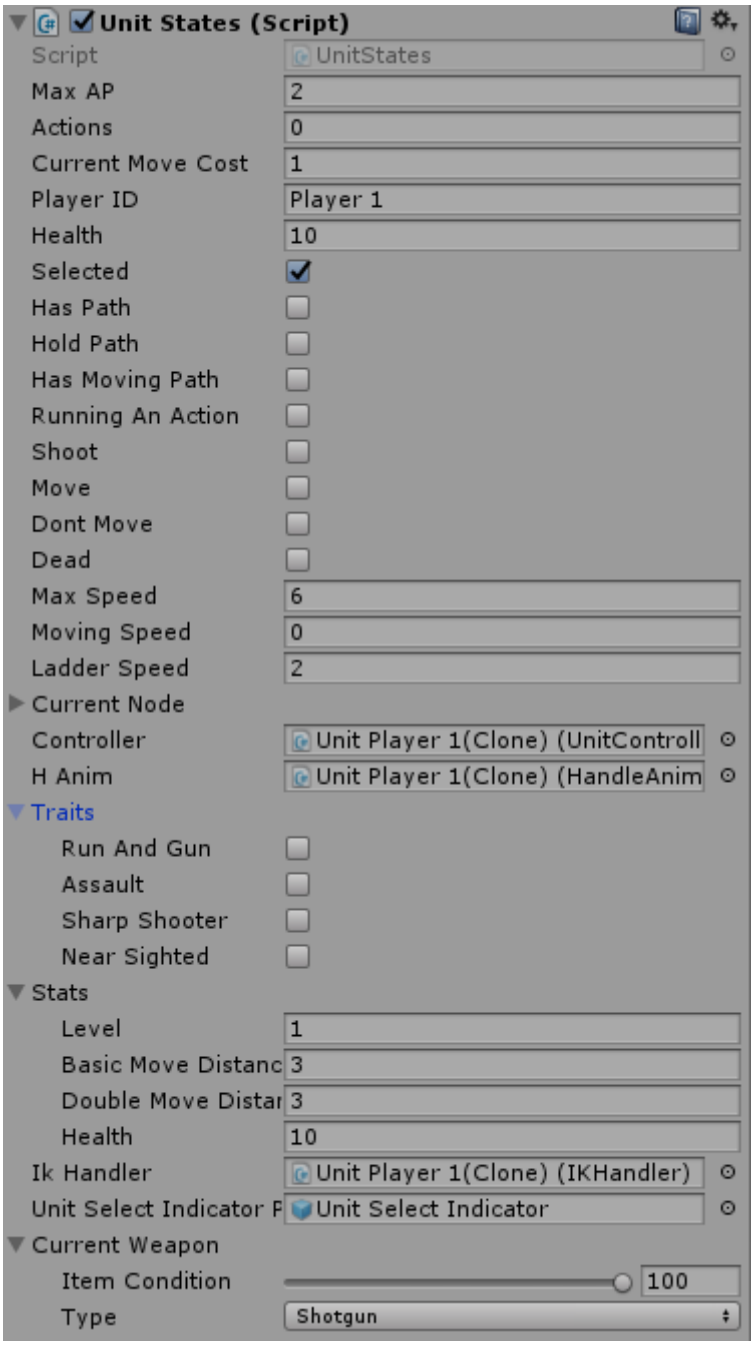
Sada kad smo pogledali korisničko sučelje, možemo se usredotočiti na samog vojnika. Postoje dvije glavne skripte koje ćemo pogledati. Prvu smo nazvali *„Unit Controller“* i ona je zadužena za kretanje vojnika. Ukoliko odredimo po kojim se poljima želimo kretati, ona će se spremiti u listu *current path*, te će se za svako polje zapisati njegove koordinate, H i G cijena, možemo li se kretati ili penjati na tom polju, nalazili se tko na njemu, te reference za izgled samog polja.



Slika 24. Unit Controller skripta [autorski rad]

Sljedeću skriptu bitnu za vojnika smo nazvali *„Unit States“*. Kao što samo ime govori ona sadrži informacije o različitim stanjima u kojima se vojnik može naći, kao sve ostale informacije o toj jedinici. Za početak vidimo koliko trenutno imamo akcija za napraviti kao i koliko je maksimalno akcija koje taj vojnik može napraviti. U našoj igri to su dvije akcije za sve jedinice koje se mogu iskoristiti za samo kretanje, kretanje i pucanje ili samo pucanje. Nadalje vidimo koliko put kojim se želimo kretati košta akcija (u igri je to vizualizirano zelenim putem za jednu akciju i narančastim putem za sve akcije) te kojem igraču ta jedinica pripada. Također vidimo i koliko zdravlja trenutna jedinica ima. Zatim imamo nekoliko *boolean* varijabli koje nam govore u kojem se stanju nalazi trenutna jedinica. Prvo imamo oznaku jeli trenutna jedinica označena odnosno aktivna. Zatim imamo nekoliko varijabli koje nam govore jeli trenutna

jedinica u pokretu. Posljednje dvije varijable nam govore puca li trenutno jedinica i jeli mrtva. Ove varijable nam većinom služe da bi zabranili igraču da čini neku akciju dok animacija za jednu ne završi. Nakon toga imamo par varijabli kojima definiramo brzinu kretanja vojnika, po zemlji i po ljestvama. To je direktno povezano sa brzinom izvođenja animacije. Postoji i nekoliko varijabli koje nam dodaju različite značajke vojniku (eng. *Traits*) kao što su *„run and gun“* kod kojeg ne dobivamo manju šansu za pogodak ako smo se kretali prije pucanja.



Slika 25. Unit States skripta [autorski rad]

6. Zaključak

Cijeli rad bio je temeljen na TBT žanru igara na poteze, izravno kopirajući mehanike iz igre *XCOM: Enemy Unknown*. Originalna zamisao je bila napraviti igru s puno većim opsegom i mehanikama koji dodaju taktičku dubinu igri. No prilikom izrade same igre sam shvatio da je takvo nešto vrlo teško izvesti zbog nekoliko razloga. Kao prvo, osnovne mehanike koje su ovdje implementirane su poprilično komplicirane same po sebi jer ima puno dijelova koji su u međusobnoj interakciji što često dovodi do mnogih *bugova*. Nadalje, za igru ovakvog opsega nužno je imati dizajnera koji će unaprijed dizajnirati svaku mehaniku koja će biti uključena u igru, prije nego li uopće počnemo s razvojem. Razlog tomu je taj što imamo toliko sustava koji međusobno moraju djelovati, te promjena jednog će vjerojatno značiti promjenu i na ostalima. Da bi se to izbjeglo, potrebno je na to računati od samog početka, jer htjeti implementirati neku funkcionalnost nakon što smo završili osnovne mehanike bit će vrlo teško, te će oduzeti puno više vremena nego da smo na to računali od početka.

Premda je ovo prvi put da koristim Unity, njega je bilo relativno jednostavno savladati te nije predstavljao prevelik problem. Puno veći problem prilikom izrade igre su bili neki od naprednih koncepata C# programskog jezika, koje do sada nikada nisam susreo. Da bi se osoba bavila ovakvim projektom trebala bi imati nekoliko godina iskustva u radu sa C# programskim jezikom, da bi mogla uspješno bez velikih problema napredovati.

Premda je finalna verzija rada manjeg opsega nego što je to originalno zamišljeno, svejedno sam zadovoljan sa postignutim, pogotovo uzevši u obzir da razvoj ovakvih projekata zna trajati i do 4 godine. Shvatio sam da je najvažnija osoba u izradi video igre njen dizajner, te je to posao kojim bih se htio pokušati baviti u budućnosti.

7. Popis literature

[1] H. Bodlaender, „Los Alamos Chess“, 2002. [Na internetu]. Dostupno: https://www.chessvariants.com/small.dir/losalamos.html [pristupano 27.08.2019.].

[2] Unity (bez dat.) Public Relations [Na internetu]. Dostupno: https://unity3d.com/public-relations [pristupano 28.08.2019.].

[3] „Unity (game engine)“, (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Unity_(game_engine) [pristupano 28.08.2019.].

[4] „UFO: Enemy Unknown“, (bez dat.). U Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/UFO:_Enemy_Unknown [pristupano 29.08.2019.].

[5] Jagged Alliance 2 [Slika] (bez dat.). Dostupno: https://store.steampowered.com/app/215930/Jagged_Alliance_2__Wildfire/ [pristupano 29.08.2019.]

[6] GameSpot, (08.10.2015.) „How XCOM Came Back From The Dead“, *YouTube* [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=HvBL1AKYpe8 [pristupano 29.08.2019.].

[7] M. Klappenbach, „Best X-COM Video Games“, 2019. [Na internetu]. Dostupno: https://www.lifewire.com/xcom-series-games-812468 [pristupano 29.08.2019.].

[8] Mutant Year Zero: Road To Eden [Slika] (bez dat.). Dostupno: https://www.mutantyearzero.com/#1 [pristupano 29.08.2019.].

[9] „A* search algorithm“, (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/A*_search_algorithm [pristupano 31.08.2019.].

[10] „Heuristika“, (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: http://www.enciklopedija.hr/natuknica.aspx?id=25317 [pristupano 31.08.2019.].

[11] R. Wenderlich, „Introduction to A* Pathfinding“, 2011. [Na internetu]. Dostupno: https://www.raywenderlich.com/3016-introduction-to-a-pathfinding#toc-anchor-001 [pristupano 31.08.2019.].

[12] S. Lague, (16.12.2014.) „A* Pathfinding (E01: algorithm explanation)“, *YouTube* [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=-L-WgKMFuhE [pristupano 31.08.2019.].

[13] Unity (bez dat.) Physics.Raycast [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html [pristupano 02.09.2019.].

[14] Shotgun Clipart Vertical [Slika] (bez dat.) Dostupno: https://www.pinclipart.com/pindetail/ihwTJhx_shotgun-clipart-vertical-over-under-shotgun-png-transparent/ [pristupano 02.09.2019.].

[15] WarCraft III World Editor [Slika] (bez dat.) Dostupno: https://www.hiveworkshop.com/threads/a-beginners-guide-to-map-making.8204/ [pristupano 03.09.2019.].

[16] Sharp Accent, (2016.) „Tactical Turn Based Game“, *YouTube* [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=feLyjkYsU-U&list=PL1bPKmY0c-wmfc6k6V7M_K7RRQftSXTaU&index=16 [pristupano 03.09.2019.].

8. Popis slika

9. Popis tablica