University of Ljubljana

Univerza v Ljubljani

Fakulteta za Računalništvo in Informatiko

Tržaška 25, Ljubljana

Taktična umetna inteligenca v realnem času

Damir Arh
Uroš Čibej
Matija Jekovec
Gregor Leban
Mitja Luštrek
Martin Žnidaršič

Ljubljana, 15.6.2001

Cilji seminarske naloge

Naš cilj je bil napisati strategijo v realnem času, torej igro, pri kateri bi imel igralec na voljo neko skupino vojakov v nekem okolju. Naloga igralca je, da s svojimi vojaki uniči nasprotnika prej kot nasprotnik uniči njegove vojake (potrebno je pobiti vse kar ni tvoje).

Igra se odvija na dvorazsežni karti, sestavljene iz prehodnih in neprehodnih kvadratkov. Neprehodni kvadratki hkrati vojakom tudi zastirajo pogled, medtem ko ga prehodni ne. Kljub prisotnosti kvadratkov na karti se vojaki po njej gibljejo zvezno. Tudi igra se odvija brez potez, v zveznem času, kar simuliramo z izračuni v kratkih časovnih korakih. Potek igre se ravna po »realnih« fizikalnih zakonih. Oba igralca poznata celotno karto vnaprej, tako da jima je ni potrebno spoznavati, ne vesta pa za položaje nasprotnikovih vojakov. Te vidita le takrat, ko so v vidnem polju enega izmed njunih vojakov.

Poleg tega, da je bilo potrebno napisati okolje, v katerem se igra dogaja, nas je zanimala predvsem umetna inteligenca, ki smo si jo zamislili tako, da v bistvu računalnik ne nadomešča igralca, ampak ga samo razbremenjuje. Igralec lahko organizira vojake v skupine, ki jih pozneje lahko obravnava kot zaključene celote, ne da bi se ukvarjal s premikanjem posameznega vojaka, ali pa pazil na vsak njegov korak v neposrednem spopadu. Skupina mora biti torej dovolj avtonomna, da se zna sama odločati glede svoje poti do nekega cilja in pa da zna dobro reagirati v nepredvidljivih situacijah na taki poti (npr. napad nasprotnika).

Struktura programa

Glede na to, da je bilo zamišljeno, da bi to igro igralo več igralcev eden proti drugemu, je tudi program razbit na dve ločeni aplikaciji, in sicer na aplikacijo ki teče na strežniku, in na aplikacijo ki teče na odjemalcu. Med njima poteka mrežna komunikacija

Komunikacija med odjemalci in strežnikom

Modul za komunikacijo skrbi za to, da se prenašajo paketi med različnimi računalniki, ki sodelujejo v igri. Vsak odjemalec ima vzpostavljeno dvosmerno povezavo s strežnikom. Odjemalec pošilja strežniku ukaze, ki jih generira uporabnik, strežnik pa pošilja odjemalcu informacije o položajih vseh vojakov.

Glavne funkcije:

- za strežnik in odjemalce:
  `bool Initialize(conn_type type)`
  inicializacija, kjer povemo, ali bomo odjemalec ali strežnik
  `void Finish()`
  zapremo vtičnice
  `int Send(char *buff, int size)`
  pošljemo paket dolžine size
  `int Receive(int clientID, void **buff, int *size)`
  če funkcija IsNewDataAvailable vrne true, potem nam ta funkcija nastavi kazalec na prejete podatke in njihovo dolžino
  `bool IsNewDataAvailable()`
  funkcija vrne true če je sprejet že cel paket podatkov

- funkcije za strežnik:
  `bool Listen()`
  strežnik gre v stanje, kjer ga lahko pokličejo odjemalci

- funkcije za odjemalce:
  `bool Call(char *address)`
  pokličemo strežnik
  `void StartGame()`
  ukaz za začetek igre

Glavni zanki (kar se tiče komunikacije) na strežniku in odjemalcu:

Dokler funkcija IsNewDataAvailable() vrača true počnemo naslednje:
- Kličemo funkcijo Receive, ki nam nastavi kazalec na prejete podatke.
- Glede na tip paketa (CLIENT_ID, START, MAP, DATA) podatke ustrezno uporabimo.
- Kličemo funkcijo DisposeBuffer(), ki poskrbi za to, da se podatki ustrezno sprostijo.

Naloge odjemalca

Odjemalec skrbi za grafični prikaz dogajanja (ta je natančneje opisan na koncu poročila), uporabniški vmesnik (kreiranje skupin, pošiljanje vojakov na določen cilj, označevanje nevarnih območij, nastavljanje agresivnosti vojakom) in pa komunikacija s strežnikom, torej pošiljanje paketov z ukazi uporabnika in sprejemanje novih stanj vojakov.

Uporabniški vmesnik predstavlja večji del izvorne kode odjemalca. *Console* je glavni razred, ki predstavlja odjemalca, skrbi za koordinacijo med več prikazovalnimi okni in rokovanje z dogodki. *Viewport* je razred, ki skrbi za prikaz dogajanja in pretvarjanje med notranjimi koordinatami in koordinatami zaslona. Vsak razred *Console* lahko vsebuje več med seboj neodvisnih objektov *Viewport* tako, da vsak prikazuje svoj kos dogajanja na zemljevidu. Razred *CommandPanel* služi prikazu trenutnih nastavitev (način za dodajanje obračalnih točk, dodajanje območij nevarnosti...). Vsak razred *Console* vsebuje en objekt *CommandPanel*, ki mu določi pozicijo in velikost.

Za nizkonivojsko podporo smo uporabili knjižnico SDL, ki poleg grafičnega prikaza omogoča tudi rokovanje z dogodki. Za uporabo je dokaj enostavna, vprašljiva pa je njena zmogljivost (zmanjšana vprid prenosljivosti) in primitivnost ukazov (odsotnost prikazovanja kompleksnejših likov, fontov, višjenivojskih funkcij za risanje in zapolnjevanje...).

Naloge strežnika

Na strežniku se izvajajo vsi izračuni (fizika, iskanje poti, hierarhična umetna inteligenca) za vse igralce. Po opravljenem izračunu v enem časovnem intervalu se nova stanja pošlje odjemalcem, ki nato ta nova stanja izrišejo. Seveda mora strežnik tudi prevzeti ukaze od odjemalcev, in jih ustrezno servisirati.

Glavno vlogo na strežniku ima razred *Environment*, ki vsebuje vse ostale razrede (sezname vojakov in skupin, zasebne podatke za vsakega odjemalca posebej, razrede za pomožne izračune). V vsakem časovnem koraku se izvede metoda *update* tega razreda, ki kliče metode *update* skupin in vojakov (poskrbita za izvedbo rutin umetne inteligence inakcije vojakov) ter na koncu preveri veljavnost fizikalnih omejitev (odkrivanje trkov in simulacija vida).

Fizika

Fizikalni model skrbi za odkrivanje trkov ter uveljavljanje fizikalnih omejitev (ustreznih pospeškov in hitrosti). Obsega tudi rutine za računalniški vid.

Lastnosti vojakov

Velik del zmogljivega fizikalnega modela, vključenega v naš program, je neposredno vezanega na posamezne vojake. Sem štejejo vse omejitve, ki izhajajo iz naravnih omejitev vojakov oz. ljudi.

Tako se vojaki lahko gibljejo na tri različne načine: lahko hodijo normalno, hodijo sklonjeno ali pa se plazijo. V naštetem vrstnem redu seveda pada največja možna hitrost za posamezni način gibanja, hkrati pa seveda tudi izpostavljenost in vidnost vojakov. Poleg načina gibanja na hitrost vplivata tudi zdravje vojaka (ranjeni vojaki se gibljejo počasneje) in smer pogleda. Ta je že tako omejena glede na smer gibanja (vojak seveda ne more glave obrniti za 180° glede na svoje gibanje), poleg tega se s povečanjem kota med smerjo pogleda in smerjo gibanja zmanjšuje največja hitrost. S povečevanjem hitrosti gibanja se zmanjšuje tudi vidno polje oz. zorni kot vojaka.

Vojaki znajo tudi streljati na tri različne načine, ki se med seboj razlikujejo po natančnosti in času merjenja (za bolj natančen strel je potrebno daljše merjenje). Poleg tega je natančnost strela odvisna tudi od zdravja vojakov, hitrosti gibanja le-teh in razlike med smerjo pogleda in smerjo gibanja. Sama smer strela je normalno porazdeljena, kar pomeni, da natančnost strela v bistvu predstavlja verjetnost točnega zadetka, ki je tako delno odvisen tudi od sreče.

Zdravje vojakov se zmanjšuje, ko jih zadene strel in ko se zaletijo v kako oviro. V prvem primeru seveda bistveno bolj kot v drugem.

Odkrivanje trkov

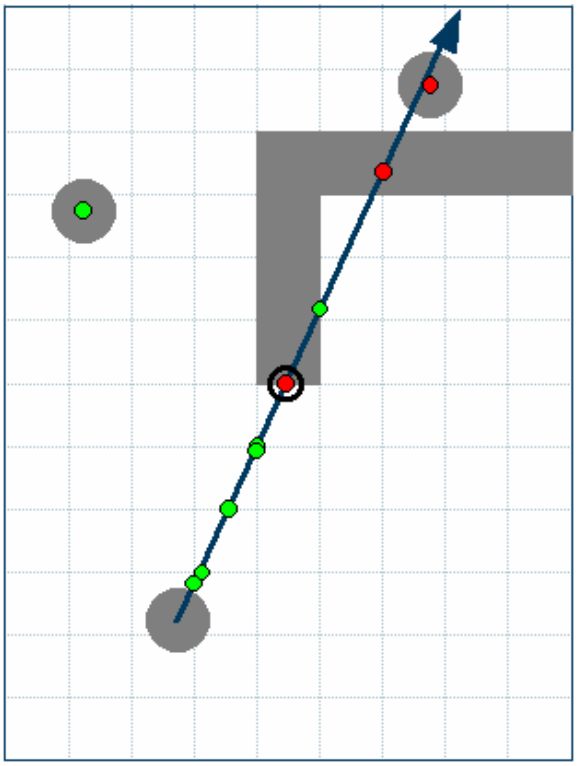
Preostali del fizikalnega modela je vključen v razred *Environment*. Sem šteje vse tisto, kar ni neposredno vezano na posamezne objekte v igri, torej interakcija med njimi. V našem primeru gre samo za odkrivanje trkov med njimi ter simuliranje pogleda vojakov. Za obe nalogi smo uspeli izkoristiti kar isti algoritem, ki ga bom opisal v naslednjih vrsticah.

Odkrivanje trkov med objekti smo razdelili v dve skupini. Prvo tvorijo trki med več vojaki ter trki med vojaki in stenami. Ker se vojaki gibljejo precej počasi glede na dolžino časovne rezine, zgolj preverjamo končni položaj vojaka po vsakem koraku. Če je ta v položaju trka z nekim drugim objektom, oba objekta prestavimo na stari položaj in ju ustavimo.

Odkrivanje trkov izstrelkov ni tako enostavno. Ti namreč v eni časovni rezini prepotujejo že kar precejšnjo razdaljo, tako da smo morali uporabiti zanje predstavitev v obliki žarka, ki predstavlja pot, ki jo izstrelek prepotuje v časovni rezini. Odkrivanje trkov tega žarka z drugimi objekti (vojaki in stenami) smo izvedli v treh korakih. V prvem koraku preverjamo trke žarka z vodoravnimi robovi sten. Za vsak vodoravni prehod v nov kvadratek preverimo, ali je v tem kvadratku stena. Če je temu tako zabeležimo trk, sicer nadaljujemo s preiskovanjem pri naslednjem prehodu. V drugem koraku podobno naredimo z navpičnimi prehodi. V zadnjem koraku preverimo še trk žarka z vsakim od vojakov. Med vsemi odkritimi trki poiščemo tistega, katerega razdalja od izhodišča žarka je najmanjša in tega zabeležimo kot dejanski trk, saj se je edini v resnici zgodil.

Za vid vojakov ravno tako uporabljamo algoritem za odkrivanje trkov žarkov, uporabimo pa žarek od vojaka, ki gleda do vojaka, za katerega preverja, ali ga vidi. Če je prvi trk tega žarka gledani vojak, ga vidi, sicer pa ne. Seveda preverimo še, ali je smer tega žarka v obsegu vidnega polja vojakov. Na podoben način pri gradnji vidljivostnega grafa si lahko z algoritmom pomagamo podobno, če izračun opravimo za žarek, ki povezuje dve vozlišči.

Na sliki so za dani primer z obarvanimi krogci prikazani vsi preverjeni kandidati za trke. Z zeleno barvo so obarvani tisti, za katere se je izkazalo, da niso trki, z rdečo pa so obarvani dejanski trki. Med temi trki je bil na koncu izbran tisti, ki je na sliki dodadatno obkrožen.

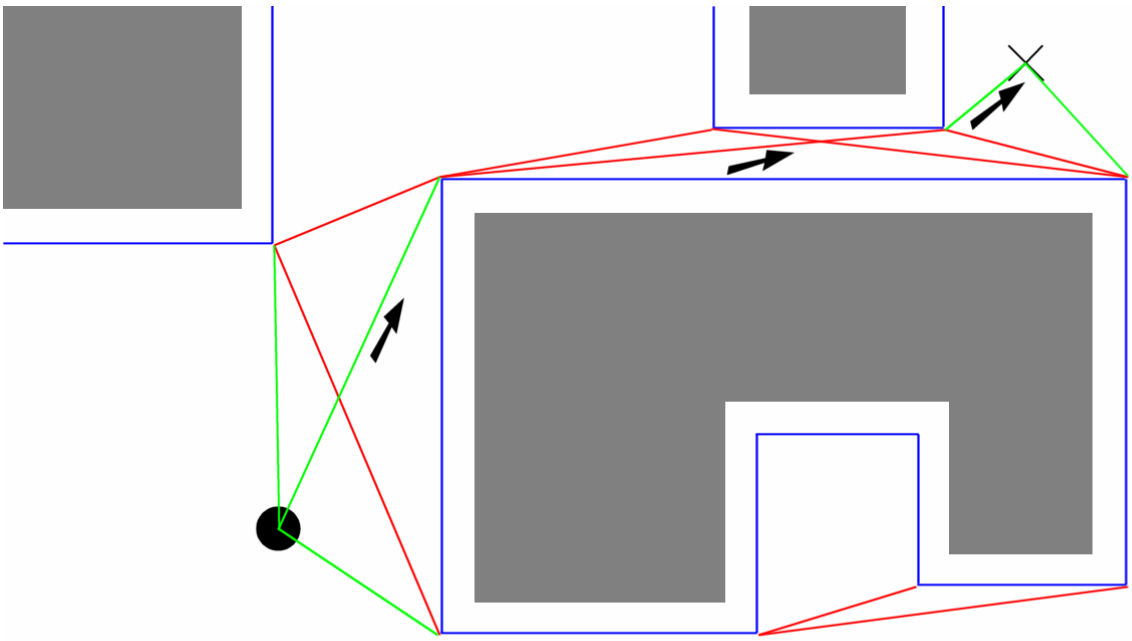


slika 1: odkrivanje trkov med žarkom in okolico

Iskanje poti

Algoritem za iskanje poti se kliče ob vsakem uporabnikovem zahtevku, naj se skupina premakne na določen cilj, in pa tudi lokalno v skupini, da vojaki najdejo pot do lokalnih ciljev, ki jim jih določi skupina.

Pri iskanju poti smo si pomagali z vidljivostnim grafom. Tega naredimo tako, da najprej razširimo stene za konstanto, ki mora biti vsaj enaka polmeru vojaka (*modre črte*) – ta razširitev pomeni, da se središče vojaka ne sme približati steni za več kot to konstanto. Vojaka nato zmanjšamo v njegovo središče. Lahko bi najbrž razširili stene samo v dveh smereh (vsaj za dvakratni vojakov polmer) in vojaka skrčili v točko v njegovem 'vogalu', a smo se za prvo rešitev odločili, ker je bolj intuitivna. Za vozlišča grafa nato vzamemo konveksne vogale, ker mora vsaka najkrajša pot med dvema točkama teči mimo njih. Te povežemo, če se med seboj vidijo, kar pomeni, da je med dvema ravna pot brez ovir (*rdeče črte*) – kako to počnemo, je podrobneje opisano v poglavju o fiziki. Vse to se postori na začetku igre.



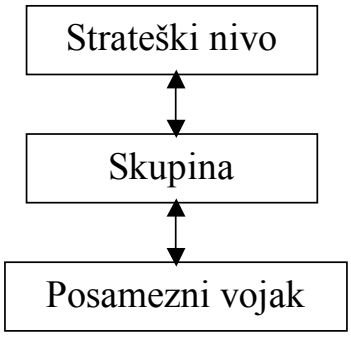
slika 2: vidljivostni graf

Ko je med igro potrebno poiskati pot med dvema točkama, ju dodamo v graf (*krog in križec*) in vzpostavimo ustrezne povezave (*zelene črte*). Nato z A* poiščemo najkrajšo pot med njima (za osnovno hevristiko smo vzeli zračno razdaljo med točkami, uporabljamo pa tudi tako, ki razdaljo poveča s številom, ki pove, kolikšen del kroga, ki predstavlja nevarno območje, pot odseka in kolikšen ta krog je). Napisali smo tudi algoritem A*, ki upošteva zgolj točke znotraj danega pravokotnika, za potrebe iskanja poti posameznih vojakov (kaj to pomeni, piše v poglavju o premikanju skupin). Na koncu dodani točki odstranimo iz grafa.

Umetna inteligenca

Umetno inteligenco smo si zamislili kot strukturo s tremi nivoji, in sicer:

- Na najnižjem nivoju se ukvarjamo samo s posameznim vojakom in umetna inteligenca skrbi, da se vojak uspešno izogiba sosednjim vojakom in oviram, hkrati pa poskuša ostati blizu svoji skupini in pa tudi svoji začrtani poti (princip »boidov«).
- En nivo više je skupina, ki skrbi, da vojake znotraj nje pripelje do cilja, ki ga je določil igralec. Vojakom določa neke lokalne cilje glede na varnost območja, razporeja pa jih tudi tako, da se ne ovirajo med seboj.
- Na najvišjem nivoju umetne inteligence se nahaja končni avtomat, ki pa skrbi za to, da skupina dobro reagira glede na okoliščine, ki se pojavijo. Te okoliščine predstavljajo vrednosti določenih atributov in glede na te vrednosti avtomat prehaja iz stanja v stanje. Vsakemu stanju so dodeljene tudi akcije, ki jih mora skupina izpolniti, ko se avtomat nahaja v tistem stanju.



slika 3: hierarhična umetna inteligenca

Prvotna zamisel je bila, da bi realizirali umetno inteligenco samo na dveh nivojih (brez nivoja, ki ga sedaj predstavlja skupina), vendar se je to izkazalo kot neuspešen poskus. Razlog za to se skriva v preveliki lokalnosti najnižjega nivoja (»boids«), zato smo potrebovali še en nivo, ki uporablja veliko več informacij iz same mape in zato veliko lažje »pametno« razporedi vojake.

Boids

Princip, ki smo ga uporabili na najnižjem nivoju umetne inteligence, izhaja iz simulacije obnašanja živalskih skupin. Princip deluje tako, da vsak osebek gleda svojo okolico, in glede na stanje te okolice prilagaja svojo hitrost in smer gibanja. Pri živalskih skupinah je zaželeno, da se gibljejo dokaj strnjeno in da lahko ob neki nepredvidljivi situaciji hitro spremenijo smer in hitrost (delovati morajo homogeno).

Pri simulaciji takega vedenja upoštevamo tri pravila (uvedel jih je Craig Reynolds):

1. Vojak se poskuša premakniti proti središču sosednjih vojakov.
2. Vojak poskuša ohraniti neko razdaljo do bližnjih objektov (vojakov, ovir).
3. Poskušamo izenačiti smer in hitrost sosednjih vojakov.

Vsako tako pravilo nam da svoj vektor, ki ga z določenimi utežmi prištejemo trenutnemu vektorju premikanja. Če pa želimo vojaka pripeljati še proti nekemu cilju, mu moramo prišteti vektor (utežen vektor) smeri, proti kateri naj bi se premikal.

Prvotna ideja naše umetne inteligence je predvidevala dodajanje podobnih pravil, s katerimi bi dosegli dobro obnašanje vojakov.

Izkaže se, da celo ta tri pravila delujejo zelo slabo v okoljih, kjer so se znašli naši vojaki. Zelo slabo so se obnašali med zapletenimi ovirami in nastavljanje določenih uteži njihovega vedenja ni popravilo ali pa ga je popravilo samo za določen primer, nikakor pa ni bilo mogoče nastaviti uteži tako, da bi se vojaki obnašali v vseh primerih vsaj zadovoljivo. Tako smo opustili idejo o dodajanju novih pravil, in uvedli nov nivo (skupino). Še vedno pa se »boidi« uporabljajo na najnižjem nivoju – za premikanje vojaka do lokalnih ciljev (predvsem za izogibanje oviram in drugim vojakom).

Grupe

Grupe so vmesni člen med posameznimi vojaki in taktično umetno inteligenco. Skrbijo za to, da se umetni inteligenci ni treba ukvarjati s posameznimi vojaki. Grupa ima za vsakega vojaka, ki ji pripada, instanco razreda CSoldierItem. V tem razredu so podatki o vojakovem naslednjem waypointu, o vseh njegovih točkah, ki naj bi jih obiskal, o njegovem stanju (hodi, miruje) ter seznam že obiskanih točk.

Glavne funkcije:

```
void update();
```
funkcija, ki se kliče vsakih n milisekund in opravi naslednje:
- odstrani mrtve vojake
- sprehodi se najprej po seznamu vseh vojakov, ki so trenutno v gibanju in jim izračuna nove koordinate
- če je že minilo dovolj korakov odkar smo poslali v gibanje zadnjega vojaka izvede:
    - GetFreeSoldier - izmed vseh mirujočih vojakov izbere tistega, ki je najbolj oddaljen od naslednjega waypointa in "ni v sredini grupe"
    - GetSafePosition - izmed vseh varnih pozicij glede na stanje grupe vrne vektor, kamor naj gre soldier

```
void addSoldiers (List * list);
```
grupi priredi vojake, ki so v seznamu list

```
void addWayPoints (List *way);
```
grupi nastavi globalno pot, po kateri naj bi vojaki približno hodili

```
Vector getCurrentWayPoint ();
```
Vrne vektor, ki pove h kateri točki vojaki trenutno težijo

```
void attachAutomaton (Automaton *aut1);
```
poskrbi za to, da imajo lahko različne grupe različno taktiko

```
bool BeenThere (int x, int y, int soldier);
```
pogledamo, če je bil vojak soldier že v okolici točke (x,y)

```
int GetProgression (Vector pos, Vector dest, Vector wp);
int Evaluate(int safety, int avail, int dist, int progress);
int GetAvailability(int i, int j);
bool FreeArea (Vector pos, int area);
```
funkcije, ki vplivajo na izbiro naslednjega waypointa posameznega vojaka

```
Vector GetSafePosition (int index);
```
funkcija, ki ustrezno uteži vrednosti zgornjih funkcij in izmed možnih lokacij izbere najboljšo

```
int GetFreeSoldier();
```
vrne index vojaka, ki
- je najbolj oddaljen od naslednjega waypointa (da vojaki ne bodo zaostajali)
- ni v sredini grupe (da se ne bodo prerivali)
- miruje

```
void GetNextCoords(int index);
```
izračuna naslednjo lokacijo vojaka z indeksom index

Kaj se zgodi, ko uporabnik nastavi grupi nov cilj?

Grupa dobi nov seznam točk, ki naj bi jih vojaki obiskali. Ta seznam je generiran z algoritmom A* in predstavlja najkrajšo pot od trenutnega centra grupe do cilja. V naslednjem koraku se grupi nastavi vrednost spremenljivke m_MotionType na vrednost MOTION_MOVING, kar povzroči to, da se bodo pri klicu funkcije Update iz razreda Environment, vojakom dejansko začeli spreminjati cilji.

Opis procedure update():

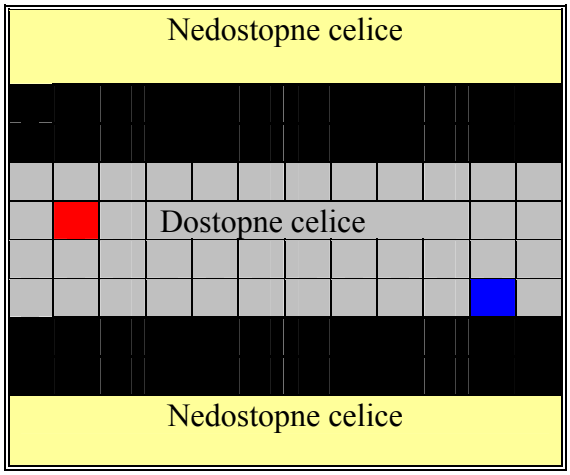
Procedura update() se kliče iz Environment::update(). Najprej se iz grupe odstrani vse mrtve vojake. Nato se za vsakega vojaka, ki je v stanju MOVING izračunajo nove koordinate, kam naj se premakne. Če je nato center grupe dovolj blizu waypointa, kamor se grupa premika, se kot naslednji cilj izbere naslednji waypoint na seznamu točk. Če je center grupe dovolj blizu končnemu waypointu, se postavi grupa v stanje MOTION_NO_WAYPOINT. V tem stanju se vojaki ustavijo tam, kjer so, oz. ne dodeljujejo se jim več novi cilji. Če pa grupa se ni dovolj blizu cilju, potem se po določenem številu klicev funkcije update() poišče naslednjega mirujočega vojaka (funkcija GetFreeSoldier()) in se mu dodeli neko pot v smeri proti končnemu cilju (GetSafePosition()).

Opis procedure GetFreeSoldier():

Izmed vseh mirujočih vojakov se izbere tistega, ki ima prost prehod vsaj v enem kvadrantu (FreeArea()) ter ni bil premaknjen največ časa. To, da mora imeti vojak prehoden vsaj 1 kvadrant, je potrebno zato, ker nočemo izbrati vojaka, ki je v središču grupe, saj bi imeli lahko probleme s trki.

Opis procedure GetSafePosition(int index):

Index nam pove, katerega vojaka bi radi premaknili naprej. Izračun naslednjega vojakovega cilja se izvede tako, da najprej označimo pravokotnik, ki ga dobimo iz vojakovih koordinat ter naslednjega waypointa celotne grupe. Ta pravokotnik napihnemo tako, da gledamo malo širšo okolico tega področja. Nato uporabimo razred CFlood, ki nam označi področja, do katerih lahko pridemo iz trenutnega vojakovega stanja. Na sliki je z rdečo označen trenutni položaj vojaka, z modro pa njegov cilj. Rumeno označene celice so lahko prehodna ali neprehodna območja, vendar ker ne obstaja v tem pravokotniku pot od vojaka do rumenih celic, so ta področja nedostopna in se obravnavajo kot neprehodna.



slika 4: iskanje varnega položaja

Za vsako dostopno celico pa se ovrednotijo naslednji parametri mape: varnost, praznost področja, razdalja do celice in napredovanje. Varnost je parameter, katerega vrednost izračunamo ob začetku igre in se med igro ne spreminja (razred DangerMap). Praznost področja nam pove, ali je na to

celico že namenjen kakšen vojak, ali pa je celica na voljo. Razdalja do celice in napredovanje pa sta funkciji razlik vektorjev testirane celice in trenutnega položaja vojaka. Na podlagi teh parametrov funkcija Evaluate izračuna željenost premika na to področje in vrednost se shrani v nek urejen seznam (OrderedList). Iz seznama se na koncu vzame najboljšo celico (vendar samo v primeru, če vojak v tej okolici še ni bil – tu se uporabi seznam že obiskanih celic v razredu CSoldierItem).

Algoritem za izračun varnosti področja:

Postopek za vsako celico mape izračuna število nedostopnih celic v okolici te celice. Nato število nedostopnih celic * 10 delimo s številom vseh celic v tej okolici in tako dobimo kot varnost število od 1 do 10, pri čemer je 1 nevarno prodročje, 10 pa popolnoma varno (sicer je ta vrednost praktično nedosegljiva). Rezultat algoritma vidimo na izpisu vrednosti te mape. Z X so označena neprehodna območja. Kot vidimo, so željena območja koti, v katerih je pregled, ki ga ima vojak največji.

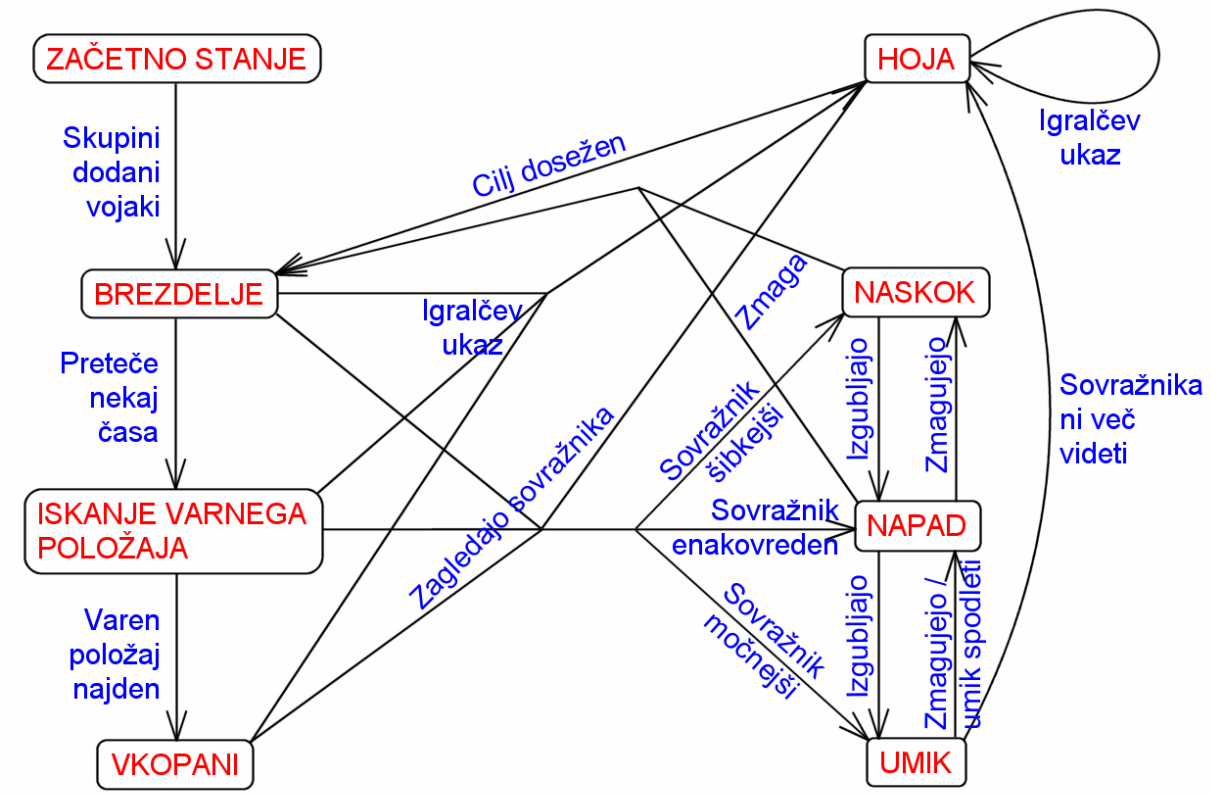
```
XXX5311111111111111111111111111111111135XXXXXXXX
XXX5311111111111111111111111111111111124XXXXXXXX
XXX5311111111111111111111112233333211123XXXXXXXX
XXX5311111111111111111111122555554321112345 67XXX
XXX5311111111111111111111111XXXXXXXXX3211122346XXX
XXX5311111111111111111111122555567X4211111135XXX
XXX5311111111111111111111122333346X5311111135XXX
XXX6433333333333333332211111111122X2211111135XXX
XXX7655555555555555543211111111122X2211111135XXX
XXXXXXXXXXXXXXXXXXXXXXXX3211111111111111111135XXX
XXXXXXXXXXXXXXXXXXXXXXXX4211111111111111111135XXX
XXXXXXXXXXXXXXXXXXXXXXXX5311111111111111111135XXX
55555555555555555567XXX5311111111111111111135XXX
33333333333333333346XXX5311111111111111111135XXX
11111111111111111124XXX4211111111111111111135XXX
11111111111111111123XXX3211111111111111111124XXX
11111111111111111123332111111111111111111123XXX
11111111111111111122211111111111111111112345
11111111111111111111111111111111111111111111223
11111111111111111111111111111111111111111111111
```

slika 5: varnost področij na mapi

Končni avtomat

Vojake ene skupine vodi končni avtomat. To je množica stanj, ki imajo določeno akcijo ob vstopu, med vsako potezo in ob izstopu. Te akcije lahko počno skorajda karkoli. Poleg tega ima vgrajen mehanizem za vzdrževanje seznama uteži (oziroma števil, ki imajo lahko kakršenkoli pomen), ki so odvisne od stanja. Njihove vrednosti se na začetku (ali pa na ukaz med igro) naložijo iz datoteke. Uporabljajo se lahko na vseh mestih, ki imajo dostop do avtomata.

Končni avtomat skupine MDM (Mitja, Damir, Matija)



slika 6: stanja končnega avtomata skupine MDM

Začetno stanje – nezanimivo in potrebno zgolj zato, ker se avtomat lahko določi skupini, ki še nima vojakov.

Brezdelje – v tem stanju so vojaki, ki nimajo ukazov in ne vidijo sovražnika; ne delajo ničesar.

Iskanje varnega položaja – ob vstopu poiščejo varen položaj v bližini, nato pa so na poti tja.

Vkopani – v varnem položaju ležijo na tleh.

Hoja – premikajo se na mesto, kamor jih je poslal igralec.

Naskok – streljajo na sovražnika in se med tem premikajo proti njemu; na začetku in nato peridoično se razvrstijo v strelsko postavitev, o kateri več kasneje (med premikanjem proti sovražniku se utegne postavitev skvariti, zato jo je treba obnavljati)

Napad – ob vstopu se postavijo v strelsko postavitev in se uležejo, nato pa streljajo na sovražnika.

Umik – hodijo proti točki, v kateri so dobili zadnji igralčev ukaz in streljajo na sovražnika.

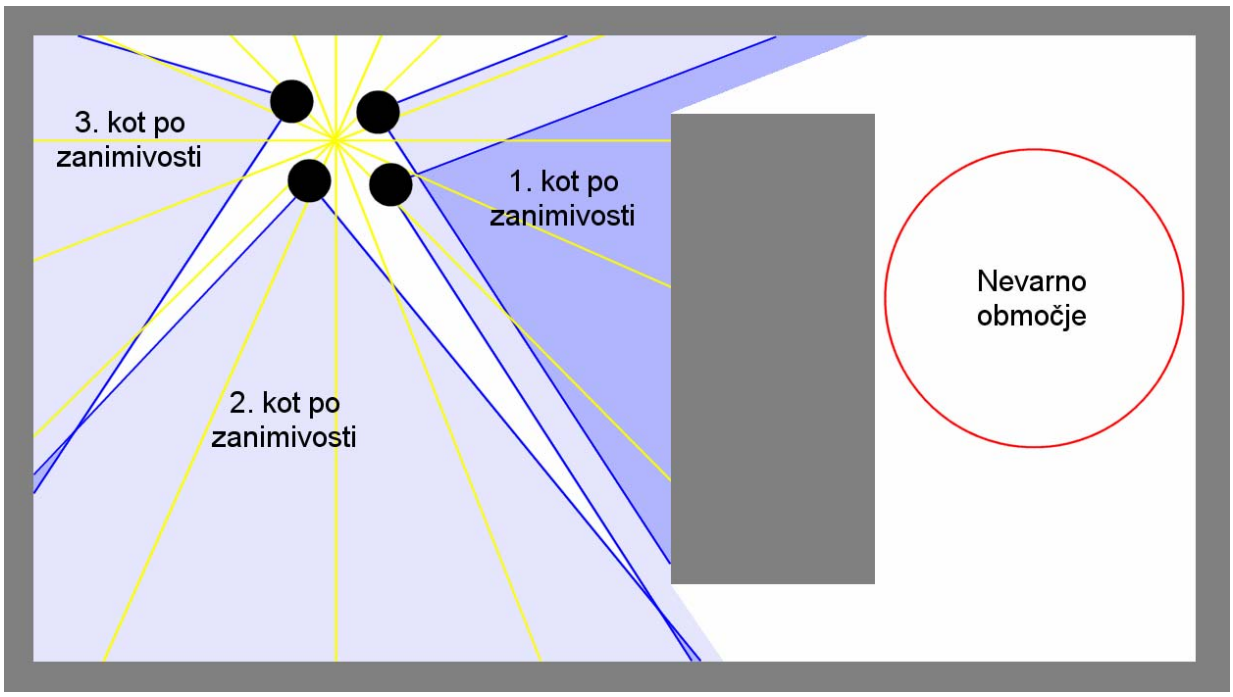
Kadar vojaki ne vidijo sovražnika, si prizadevajo s pogledom pokrivati čim več terena (več o tem kasneje). Kadar ga vidijo, pa se obrnejo proti njemu in streljajo nanj. Takrat tudi ne upoštevajo najbolj ukazov igralca.

Moč sovražnika ocenjujemo tako, da primerjamo vsoto vrednosti zdravja sovražnih in lastnih vojakov (k vsakemu zdravju pa prištejemo konstanto, ker že to, da je vojak prisoten, nekaj pomeni, saj te lahko ustreli tudi ranjenec).

Pokrivanje terena s pogledom

Če se skupina premika, sprednji vojak (tisti, pri katerem je vektor od središča skupine do njega najbližji vektorju premikanja) postane vodja. Ta glavo obrača levo in desno (če tam ni zidu, ker gledanje v zid pač ni koristno). Kako daleč je zid, ugotavljamo v 16 smereh od središča skupine (*rumene črte*).

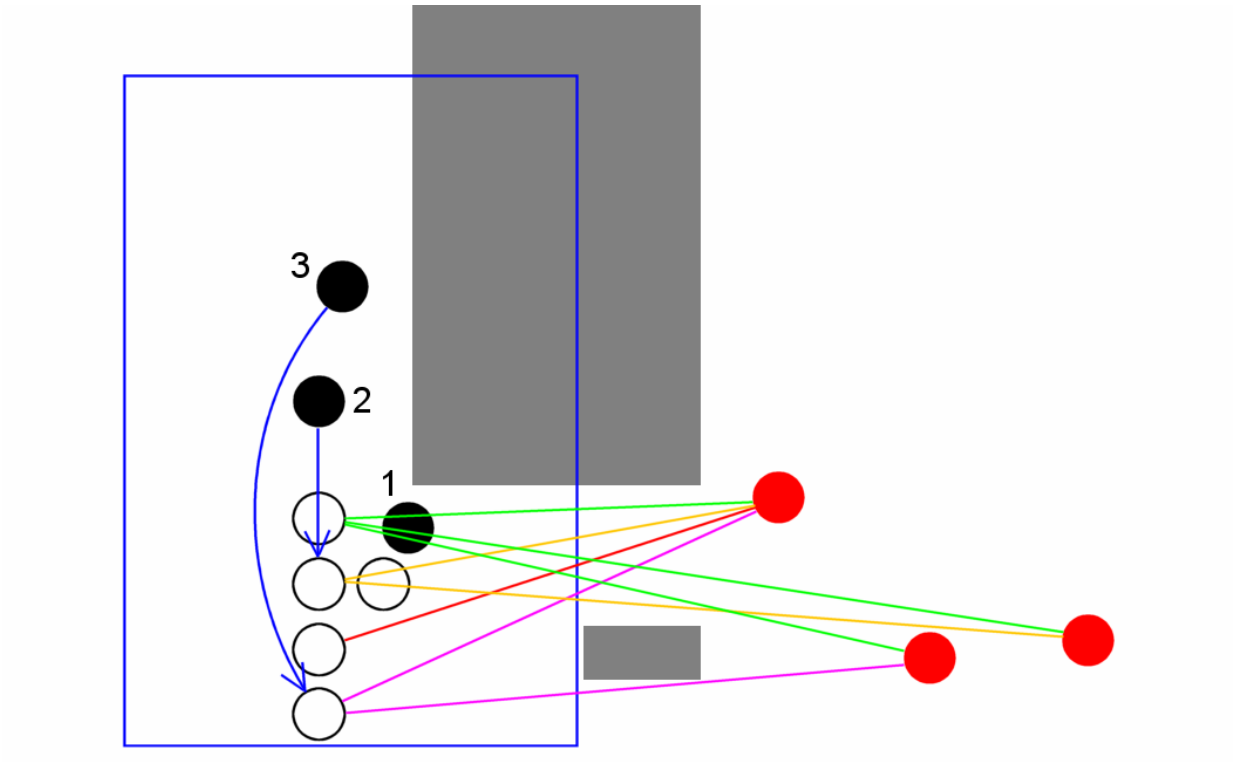
Ostali (kadar skupina miruje, vsi) vojaki si prizadevajo pokriti čim večji skupni kot (v smereh, kjer ni preblizu zidu). Odsek, ki se dodeli posemeznemu vojaku, je enak njegovemu zornemu kotu (o tem pišemo v poglavju o fiziki). Vsak gleda v smeri stran od središča skupine, da mu tovariši ne zakrivajo pogleda (kote vojakov od središča skupine in želeno kote gledanja sortiramo ter jih nato prirejamo po vrsti). Če že del vojakov v skupini pokrije vse teren, dodatne dodelimo bolj zanimivim smerem – to so take, kamor se vidi dlje ali kjer je nevarno območje (uporabi se utežena vsota tega dvojega).



slika 7: pokrivanje terena s pogledom

Strelska postavitev

Ko kak vojak v skupini zagleda sovražnika, vsi, ki ga vidijo, obmirujejo (*1*). Ostali se postavijo v položaj primeren za streljanje. To je tak, od koder bo vojak videl čim več sovražnikov (da se mu, ko prvi umre, ni treba spet premikati), kjer ne bo ustrelil nobenega tovariša in ne bo noben tovariš ustrelil njega ter je seveda čim bliže mestu, kjer trenutno je. Pregledamo položaje znotraj pravokotnika, ki je za konstato razširjen skupini očrtan pravokotnik (*moder*).

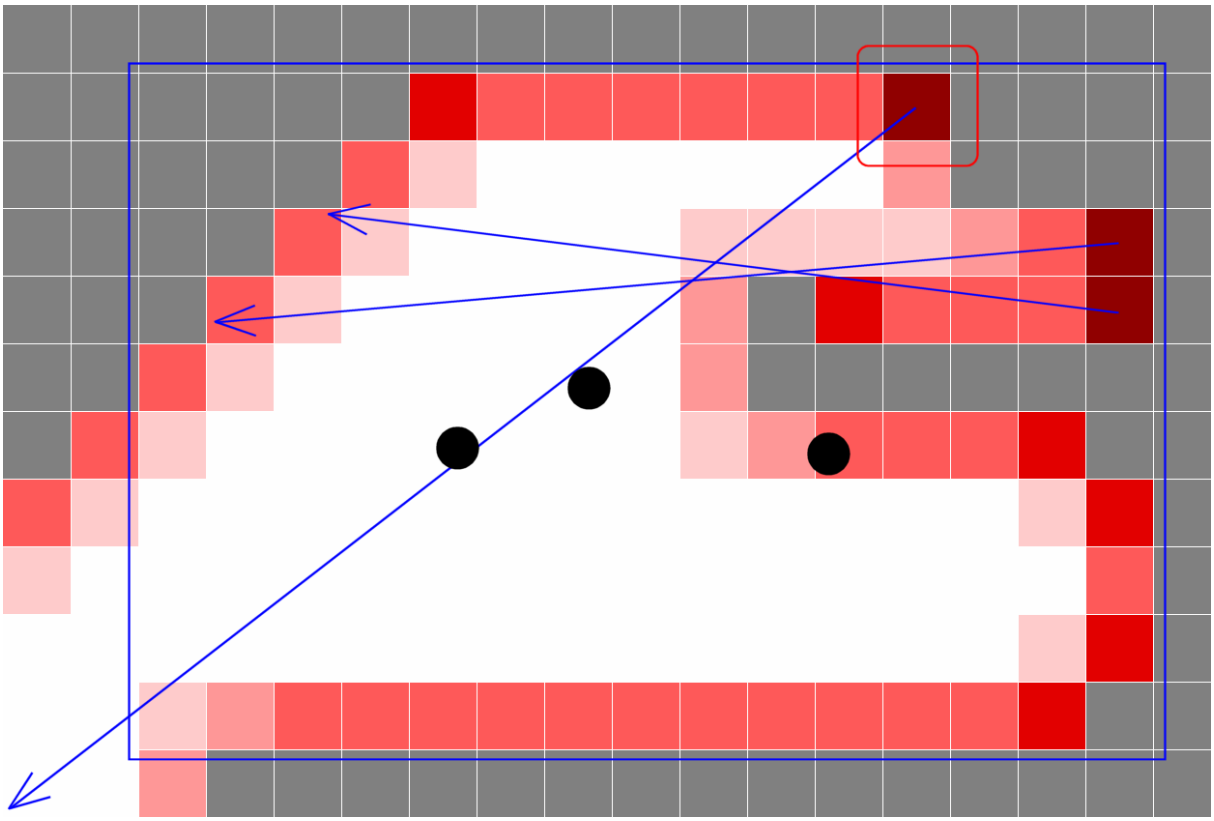


slika 8: strelska postavitev

Po zadnjem kriteriju bi bilo za vojaka *2* najprimernejše mesto za vojakom *1*, a bi slednjega tam ustrelil, zato se bo postavil na označeno mesto (*modra puščica*). Vojak *3* bi prav tako lahko izbral ti dve mesti, a prvega ne bo iz istih razlogov kot ga ni *2*, drugega pa zato ne, ker ga je že *2*. Prav tako se ne bo postavil pred mesto, kamor se bo *2*, ker bi tam utegnil biti ustreljen. Ostala mesta iz podobnih razlogov niso primerna, zato se pomudimo pri zadnjih dveh, ki sta označeni (*s praznim črnim krogom*) – zgornje je sicer malce bliže, a spodnje ni bistveno dlje, se pa od tam vidita dva sovražnika, zato bo izbral tega (*modra puščica*).

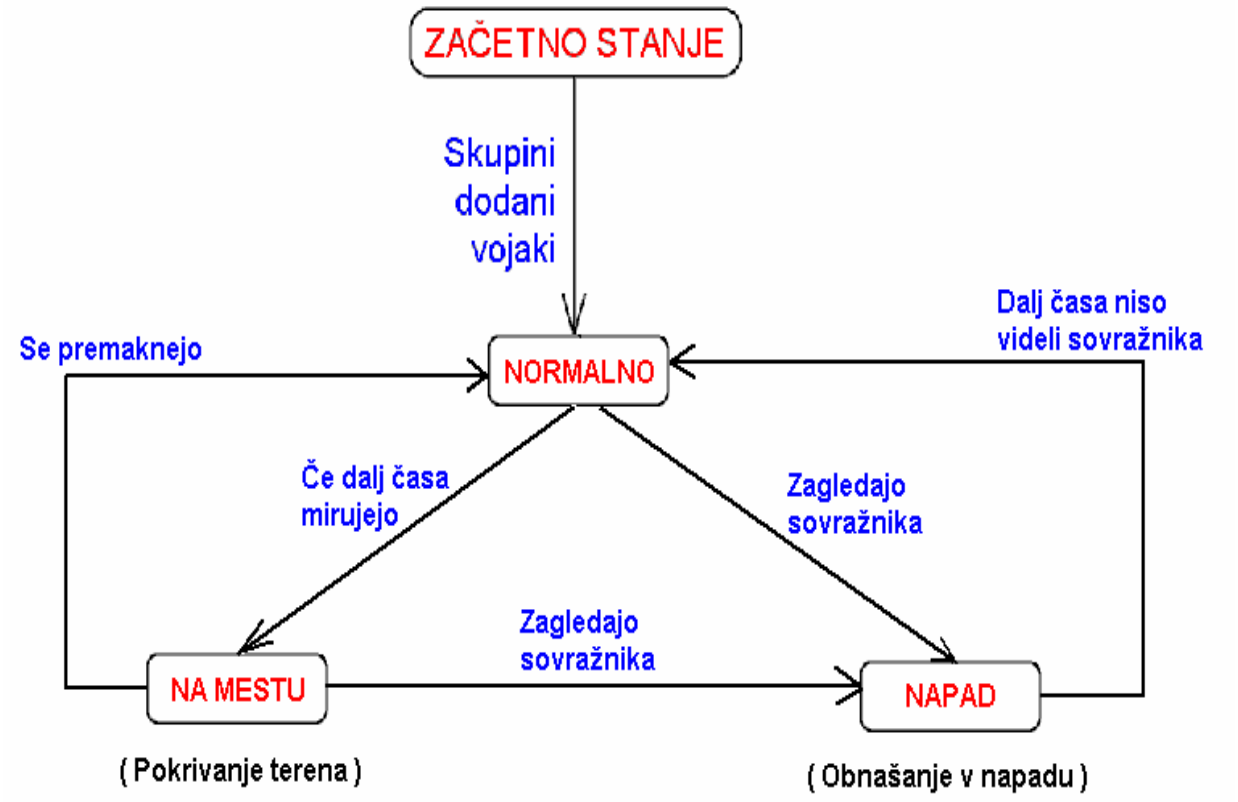
Iskanje varnega položaja

Koliko je položaj varen, je odvisno od tega, koliko sten je v soseščini (*odtenki rdeče barve*). Ko želimo poiskati varen položaj za skupino, pregledamo vse kvadratke znotraj pravokotnika, ki je za konstanto razširjen skupini očrtan pravokotnik (*moder*). Primernost položaja je utežena vsota njegove varnosti in razdalje, kamor se najdlje vidi (*modre puščice*)- skupina izbere najugodnejšega glede na to (*rdeč zaobljen kvadrat*).



slika 9: iskanje varnega položaja

Avtomat skupine GUM (Gregor, Uroš, Martin



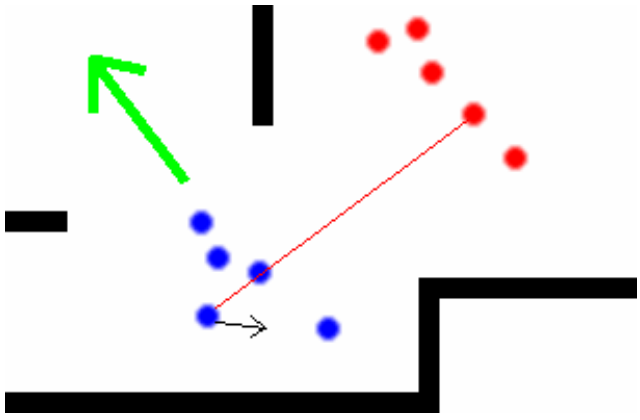
slika 10: stanja končnega avtomata skupine GUM

Avtomat skupine GUM je narejen v skladu z navodili mentorja, ki je predlagal malo stanj, ki pa naj bodo močna. Tako je lažje ohraniti kontrolo nad avtomatom, ki ga dopolnjujemo z dopolnjevanjem vsebine stanj in ne z novimi stanji ali novimi prehodi.

Stanje NORMAL je normalno stanje v katerem se vojaki premikajo in gledajo v vse smeri ali le v smeri iz katerih pričakujemo nevarnost (po želji igralca).

Če so nekaj (utež) časa pri miru, preidejo v stanje STATIC v katerem se poležejo po tleh in s pogledi pokrivajo ves teren.

Če v katerem koli od prej omenjenih dveh stanj zagledajo sovražnika, gre avtomat v stanje ENEMY, v katerem vsi vojaki gledajo v smer, kjer je bil sovražnik nazadnje opažen in streljajo na sovražne vojake. Če se pri tem ovirajo (glede na to, da jih igralec med tem lahko premika se to dogaja), se razmaknejo upoštevaje pravokotnico na smer strela (premakne se vstran in malo proti sovražniku). Tako preprečimo, da bi se med premikanjem medsebojno preveč ranili. Ko nekaj časa (utež) ne vidijo več nobenega sovražnika, spet preidejo v stanje NORMAL ali STATIC.

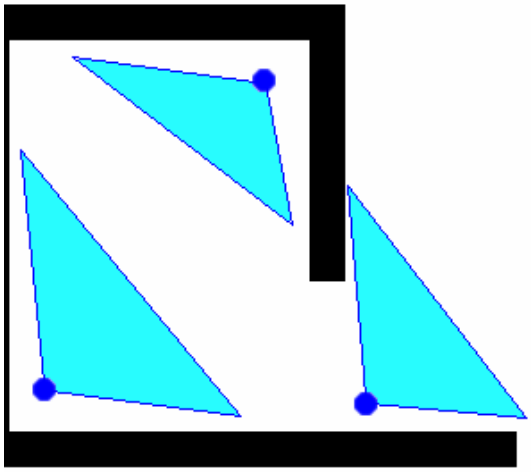


slika 11: obnašanje v stanju ENEMY

Ob gradnji avtomata za našo skupino smo že poznali obnašanje avtomata druge skupine, zato smo ga zastavili popolnoma drugače. Dva zelo različna avtomata je namreč lepše in bolj zanimivo primerjati, analiziramo pa lahko dobre in slabe strani obeh realizacij.

Ker smo vedeli, da avtomat MDM med napadom postavi vojake na pameten način v fiksno formacijo in da med napadom igralec praktično nima kontrole nad njimi, je bil naš prvi cilj ohraniti gibljivost vojakov in kontrolo igralca med napadom. Kasneje se je izkazalo, da je nujno vsaj toliko vplivati na postavitev, da se med sabo ne ranijo, zato prej omenjeno razmikanje.

V začetku so v stanjih NORMAL in STATIC vojaki sicer gledali v vse smeri, kar je s praktičnega vidika (čimprej opaziti sovražnike) sicer najboljše, ampak večkrat je izgledalo grdo, saj je kateri od njih gledal naravnost v zid. Zato smo za vsakega vojaka izračunali štiri smeri pogleda, urejene po pomembnosti. Pri tem smo upoštevali odprtost terena in nevarna območja v tisti smeri. Tako so vojaki lahko gledali v smereh, od koder je pričakovati prihod nasprotnikov in ne v zidove.



slika 12: smeri pogleda

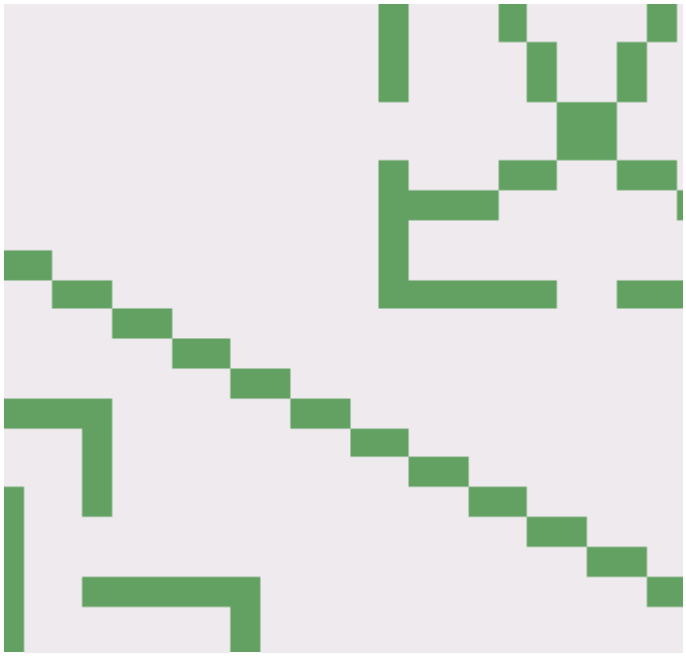
Naš avtomat je bil torej zastavljen enostavno in jasno in je kot tak tudi enostavno razširljiv. Dodali bi mu lahko še stanje za umik, vendar to v trenutni verziji fizike in pogojih v katerih smo testirali uspešnost avtomatov, ni imelo nobenega smisla.

V boju s skupino, ki jo je vodil avtomat MDM se je na začetku izkazal odlično, saj skoraj nismo mogli izgubiti (za to je bila kriva tudi fizika, ki seveda ni mogla biti popolnoma realistična – hitrost metkov). Po nekaterih spremembah kode in igralcev pa so rezultati postali slabši in je naša skupina le težko zmagala.

Slabost se je kazala v tem, da med napadom nimajo dobre formacije in so manj učinkoviti. Prednost avtomata GUM pa je gotovo to, da so vojaki med napadom premični in spreten igralec lahko to dobro izkoristi – še posebej, če si ustvari več grup, ki jih med napadom premika.

Uporabniški vmesnik

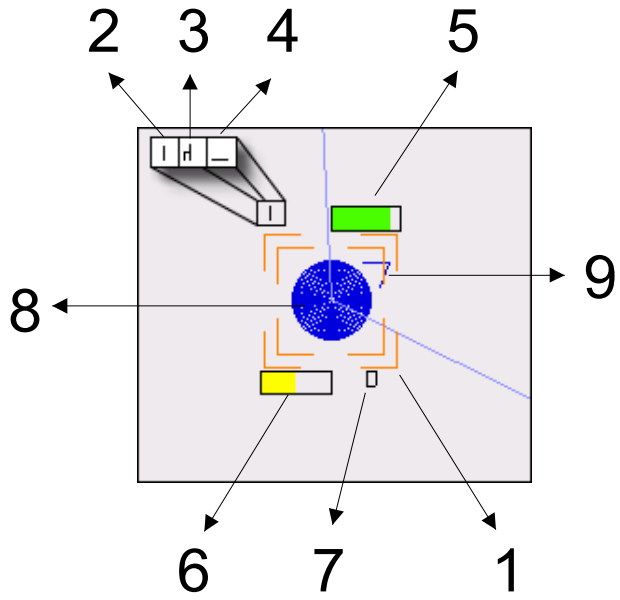
Grafični prikaz



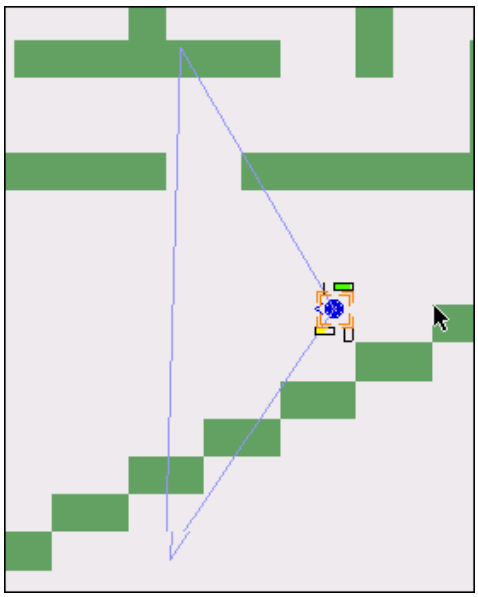
slika 13: Zemljevid

Splošni prikaz zemljevida

Zemljevid je sestavljen iz polj svetle barve, ki predstavljajo prehodna območja in iz polj temnejše barve, ki predstavljajo neprehodna območja.



slika 14: prikazani atributi enote



slika 15: vidno polje enote

Enota

Vsaka enota ima svoje atribute, nekateri med njimi so prikazani tudi med igro. Če je enota označena, je obkrožena z dvojno črto na vogalih (1). V zgornjem levem kotu je prikazano stanje vojaka (2,3,4). Nad vojakom vidimo preostanek energije, ki ga ohranja pri življenju. Ustrezno s krajšanjem črte se spreminja tudi njena barva od zelene prek rumene v rdečo (5). Črta pod enoto ponazarja nastavljeno agresivnost vojaka (6). Številka v spodnjem desnem kotu predstavlja številko skupine, ki ji enota pripada (7). Barva enote sovpada s stranjo, na kateri se bojuje (8). Puščica (9) predstavlja smer, v katero se enota giblje, modri trikotnik pred enoto (desna slika) pa prikazuje vidno polje, ki ga enota pokriva.

Prikaz nastavitev

V igri je lahko uporabniški vmesnik v enem izmed štirih načinov (za načinom je dana tudi ikona, ki je prikazana na zaslonu med igro)

- *Normalni način* (brez ikon)
- *Način za premikanje* 
- *Način za določanje obračalnih točk* 
- *Način za določanje nevarnih območij* 

Uporaba

Označevanje

Enote se označuje s klikom na enoto v *normalnem* ali *načinu za premikanje*. Za označevanje več enot lahko med klikanjem po enotah držimo tipko '*shift'*. Zadnja možnost pa je premikanje miške s pritisnjenim levim gumbom, kar nariše pravokotnik na zaslonu. Ob spustu gumba se označijo vse enote znotraj pravokotnika.

Delo s skupinami

Za lažje delo z enotami se enote lahko združuje v skupine, nad katerimi se kasneje izdajajo ukazi. Označeni vojaki, ki ne pripadajo nobeni skupini so začasno dodeljeni privzeti skupini 0. S pritiskom '*ctrl'* + številka med '1' in '9' se lahko enote dodeli skupini, ki je za naprej predstavljena s pritisnjeno številko. Če želimo priklicati skupino (naenkrat označiti vse vojake, ki ji pripadajo), pritisnemo ustrezno številko skupine.

Premikanje

V *načinu za premikanje* enostavno kliknemo na prazen prostor na zemljevidu. Vsem označenim enotam je tako izdan ukaz za premikanje proti kliknjeni točki. Če smo v *načinu za dodajanje obračalnih točk*, s pritiskom na tipko '*E*' zahtevamo izvedbo ukaza o obhodu vseh točk.

Obračalne točke

So navidezne točke, ki jih določimo na zemljevidu in predstavljajo pot vojakov v zaporedju, ki ga določimo. S tipko *'W'* preidemo v način za dodajanje obračalnih točk, kar opazimo tudi po prikazani ikoni (glej *grafični prikaz* → *prikaz nastavitev*). S klikanjem po zemljevidu postavljamo točke (), s pritiskom *'E'* pa zahtevamo obhod poti. Točke lahko tudi brišemo in sicer v ravno obratnem vrstnem redu s pritiskanjem tipke *'R'*. Izhod iz *načina za dodajanje obračalnih točk* in prehod nazaj vanj zbriše vse obračalne točke.

Povečevanje in zmanjševanje zemljevida

V aktivnem prikaznem oknu lahko pogled zmanjšamo ali povečamo s pritiskom tipk '+' in '–' na numerični tipkovnici. Manjše prikazno okno lahko tako uporabimo za prikaz splošnega dogajanja, večje pa za interakcijo.

Nevarna območja

Nevarna območja dodajamo tako, da najprej vključimo *način za dodajanje nevarnih območij* (prikaže se ustrezna ikona)s pritiskom na tipko 'A', nato pa najprej kliknemo za določitev središča nevarnega območja, nato pa še za polmer nevarnega območja. Nevarna območja brišemo, v nasprotnem vrstnem redu od dodajanja, s pritiskanjem tipke 'D'.

Agresivnost

Agresivnost enote nastavljamo za vse označene enote naenkrat s pritiskom *shift* + številka med '0' in '9', kjer '0' pomeni nič odstotno, '9 'pa sto odstotno agresivnost.

Avtomatično drsenje prikaznega okna

S pritiskom tipke '*F2*' vklopimo avtomatično drsenje, ki označeno enoto postavi v središče dogajanja, če pride preveč na rob prikaznega okna. S ponovnim pritiskom način izklopimo.

Zajemanje zaslona

S pritiskom na tipko '*F1*' začnemo z zajemanje zaslona v danih intervalih. Slike se shranjujejo v bitne slike v trenutni globini in ločljivosti z imenom 'CaptureXXX.bmp', kjer XXX predstavlja zaporedno številko zajete slike. S pritiskom tipke '*F10*' naenkrat zbrišemo vse zajete slike naenkrat.

Zajemanje zaslona v AVI datoteko

S pritiskom tipke '*F3*' začnemo s snemanjem dogajanja v AVI datoteko. Prav tako s ponovnim pritiskom iste tipke nehamo s snemanjem, film pa je shranjen v 'movie.avi' datoteko.

Inicializacijska datoteka

Datoteka se mora nahajati v Windows direktorijo in je sestavljena:

Sekcija	**Argument**	**Pomen**
Global		splošne nastavitve programa
	MapName	datoteka, ki vsebuje podatke o ozemlju – zemljevid
Server		nastavitve specifične za strežnik
	ClientCount	število odjemalcev, ki se morajo priključiti za začetek igre
Client		nastavitve specifične za odjemalca
	IP	IP strežnika
	PicName	datoteka s teksturami (se ne uporablja, a je potrebna)

Hitri pregled tipk v igri

Tipka	**Pomen**
M	*način za premikanje* (in izhod iz njega)
W	*način za obračalne točke* (in izhod)
E	obhod obračalnih točk (deluje v *načinu za obračalne točke*)
R	zbriši zadnjo obračalno točko
A	*način za dodajanje nevarnih območij*
D	zbriši zadnje nevarno območje
ctrl + številka med 1 in 9	tvorjenje ustrezne skupine
številka med 1 in 9	priklic ustrezne skupine
shift + številka med 0 in 9	nastavitev agresivnosti za označene enote
F1	začetek in prenehanje zajemanja slik
F2	vklop in izklop avtomatičnega drsenja okna
F3	začetek in prenehanje zajemanja videa
F10	brisanje vseh zajetih slik