»Mladi za napredek Maribora 2015«
32. srečanje

RAČUNALNIŠKA OSEBNOST

Raziskovalno področje: RAČUNALNIŠTVO
RAZISKOVALNA NALOGA

PROSTOR ZA NALEPKO

Avtor: LEON GORJUP

Mentor: BRANKO POTISK

Šola: SREDNJA ELEKTRO-RAČUNALNIŠKA ŠOLA

2015, Maribor

»Mladi za napredek Maribora 2015«
32. srečanje

RAČUNALNIŠKA OSEBNOST

Raziskovalno področje: RAČUNALNIŠTVO
RAZISKOVALNA NALOGA

2015, Maribor

Kazalo

Kazalo slik

1 Povzetek

Ali je mogoče poustvariti realistično razmišljanje znotraj računalniškega programa? Cilj te raziskovalne naloge je ugotoviti odgovor na to vprašanje.

V nalogi smo raziskovali, ali lahko na preprost način ustvarimo elemente igre (osebe), ki bi se samostojno odločali v različnih okoliščinah računalniške igre. Vsaka oseba bi imela svojo osebnost, ki bi vplivala na njene odločitve. Naloga igralca bi bila upoštevati te lastnosti in sestaviti ekipo. Želeli smo, da bi z določitvijo lastnosti posameznih osebnosti igra potekala bolj realistično. Rezultat takšne zasnove igre je večje zanimanje potencialnih igralcev. V velikem številu današnjih strateških iger so vojaki le preproste lutke, ki storijo točno to, kar jim je ukazano. Mi smo želeli ustvariti vojake, ki bi imeli neko svojo identiteto in bi ne bili čisto enaki drug drugemu. Bili bi tudi, vsaj do neke mere, sposobni samostojnih odločitev.

Kot rezultat te naloge je nastala igra, ki poskuša simulirati obnašanje posameznika v neki situaciji.

2 Uvod

Igre in resničnost se zelo razlikujejo. Nekatere igre so bolj realistične kot druge, vendar je tudi najbolj realistična igra težko enakovredna resnični upodobitvi enake situacije. Če vzamemo na primer strateške igre, je v večini teh vojak le marioneta, ki slepo posluša igralčeve ukaze. Za razliko od resničnega življenja, kjer imajo velik vpliv poleg sposobnosti poveljnika tudi osebne izkušnje, sposobnost, motivacija, volja in značaj posameznika, so v večini iger te lastnosti zanemarjene. Nekatere igre pa zajamejo nekaj teh lastnosti: sposobnost s pomočjo nivoja izkušenosti vojakov (primer: Command & Conquer igre), volja *(ang. morale* - primer: Total War), vendar je dejstvo, da v realni situaciji ne odreagirajo vsi na enak način.

Bi šlo narediti podobno igro, brez posebnega predznanja, ki bi vsebovala neke osnove umetne inteligence? Na to vprašanje bomo skozi našo nalogo skušali najti odgovor.

3 Hipoteza

Ali je mogoče poustvariti realistično razmišljanje znotraj računalniškega programa?

Na to vprašanje bomo skušali odgovoriti na ta način, da bomo izdelali lastno simulacijo spopada v obliki preproste računalniške igre.

Postavljamo naslednjo hipotezo:

S pravilno zasnovo bodo objekti – vojaki sprejemali bolj realistične – človeške odločitve in s tem vplivali na potek igre.

4 Teoretični del

4.1 Računalniške igre

Ljudje že od nekdaj igramo igre. To je lahko za sprostitev, za zabavo ali za druženje. Igre so najpogosteje namenjene zabavi, ampak včasih so pa obenem tudi poučne. V današnji digitalni dobi nastaja vedno več iger poučnega značaja. Iz njih se lahko naučimo marsikaj: o naravi, vesolju, človeškem telesu, tudi matematiko ali jezike. Vse igre pa imajo nekaj skupnega: imajo neka pravila, katerih se morajo vsi igralci držati. Igre imajo tudi cilj - nekaj, kar mora igralec doseči. Večina se jih tudi tako ali drugače zaključi, lahko z zmago ali s porazom. Če bi igralec moral le opazovati dogajanje, to ne bi bila nikakršna igra, torej potrebuje tudi neko interakcijo. Ta je v primeru računalniških iger lahko tako enostavna, da mora igralec le pravi trenutek pritisniti eno tipko. Lahko pa tudi tako zapletena, da če igralec ne prebere prej priročnika, ne bo vedel, kaj delati - npr. letalske simulacije.

Kljub podobnostim so pa lahko ogromne razlike v delovanju in videzu različnih vrst iger. Nekatere igre so sestavljene iz mreže kock, ki jih lahko igralec uničuje, premika in drugače manipulira z njimi. V nekaterih je igralec postavljen v vlogo vojaka (na primer na vesoljski ladji) in se mora obraniti pred napadalci. Spet druge igre zahtevajo od igralca, da mora razmišljati in reševati uganke, če želi nadaljevati.

Danes smo tehnološko že tako napredni, da lahko znotraj računalnika ustvarimo celoten izmišljen virtualni svet. Na podlagi tega dejstva je nastalo že ogromno iger, kjer igralec prevzame vlogo junaka v domišljijskem svetu. Najpogosteje lahko igralec v takšnih igrah hodi okoli sveta in si razvija tega junaka. Če lahko ustvarimo izmišljen svet, lahko tudi poustvarimo resničnega. Obstajajo simulacije za skoraj vse, česar si je možno domisliti – vožnjo, letenje, vodenje mesta, kmetovanje, akvarij, lov, vojno. Obstaja celo simulator življenja predmestne koze. Meje iger so skoraj neskončne.

Mnogim se zdi, da je ustvariti računalniško igro zelo težek in dolgotrajen proces. Ta izjava je morda včasih držala, vendar je danes toliko različnih orodij za izdelavo iger, da bi se dalo s pravim orodjem tudi brez posebnega predznanja v manj kot enem tednu narediti delujočo igro, če ta ni preveč zapletena. Nekateri preprosti programi za izdelavo iger so na primer Scratch, GameMaker, Construct. Preko njih se lahko tudi naučimo razmišljanja, potrebnega za izdelavo iger.

5 Problem

Prvotna ideja je bila igra, ki bi bila na prvi pogled preprosta, vendar bi v ozadju skrivala kompleksne algoritme, ki bi igralca spodbujali k razmišljanju, če bi ta želel dober izid. Kasneje se je izoblikovala zamisel za strateško igro z vojaki, ki pa za razliko od večine strateških iger ne bi bili popolnoma pod igralčevim nadzorom. Poskusiti smo tudi želeli uvesti človeški faktor v razmišljanje teh vojakov in ugotoviti, ali se je sploh potrebno tako poglabljati v njihovo razmišljanje.

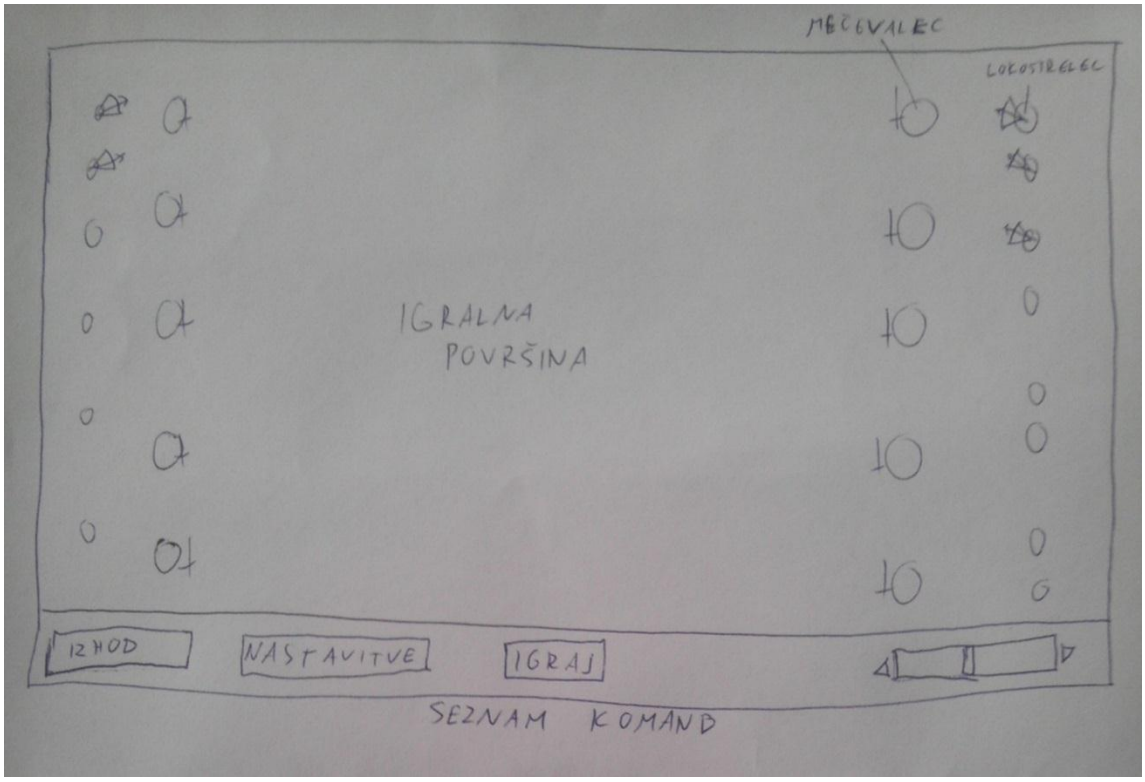
5.1 Metodologija dela

Odločili smo se za metodo poskusov in napak, ker lahko s tem pridobimo veliko izkušenj in je ena od možnosti, da pridemo do drugačne poti do rešitve, kot bi morda, če bi sledili že obstoječemu vzorcu. Pri tem je pomembno, da iz napak zberemo zaključke, ki se jim zato lahko v bodoče izognemo.

Mi smo se odločili ustvariti igro na klasičen način, t.j. sprogramirati celotno delovanje igre s splošnim programskim jezikom tretje generacije. Med te spadajo znani jeziki kot so C, C++, Java, C#, FORTRAN. Mi smo izmed njih izbrali Javo, ki se je učimo v šoli. Prednosti Jave pred drugimi jeziki so na primer dobra prenosljivost kode, poleg tega pa se zaradi garbage collector-ja ni treba ukvarjati z brisanjem nepotrebnih objektov. Ta je sprožen v določenih časovnih intervalih in gre skozi vse obstoječe objekte ter izbriše vse, za katere meni, da več niso nikjer uporabljeni. Ta način seveda ni popoln, je pa zelo dober in učinkovit. Java je tudi eden izmed najbolj učinkovitih programskih jezikov po zaslugi svojega JIT prevajalnika, ki lahko celo med izvajanjem programa prevaja delčke izvorne kode in optimizira delovanje. Potrebovali smo tudi nekaj za vizualizacijo dogajanja, za kar smo uporabili že obstoječo Swing knjižnico, saj je dokaj preprosta in že vključena v sam programski jezik. Izbira jezika nam omogoča tudi neodvisnost od strojne opreme, saj je za izvajanje potreben le Java stroj.

5.2 Načrt

Od začetka same ideje in scenarija igre smo imeli v mislih programsko logiko, zato je bil izgled na drugem mestu in skica tudi temu primerno preprosta.



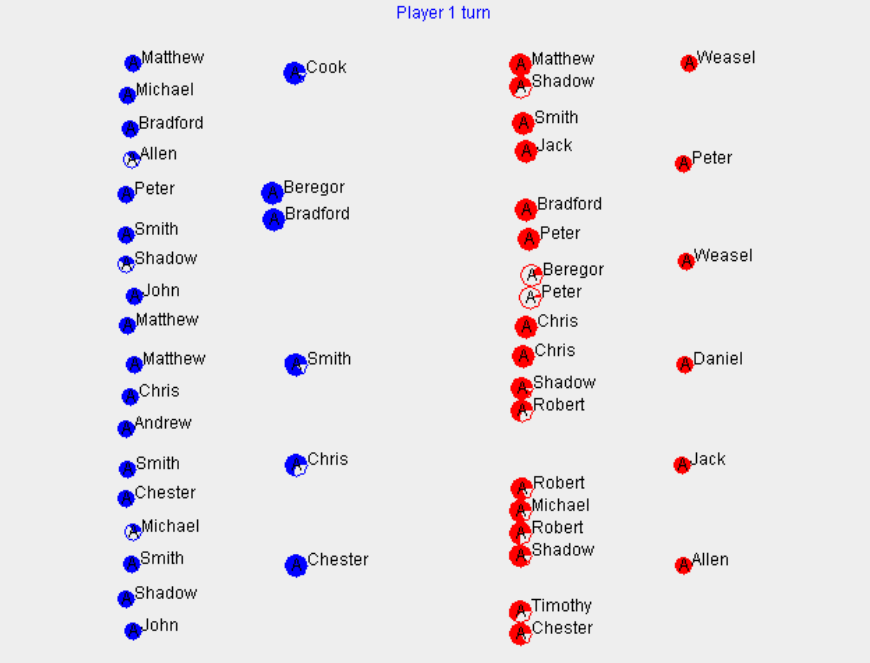
Slika 1: Idejna skica igre

V mislih smo imeli le delujočo igro, vizualni del pa se ni zdel tako pomemben. Poleg tega je brez zapletenih grafik igra tudi bolj prenosljiva. Planiranih je bilo le nekaj gumbov: za začetek, razne nastavitve in konec igre.

6 Opis igre

Za programski jezik smo zaradi zgoraj navedenih razlogov izbrali Javo. Grafični vmesnik smo si zamislili precej osnoven; le toliko, da je sposoben zadovoljivo prikazovati dogajanje. Barvni krogci predstavljajo vojaka. Bolj, kot je krog prazen, manj življenja ima. Na sredini kroga je izrisana črka, ki bi naj predstavljala, kaj vojak trenutno namerava (A – attacking/napad, D – desperate/obup, R – running/beg, M – moving/premik). Zraven krogca pa je tudi naključno ime iz vnaprej pripravljenega seznama, zato da si je malo lažje zapomniti, kdo je kdo.

V končni izvedbi bi lahko eventualno dodali še kakšne grafike za ozadje in namesto krogcev prave ljudi in puščice, vendar se je to zdelo manj pomembno, kot pa njihovo dejansko obnašanje. Zdelo se nam je pomembneje posvečati programskemu delu kot pa vizualnemu izgledu, pri tem pa se zavedamo, da tak izdelek ni primeren za trženje. Iz vsakdanjega življenja vemo, da je včasih izgled bolj pomemben od vsebine.



Slika 2: Grafični izgled igre

Čisto na začetku bi bila prva naloga, da se igralec odloči, kje bo stal kateri vojak. S klikom na vojaka se izpišejo njegove lastnosti, ki jih potem igralec oceni in se odloči, kaj bi bil njemu primeren ukaz in položaj. Igralčeva nadaljnja vloga bi se pojavila na rednih časovnih intervalih, na primer vsakih 10 sekund bi lahko igralec dajal ukaze. S tem ima tudi izbiro, ali se bo sam odločal ali prepustil odločanje posameznim vojakom. Če bi na primer bilo 10 vojakov

in bi jih igralec poslal naprej devet, desetemu pa ne bi dal nobenega ukaza, bi se morda tudi ta odločil slediti ostalim. Edini ukaz, ki bi ga imel igralec bi naj bila tarča, kamor se naj premaknejo. Če bi med potjo naleteli na sovražnika, bi se morali potem sami odločiti, ali bodo napadli ali pa nadaljevali pot proti tarči. Tako mora igralec misliti tudi na to, kam svoje vojake usmeri, saj če bi jih poslal predaleč naprej, bi mislili, da se naj izogibajo sovražniku. Lahko bi pa tudi namenoma to naredil, da bi jih uporabil kot vabo, ali pa bi se s tem izognil frontnim vojakom in napadal tiste, ki so za njimi in morda bolj ranljivi. Če vojak ne bi dobil nobenih ukazov, bi stal na mestu, vendar bi, ko bi se sovražnik preveč približal, začel le-tega napadati ne glede na ukaz.

7 Potek izdelave programa

Prvi pogoj za izdelavo kakršnegakoli programa ali igre je neko okno, ki prikazuje njegovo delovanje; zato smo začeli programiranje s prikaznim oknom. Naredili smo samostojen razred *Window*, preko katerega smo ustvarili nov objekt *Game*, izpeljan iz osnovnega JPanel objekta iz Swing knjižnice. Ta potem v zanki posodablja in izrisuje vse igralne objekte, dokler se okno igre ne zapre. Dodan je tudi časovni števec, ki v primeru zelo kratkega izvajanja posodobitve zanko za kratek časovni interval zaustavi, saj bi drugače hitrost izvajanja na čase močno nihala.

Naslednji korak je bil *Game* objekt. Ta skrbi za stanje igre ter za izrisovanje in posodabljanje vseh spremenljivk in objektov. Metoda za posodabljanje gre ob vsaki izvedbi skozi vse vojake in pokliče njihovo metodo obnašanja, poskrbi za fiziko (premiki puščic in trki med vojaki) ter za vsakega pogleda, ali je mrtev, in če je, ga odstrani iz igre.

Konstruktor tega razreda poskrbi, da se inicializirajo vsi potrebni podatki. To vključuje generiranje vseh vojakov in pa poslušalce tipk (KeyListener in MouseListener). Igra je na začetku zaustavljena in se začne koma ob pritisku tipke za presledek. Zato, da bi igralec dejansko lahko vplival na igro, smo morali ustvariti nova *KeyListener* in *MouseListener* objekta. V prvem smo naredili kontrole za pavzo igre in za preklic izbora vojakov, drugi pa je skrbel za vse ukaze, povezane z miško. To pomeni izbor vojakov in dajanje ukazov.

Problem v izdelavi sistema, kjer bi vsak posamezen vojak dal občutek individualnosti je, kako to implementirati v programski kodi. Prva misel za izvedbo mislečih vojakov je bila, da pred vsako odločitvijo program generira naključno število med 0 in 1 in se preko te odloči, kaj storiti. Ker pa je to bilo preveč naključno, smo poskusili ustvariti polje za vrednost vsake lastnosti za vsakega vojaka posebej. Ob inicializaciji vojaka bi se temu potem naključno

nastavile vrednosti atributov glede na to, kakšne vrste vojak je. Tak sistem bi potem omogočil, da bi se različni vojaki v dani situaciji različno odločili. Če bi bili znova v enakem položaju je velika verjetnost, da bi se enako odločili - kar v primeru popolnoma naključne vrednosti ne bi držalo.

Če bi bila za inicializacijo lastnosti vojakov uporabljena navadna *Math.random()* metoda, bi bile lastnosti zelo razpršene in bi v končni fazi stvar izpadla precej kaotično. Zato smo naredili statično metodo *gaussRandom(int min, int max)*, ki vrne naključno število z vrednostmi razporejenimi po gaussovi krivulji – torej je najpogostejša srednja vrednost, ekstremne vrednosti pa so redke. Tako je na primer mečevalec v večini primerov močnejši od lokostrelca, medtem ko je le-ta hitrejši, vendar se še vseeno najde kakšen izjemno močan lokostrelec. To metodo smo implementirali tako, da smo vzeli povprečje med več generiranimi števili med 0 in 1 in potem to vrednost uporabili kot generirano naključno število, razporejeno med dano minimalno in maksimalno vrednostjo. Odločili smo se za povprečje 2 števil, saj se nam je pri 3 številih zdelo že preveč nagnjeno proti srednji vrednosti. Kasneje smo tej metodi dodali še dve dodatni spremenljivki: eno, ki vpliva na razpon generirane vrednosti, da smo lahko preizkusili tudi delovanje v primeru, če imajo vsi vojaki enake ali zelo raznolike sposobnosti, in drugo, ki spremeni generirano vrednost navzgor ali navzdol. S to smo lahko preizkusili, kolikšen vpliv imajo večje oziroma manjše vrednosti na končen izid.

Tako smo potem ustvarili osnovni razred *Actor*, na katerem temeljijo vsi različni tipi vojakov, kot sta na primer *Warrior* in *Archer*. Ta osnovni razred vsebuje parametre za razmišljanje. Zaradi očitnih razlogov smo se odločili, da bodo imeli določeno moč, spretnost in pogum. Poleg teh treh smo dodali tudi zdravje, saj je v večini primerov potrebno več udarcev, da vojak umre, in pa hitrost ter velikost, saj nimajo vsi vojaki enakega telesa. Kasneje smo še dodali voljo v obliki numerične vrednosti, ta pa predstavlja, koliko je določen vojak zavezan k skupnim ciljem in koliko se je pripravljen dalje boriti. S tem smo lahko tudi nekako ustvarili občutek, da se vojaki izmučijo in naveličajo ter izgubijo voljo do bojevanja. Poleg parametrov pa ima osnovni Actor implementirano tudi osnovno razmišljanje (izbiro tarče, premikanje) in izris. V nadaljnjih, specializiranih podrazredih se potem spremeni le način napadanja, prioriteta tarč in začetne vrednosti parametrov.

V prvi izvedbi je bilo celotno obnašanje definirano v vsakem podrazredu posebej. Izkazalo se je, da je bila stvar za različne vrste vojakov zelo podobna in smo zato potem ločili način napadanja od ostale logike, in samo logiko prestavili kar v osnovni razred. Uvedli smo le nekaj binarnih spremenljivk, ki vplivajo na kak specifičen del odločanja. En primer takšne spremenljivke je *close_range_square_danger* v primeru lokostrelca. Ker je ta zelo ranljiv na

manjših razdaljah, se njegov občutek varnosti zelo močno zniža, kadar je sovražnik tik ob njem. V primeru bojevnika pa je ta nastavitev izklopljena in se ne počuti tako ogroženega v bližini sovražnika.

Vsak posamezen vojak vsebuje eno spremenljivko za cilj, ki mu ga da igralec. Glede na bližino in nevarnost najbližjega sovražnika ter lastno voljo in zdravje se vojak odloči, ali bo šel proti cilju ali napadel sovražnika. Če se ne počuti preveč ogroženega, se ne bo menil za sovražnika. Manj, kot ima volje, bolj se bo počutil ogroženega in posledično bo večja možnost, da bo ignoriral ukaz in se umaknil. Ko je dovolj blizu cilja, se ta ponastavi in so vojakova dejanja spet v samodejnem načinu, kar pomeni, da če je kdo dovolj blizu za napad, bo tistega napadel, drugače pa bo stal na mestu in čakal na ukaz.

Kadar se vojak počuti zelo ogroženega in ni nikjer v bližini zaveznikov, se lahko zgodi, da bo obupal. V takem primeru se bo lahko začel umikati ali pa kar slepo napadel najbližjega sovražnika. Verjetnost da vojak obupa je povezana z njegovim pogumom in voljo, izražena s pogojem *nevarnost/varnost > 5+pogum+volja/2*. Pri lokostrelcih lahko to pomeni, da bodo pustili lok in sovražnika napadli kar od blizu.

8 Rezultati

Posamezen vojak ima 3 lastnosti: pogum, moč in spretnost. V igri so te lastnosti ovrednotene s števili od 1 do 10, pri čemer je 1 najslabša vrednost in 10 najboljša. Za potrebe testiranja smo vrednosti ene ekipe (modre) postavili na srednjo vrednost (5) in spreminjali vrednosti lastnosti druge (rdeče) ekipe. Ker tudi v resničnem svetu ne moremo vsega predvideti, smo v igro vključili tudi lastnost presenečenja in s tem igro naredili bolj realistično. Hkrati pa to pomeni, da gornje lastnosti pri istih vrednosti ne dajo vedno enakega izida.

V poskusih je posamezna ekipa imela po 24 vojakov – 12 mečevalcev in 12 lokostrelcev.

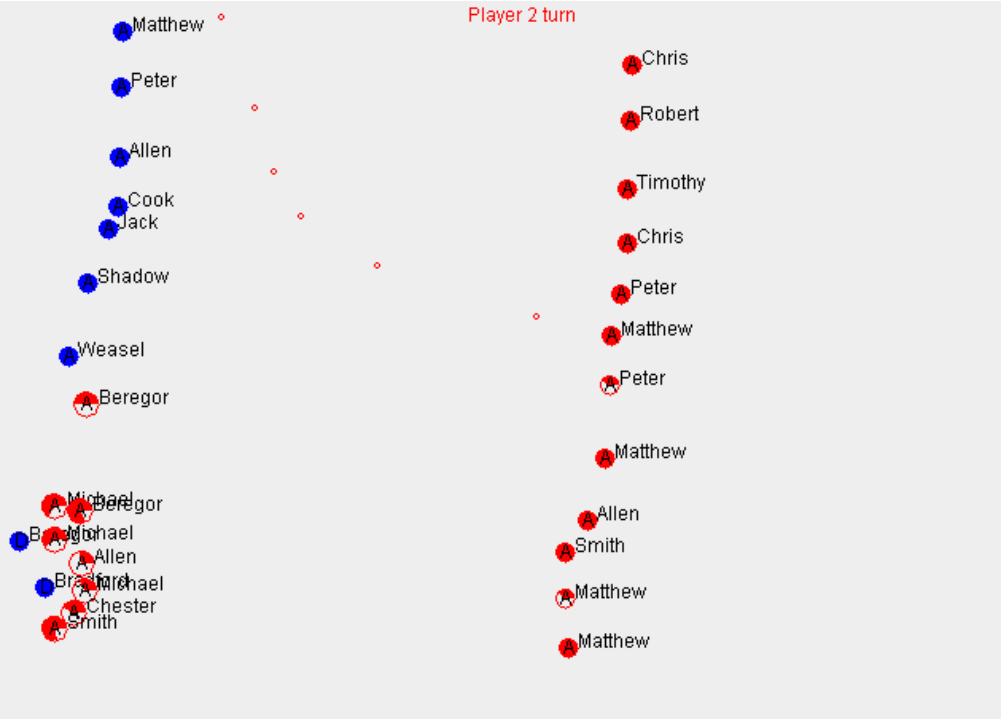
Pričakovani rezultati so naslednji:

Moč in spretnost imata vpliv na sposobnost bojevanja, medtem ko ima pogum vlogo pri odločanju v krizni situaciji (mnogo sovražnikov). Težko bi vnaprej predvideli, kako se bo kdo odločil v trenutni situaciji, če ne poznamo njegovih lastnosti. V tej točki smo prišli do problema, kako pravilno predvideti izid spopada glede na dane lastnosti vojaka. S tem tudi težko vnaprej ovrednotimo delovanje simulacije spopada. Ko med testiranjem nastavimo vse vrednosti na minimalne, je pričakovan izid popolni poraz, ko so pa vrednosti na maksimumu, pa bi morali zmagati. Vojak z veliko poguma in spretnosti bo nepremagljiv. Raznolike kombinacije lastnosti pa vodijo k izidu, ki ga je težko predvideti brez poznavanja psihologije. Predvidevamo pa, da bo z višjimi vrednostmi vseh lastnosti posameznega vojaka izid boljši.

Za vsako izbrano vrednost smo naredili po 3 poskuse in izračunali povprečje. Če bi vse lastnosti spreminjali v najmanjšem koraku (1), bi morali naredili 10^3 poskusov, in ker smo se odločili vsakega delati trikrat, bi to pomenilo 3000 poskusov. Po razmisleku smo se odločili narediti poskus le pri določenih mejnih vrednostih, na primer pri maksimalnem pogumu, moči ali spretnosti.

Glede na izkušnje in poznavanje realnosti so izbrane mejne vrednosti zelo netipične, predpostavljamo pa, da so možne. Težko bi našli osebo, ki bi bila pogumna, ampak bi ne imela nobenih spretnosti in moči.

Različica, kjer so imeli vsi vojaki enake sposobnosti, je izpadla malenkost bolj monotona v primerjavi z različico, pri kateri so bili vojaki raznoliki. Vsi so hodili v enakem tempu, bili enako učinkoviti v boju. Če smo samo malo povečali povprečje sposobnosti ene ekipe, je ta imela veliko prednost pred drugo ekipo.



Slika 3: Primer situacije, kjer imajo rdeči vojaki maksimalno prednost pred modrimi

Tabela: Rezultati testiranja vpliva lastnosti

pogum	moč	spretnost	preživeli	nasprotnih	preživeli	nasprotnih	preživeli	nasprotnih	preživeli AVG	nasprotnih AVG
0	0	0	0	24	0	24	0	22	0,0	97,2
0	0	10	20	0	14	0	18	0	72,2	0,0
0	10	0	0	16	0	20	0	11	0,0	65,3
0	10	10	24	0	20	0	22	0	91,7	0,0
10	0	0	0	23	0	24	0	24	0,0	98,6
10	0	10	4	0	11	0	19	0	47,2	0,0
10	10	0	0	20	0	18	0	13	0,0	70,8
10	10	10	24	0	23	0	24	0	98,6	0,0
5	5	5	8	0	0	6	0	7	11,1	18,1

Legenda:

- pogum (faktor med 0 in 10),
- moč (faktor med 0 in 10),
- spretnost (faktor med 0 in 10),
- preživeli AVG – povprečje preživelih v treh poskusih (v procentih),
- nasprotnih AVG – povprečje preživelih v nasprotni ekipi v treh poskusih (v procentih).

V poskusu, ko so bile lastnosti obeh ekip enake (v zadnji vrstici tabele), smo pričakovali izenačene rezultate, česar pa rezultati v tabeli ne kažejo. Sklepamo, da bi morali opraviti še več poskusov, da bi s tem zmanjšali vpliv naključnosti, ki smo jo dodali v igro. Zato tudi ostalim izidom ne moremo popolnoma zaupati.

Na podlagi rezultatov poskusov lahko sklepamo, da imajo ekipe, katerih vojaki so maksimalno spretni, ampak niso ne močni in ne pogumni, prednost pred tistimi, ki imajo maksimalno le moč oziroma pogum. Največji vpliv na izid spopada ima spretnost, sledi ji moč, šele potem pa pogum. Ekipa samih močnih in spretnih vojakov, tudi če so strahopetci, ima veliko prednost pred drugo ekipo, ki je sestavljeno iz samih povprečnih vojakov. Vojaki, ki so močni in pogumni, ampak ne znajo uporabljati orožja, so prav tako doživeli poraz. Večino rezultatov poskusov se da logično interpretirati v primerjavi z realnostjo.

Sklepamo, da igra daje rezultate, delno skladne s pričakovanji. Kljub temu smo optimistični in menimo, da bi lahko igro nadgradili v zahtevnejšo aplikacijo.

Kako bi uporabljene metode uporabili v drugih situacijah? (dijaki v razredu, ljudje v množici, požar, …)

S pravilno zasnovo bodo objekti – vojaki sprejemali bolj realistične – človeške odločitve in s tem vplivali na potek igre.

S pridobljenimi izkušnjami bi se dalo - z nekaterimi spremembami - izdelati preprosto simulacijo ene zgoraj navedenih situacij. Vendar bi se z upoštevanjem večjega števila parametrov tudi kompleksnost algoritma povečala. Pri tem pa je velik izziv, kako posamezen parameter ustrezno vstaviti kot matematično funkcijo v algoritem.

9 Družbena odgovornost

Upamo, da bo naša raziskovalna naloga koga navdušila, da bi sam poskusil ustvariti igro ali kakšno drugo preprosto simulacijo.

Z ustreznimi simulacijskimi programi bi se dalo predvideti nevarnosti in minimizirati posledice nepredvidljivih nesreč, npr. požar, poplave, potres.

10 Zaključek

S to raziskovalno nalogo smo se prvič soočali s programiranjem umetne inteligence. Sprva se je zdelo bolj preprosto, vendar smo se motili. Zares je zapleteno povezati vse faktorje, ki vplivajo na odločanje, le na podlagi matematičnih enačb, tako kot smo mi to poskušali. Na koncu je stvar izpadla drugače, kot smo pričakovali. Pri večjem številu vojakov se igralec veliko težje ravna po njihovih lastnostih.

Naloga je bila zgrajena na osnovi igre, kjer se borijo vojaki drug z drugim, vendar bi se dalo podoben sistem prilagoditi tudi za druge situacije oz. simulacije. S čim boljšim razumevanjem okoliščin se da izdelati tem boljši model za predvidevanje in tem bolje vključiti vse faktorje, ki vplivajo na izid nekega dogodka. S preprosto simulacijo ne moremo dobro predstaviti kompleksnega problema.

11 Priloge

Del izvorne kode

```
// metoda za ocenitev situacije
public void assess(ArrayList<Actor> units) {
	int lasttype = target.type;
	target.setnull();

	float targetdst = 100000;
	float friendlydst = 100000;
	Actor friendly = null;
	Vector2 dangerdir = new Vector2();
	float danger = 0;
	float safety = morale/10f;
	for(Actor a : units) {
		if(a == this) continue;
		float dst = distance(a);

		if(a.team == team) {
			// find closest friendly
			if(dst > stats.speed*3) continue;
			if(dst < friendlydst && (a.getClass().equals(getClass()))) {
				friendlydst = dst;
				friendly = a;
			}
			safety += (3-dst/stats.speed)*8;
		}
		else {
			// find best target
			float curdist = dst / getaggro(a);
			if(curdist < targetdst) {
				target.set(a);
				targetdst = curdist;
			}
			float d = (5-dst/stats.speed)*3 * getdanger(a);
```

```
                        if(d < 0) d = 0;
                        dangerdir.x += (a.posx-posx)*d/32;
                        dangerdir.y += (a.posy-posy)*d/32;
                        danger += d;
                        if(close_range_square_danger && dst < stats.speed)
                                danger += d*d/2;
                }
                reassesstime = System.currentTimeMillis() + 100;
        }

        // if target is enemy
        if(target.type == 2) {
                if(danger/safety > 1+stats.courage/2+morale/2) {
                        // can_desperate = 0,1
                        if(danger/safety > 5+stats.courage+morale/2) {
                                // desperation attack
                                target.type = 3;
                                if(lasttype != 3)
                                        nextattack = 0.1f;
                        }
                        // can_run = 0,1
                        else {
                                // run
                                target.set(new Vector2(posx-dangerdir.x, posy-
dangerdir.y));
                                if(friendly != null && friendlydst < stats.speed*3) {
                                        target.vector().add(friendly.posx-posx,
friendly.posy-posy);
                                }
                        }
                        reassesstime = System.currentTimeMillis() + 1000;
                }
                else if(run_to_same_class && targetdst > stats.size*5 && friendlydst <
targetdst && stats.size*1.3f < friendlydst && friendlydst < stats.size*4) {
                        // move to friendly
                        target.set(friendly);
                        reassesstime = System.currentTimeMillis() + 10;
```

```
                }
                else // attacking
                        reassesstime = System.currentTimeMillis() + 200 +
(int)(Math.random()*100);
        }
        // target is position or something
        else {
                reassesstime = System.currentTimeMillis() + 250;
        }
    }
```

12 Viri in literatura

Brigita Kotar, 2012. Resne igre. Diplomsko delo Univerza v Ljubljani 2012.

O javi (Internetni vir) https://www.java.com/en/about/ (9.1.2015)

Lastni viri (slike, tabela)