SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1208

3D računalna igra s elementima autonomnog ponašanja

Luka Hrabar

Zagreb, lipanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 8. ožujka 2016.

DIPLOMSKI ZADATAK br. 1208

Pristupnik: **Luka Hrabar (0036464514)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **3D računalna igra s elementima autonomnog ponašanja**

Opis zadatka:

Cilj diplomskog zadatka je osmisliti i u potpunosti implementirati 3D računalnu igru. Sadržaj igre je slobodan. Predviđeno je da igra treba sadržavati elemente autonomije u ponašanju likova (engl. non-player characters) te da se i samo polje igre i razmještaj objekata u njemu stvaraju algoritamski kako bi igra bila što manje predvidiva. U implementaciji je moguće koristiti neki od dostupnih pokretačkih sustava za igre ili zasnovati implementaciju direktno na normi OpenGL. Poželjno je da igra podržava više igrača u umreženom načinu rada.
Vaša je zadaća proučiti dostupne sustave za razvoj igara, osmisliti igru i u potpunosti je implementirati. Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 1. srpnja 2016.

Mentor:

Prof. dr. sc. Igor Sunday Pandžić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

SADRŽAJ

1. Uvod

1.1. Video igre

Video igra je igra na elektroničkom uređaju koji prima korisničke unose te vraća rezultate interakcije na nekom obliku video uređaja, primjerice na računalnom monitoru. Razvoj prvih video igara počeo je u drugoj polovici 20-og stoljeća, a najveći rast popularnosti dobivaju u posljednjih 15 godina. Danas su video igre jedna od najvećih grana industrije zabave te se projicira da će već 2016. godine svjetski prihod od video igara preći 100 mlrd. USD.

Umjetna inteligencija i video igre su nerazdvojni pojmovi već desecima godina. Od jednostavnijih igara za zabavu, do teških igara poput šaha i Go [1], umjetna inteligencija u video igrama je sve češća i razvijenija. Više ćemo o tome raspravljati u poglavlju 4.

Prije svega ćemo početi s uvidom u podrijetlo ove igre te proći kroz većinu pravila i mehanika. Zatim slijedi pregled implementacije same aplikacije, od glavnih sustava do poslova dretvi. U poglavlju 3 ćemo predstaviti generator mapa za igru te komentirati probleme i rješenja za uravnoteženo generiranje. Naposljetku ćemo razgovarati o implementaciji umjetne inteligencije u našoj igri i općenito - od razvoja do konkretnih algoritama i pravila za rješavanje problema i oblikovanje ponašanja *botova*.

1.2. OpenGL

Open Graphics Library (hrv. otvorena grafička biblioteka) je specifikacija koja opisuje višeplatformsko programsko sučelje (engl. *Application programming interface; API*) za rad s dvodimenzionalnom i trodimenzionalnom vektorskom grafikom. Otkad je predstavljen 1992. godine, OpenGL je postao najrašireniji API za 2D i 3D grafiku s preko 250 različitih funkcija koje sežu od osnovnih rasterskih funkcija do specijalnih efekata i drugih vizualizacijskih pomagala. Konkurira Microsoft Direct3D-u, a oba su

[1] https://deepmind.com/alpha-go

nezamjenjivi temelji za mnoštvo grafičkih aplikacija, posebice u industriji video igara, simulacijama i projektima virtualne stvarnosti.

Kako bi mogao ostati nezavisan o arhitekturi, jezicima i platformama, OpenGL ne pruža nikakvu podršku za zvuk, unose i baratanje prozorima već se za to treba pobrinuti na drugi način, primjerice bibliotekama kao što su OpenAL (Open Audio Library) i GLUT (OpenGL Utility Toolkit). Detaljnije o OpenGL može se pročitati u [3].

GLUT je jedna od prvih biblioteka stvorena kao dodatak uz OpenGL kako bi se olakšala izrada aplikacija koje nemaju pretjerano zahtjevnu funkcionalnost i opseg. Pruža osnovnu podršku za baratanje prozorima operacijskog sustava i alate za korištenje I/O (engl. *Input/Output* = *ulaz/izlaz*) na razini sustava. Uz navedeno pruža i funkcije za prikazivanje osnovnih geometrijskih tijela i oblika, ali to otprilike pokriva cijeli opseg njezine funkcionalnosti. Rad i održavanje ove biblioteke su prestali te je uglavnom nasljeđuju druge, od kojih je najpoznatija freeglut.

2. Igra

2.1. Podrijetlo igre

Naša je igra nastala po uzoru na Heroes of Might and Magic III: The Restoration of Erathia, a najbliže bismo poveznicu opisali kao "pojednostavljena verzija". Navedena igra je treći nastavak iz serijala Heroes of Might and Magic. Tematski, cijeli se serijal odvija u svijetu mitoloških bića, a većinu ćemo pravila i mehanika opisati u kasnijim poglavljima. Slika 2.1 je slika iz novoizdane originalne igre (preuzeto s [1]).

Slika 2.1: Slika originalne igre.

Sve igre iz serijala, pa tako i ova, su strategije potez-po-potez (engl. *turn-based strategy*). Općenito, turn-based strategije su igre podijeljene na runde u kojima svaki igrač ima po jedan potez tijekom kojeg su drugi igrači inaktivni. Do istovremenog igranja dolazi isključivo kada neki igrač tijekom svog poteza izvrši interakciju s likovima koji pripadaju nekom drugom igraču.

[1] https://www.ubisoft.com/en-AU/game/heroes-of-might-and-magic-3-hd/

2.2. Pravila igre

Igra se odvija na kvadratnoj mreži polja koju zovemo *mapa* (engl. *adventure map*). Igrači kontroliraju likove koji se zovu heroji (engl. *heroes*) te ih koriste kako bi istraživali mapu, skupljali resurse (engl. *resources*) te napadali protivnike. Neka polja na mapi su blokirana okolišem ili dijelovima građevina dok su neka polja pod napadom neutralnih stvorenja, a ostala polja su prazna ili interaktivna na neki način. Neutralna stvorenja su stvorena na mapi kako bi se igračima stvorio osjećaj napretka, kroz sve teže borbe od igrača se zahtjeva sve veća snaga vojske i heroji dobivaju sve više bodova iskustva (engl. *experience points*). Kretanje po mapi je ograničeno, svaki heroj dobije određeni broj koraka kretanja (engl. *movement points*) na dan (dan je ekvivalentan jednom krugu poteza svih igrača).

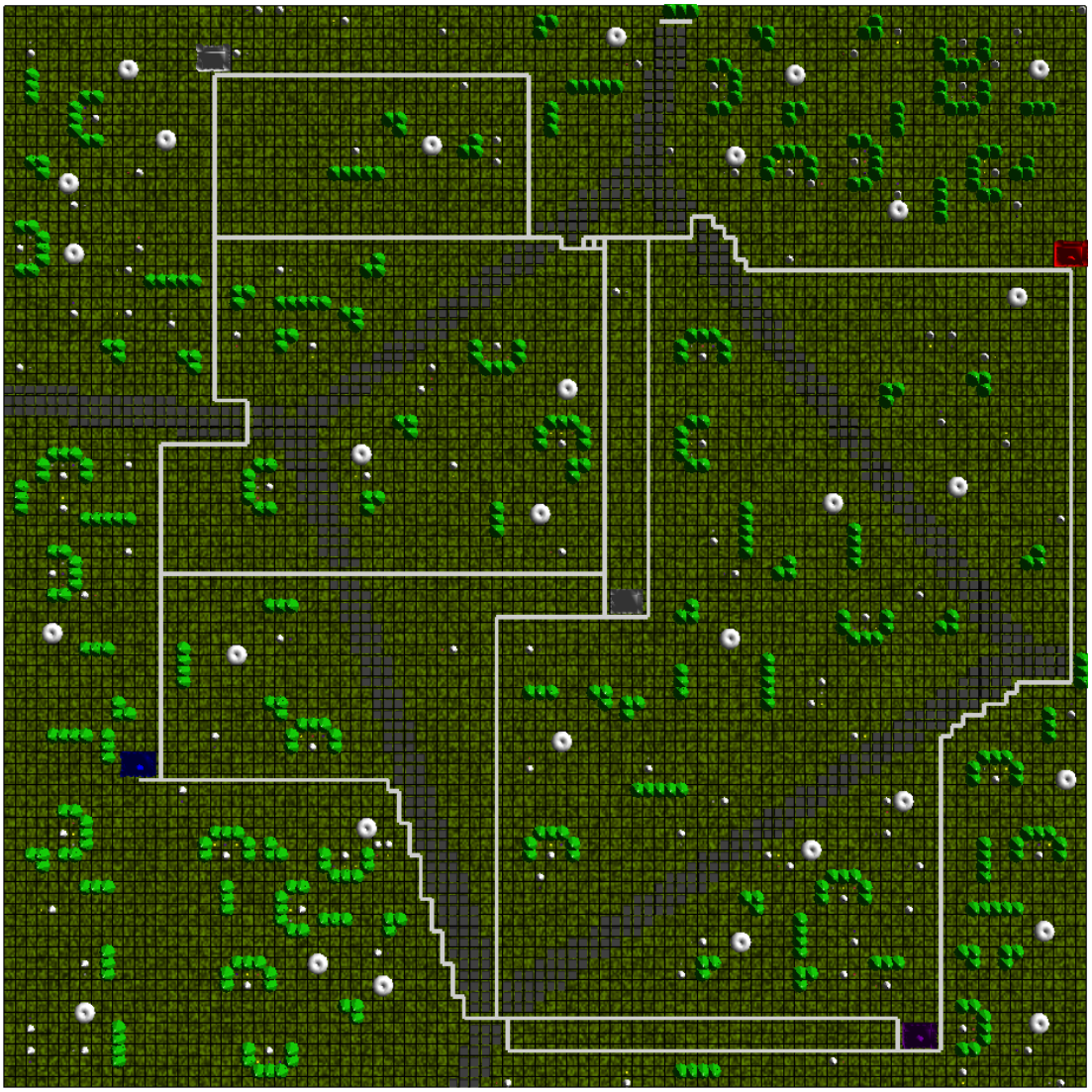
Igračima nije automatski vidljiva cijela mapa. Ukoliko igrač neko područje nije prethodno istražio ono će biti prekriveno maglom (engl. *fog of war*). Maglu otkrivaju heroji tijekom kretanja, ali samo na malom promjeru oko sebe.

Heroji uza sebe vode do nekoliko grupa borbenih jedinica (engl. *combat units*). To su grupe stvorenja s podacima o snazi u različitim aspektima borbe. Borba je mini-igra unutar velike igre. Na nešto manjoj pravokutnoj mreži polja se pozicioniraju borbene jedinice napadačke i obrambene strane. Heroji ne sudjeluju u borbi kao jedinice. Kretnjama sličnima šahu igrači približavaju svoje jedinice protivničkima te, ako dođu dovoljno blizu, napadaju. Igrač koji izgubi sve jedinice izgubio je i borbu.

Na mapi se pojavljuju građevine, u slučaju naše igre to su rudnici (engl. *mines*) i dvorci (engl. *castles*). Obje vrste građevina se mogu zauzeti herojem i pružaju neke funkcionalnosti igraču. Rudnik ima jednostavnu funkciju - svaki dan igraču koji ga kontrolira daje neku količinu resursa, ovisno o vrsti koja je predodređena za taj rudnik. U dvorcima se, jednom dnevno, mogu izgraditi dodatne građevine za regrutiranje borbenih jedinica ili za dobivanje resursa, a mogu se unajmiti i novi heroji. Građevine koje prodaju jedinice obnavljaju svoju rezervu jednom tjedno.

Postoje 4 vrste resursa: drvo, kamen, dragulji, zlato. Resursi se, u našem slučaju, troše isključivo u dvorcima. Zlato ulazi u sve cijene, a jedinice se kupuju isključivo zlatom. Drvo i kamen su jeftini materijali koji se često koriste u izgradnji, a dragulji su rjeđi i rijetko su potrebni. Sve 4 vrste mogu se pronaći u obliku kockica resursa na mapi te postoje rudnici za svaku vrstu.

Pobjednik je onaj igrač koji ostane zadnji nakon što svi ostali igrači izgube svoje dvorce i heroje. Ako igrač izgubi sve heroje, a kontrolira dvorce, još uvijek se može vratiti u igru ako regrutira novog heroja. Originalna verzija igre se od ove razlikuje



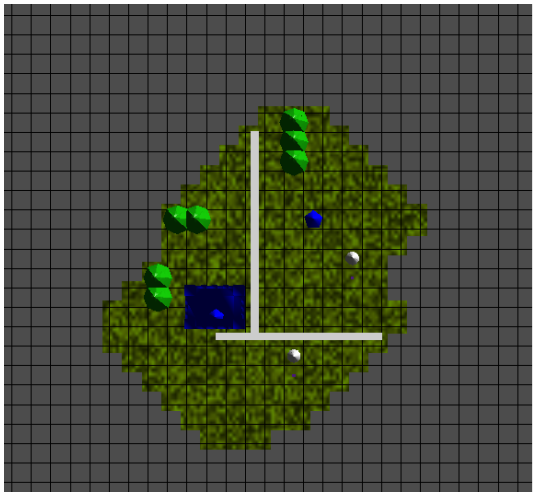
Slika 2.2: Udaljeni prikaz cijele mape za igru. (slučajno generirana)

uglavnom u broju različitih vrsta građevina i stvorenja koje postoje.

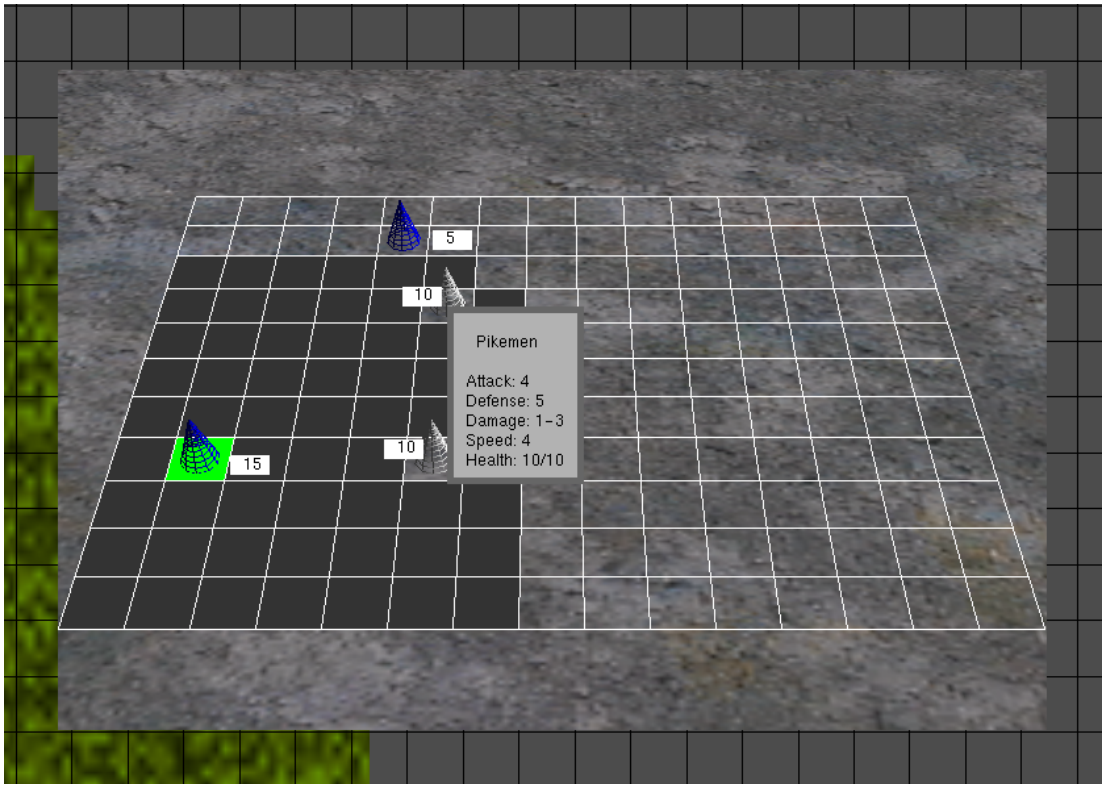
2.3. Pokretački sustav

Svaka video igra, odnosno svaka složenija grafička aplikacija, u izgrađena je na sustavu koji povezuje sve različite funkcionalne aspekte igre. Zovemo ga pokretački sustav za igre (engl. *game engine*) i općenito se sastoji od sustava za grafiku, fiziku, animaciju, AI, zvuk, i sličnih. U slučaju naše igre, ovaj sustav se sastoji od četiri dijela kao što je vidljivo na slici 2.6.

Neki od ovih modula su manageri, što znači da dodaju osnovne funkcionalnosti aplikacije koje nisu raspoložive u korištenim bibliotekama te nisu pretjerano povezani



Slika 2.3: Magla oko plavog igrača.



Slika 2.4: Borba između plavog igrača i neutralnih stvorenja na mapi. Vidljiv je i *tooltip* s podacima o jednoj od grupa stvorenja.

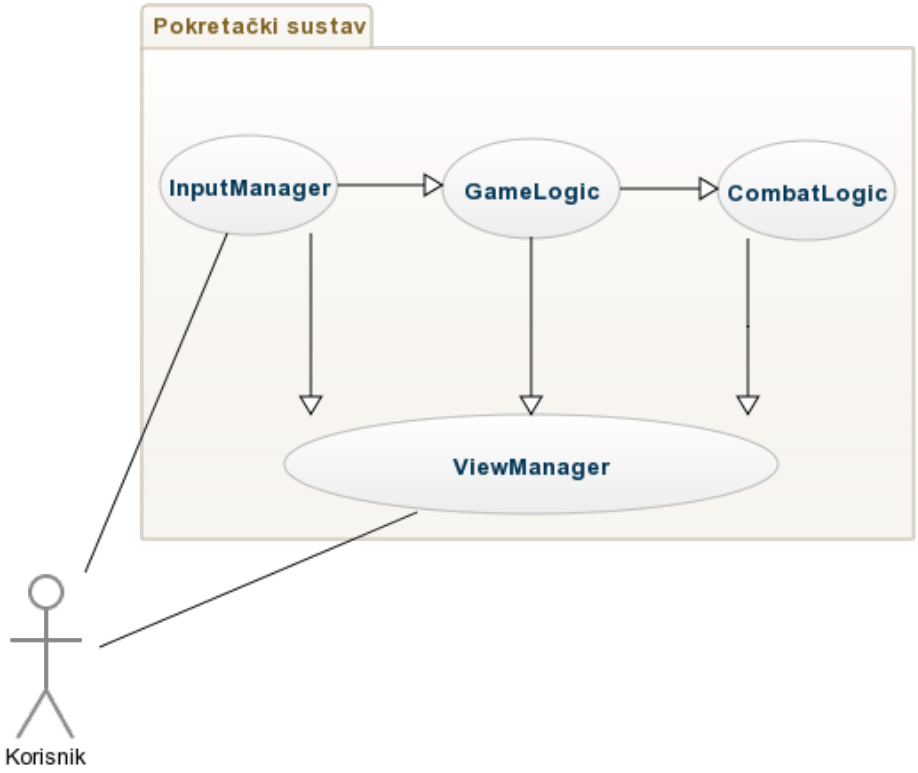
sa samom igrom. Ostali moduli su logike što znači da opisuju tok igre, pravila i mehanike. Dakle, manageri su u interakciji s korisnikom aplikacije dok logike "igraju" igru. Slijede opisi funkcionalnosti navedenih dijelova sustava i načina na koji oni međusobno surađuju tijekom pokretanja aplikacije.

Slika 2.5: Prikaz sučelja dvorca. Gornji dio sadrži listu građevina koje ovaj dvorac može sadržavati, druga građevina u prvom redu je izgrađena i može prodavati jedinice. Donji dio sučelja sadrži listu jedinica u dvorcu i listu jedinica na heroju koji je posjetio dvorac.

2.3.1. InputManager

Kao što se vidi iz poglavlja o OpenGL, ta biblioteka sama po sebi nema nikakvu funkcionalnost vezanu uz korisnički ulaz pa ćemo za pomoć koristiti GLUT. Za tipkovničke ulaze jednostavno mapiramo tipke na funkcije koje želimo da izvršavaju jer je jedini varijabilan podatak ime same tipke. Tipkovnica je korisna za otvaranje sučelja dvoraca i heroja te za druge jednostavne kratice (engl. *hotkeys*). Za složeniju interakciju kao što je pomicanje likova na mapi, pomicanje trupa u borbi ili napadanje drugih trupa u borbi potreban nam je nekakav kontroler, u našem slučaju miš. Uz navedeno moramo imati i način za interakciju s dijelovima korisničkog sučelja (engl. *UI*) gdje ćemo opet koristiti unos mišem.

Korisnički ulaz mišem podijeljen je na dva dijela, ovisno o vrsti projekcije koja je korištena u podprozoru gdje je očitan pritisak miša. Dakle, ako imamo "normalnu" (trodimenzionalnu perspektivu) projekciju potrebno je koristiti neku vrstu praćenja zrake (engl. *raytracing*) kako bismo otkrili koji je objekt korisnik želio odabrati. Za dvodimenzionalnu ortografsku projekciju je trivijalno očitati što se nalazi ispod pokazivača miša jer se koordinate direktno prevode u prostor podprozora. Stoga, cilj nam je



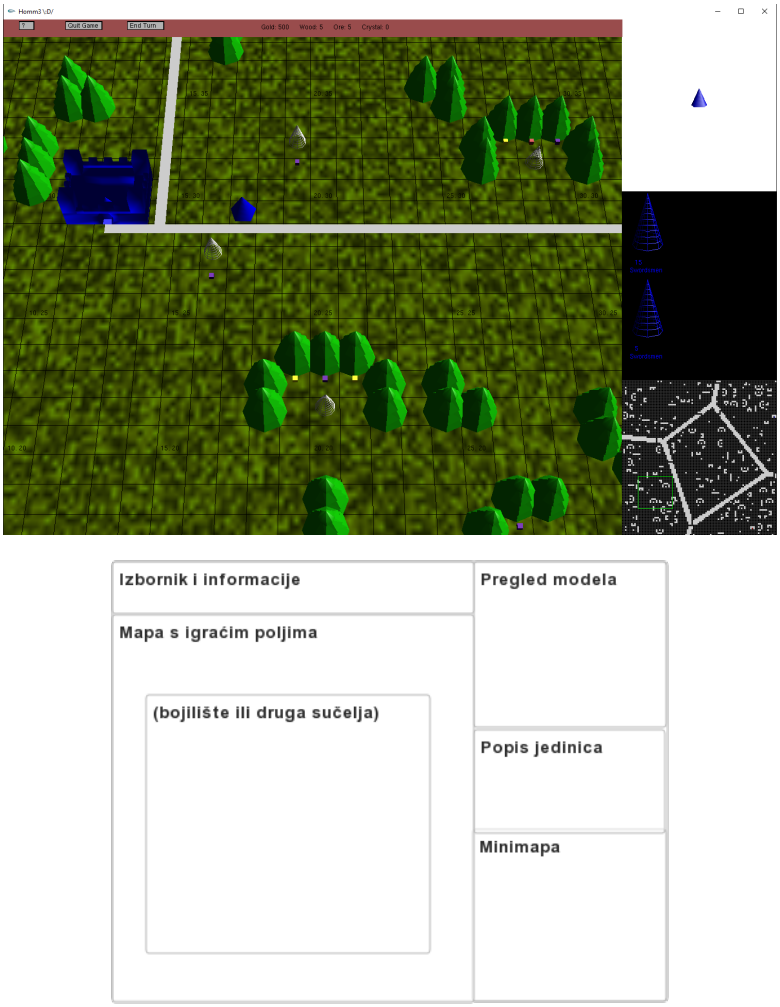
Slika 2.6: Dijelov našeg pokretačkog sustava.

bio što više sučelja u igri prikazati dvodimenzionalno i ispostavilo se da samo sučelje za borbu i mapa s objektima traže trodimenzionalni tretman. Izračuni za trodimenzionalni slučaj se mogu donekle pojednostavniti. Igraća mapa i prostor za borbu se zapravo nalaze na mreži polja te možemo ciljanje mišem svesti na određivanje koordinata polja koje se nalazi ispod pokazivača. Dodatno pojednostavljujemo izračun tako da mreže polja uvijek fiksiramo u ravninu z = 0 jer je tako lakše odrediti presjek zrake s ravninom. Na kraju je samo potrebno provjeriti je li odabrano polje legalan potez za trenutnu akciju kretanja po ploči ili u borbi.

InputManager koristi podatke o prozorima i kamerama iz ViewManagera kako bi odredio u kojem kontekstu treba obraditi korisnički unos. Ukoliko se ispod pokazivača nalazi više pogleda, primjerice kada je prikazano sučelje neke zgrade iznad igraće mape, očigledno ulaz gledamo u kontekstu sučelja jer je ono potpuno vidljivo. Rezultate interakcija uglavnom osvježuje u podacima koje čuva GameLogic.

2.3.2. ViewManager

Kao i za obradu ulaza, potrebna nam je poseban modul za baratanje prozorima, pogledima i kamerama u aplikaciji. Konkretno, aplikacija ima samo jedan pravi prozor operacijskog sustava i njega otvaramo i koristimo preko GLUT funkcija. Unutar prozora imamo sljedeću podjelu na poglede kako je vidljivo na slici 2.7.



Slika 2.7: Podjela prozora aplikacije na poglede (podprozore).

Kao što je spomenuto ranije, neke od ovih pogleda tretiramo trodimenzionalno, ali uglavnom prevladavaju dvodimenzionalni s UI elementima na koje ćemo gledati kao tipke. Sve poglede prikazujemo u okvirima u kojima su definirani (engl. *viewports*), za 3D koristimo perspektivnu projekciju, dok za 2D natpise i sučelja koristimo ortografsku projekciju. U slučaju otvaranja sučelja za bojilište, sučelja za dvorac ili sličnih sučelja, aktivira se pogled u sredini kako je prikazano na 2.7 te se iscrtava preko svih

pogleda koji su aktivirani ispod.

Za prikazivanje elemenata mape uglavnom korištene GLUT funkcije za geometrijska tijela. Objekti koji imaju složenije modele u obj formatu iscrtavaju se trokut po trokut s OpenGL funkcijama. ViewManager koriste drugi moduli kada je potrebno prikazati novo sučelje, najčešće kada se dogodi odgovarajuća interakcija na mapi. Prioritet prikazivanja pogleda je predodređen tako da je, primjerice, igraća mapa uvijek prikazana ispod svega ostaloga te igrač mora prvo zatvoriti sva druga aktivna sučelja kako bi pokrenuo svoje jedinice na mapi.

2.3.3. CombatLogic

CombatLogic je modul koji opisuje interakcije tijekom borbe (sama borba je opisana u pravilima igre). Koristeći BFS, algoritam opisan u poglavlju o pathfindingu, određuje se koja polja igrač može odabrati za kretanje ili napad. Modul čeka na primanje ulaza, odnosno na signale od InputManagera, te osvježuje stanje na bojištu. Ukoliko je jedan od sudionika borbe AI igrač, odnosno bot, koristi se implementirana AI logika kako bi se odredio sljedeći potez. Nakon kraja borbe se rezultati šalju u GameLogic - uklanjaju se heroji, objekti i trupe po potrebi te se dodjeljuju odgovarajuće nagrade.

2.3.4. GameLogic

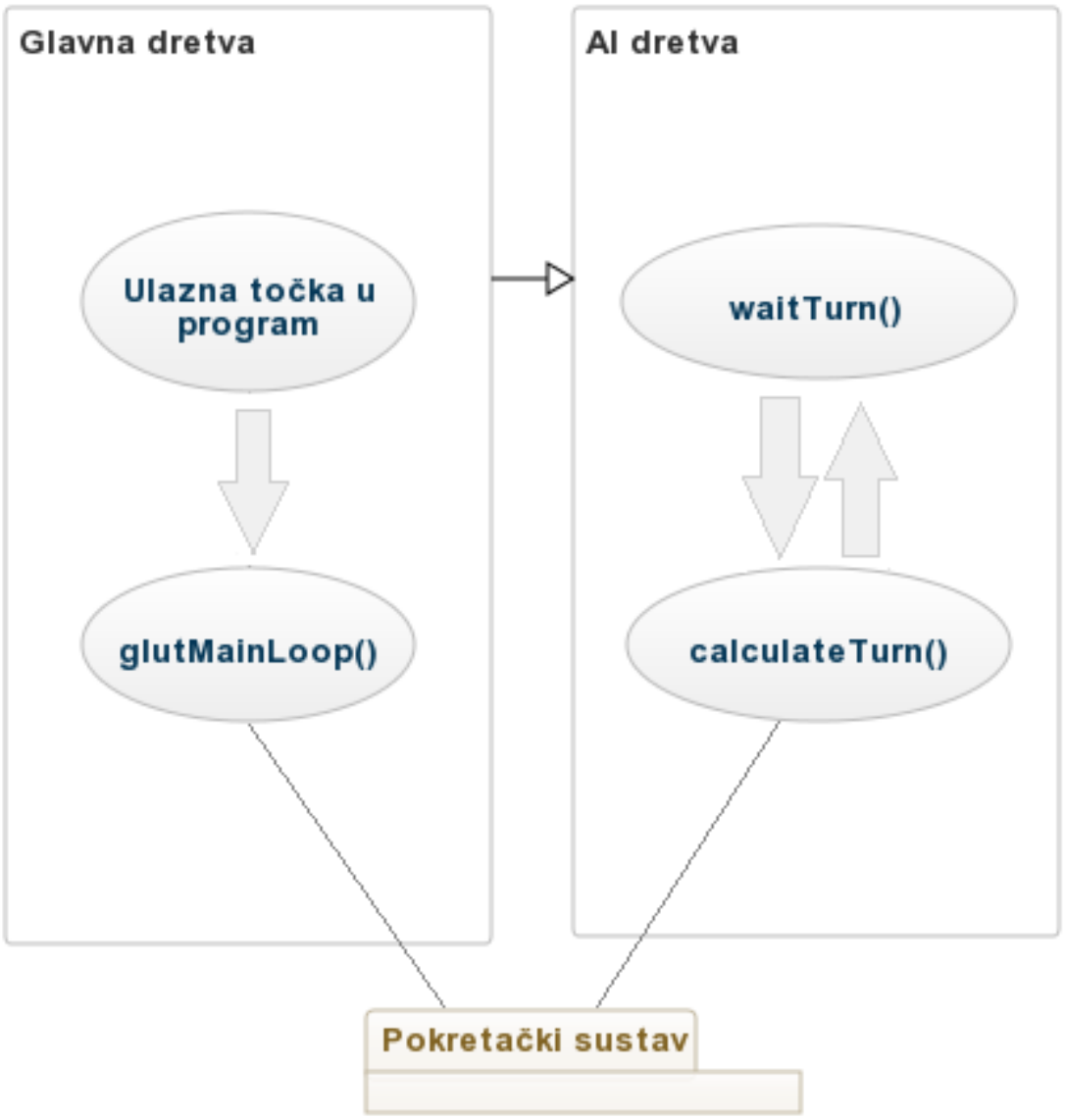
Ovo je najveći modul igre, barem u memorijskom smislu. Sadrži podatke o svim igračima i mapi te upravlja tijekom igre ovisno o korisničkim unosima. Općenito se brine o protjecanju poteza i rundi te podacima o trenutnom igraču, a za ljudskog igrača drži i informacije o legalnim potezima. Sva ne-borbena interakcija prolazi kroz ovaj modul.

2.4. Struktura aplikacije

Nakon što smo analizirali strukturu pokretačkog sustava naše igre možemo opisati rad cjelokupne aplikacije. Počevši s ulaznom točkom programa vrši se inicijalizacija kako slijedi:

- učitavaju se modeli objekata i teksture
- učitavaju se konfiguracijske datoteke za generiranje mapa te s informacijama o stvorenjima (engl. *creatures*)
- generira se mapa

– stvaraju se igrači s herojima i dodjeljuju im se početne zone i dvorci
– stvara se nova dretva koja će biti zadužena za AI igrače, kao što je vidljivo na slici 2.8
– glavna dretva prelazi u petlju definiranu u GLUT biblioteci - biti će zadužena za iscrtavanje i primanje korisničkog ulaza do kraja životnog vijeka aplikacije



Slika 2.8: Struktura aplikacije s prikazom dretvi i glavnim interakcijama između modula.

Glavna dretva dakle iscrtava podatke preko ViewManager modula, a korisničke ulaze šalje u InputManager modul na obradu. AI dretva čeka u inaktivnom stanju dok na potez ne dođe jedan od botova te se onda aktivira i počinje računati potez. Koristeći informacije iz GameLogic modula (više o tome u 4) odigrava potez te se vraća na

čekanje. Ovakva struktura je potrebna kako korisnik ne bi bio potpuno blokiran od interakcije s aplikacijom dok je bot na potezu te kako bi na ekranu uvijek bilo iscrtano trenutno stanje igre.

3. Generator mapa

3.1. Generiranje zona

Stvaranje zona temelji se na ideji Voronoi dijagrama. U matematici, Voronoi dijagram je podjela prostora u regije na temelju udaljenosti do nekog skupa točaka. Svaka točka iz skupa definira regiju koja obuhvaća sve točke prostora koje su bliže njoj nego ijednoj drugoj točki iz skupa.

Na slici 3.1[1] vidimo da obojane regije koje su nastale vrlo praktično i ravnomjerno dijele prostor. To je točno što nama treba za generiranje zona na mapi. Za neki dogovoreni broj zona Z generiramo toliko nasumičnih koordinata polja mape. Svaka od tih generiranih točaka će definirati jednu zonu te ćemo ih zvati središta zona. Sada, s obzirom na to da imamo diskretan prostor, mapu s relativno malim brojem polja, dovoljno nam je proći kroz sva polja i vidjeti kojoj zoni pripadaju te spremiti tu informaciju.

Tijekom određivanja pripadnosti polja zonama često nalazimo polja koja su jednako udaljena od više središta zona. Ta polja se nalaze na granicama i na njihovom ćemo mjestu stvoriti planine, odnosno zidove koji blokiraju prolaz herojima. Kako bi zone bile međusobno povezane neke od barijera moramo probiti pa ćemo izabrati neki broj prolaza i nasumično stvoriti toliko prolaza na granici svake dvije susjedne zone. Kako bismo osigurali uredno stvaranje zidova i prolaza proces ima još nekoliko koraka, ali nećemo ulaziti u detalje.

Kao rezultat generiranja zona dobivamo lijepe konveksne regije sličnih veličina, odvojene granicama, s uredno raspoređenim prolazima između njih. Za primjer pogledajte sliku 2.2.

[1] https://en.wikipedia.org/wiki/Voronoi_diagram

Slika 3.1: Primjer Voronoi dijagrama skupa od 20 točaka s Manhattan metrikom.

3.2. Očuvanje puteva

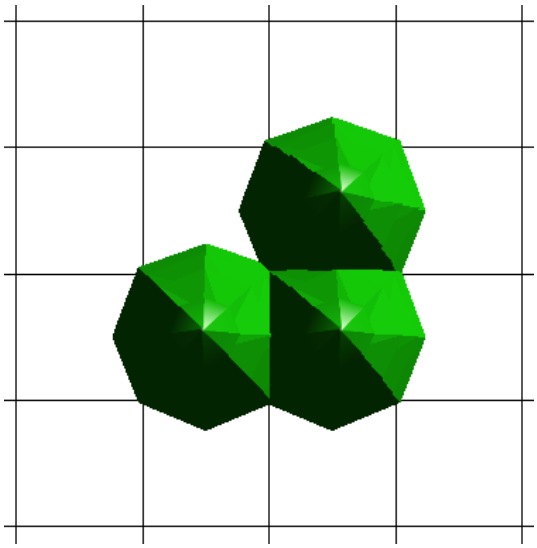
Kasnije ćemo u procesu generiranja dodavati i neke blokirajuće elemente poput zidova ili drveća. Kako bismo osigurali da sve zone ostanu povezane do kraja generiranja moramo se pobrinuti da sačuvamo barem jedan put između središta svake dvije susjedne zone. Najjednostavniji način je pomoću BFS algoritma (opisan u 4.2.2) odrediti najkraće puteve iz svakog središta zone do prolaza na granicama te zone. Zatim sva polja na tako stvorenim putevima označimo kao "prazna polja" koja se nikad ne smiju trajno blokirati tijekom generiranja mape.

3.3. Blok-generiranje objekata

Generiranje objekata na mapi temelji se na konfiguracijskoj datoteci. Datoteka sadrži blokove objekata koji se precrtavaju na mapu dok se određena zona ne popuni blokovima. Blokovi su kodirani po vrstama, a trenutno se kodiraju sljedeća svojstva:

- blok sadrži resurse - R (engl. *resources*)
- blok sadrži rudnik - M (engl. *mine*)
- blok brane stvorenja - C (engl. *creature*)

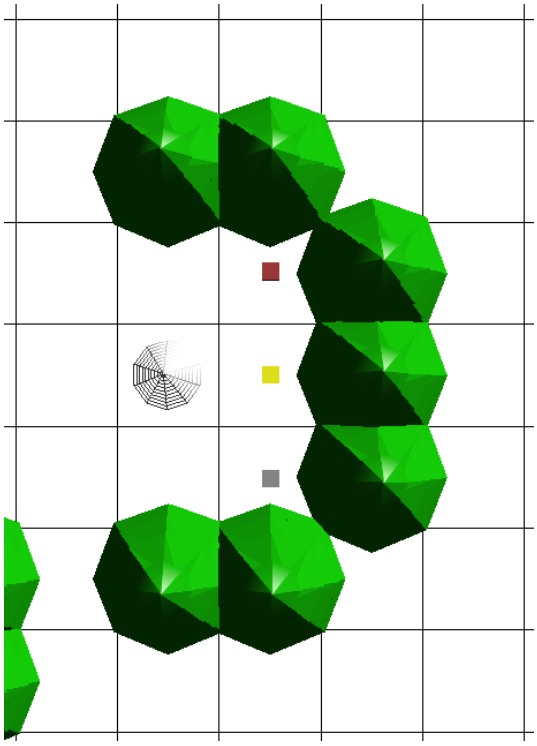
Primjeri blokova s kodovima se vide na slikama 3.3. Blokovi istog koda mogu varirati u veličini i bilo kojim drugim svojstvima sve dok su osnovna kodirana svojstva jednaka. Kodovi su zapravo zapisani binarno, ali prikazani su slovima zbog čitljivosti.



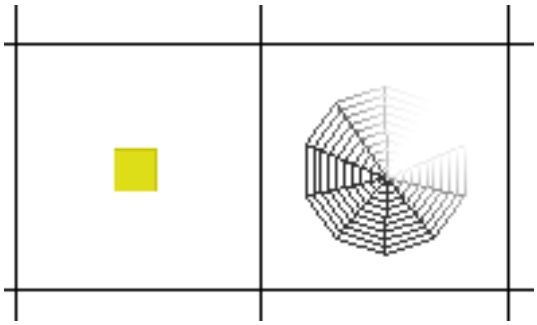
Slika 3.2: Kod bloka: - , binarno bi bio 000

Svaka bitna vrsta bloka, e.g. blokovi koji sadrže rudnik, ima predefiniran raspon broja pojavljivanja u zoni i prioritet generiranja u odnosu na ostale vrste blokova. Kao rezultat praktički je nemoguće, primjerice, da nema dovoljno mjesta za stvaranje minimalnog broja rudnika u nekoj zoni, jer blokovi s rudnicima imaju najviši prioritet. Najniži prioritet očito imaju prazni blokovi koji sadrže samo blokirajuća polja i druge neinteraktivne elemente okoline.

Postoji još jedno vrlo bitno, a praktično i jednostavno, svojstvo blokova koje nije očito sa slika. Svi blokovi u krajnjim redovima i stupcima, i.e. u okviru debljine jednog polja, uvijek imaju neblokirana polja. To direktno omogućuje da će svako neblokirano polje u svakoj zoni uvijek biti dohvatljivo iz središta zone! Zapravo, uvijek će postojati put (možda ponekad pod napadom čudovišta, ali nikad blokiran) između



Slika 3.3: Kod bloka: RC



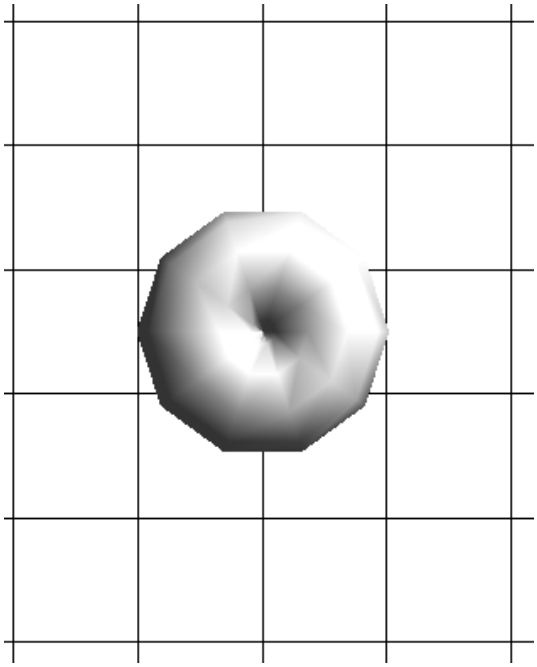
Slika 3.4: Kod bloka: RC

svaka dva neblokirana polja na mapi! Ovo svojstvo je vrlo poželjno i omogućuje da ovako generirana mapa nikad nema "mrtve" regije nego je cijela istraživa.

3.4. Postavljanje stvorenja

Tijekom blok-generiranja mape označili smo da se na nekim mjestima nalaze stvorenja koja čuvaju određene interaktivne objekte. Sada slijedi korak u kojem određujemo koja točno vrsta stvorenja i u kojoj brojnosti će se stvoriti na određenoj lokaciji.

Funkcija bi stvorenja stražara (engl. *creature guards*) trebala biti usporavanje napretka igrača tijekom istraživanja mape. Ako bismo posvuda stavili jednake čuvare



Slika 3.5: Kod bloka: M

igrač bi, nakon što dosegne određenu količinu snage, mogao proći prvog čuvara i sve ostale jednakom lakoćom. Ideja je postepeno otežavati probijanje kroz čuvare što igrač više istražuje mapu. To se u našem slučaju prenosi u ojačavanje čuvara kako se mičemo prema granicama između zona, odnosno udaljavamo od središta zona. To znači da ovisno o radiusu i, u manjoj mjeri, obliku zone raspoređujemo razine jačina generiranih stvorenja po zoni kako bismo dobili osjećaj postepenog napretka. Za brojnost stvorenja koristimo neke dogovorene raspone, a rasponi naravno opadaju s rastom jačine stvorenja.

3.5. Postavljanje građevina

3.5.1. Dvorci

Dvorci su postavljeni u središta zona koja smo definirali ranije kod generiranja zona. Razlog tome je što su zone zapravo generirane iz središta, a ne obrnuto, pa se iz takvih pozicija može, u prosjeku, brzo stići do svih polja u zoni. Očigledno je da bi medijan polja zone zapravo bio idealno središte u smislu udaljenosti prema ostalim poljima, ali zapravo ga nije bitno pomaknuti u medijan. U prosječnom se slučaju ne bi dobilo puno na udaljenosti prema ostalim polja, a izgubila bi se ključna funkcija magle - igrači bi vrlo jednostavno po obliku zone znali preciznu lokaciju dvorca makar im cijela zona,

osim zidova koji je odjeljuju od drugih zona i kraja mape, bila prekrivena maglom. Taj razlog je dovoljan da središta ostavimo kao središta Voronoi ćelija. Da bi se sada pronašle precizne lokacije dvoraca potrebno je poznavati lokacije svih graničnih zidova oko neke zone i lokacije svih dvoraca u zonama koji graniče sa spomenutom zonom.

Novi dvorci nemaju izgrađenu niti jednu građevinu niti imaju vojsku u rezervama. Početno su svi dvorci koji se nalaze izvan početnih zona igrača stavljeni pod kontrolu neutralne fakcije te ih igrači moraju zauzeti.

3.5.2. Rudnici

Rudnici služe kako bi unijeli temeljnu količinu resursa u svaku zonu. Potrebni su jer slučajno generirani resursi nisu dovoljni kako bi održali ravnotežu u igri - igrači trebaju imati jednaki pristup potrebnim valutama kako se bi dogodila nesprječiva nestašica i nemogućnost nastavka igre. Generiramo ih s obzirom na rijetkost pojavljivanja u slučajnom postavljanju resursa i s obzirom na potražnju u cijenama zgrada i jedinica. Dakle, drvo i kamen se najviše generiraju jer su najpotrebniji, zatim dragulji, zatim zlato. Zlato, iako je najpotrebnija valuta, najmanjeg je prioriteta u generiranju rudnika jer zona s rudnikom zlata dobiva vrlo veliku prednost nad zonama koje ga nemaju.

Konkretno, redoslijed generiranja je drvo - kamen - dragulji - kamen - drvo - zlato, čime se osigurava da samo najveća, ili više najvećih zona ako je mapa velikih dimenzija, dobije rudnik zlata. Spomenuli smo dodjeljivanje zona po veličini iz čega se zaključuje da se, zbog ovakvog načina generiranja, vjerojatno neće dogoditi da jedan igrač ima pristup rudniku zlata, dok drugi igrači nemaju.

3.6. Dodjeljivanje zona

Ravnoteža u igri (engl. *game balance*) je pojam koji se vrlo često pojavljuje u igrama svih vrsta pa tako i video igrama. Predstavlja ideju da bi svi igrači trebali biti u ekvivalentnoj početnoj poziciji i da bi im na raspolaganju trebale biti jednake mogućnosti kako bi igra bila poštena. Stoga, u slučaju naše igre, prvo smo se pobrinuli da su sva područja mape generirana na što je moguće sličniji ačin. To je glavni korak jer je preduvjet za uravnoteženu igru da svi igrači imaju sve potrebne elemente mape na raspolaganju.

Kako smo vidjeli ranije, prvi korak kod generiranja mape je generiranje zona. Svaki igrač će započeti u nekoj od zona te će dobiti dvorac u toj zoni. Sada moramo odrediti način dodjeljivanja početnih zona igračima tako da svi imaju što sličnije strate-

ške pozicije na mapi. U ovoj implementaciji smo koristili dva parametra za rangiranje početnih zona, a njihovi opisi slijede u 3.6.1 i 3.6.2.

3.6.1. Površina zone

Najočitiji čimbenik za rangiranje je površina zone, u našem slučaju broj polja koje ona sadrži. Veća površina, zbog načina na koji se generiraju elementi nakon razdvajanja zona, praktički osigurava da će se u zoni stvoriti svi elementi i resursi koji su zadani. Kako bi uravnotežili površine zona mogli bi doći na ideju da generiramo zone istih veličina te ih onda nekako spojimo u čitavu mapu. To je, u najmanju ruku, spor način da se igračima dodijele jednako velike zone, a kao rezultat bi dobili vrlo pravilne i dosadne oblike. Slična veličina dovoljno povećava vjerojatnost da će zone imati i sličnu zastupljenost i raspored objekata. Slijedi da početne zone ne moraju biti u polje jednake pa ćemo jednostavno pronaći N generiranih zona (gdje je N broj igrača) koje su najbliže po veličini te najsličnije po parametru opisanom u 3.6.2.

3.6.2. Prohodnost zone

Dijelovi jedne zone, odnosno objekti unutar zone, su dovoljno međusobno povezani te svi su dostupni iz glavne točke zone zbog načina generiranja mape. Ovdje govorimo o prohodnosti između zona - o dostupnosti drugih zona ako krenemo iz neke početne zone. Situacija igrača čija početna zona graniči (i.e. ima prolaze prema) s više susjednih zona očigledno je različita od situacije igrača koji počinje u kutu mape i ima pristup samo jednoj susjednoj zoni. Prva posljedica je da je prvi igrač izloženiji napadu iz susjednih zona, odnosno veća je vjerojatnost da će se drugi igrači slučajno ili namjerno naći blizu njegovog glavnog dvorca. S druge strane, prvi igrač ima više izbora kod istraživanja te ima veću vjerojatnost brzo pronaći nešto što mu je korisno. Ovakve pozitivne i negativne strane je teško uskladiti pa je jednostavnije zaobići takvu situaciju te igračima osigurati sličnu prohodnost zona. Stoga, ovo kao parametar znači da su sve dodijeljene zone povezane s nekim minimalnim brojem drugih zona s nekim minimalnim brojem prolaza, ali da nisu puno prohodnije od ostalih dodijeljenih zona. Ukratko, svi bi igrači trebali dobiti srednje-prohodne zone.

4. Umjetna inteligencija

4.1. AI u igrama

4.1.1. Razlog nastanka

Današnje najraširenije video igre uvelike duguju svoju popularnost mogućnosti mrežnog igranja. S druge strane, prve video igre su se pojavile dok internet još nije bio sveprisutan, odnosno nije bio raširen i svugdje dostupan kao što je to danas slučaj. Na osobnim računalima je mrežno povezivanje krenulo 1990-ih godina (primjerice Doom iz 1993. godine). Uzmemo li u obzir da su igre u početku bile raširenije na konzolama nego na osobnim računalima možemo spomenuti i da se prvi pokušaj povezivanja konzola preko interneta (engl. *online*) pojavljuje tek 1990. godine, a pravim se početkom mrežnog povezivanja konzola smatra Xbox Live mreža 2002. godine. Iz navedenog vidimo da su prvi igrači suigrače morali tražiti u neposrednoj okolini, ili su bili "osuđeni" na samostalnu zabavu. Suigrača, protivnika ili saveznika, predstavljala je sama igra. Stoga, uspjeh tadašnjih igara je velikim dijelom ovisio o sustavima "umjetne inteligencije" koji su predstavljali botove (kolokvijalizam za umjetne sudionike u video igrama). Botovi su se do danas održali kao glavni sudionici u igrama za jednog igrača (engl. *singleplayer*), odnosno sporedni sudionici u igrama za više igrača (engl. *multiplayer*). U igrama kao što je šah, umjetna će inteligencija (engl. *Artificial Intelligence; AI*) uvijek biti vitalan element te će se uvijek težiti poboljšanju takvog AI igrača.

4.1.2. Razvoj

Prvi AI u igrama se pojavljuje u obliku bota koji igra šah, Nim ili druge "misaone" igre, oko 50-ih godina prošlog stoljeća. Razvoj takve inteligencije je do danas usmjeren cilju da se napravi što bolji igrač određene igre. U industriji zabave vezanoj uz igre 70-ih se godina pojavljuje AI s drugom motivacijom - da ljudski igrač dobije dojam igranja protiv drugog čovjeka ili nečega što razmišlja "ljudski". Navedimo primjere

igara kao što su Space Invaders (1978.) i Pac-Man (1979.). U Space Invaders se pojavio AI koji reagira na unose (engl. *input*) igrača te su posljedično letjelice u igri razvijale zanimljive formacije i ostavljale dojam individualnog ponašanja. Pac-Man je koristio jednostavan algoritam za prolaženje puteva uz dodatak slučajnog odabira puteva kako neprijatelji ne bi bili predvidljivi i dosadni. Dojam "razmišljanja" botova je pojačan kada se uzme u obzir da je tvorac originalne igre svakom čudovištu dao vrlo jednostavna individualna pravila. Obje igre su u osnovi koristile vrlo jednostavne algoritme, ali postignut je željeni učinak - igrači tih igara su prepoznavali "ljudske" uzorke u ponašanju botova te su čak racionalizirali njihove nasumične kretnje.

Detaljnije o razvoju i nastanku AI u igrama može se naći u [2]. Uz navedeno se spominje i pregršt algoritama i ideja za sve aspekte AI u igrama, od traženja puteva do sustava učenja i zaključivanja. Ovo je bio glavni izvor inspiracije za naš rad.

4.2. Implementacija

4.2.1. Pathfinding

Možda glavna poveznica svih AI sustava koji se pojavljuju u igrama s pokretnim likovima je njihov sustav za traženje puteva (engl. *pathfinding*). Složenosti ovakvih sustava uvelike ovise o primjeni, a možemo ih podijeliti na dvije vrste:

- sustavi za pathfinding koji isključivo koriste algoritme iz teorije grafova kako bi riješili probleme vezane uz puteve i putanje likova
- sustavi koji u traženje puteva implementiraju dodatne parametre ili složene heurističke algoritme kako bi iskoristili sve na raspolaganju i pronašli najbolje radnje

Druga vrsta sustava je očigledno nešto složenija i sastoji se od kombinacije odlučivanja i traženja puteva. Mi smo koristili prvu vrstu, s odvojenim sustavom za odlučivanje o kojemu ćemo više pričati u 4.2.4.

Bez pretjeranog ulaženja u terminologiju teorije grafova, pathfinding sustave možemo raščlaniti na sljedeće dijelove, odnosno glavne funkcije: određivanje legalnih pozicija u igri i prijelaza između njih - konkretno, u kontekstu ove igre govorimo o slobodnim poljima na mapi i prijelaze u susjedna polja, gdje slobodna polja predstavljaju polja bez zidova i drugih objekata koji blokiraju kretanje određivanje "cijena" za prijelaze, bilo vremenskih ili materijalnih ili nekih drugih, koje predstavljaju trošak za kretanje - u našem slučaju ćemo govoriti o dnevno dodijeljenoj količini kretanja za

likove mjerenoj u prijelazima između susjednih polja korištenjem odgovarajućeg algoritma iz teorije grafova izračunati podatke o udaljenosti i dosezljivosti pozicija koje nas zanimaju iz pozicije u kojoj se neki lik trenutno nalazi - spomenuti ćemo ukratko BFS (eng. Breadth-First Search), algoritam korišten u ovoj igri pomicanje likova i ponovno izračunavanje puteva, udaljenosti i dosezljivosti, kada je potrebno, ovisno o događajima u igri i drugim dinamičnim elementima igre koji traže ponovno razmatranje kretanja.

Ostali se pokretni likovi u igrama obično kreću po predodređenim rutama. Takvo ponašanje je jednostavno implementirati i idealno je za animiranje sporednih likova i događaja. Ali, čim u priču uđu složeniji i bitniji pokretni objekti moramo imati i prikladan sustav za pathfinding jer predodređene rute nisu pogodne za interakciju. Primjerice, javljaju se problemi i nepoželjno ponašanje ukoliko se nešto stvori na putu, a usto predodređene rute mogu izgledati vrlo nezanimljivo i neprirodno, posebice ako su pridjeljene "živim" objektima.

4.2.2. Algoritam BFS

Jedan od najjednostavnijih (efikasnih) algoritama za traženje puteva u grafu je zasigurno pretraživanje u širinu (engl. *Breadth-First Search; BFS*). Ime je dobio zbog načina rada - počevši iz početnog stanja ujednačeno se širimo po granici između obrađenih i neobrađenih stanja, obrađujući prvo sva stanja na udaljenosti k prije nego obradimo ijedno stanje na udaljenosti $k + 1$. Pod udaljenost do nekog stanja naravno mislimo na duljinu najkraćeg puta od početnog stanja do tog stanja. Također, nadalje ćemo za stanja ili pozicije ponekad koristiti termin čvorovi, a za prijelaze termin bridovi, kako bismo bili u skladu s terminologijom teorije grafova. Opišimo ukratko kako radi algoritam BFS (čitati uz sliku 4.1):

(a) počinjemo stavljanjem početnog čvora s u red (engl. *queue*) za obradu te mu pridružujemo udaljenost 0 jer se radi o početnom čvoru; čvor s je obojan sivo jer je pripremljen za obradu

(b) obrađujemo čvor s tako da sve njegove bijele (nisu obrađeni niti pripremljeni za obradu) susjedne čvorove, dakle w i r, stavimo u red za obradu i pridružimo im udaljenost $dist(s) + 1$; čvor s je obojan crno jer je sada obrađen

(c) obrađujemo čvor w jer je prvi u redu za obradu - čvorove t i x stavljamo u red i spremamo njihovu udaljenost kao $dist(w) + 1 = 2$; čvor w je obrađen

(d) obrađujemo čvor r, posljedično u red dodajemo čvor v

(e) obrađujemo čvor t, posljedično u red dodajemo čvor u

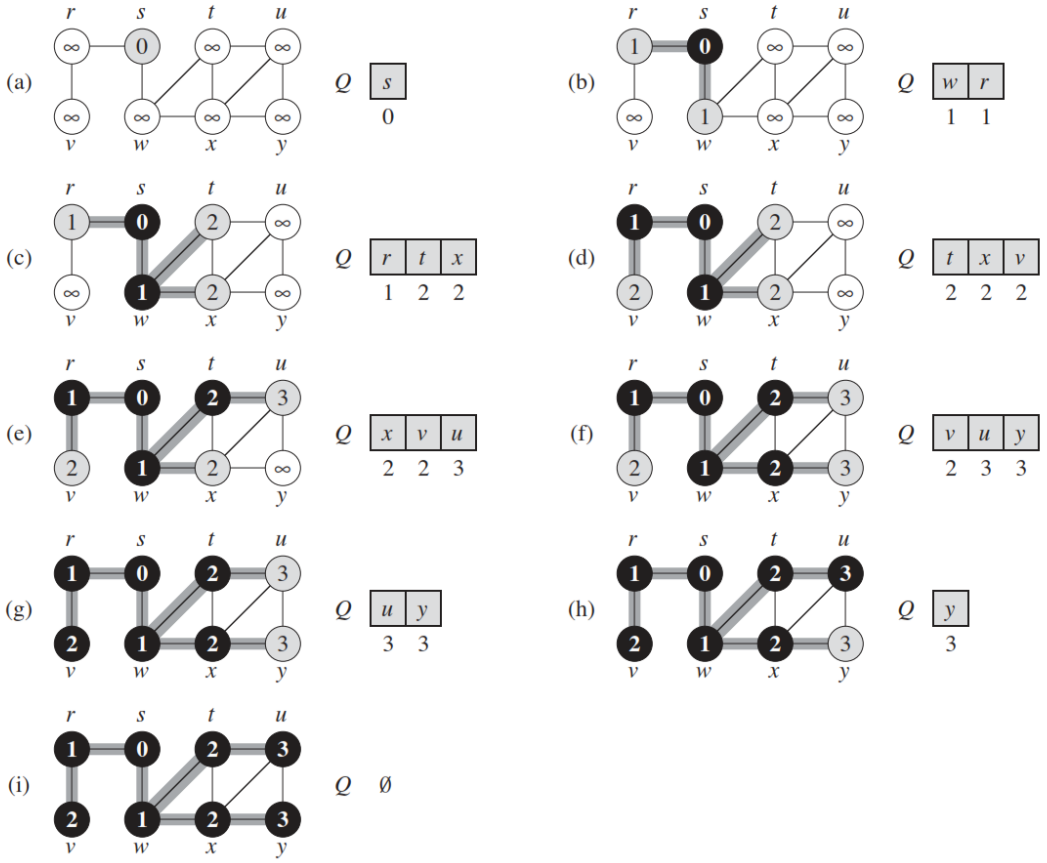
(f) obrađujemo čvor x, posljedično u red dodajemo čvor y

(g) obrađujemo čvor v, nema bijelih susjednih čvorova

(h) obrađujemo čvor u, nema bijelih susjednih čvorova

(i) obrađujemo čvor y, nema bijelih sujednih čvorova; red je prazan, gotovo je izvršavanje algoritma

Nakon što je izvršavanje algoritma gotovo imamo spremljen niz udaljenosti $dist$ koji označava najkraće udaljenosti do svih čvorova. Ukoliko bismo htjeli imati mogućnost reprodukcije najkraćih puteva samo je potrebno za svaki čvor spremiti i informaciju o čvoru koji ga je pogurnuo u red za obradu.



Slika 4.1: Primjer izvršavanja BFS algoritma. Preuzeto iz [1]

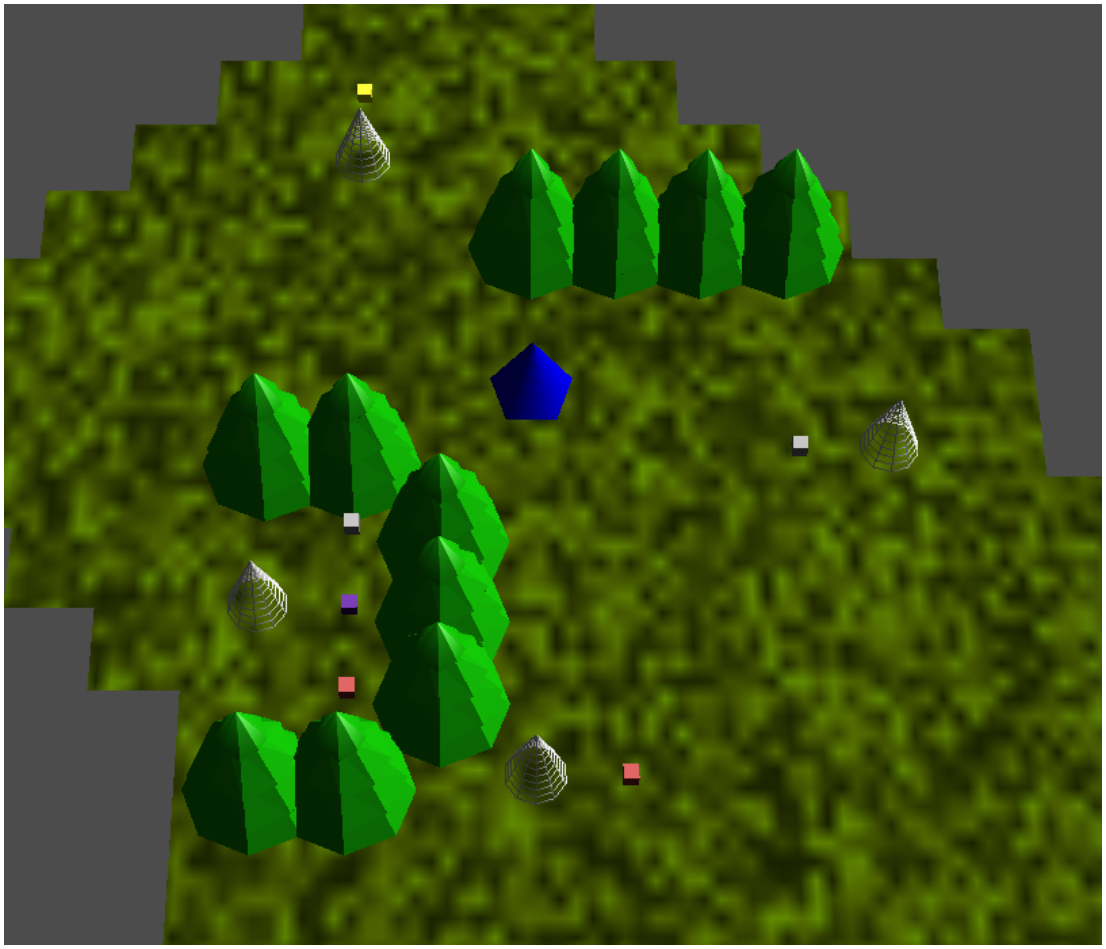
Uz pretpostavku da su cijene svih prijelaza između stanja, odnosno pozicija u kojima se nalazimo, jednake, odnosno mogu se svesti na jediničnu cijenu, ovaj deterministički algoritam može u jednom prolazu kroz sve pozicije (i.e. u linearnoj složenosti) pronaći najkraće puteve iz trenutnog stanja u sva druga stanja. To je teoretski najniža granica složenosti za spomenuti problem pa je stoga ovaj algoritam nezamjenjiv ako su ispunjeni spomenuti uvjeti. Više o matematičkoj pozadini iza ovoga algoritma može se pronaći u [1].

4.2.3. Implementacija pathfindinga

Očigledno je zašto smo odabrali baš BFS za računanje mreže puteva - kretanje po poljima se odvija samo između susjednih polja i cijena za svaki prijelaz je jedna jedinica kretnje (engl. *movement point*). Na slici 4.2 vidimo stanje na mapi u nekom trenutku igre, a na slici 4.3 vidimo koja su polja dostupna, u kontekstu prepreka, i dovoljno blizu da bismo se do njih pomaknuli herojem. Ovakav izračun se koristi za pomicanje ljudskog igrača, odnosno za provjeru je li mišem uneseno valjano polje za kretanje.

Za pokretanje AI igrača potrebno nam je još podataka jer ovakva pretraga pronalazi samo polja do kojih možemo doći bez da očistimo ikakve blokade. Primjerice, ako bismo na osnovi 4.3 pokušali izračunati kretnje bota znali bismo za postojanje samo dvije kockice resursa kako je zaokruženo na slici. Dakle, to ne bi bilo vrlo efikasno jer ne bismo mogli odrediti u kojem smjeru se isplati čistiti put - efektivno bismo imali "kratkovidnog" bota koji bi iscrpio lokalne mogućnosti i onda "na slijepo" napadao blokirajuća stvorenja.

Stoga, pokrećemo još jednu BFS pretragu kako bismo saznali što je sve moguće dosegnuti kada bi nam jedini faktor sputavanja bili nepomične prepeke i magla. Pod nepomične prepreke smatramo zidove i drveće te dijelove građevina (polja) na koje se ne može stati herojem. Nova vidljiva mjesta bi izgledala kao na slici 4.4 te bismo vidjeli sve zaokružene resurse. Potrebno je napomenuti da smo u ovome primjeru koristili malo ograničenje udaljenosti kako bi mogli pokazati razliku između dva izračuna, inače se za vidljivost objekata koristi veća udaljenost tako da možemo planirati skupljanje objekata koji su nekoliko poteza daleko. Dakle, zato na slici 4.4 najniža u grupi od 3 kockice resursa nije zaokružena, nalazi se korak predaleko od trenutne pozicije heroja.



Slika 4.2: Trenutno stanje na mapi: plavi heroj, drveće (stalne prepreke), stvorenja koja priječe prolaz (bijeli žičani stošci; privremene prepreke), i resursi koje je moguće pokupiti (kockice).

4.2.4. Odluke i ciljevi

Botovi u našoj implementaciji stvoreni su s namjerom da što bolje oponašaju ljudskog igrača i donošenje odluka. Istovremeno smo htjeli da sve odluke budu jednostavno rastavljive na osnovne elemente kako bi se moglo prirodnije parametrizirati neke karakteristike, primjerice agresivnost. Razmišljajući o strategijama koje ljudi koriste igrajući ovu igru možemo nabrojati par pitanja koja se javljaju:

- Gdje možemo pronaći resurse?
- Kako možemo istražiti najviše područja?
- Postoje li neke građevine u blizini koje možemo zauzeti?
- Možemo li pobijediti borbu protiv neke vojske?
- Možemo li unaprijediti svoje dvorce?
- Možemo li unajmiti još heroja ili kupiti još vojske?

Slika 4.3: Prikaz direktno dohvatljivih polja kao rezultat BFS algoritma. Prisjetimo se da stvorenja na mapi napadaju polja oko sebe i tako zaustavljaju heroje koji stanu na ta polja.

Navedena pitanja kao odgovore povlače radnje koje možemo podijeliti na sljedeći način:

- brze radnje: regrutiranje heroja, kupovanje stvorenja (borbenih jedinica), unaprjeđivanje dvorca
- spore radnje: skupljanje resursa, zauzimanje građevine, borba, prebacivanje vojske, istraživanje mape

Brze radnje možemo izvršiti bilo kad, uz pretpostavku da možemo platiti cijenu toga što kupujemo - primjerice, za regrutiranje heroja moramo imati dovoljno zlata i barem jedan dvorac. Takve radnje ne ovise o trenutno aktivnom heroju. S druge strane, za sve spore radnje moramo se kretati nekim herojem po mapi kako bismo došli do odgovarajućeg objekta - primjerice, za skupljanje resursa moramo doći na polje gdje se taj resurs nalazi i pokupiti ga.

Za brze radnje koristimo jednostavna pravila koja određuju kad će se koja radnja izvršiti. Primjerice, heroje možemo kupovati ako imamo dovoljno novaca nakon što smo odlučili želimo li unaprijediti dvorce te je to najčešće dobra strategija. Vrlo jed-

Slika 4.4: Prikaz dohvatljivih polja do udaljenosti 5 uz ignoriranje svih privremenih blokada (u ovom slučaju stvorenja).

nostavno možemo unaprijediti logiku s pravilom poput "Nemoj kupiti heroja ako ih imaš više od 5, a još nije prošlo 2 tjedna igre". Na taj se način bot jednostavno može poboljšavati i dalje, ali zasad implementacija ne koristi pretjeran broj ovakvih pravila.

Spore radnje se velikim dijelom oslanjaju na pathfinding logiku opisanu u 4.2.3. Slijedi okvirni opis postupka kojeg će bot izvršiti za svakog svog heroja kako bi mu odredio radnje za sljedeći ili više sljedećih poteza.

1. Pomoću BFS algoritma pronađi sve resurse, građevine i neprijateljske heroje koje ovaj heroj može teoretski dosegnuti (ukratko: ciljni objekti), odnosno postoji neki put do njih koji ne prolazi kroz nepomične prepreke niti maglu.

2. Stvori vektor kandidata za ciljeve - u njemu ćemo držati sve ciljeve koje bi potencijalno mogli zadati trenutnom heroju uz ukupnu udaljenost koju heroj mora proći da bi izvršio cilj.

3. U vektor kandidata ubaci sve ciljne objekte do kojih heroj može neometano doći

(neće naletiti na borbu na putu) te im pridruži udaljenost od heroja.

4. U vektor kandidata ubaci sve ciljne objekte do kojih se heroj može uspješno probiti te im pridruži udaljenost probijanja. Probijanje se računa na osnovi dvaju nizova s udaljenostima iz 4.2.3 - pronađemo najbliže polje ciljnom objektu na koje možemo doći bez borbi te se onda probijemo do kraja (udaljenošću probijanja zovemo zbroj tih dviju kretnji).

5. Za ostale ciljne objekte izračunamo koliko vojske nedostaje na heroju kako bi se mogao uspješno probiti. Svaki ciljni objekt za koji postoji neki dvorac u kojem možemo regrutirati dovoljno vojske za probijanje ubacujemo u vektor kandidata. Kao udaljenost objektu pridružujemo zbroj udaljenosti putovanja do optimalnog dvorca i udaljenosti probijanja od tog dvorca do ciljnog objekta.

6. Sve kandidate sortiramo po ukupnoj udaljenosti te heroju zadajemo prvi ciljni objekt. Ako je potrebno više radnji kako bi se došlo do tog ciljnog objekta, sve radnje su spremljene po redu i heroju se zadaje prva na izvršavanje.

Sve međuradnje koje su iznad samo spomenute, a nisu objašnjene, obično se svode na parametrizirane procjene situacije. Primjerice, ako bot ima priliku napasti neprijateljskog heroja, a botu smo dodijelili veliku agresivnost - mislit će da će napad uspjeti ako ima barem 10% jaču vojsku nego neprijateljski heroj, gdje je snaga vojske određena drugim parametrima.

Kao rezultat cjelokupne strategije pokušaja prevođenja ljudskih smjernica u sustav odlučivanja, bot pokušava donositi odluke na način sličan ljudskim igračima, ili barem neiskusnim ljudskim igračima. Ne koristi nikakve dodatne informacije koje čovjek ne bi imao i svi njegovi potezi se mogu jednostavno racionalizirati. Naposljetku, ovakvog bota je zanimljivo promatrati i uvijek se mogu dodati nova pravila kako bi ga poboljšali, a učinak je vrlo dobar makar je "inteligencija" iza svega poprilično jednostavna.

Ponašanje botova u borbi ćemo samo ukratko opisati. Vrlo je jednostavno i maksimalno agresivno - bot svoje jedinice prije borbe podijeli u grupe ovisno o odnosu snaga između sučeljenih vojski te se sa svakom grupom uvijek kreće prema najbližoj neprijateljskoj grupi. Postoje mnoge strategije koje bi se mogle implementirati za poboljšanje borbe, ali neke mehanike borbe iz originalne igre nisu prenesene pa je ova strategija trenutno dovoljno dobra.

5. Zaključak

Kao rezultat rada stvorena je 3D računalna igra s elementima autonomnog ponašanja. Autonomno ponašanje je implementirano u obliku botova koji pokušavaju igrati igru s idejama sličnim ljudskima. Implementirani botovi koriste sve mehanike igre te se većina njihovog ponašanja može jednostavno modificirati.

Postoji mnogo aspekata igre koji bi se mogli unaprijediti. Sljedeći je korak dodati ostale mehanike iz originalne igre i prilagoditi botove kako bi ih znali koristiti. Postojeće ponašanje botova se uvijek može poboljšati dodavanjem novih pravila zaključivanja. U krajnjem slučaju, potrebno je poraditi i na vizualnom aspektu igre dodavanjem više korisničkih sučelja, a i modela i tekstura za objekte.

LITERATURA

[1] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[2] Ian Millington i John Funge. *Artificial intelligence for games*. CRC Press, 2016.

[3] Dave Shreiner, Bill The Khronos OpenGL ARB Working Group, et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education, 2009.

3D računalna igra s elementima autonomnog ponašanja

Sažetak

Rad opisuje 3D računalnu igru stvorenu po ideji nekoć popularne strateške igre Heroes of Might and Magic III. Igra se odvija na potez-po-potez logici, a svaki igrač tijekom svog poteza upravlja herojima i vojskom te izgrađuje dodatne građevine u svojim dvorcima. Implementacija pokretačkog sustava za igru je zasnovana na OpenGL i GLUT bibliotekama. Mape za igru su automatski generirane slažući predefinirane blokove objekata, uz korištenje određenih parametara kako bi svi igrači bili u ravnopravnim pozicijama. AI igrači, odnosno botovi, koriste kombinaciju baze pravila i sustava za traženje puteva kako bi isplanirali sljedeći potez i buduće ciljeve.

Ključne riječi: 3D računalna igra, strategija potez-po-potez, pokretački sustav za igre, OpenGL, GLUT, blokovno generiranje mapa, botovi, baze pravila, algoritmi za traženje puteva

3D Computer Game with Elements of Autonomous Behavior

Abstract

This paper presents a 3D computer game based on Heroes of Might and Magic III. The gameplay is turn-based. Each player gets to control several heroes during his turn and aims to explore the map using his heroes' fighting units while upgrading his castles. The game engine is based on OpenGL, using GLUT for added functionality. The maps are randomly generated using predefined blocks of objects while trying to keep all players on equal ground with respect to their starting zones. AI players, also called bots, use rule-based decision making and pathfinding algorithms to prioritize their future goals and calculate movement.

Keywords: 3D video game, turn-based strategy, game engine, OpenGL, GLUT, block-based map generation, game bots, rule-based decision making, pathfinding algorithms