Neuro Asteroids

Studentski tim:

Fredi Šarić

Andrija Miličević

Domagoj Nakić

Filip Gulan

Luka Žmegač

Mentor:

Marko Čupić

Projekt NeuroAsteroids rezultirao je igricom, koja je inspirirana arkadnom igrom „Asteroids“ (http://www.freeasteroids.org/). Uz igricu je razvijeno i više različitih računalnih igrača koji također mogu igrati. Reprezentacije igrača su napravljene kao neuronske mreže.

Cilj igrice je skupiti što više zvjezdica i uništiti što više meteora.

Igra:

Igrica je rađena u potpunosti u programskom jeziku Java 8, a grafičko sučelje je napravljeno u JavaFX programskoj okolini. Projekt smo napravili u potpunosti od nule. To je uključivalo i izradu animacija, fizike, zvuka, kontrolora za upravljanje sa brodom i izradu menija.

Fiziku smo napravili koristeći vektore i linearnu algebru za koju smo sami napisali biblioteku.

Animacije objekata napravili smo tako da smo u programu Blender kreirali osnovni 3D objekt koji samo „zamotali“ u određenu sliku kako bi dobili željeni objekt (npr. zvjezdica je zapravo cilindar oko kojeg je zamotana sličica zvjezdice) te smo nad tim objektom napravili neku transformaciju (npr. rotaciju) i snimili 30 - 60 sličica tijekom trajanja transformacije. Nakon toga, sličice su spremljene kao jedna filmska traka (eng. sprite-sheet). Da bi smo dobili animaciju morali smo pomicati „kameru“ (koja gleda na jednu od sličica). U svakom vremenskom okviru kamera se pomaknula na sljedeću sličicu. Time smo dobili iluziju da je taj objekt animiran u 3D-u. Za tranzicije između menija koristili smo razrede iz JaveFX za transformaciju objekata scene, poput „TranslateTransition“.

Zvuk je bilo vrlo jednostavno implementirati zato što JavaFX okruženje pruža vrlo dobru potporu za zvuk, uporabom razreda iz javafx.scene.media paketa. Za pozadinsku muziku koristili smo javafx.scene.media.MediaPlayer razred jer on pruža potporu za učitavanje zvučnih datoteka. Zvučne datoteke se ne učitavaju u memoriju u cijelosti, već se učita samo mali komadić i tako je napravljeno bolje upravljanje memorijom. Time je omogućeno da se učitana zvučna datoteka pokrene ispočetka kada završi te smo tako ostvarili to da se pozadinska muzika nikad ne ugasi. MediaPlayer ne dopušta da se zvučni zapis pokrene prije završetka te takvu implementaciju nismo mogli koristiti za efekte unutar igrice (npr. ispaljivanje rakete ili eksplozija meteora). Iz tog razloga smo koristili razred AudioClip. AudioClip razred je namijenjen zvučnim datotekama koje nisu memorijski zahtjevne te ih učitava u memoriju odjednom. Zbog toga je omogućeno da se zvučni zapis pokrene prije nego što je prošli završio.

Za upravljanje brodićem u svijetu bili su zaduženi kontroleri. Imali smo dvije vrste kontrolera. Prvi tip kontrolera reagira na ulaze sa tipkovnice. Podržali smo dvije vrste takvih kontrolera, jedan koji upravlja brodićem sa strelicama, a drugi koji koristi tipke WAD. Kontroleri koriste tipku SPACE za pucanje. Morali smo koristiti te dvije vrste kontrolera kako bi se igrica mogla izvršavati na svim operacijskim sustavima, jer operacijski sustav Linux ne može očitati simultani unos više tipki (kao što su lijevo, gore i space – pa nije moguće napraviti akciju naprijed, lijevo, pucaj za kontroler s kojim se upravlja sa strelicama). Drugi tip kontrolera je koristilo računalo, također smo napravili više primjera takvih kontrolera. To su zapravo bili dekoratori oko neuronske mreže. Takvi kontroleri su služili da svaki put kad se zatraži od njih akcija pripreme informacije o svijetu i predaju ih neuronskoj mreži koja na kraju odlučuje koja akcija se treba izvršiti.

Menije smo napisali direktno u Javi. U kôdu smo sami pisali raspored komponenti, dok smo za uređivanje komponenata koristili Javinu verziju CSS-a. CSS-om je bilo moguće detaljno opisati svaki element unutar pojedinog menija. Nakon što smo napravili sve menije, shvatili smo da JavaFX podržava još jednu mogućnost, a to je „FXML“ koji u kombinaciji sa programom „SceneBuilder“ može puno lakše kreirati poglede (menije u slučaju igrice) nego kad se oni programski pišu. Takva izgradnja pogleda se drži MVC programske paradigme.

Evolucijski algoritmi i neuronske mreže:

Svima u timu ovo je bio prvi put da smo se susreli s evolucijskim algoritmima te smo u početku morali pročitati puno literature kako bi počeli pisati smisleni kôd. Koristili smo samo genetske algoritme za razvoj mreža, a to su bile implementacije klasičnog generacijskog genetskog algoritma i genetskog algoritma za višekriterijsku optimizaciju NSGA I. Kod klasičnog algoritma uvodili smo elitizam tako što smo uvijek prenosili najbolju jedinku u sljedeću generaciju. Elitizam nismo mogli uvesti kod NSGA I algoritma, iz razloga što on ne podržava elitizam jer je kod višekriterijskih problema teško reći koja je jedinka najbolja u populaciji. Da smo implementirali NSGA II algoritam dobili bi elitistički višekriterijski algoritam, ali to nam i dalje nije garancija da bi se pojavljivala bolja rješenja pojavljivala jer elitistički algoritmi imaju tendenciju lakše zapeti u lokalnom optimumu, i ako se parametri ispravno ne podese, mogu pretraživati mali prostor rješenja, a nama je prostor rješenja bio višedimenzionalni realni prostor.

Kao reprezentaciju neuronskih mreža koristili smo potpuno povezanu unaprijednu neuronsku mrežu i reprezentaciju temeljenu na NEAT algoritmu opisanu u rada Kenneth O. Stanley-a. Bolje rezultate dobili smo sa prvom implementacijom, ali smo toj implementaciji posvetili više vremena te smo ju više trenirali. Mreža je imala ulogu da ovisno o informacijama koje dobije od kontrolera odluči koju akciju treba napraviti. Svaka mreža je imala N ulaza i 4 izlaza koji su određivali koju akciju će u sljedećem trenutku brodić napraviti.

```
1.  t ← 0
2.  InicijalizirajPopulaciju( P[t] );
3.  EvaluirajPopulaciju( P[t] );
4.  Dok ( nije zadovoljen uvjet zaustavljanja ) radi {
5.      P1 = OdaberiRoditelja( P[t] );
6.      P2 = OdaberiRoditelja( P[t] );
7.      C  = KrižajRoditelje( P1, P2 );
8.      Mutiraj( C );
9.      DodajUPopulaciju( P' , C );
10.     t = t + 1;
11.     P[t] = P';
12. } Završi
```

```
1.  NSGA_I {
2.      t ← 0;
3.      P[t] = inicijalizirajPopulaciju();
4.      evaluirajPopulaciju( P[t] );
5.      Dok nije zadovoljen uvjet zazstavljanja radi {
6.          P1 = odaberiRoditelja( P[t] );
7.          P2 = odaberiRoditelja( P[t] );
8.          C  = križajRoditelje( P1, P2 );
9.          mutirajDijete( C );
10.         dodajDjeteU( P', C );
11.     }
12.     evaluirajPopulaciju( P' );
13.     t = t + 1;
14.     P[t] = P';
15. }
16. evaluirajPopulaciju( P ) {
17.     za jedinku C u populaciji P {
18.         odrediVrijednostiCiljnihFunkcija( C );
19.     }
20.     Fronte = podijeliUFronte( P );
21.     fit = brojFronti( Fronte );
22.     Za frontu F iz Fronte {
23.         za svaku jedinku iz F {
24.         nc = izračunajGustočuNišeOkoJedinke( C );
25.         dodjeliDobrotuJedinci( C, fit / nc );
26.       }
27.       fit  = fit_min * ε;
28.     }
29. }
```

Napravili smo više implementacija kontrolera koji su mrežama davali različite ulaze. Kontroleri koji su davali informacije mrežama od najboljih prema lošijima su:

1. Najbolji kontroler:
    - Poziciju zvjezdice u obliku, zvjezdica je lijevo ili je desno (binarno)
    - Kolika je opasnost s desne i lijeve strane na eksponencijalnoj skali od 0-1
    - Nalazi li se ispred igrača neki meteor ili ne (pod ispred se smatra polje čija je širina jednaka širini igrača, a dužina jednaka 3 duljine meteora. Ovakva informacija je predana igraču u nadi da će čuvati metke i pucati samo kada se neki meteor nalazi dovoljno blizu da ima dobru šansu da ga pogodi)
    - Informacija modelirana eksponencijalnom funkcijom koja označava prije koliko je igrač ispalio prošli metak (ova informacija je dana igraču u nadi da kad ispali metak jer mu je neki meteor u vidnom polju, sljedeći put kada bude pitan za akciju ne ispali novi metak)
2. Drugi kontroler:
    - Poziciju zvjezdice u obliku, zvjezdica je lijevo ili je desno (binarno)
    - Poziciju najbližeg asteroida, u obliku asteroid je lijevo, desno ili ispred tebe
    - Informaciju je li ispalio metak u zadnjih 10 puta ili nije
3. Treći kontroler:
    - Brzinu broda
    - Informacija što se nalazi ispred igrača, zvjezdica ili meteor
    - Informacija što se nalazi lijevo od igrača, zvjezdica ili meteor
    - Informacija što se nalazi desno od igrača, zvjezdica ili meteor
4. Četvrti kontroler:
    - Brzinu broda
    - Kut pod kojim igrač vidi najbližu zvjezdicu, i udaljenost od nje
    - Kut pod kojim igrač vidi najbliži meteor s lijeva, desna, ispred i iza broda, i njihove udaljenosti

Sve ove algoritme smo više manje izvršavali u višedretvenom okruženju kako bi dobili ubrzanje. Iz razloga što smo napravili dobru abstrakciju algoritma, mogli smo nakon svake generacije izvršavati operacije nad populacijom rješenja. Mogli smo gledati napredak algoritma kroz generacije, i ako bi primijetili da je algoritam zapeo u lokalnom optimumu mogli smo ga resetirati u bilo kojem trenutku.

Za svaki od ovih kontrolera dobili smo zanimljiva ponašanja, no nismo dobili niti jednu mrežu koja bi igrala igricu bolje od čovjeka. Mislimo da je na to utjecalo više faktora, kao što su:

- **Utjecaj sreće**: ponekad su se znali pojaviti igrači koji su bili vrlo loši, ali su imali sreće da ih baš svi meteori izbjegavaju te da budu ocijenjeni kao dobri igrači. Takve mreže bi proširile svoj genetski materijal u sljedeću generaciju koja bi baš zbog toga možda bila lošija nego prošla. Protiv toga smo se borili tako da smo svaku mrežu ocjenjivali više puta i kao završnu ocjenu uzeli bi aritmetičku sredinu dvije najlošije ocjene koje je mreža dobila. Kao daljnji prijedlog protiv ovakvog ponašanja predložili bismo da se tokom evolucije mijenja težina igre (povećava se broj meteora u svijetu i/ili se smanjuje broj zvjezdica) ili tokom same evaluacije. Tako da na početku evaluacije u svijetu bude vrlo malo meteora i više zvjezdica, a pri kraju evaluacije da se broj meteora u svijetu poveća, a broj zvjezdica smanji. Takav pristup bi onda dozvoljavao i različite načine ocjenjivanja. Npr. za svaku pokupljenu zvjezdicu na početku se dobije malo bodova, a za svaki pogođeni meteor na početku se dobije puno bodova, dok se na kraju evaluacije za svaku pokupljenu zvjezdicu dobije puno bodova, a za svaki meteor malo. Takav način ocjenjivanja bio bi dosta pogodan za višekriterijsku optimizaciju.
- **Puno informacija**: svijet u kojem se mreža nalazila sadrži puno informacija:
  - položaj svih zvjezdica i meteora
  - brzina i smjer kretanja meteora
  - brzina i smjer kretanja broda
  - količina goriva koje brod ima na raspolaganju
  - količina metaka koje brod ima na raspolaganju
  - udaljenost od „kraja“ mape

  Da bi netko postao najbolji igrač morao bi znati sve te informacije i na temelju njih donositi odluku, no ako svaku tu informaciju preslikamo u jednu dimenziju realnog prostora, dobit ćemo n-dimenzijski hiperprostor koji je prevelik i besmislen za pretraživanje. Zato smo morali pažljivo odabrati informacije koje ćemo predati neuronskoj mreži kako bi ona imala šanse pronaći dobra rješenja. Morali smo naći i dobar balans, jer što smo manje informacija dali mreži o svijetu, to je ona lakše pronalazila dobra rješenja koja su bila ograničena na osnovne akcije.
- **Dugo vrijeme učenja**: pošto smo svaki put kad smo pokrenuli algoritam učenja moral čekati 3-4 sata da bi on napravo 1000 generacija, mogli smo rijetko ugađati parametre algoritma
- **Kompleksnost mreže**: pošto smo više koristili potpuno povezanu unaprijednu mrežu kojoj smo morali u početku definirati topologiju, nismo znali kako će se ta topologija ponašati za problem koji rješavamo te smo to mogli vidjeti tek na kraju algoritma kada dobijemo rješenja.

Rad u timu:

Ovo nam je svima, više manje, bio prvi put da smo radili na timskom projektu i tek smo sada vidjeli koliko je važno imati dobar tim i dobru komunikaciju između članova tima i s mentorom da bi projekt uspješno završio. Tim je bio podijeljen u manje grupe od kojih je jedna grupa bila zadužena za izgradnju igrice, a druga grupa za evolucijske algoritme i neuronske mreže. Za komunikaciju smo koristili program „Slack“ koji može imati više kanala kako bi bolje odvojio komunikaciju po temama, pa smo imali kanal za komunikaciju vezanu samo za izgradnju igrice, i kanal samo za komunikaciju vezanu uz evolucijske algoritme i neuronske mreže. Za upravljanje kodom smo koristili Git, a kao Git klijent smo odabrali „Bitbucket“ koji smo mogli integrirati u Slack kao „hook“ te smo time dobili mogućnost da vidimo svaki put kad netko nešto mijenja po repozitoriju, odnosno da nam dođe na Slack kao notifikacija.