QRsnek

Tööst

Töö eesmärk oli kasutada erineval moel tehisintellekti, et teha huvitavaks traditsiooniline ussimäng. Loodud mängus QRsnek, saab ussi juhtida QR koodi kallutades ja põnevuse lisamiseks võib endale vastaseks lisada arvuti poolt juhitava ussi.
Mängu käivitamiseks ja mängimiseks on mõned eeldused. Pythonis peab olema paigaldatud teegid pygame versioon 2.0.1 ning opencv versioon 4.4.0. Riistvaraliselt peab arvutil olema veebikaamera, et QR koodi tuvastada. Kui need eeldused on täidetud, siis mängu alustamiseks peab käivitama game.py faili. Mängimiseks peab olema kasutajal kaamerale näidata QR kood, kas paberil (soovitatav) või ekraanil. Kui kasutada teist ekraani QR koodi näitamiseks, siis tuleb selle heledust muuta mängu alguses kalibreerimise ekraanil. Tavaliselt töötab tuvastamise algoritm hästi, kui heledus on madalamapoolne.
Programmi testimiseks tuleb mängida mängu. QR koodiga on juhtimine edukas kui suudad ussi niimoodi pöörata, et see jõuaks söögini. Samuti saad kontrollida, et QR kood leitakse süsteemi videotöötluse osa poolt ülesse, kui vaatad kaamera akent ja näed seal QR koodi ümber värvitud punast kontuuri. ****KAS AI'd saab ka kuidagi lihtsat moodi mängimise teel testida****

Autorite panus

Jürgen Leppsalu:
Minu peamine panus QRsnek projektis oli videotöötluse loogika väljatöötamine (fail qr_reader.py). Kasutasin OpenCV võimalusi, et avada arvutis vaikimisi leitud veebikaamera videovoog, nende kaadrite põhjal leida QR koodi koordinaadid ning koordinaatide põhjal arvutada QR koodi kalle. Kui alguses oli plaan teha videotöötlust peameetodiga samas lõimes, siis testimise käigus selgus, et sellise variandi puhul ei tuvastata QR koodi kallutamist piisavalt efektiivselt. Tõstsin seepeale videotöötluse eraldi lõimele ja mängu mängitavus paranes märgatavalt.
Videotöötlusele **lisaks tegin peamängus kalibreerimise ekraani**, mida saab kasutada mängija, kes tahab QR koodi näidata näiteks telefoniekraanilt. Seda oli vaja, kuna kaamera ei tuvasta hästi heledal ekraanil olevat QR koodi ja seetõttu oli vaja kuidagi kasutajat suunata ekraani heledust muutma. **Peamängus parendasin ka mängu lõpu ekraani**, et seal näitaks kasutaja lõppskoori ehk söödud toiduplokkide arvu. Samuti oli ekraan minu hinnangul kole ning tegin selle ilusmaks. See nõudis muu koodi mõningast ümber kirjutamist ja game.py faili loogikast arusaamist.
Kasutatud materjalid olid OpenCV videotöötluse artikkel, OpenCV QRCodeDetector klassi dokumentatsioon ning RealPython veebilehe artikkel mitmelõimeliste programmide loomisest. Materjale kasutasin, et õppida teekide funktsionaalsust õigesti kasutama, kuid kogu suurema loogika kirjutasin ise.

Hain Luud:

Minu panus QRsnek projektis oli **A* algoritmi kirjutamine** (Node.py ja main.py-s aiMove(), openDoesntContain(), setNodeDistance()), **tehisnärvivõrgu jaoks treeningandmete kogumine** (main.py-s generateTrainData()) ja muud väiksemad koodi parandused. Kasutasin A* algoritmi, sest ussimängu peamiseks ülesandeks on leida tee toiduni ja teadsin, et A* on üks optimaalsemaid viise selle leidmiseks ning seda on ka teised selles mängus AI loomiseks kasutatud. Kasutasin koodi kirjutamiseks veebist leitud näidislahendust, mille kohandasin vastavaks meie enda projektile.
Kuna A* algoritm ei olnud ülemäära keeruline kirjutada, siis töötasin ka selle kallal, et saaks luua ussi juhtiva tehisnärvivõrgu mudeli. Varasemalt loodud ja veebist leitud materjalide põhjal tundus Reinforcement Learning (ei tea kuidas see eesti keeles oleks) strateegia hea lähenemisviis olevat. Kirjutasin generateTrainData() funktsiooni, mis mängib 1000 ussimängu 300 juhukäiguga ja salvestab iga mängu, mille punktisumma ületab 100 punkti. Igas ajaühikus kaotab uss 0.5 punkti, aga toidu leidmisel saab ta 100 punkti juurde. Välja valitud mängud salvestati saved.npy faili, nii et neid saaks edaspidi kasutada mudeli õpetamiseks.

Siim Markus Marvet:
Mina kirjutasin ussi mängu selgroo ning suurema osa GUIst. Selle juures oli mu põhiline eesmärk, et mängu modulaarsus lubaks teistel tiimiliikmetel sellele enda käiguotsustus. Samuti oli mul eesmärk, et mäng töötaks võimalikult efektiivselt, tänu millele oleks selle peal hiljem võimalik suures koguses näidisandmeid genereerida.
Täiendasin A* algoritmi, et uss suudaks igas suunas ka mängulaua servi ületada, kui see annaks lühema tee toiduni.
Proovisin luua ussi käike otsustavat närvivõrku, mis toimiks enam-vähem A* põhimõttel - lõppkokkuvõttes edutult. Närvivõrgu sisendiks oli 8 väärtust:

- ussi pead ümbritsevate ruutude hõivatus(4 väärtust [üleval, alla, vasakul, paremal] {0,1}),
- toidu suund ussi suhtes x ja y telge mööda (2 väärtust {-1, 0, 1}),
- hüpoteetilise käigu x ja y koordinaadi muutus (2 väärtust {-1, 0, 1}).

Nende põhjal pidi närvivõrk andma väljundina hinnangu hüpoteetilisele käigule(sisendi viimased kaks väärtust), kas käik lõppeb:

- -1 - surmaga;
- 0 - ellujäämisega;
- 1 - ellujäämisega ja toidule lähenemisega.

Pärast kolme ööd ja päeva erinevate mudelite, mudeliparameetrite ja andmehulkade katsetamist pidin tõdema, et ükski loodud närvivõrk ei suutnud edukalt ussi juhtida. Proovisin kõiki viise kursuselt, mis mul tuli pähe treeningandmete eeltöötluseks ja mudelite loomiseks, aga ükski neist ei andnud märkimisväärselt paremat tulemust.
Põhilise probleemina õppis uss küll ära ellujäämise, kuid hakkas alati eelistama ühte hüpoteetilist käiku. Ehk närvivõrgu ennustustest oli näha, et hüpoteetiliste käikude hinnangud muutusid oodatud suunas vastavalt toidu asukohale, ei olnud see muutus kunagi piisavalt suur, et "domineeriva" käigu hinnangut ületada.

The truth is out there, aga sellel kursusel käsitletud materjalide põhjal said mul ideed otsa, kuidas närvivõrku paremaks teha.