Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jasmina Satler

Učenje Pythona z računalniško igro Minecraft

magistrsko delo

magistrski program druge stopnje
pedagoško računalništvo in informatika

Mentor: prof. dr. Janez Demšar

Ljubljana, 2018

Zahvala

Zahvaljujem se svojemu mentorju prof. Janezu Demšarju za vso pomoč in vodenje tekom izdelave magistrske naloge. Zahvala gre tudi vsem prijateljem in sodelavcem, ki so poslušali moje "jamranje" in mi v težkih trenutkih niso pustili odnehati. Prav posebna zahvala pa gre moji družini, ki mi je vsa leta tekom študija stala ob strani in me spodbujala, saj mi brez njih ne bi uspelo.

Jasmina Satler, 2018

Kazalo

Povzetek

Naslov: Učenje Pythona z računalniško igro Minecraft

Znanje programskih jezikov postaja z leti vse bolj pomembno, zato moramo poskrbeti, da se bodo mlajše generacije zgodaj srečale s tem področjem in se naučile algoritmično razmišljati. Čeprav se je v zadnjih letih povečalo število gradiv in nalog za učenje programiranja v osnovni šoli, je le-tega še vedno premalo ali pa so otrokom nezanimive. Ker imajo otroci radi računalniške igre, smo pripravili paket programov in nalog, s katerimi se lahko učijo programskega jezika Python s pomočjo priljubljene računalniške igre Minecraft.

Ključne besede

poučevanje, programiranje, Python, računalniška igra Minecraft

Abstract

Title: Thesis template

The knowledge of programming languages is becoming more and more prominent over the last decade, so we need to make sure that younger generations meet with this field early on and learn how to think logically. Although in recent years the amount of materials and tasks used for teaching programming in elementary schools has increased, it is still insufficient, or it might not be interesting to children. Because children love computer games, we have prepared a suite of programs and tasks that can teach the Python programming language through the popular Minecraft computer game.

Keywords

teaching, programming, Python, computer game Minecraft

Poglavje 1

Uvod

Katera je vaša najljubša igrica? Ste imeli kdaj željo, da bi v njej počeli več kot lahko? Ste hoteli spremeniti ali dodati kakšne posebne možnosti? Ste si obupano želeli "goljufati", pa niste imeli dovolj znanja za to? Vse to je mogoče. Ravno zato moramo našim otrokom pomagati in ponuditi možnost, da osvojijo programersko znanje. V tem magistrskem delu se bomo poigrali z računalniško igro Minecraft in ustvarili gradivo, ki učencem predstavi nekatere izmed možnosti, ki so jim ponujene. Poleg ustvarjalnosti bodo z njim razvijali še logično razmišljanje in se učili programskega jezika Python.

Današnji otroci se s programiranjem srečajo že zelo zgodaj. Programersko znanje postaja vse bolj pomembno [23], delo z računalnikom spremlja že skoraj vsako delovno mesto [8]. Pri nas se otroci s programiranjem spoznajo v okviru izvenšolskih dejavnosti (večinoma na raznih tečajih) ali preko izbirnega predmeta v četrtem, petem in šestem razredu [31]. Pri učenju programiranja se srečajo predvsem s programskim okoljem Scratch. V tem delu smo želeli raziskati možnosti učenja v drugih jezikih, ki ne bi bili grafični (sestavljanje v bloke), saj bi želeli, da se učenci srečajo tudi s pisanjem krajših ukazov oz. kode.

Scratch je privlačen, ker je narejen tako, da se otroci učijo preko igre. Ker pa postane po določenem času neučinkovit in zato nezanimiv, bomo v tem delu raziskovali možnosti za poučevanje programskega jezika Python.

Osnovni problem pri poučevanju programiranja za otroke je iskanje primernih nalog za poučevanje posameznih konceptov (npr. pogoji, ponavljanje/zanke, spremenljivke), ki so privlačne za otroke. Scratch je v tem pogledu uspešen, ker otrokom nudi okolje, v katerem lahko sestavljajo animacije, zgodbe, igre in podobno. Obenem je okolje Scratch takšno, da si lahko otrok sam prilagaja naloge, ki jih rešuje, oziroma njihovo zahtevnost. Otroku tudi ni potrebno razmišljati o tehničnih podrobnostih, na kakršne naletimo pri resničnem programiranju, kot so različni robni primeri, ki niso povezani z razumevanjem koncepta, ki ga želimo naučiti in zato le zmedejo otroka.

V jezikih, kot je Python, še ni nam poznanih okolij, ki bi bila usmerjena v poučevanje, skladno z zgoraj opisanim pristopom. V okviru magistrskega dela bomo raziskovali eno od potencialnih takšnih okolij za Python. Ker vemo, da igra otroke pritegne, bomo kot okolje izbrali popularno računalniško igro Minecraft. Raziskovali bomo, katere koncepte je mogoče smiselno vključiti v izzive, ki bi jih zastavili otrokom.

Na spletu najdemo kar nekaj jezikov in okolij, ki učence navdušijo nad programiranjem [27]. Veliko študentov računalniških smeri je za svoje zaključno delo izbralo izdelovanje učnih gradiv in iskanja načinov uporabe programskega okolja za izobraževalne namene, glej Prevc [10], Mihalič [30] in Ropret [18]. Veča se tudi število knjig za poučevanje programiranja v slovenskem jeziku. Nekatere izmed njih so osredotočene prav na naš izbirnik Python, in sicer S. Lajovic: Python za otroke [36], P. Krebelj: Spoznavamo programski jezik Python [29] itd. Na Fakulteti za računalništvo in informatiko vsako leto poteka poletna šola. V okviru te poteka več tečajev, namenjenih učencem različnih starosti. Na njih se učenci učijo programiranja z več jeziki in uporabe različnih okolij. Naslov ene izmed tem letošnje leto je bil `Sprogramiraj svoje Minecraft mesto`, namenjena pa je bila učencem zadnje triade osnovne šole in dijakom [12].

Vse bolj se zavedamo pomembnosti programerskega znanja. Nekatere države imajo učenje programiranja že vključeno v predmetnik v osnovnih šolah [51]. Večje kot bo zanimanje, večja bo želja po znanju in posledično

tudi potreba po gradivih, s pomočjo katerih bi to znanje lahko osvojili. V našem delu bomo ustvarili učno gradivo, ki bo primerno za učence vseh treh triad in bo v pomoč tako učencem kot izobraževalcem.

V poglavju 2 bomo v prvem delu raziskovali jezike, primerne za učenje programiranja. V drugem delu bomo pregledali še druga obstoječa okolja, ki se lahko uporabljajo za poučevanje. V zadnjem delu poglavja pa bomo opisali, zakaj smo se odločili za računalniško igro Minecraft in programski jezik Python.

Poglavje 3 je namenjeno računalniški igri Minecraft. Najprej bomo naredili kratek opis igre, ki predstavlja izbrano učno okolje. Sledi opis uporabljenega Minecraft Python API-ja in strežnika Spigot, brez katerih pisanje funkcij v Pythonu in njihova uporaba v igri ne bi bila mogoča. Opisali bomo tudi pripravo učnega okolja in naše spremembe omenjenega API-ja.

Narejeno učno gradivo ter njegovo pripravo in testiranje bomo opisali v poglavju 4. Najprej se bomo lotili priprave nalog in učnega gradiva, ki je cilj tega magistrskega dela. V začetku bomo zbrali ideje za naloge, nato pa ugotavljali primernost in težavnost izbranih tem.

Poglavje 2

Jeziki in okolja za poučevanje programiranja

Za poučevanje programiranja moramo izbrati jezik in okolje. S slednjim ne mislimo zgolj razvojnega okolja (kot npr. Eclipse, Visual Studio, IntelliJ), temveč predvsem kontekst, v katerem bodo učenci razvijali projekte (npr. programiranje želve, spletnih strani, mikrokrmilnikov). Izbor jezika in okolja sta povezana, saj je za določena okolja primeren ali celo edini možen le določen jezik ali tip jezika. Spletne strani (na strani odjemalca) običajno programiramo v Javascriptu ali kateri od njegovih izvedenk, šibkejše mikrokrmilnike programiramo v C, C++ ali pa v zanje napisanih in prilagojenih jezikih, kot sta Lua ali MicroPython.

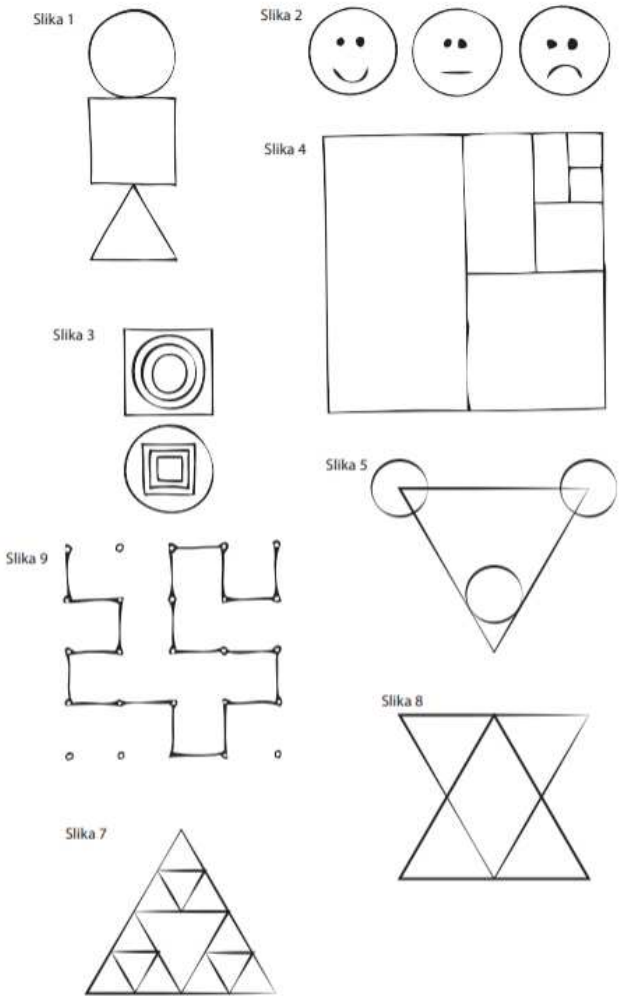
2.1 Programski jeziki

Med jeziki za poučevanje programiranja lahko prepoznamo tri zvrsti; neračunalniške jezike ter grafične in besedilne računalniške jezike.

Neračunalniški jeziki. V zadnjih letih je predvsem po zaslugi projekta CS Unplugged [9] (v slovenskem prevodu Vidra [43]) popularno poučevanje osnovnih idej računalništva in razvoja logičnega razmišljanja brez uporabe

računalnikov [41]. V sklopu takšnih aktivnosti je programiranje predstavljeno v svojem bistvu, kot podajanje navodil.

Ta so lahko podana v naravnem jeziku. Naloga učenca je opisati določen postopek tako, da ga bodo lahko drugi učenci ali učitelj izvedli/ponovili. Primer iz Vidre [44] je opisovanje določene slike, ki jo morajo drugi učenci, ne da bi jo videli, narisati (glej sliko 2.1).



Slika 2.1: Učni list za uro Risanje po navodilih *(vir: http://vidra.si/risanje-po-navodilih/).*

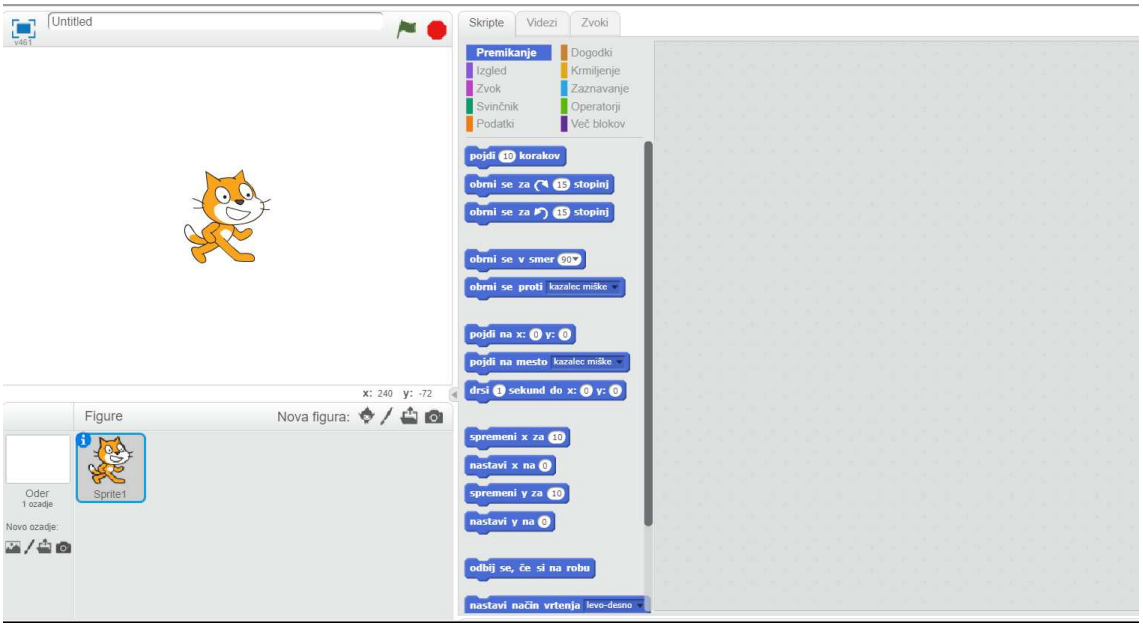
Drug popularen zgled je mazanje marmelade na kruh. Ker je ta postopek učencem že znan, navodila navadno izvaja učitelj, ki pa jim sledi slepo ali tako, da v primeru dvoumnosti namerno ne stori tega, kar se implicitno pričakuje. Če na primer učenec pozabi povedati na kateri strani naj učitelj prime nož za mazanje, ga učitelj seveda prime na napačnem koncu. Če učenec pozabi povedati, da je potrebno odpreti kozarec z marmelado, ga učitelj seveda ne odpre in poskuša zariniti nož skozi pokrov. Če v navodilih ne piše, da je potrebno odrezati kos kruha, temveč učenec napiše "namaži kruh z marmelado", učitelj namaže celo štruco kruha (ali pa, iz spoštovanja do hrane, le nakaže, da jo bo).

Pri podajanju navodil v naravnem jeziku je poudarek predvsem na tem, da demonstriramo njihovo dvoumnost, kar pokaže potrebo po formalnejših jezikih in natančnih navodilih.

Primer formalnejših, a še vedno neračunalniških jezikov so različne kartice, ki predstavljajo posamezne ukaze. Najpopularnejši primer so različne namizne igre, kot na primer Robo Turtles [34], ki je najbolj podprta namizna igra v zgodovini Kickstarter-ja [16]. V takšnih igrah igralec navadno upravlja s figuro, kartice pa predstavljajo različne premike, pogoje in v nekaterih primerih celo zanke in funkcije.

Grafični jeziki. Podobni so jezikom pri katerih so ukazi zapisani na karticah, le da te niso fizične, temveč jih učenec sestavlja na računalniku. Predvsem je pomembno, da jih računalnik izvaja sam.

Najbolj znan predstavnik te vrste jezikov in okolij je Scratch [37]. V Scratchu (slika 2.2) določimo enega ali več objektov in programiramo odzive tega objekta na dogodke v okolju, kot so začetek izvajanja programa, pritisk tipke, prejem sporočila od drugega objekta in podobno. Bloki spreminjajo koordinate, orientacijo in videz objekta, izpisujejo besedilo in predvajajo zvoke, poleg tega so na voljo kontrolni bloki za pogoje in zanke, matematične funkcije. Možno je tudi definiranje novih funkcij.



Slika 2.2: Scratch.

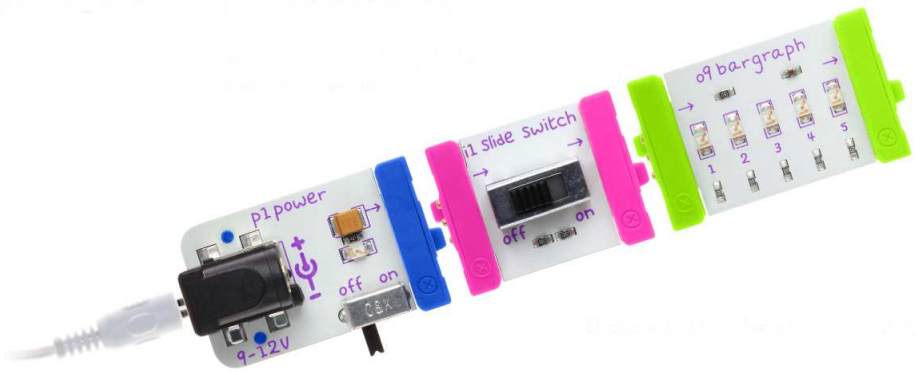
Programi v Scratchu so vedno sintaktično pravilni, saj se bloki, ki se ne združujejo, grafično ne ujemajo in tudi ne "sestavijo", ko jih učenec poskuša zložiti. Scratch vsebuje tudi osnovno kontrolo podatkovnih tipov (pozna namreč števila in nize). Učencu se tako ni potrebno ukvarjati s pravili, temveč le s semantiko, kar je prednost pred običajnimi, tekstovnimi jeziki.

Kot vmesno stopnjo med namiznimi igrami in Scratchom omenimo še ScratchJr [38], ki je namenjen mlajšim in vsebuje le nekaj osnovnih blokov za premikanje izbrane figure, ne pa tudi blokov za kontrolo toka.

Scratch so razvili v MIT Media Lab pod vodstvom Mitchella Resnicka, ki je bil doktorski študent Samuela Paperta, začetnika konstrukcionističnega pristopa k učenju [26]. V smislu pedagoških načel Scratch črpa iz idej programskega jezika Logo. Za razliko od namiznih iger je Scratch torej izrazito konstrukcionistično okolje, saj učenca spodbuja k raziskovanju. Učitelj naj bi učencem dal zgolj ohlapne usmeritve, kaj naj bi naredili (npr. igro, v kateri igralec išče pot skozi labirint), način reševanja in podrobnosti (bodo v labirintu tudi pošasti, ali bo igralec v njem pobiral sladolede itd.) pa prepusti njim [39, 30].

Scratch je navdihnil množico podobnih projektov. Eden izmed njih je Blockly, ki ga razvija Google in je uporabljen v več sto projektih, od katerih je bila večina narejena v izobraževalne namene [13]. Blockly v osnovi ni ne jezik in ne okolje (v smislu konteksta), temveč zgolj orodje, v katerem lahko definiramo bloke in njihov pomen. Primer uporabe Blocklyja je AppInventor [4], v katerem iz blokov sestavimo aplikacijo, ki teče na mobilnih telefonih in tablicah z operacijskim sistemom Android. *Jezik* AppInventor-ja so torej bloki, ki ustrezajo pogojem in operacijam, smiselnim za aplikacije na Androidu, *okolje* oz. kontekst pa so aplikacije za mobilne telefone in tablice [46].

Če sicer nejasno mejo med *neračunalniškimi* in *grafičnimi* jeziki začrtamo glede na to, ali izvajanje programa simulira človek ali pa se program izvaja avtomatsko, sodijo med slednje tudi pripomočki, kot je LittleBits [17]. Ta, podobno kot Scratch, vsebuje bloke, ki izvajajo določene operacije in ki jih je potrebno zložiti v program. Razlika je v tem, da so LittleBits fizični bloki (slika 2.3), program pa se vseeno izvaja avtomatsko. LittleBits so bolj podobni grafičnem jezikom tudi v tem smislu, da gre za izrazito konstrukcionistično okolje. Učenec se uči z raziskovanjem in je takoj deležen tudi (objektivne) povratne informacije, ali njegov program res deluje tako, kot si je želel [49, 17].



Slika 2.3: LittleBits bloki *(vir: https://littlebits.com/)*.

Besedilni računalniški jeziki. Poučevanje programiranja slej ko prej pripelje do klasičnih, besedilnih računalniških jezikov. Medtem ko se v grafičnih jezikih učenec ukvarja le s semantiko (in se lahko tako bolj posveti razmišljanju o tem, kaj *želi* narediti), je tu potrebno paziti tudi na pravila. To lahko postane hitro obremenjujoče in odbijajoče, saj lahko učenci večino časa preživijo ob iskanju pozabljenih oklepajev in podpičij. Zato je pomembno, da izberemo za poučevanje takšen jezik, pri katerem bo sintakse čim manj in bo za začetnika čim preprostejša.[1]

Drug kriterij za izbor jezika je njegova izrazna moč. Zbirnik ima sintakso, ki je preprosta tudi za človeka, vendar je v zbirniku težko napisati za učenca privlačen program, ki ne bi bil dolg in neobvladljiv za začetnika. Veliko sodobnih jezikov ima sintakso po zgledu C-ja, kar je za začetnika neprikladno, saj bo pozabljal dvopičja (novejši jeziki v tem slogu, npr. Kotlin in modernejši Javascript, jih na mestih, kjer niso nujni, opuščajo [2]). Bločna struktura je podana z zavitimi oklepaji, ki jih bo začetnik pozabljal, obenem pa ga jezik ne sili v pravilno zamikanje. Operatorji so večinoma zapisani z znaki, kot so `&&` in ||, ki začetniku ne pomenijo ničesar. Nadomestimo jih lahko z angleškimi besedami `and` in `or`, ki si ju je lažje zapomniti.

Jezik C ni najboljša izbira tudi zaradi izrazne moči, saj visokonivojske podatkovne strukture (npr. slovarji in množice) niso vdelane v sintakso jezika, temveč so dostopne le prek dodatnih knjižnic (kot npr. STL za C++), ki zahtevajo razumevanje objektnega programiranja in vzorcev, *template*. Tudi drugi jeziki s C-jevsko sintakso tipično ne podpirajo tako preproste uporabe podatkovnih struktur kot, na primer, Python, ki ga opišemo kasneje.

Podobne, čeprav nekoliko manjše hibe ima Pascal. BASIC je sicer primernejši, vendar izvirne različice niso dobro podpirale strukturiranega programiranja. Funkcije so bile okorne ali pa so temeljile na uporabi GO SUB in RETURN. Novejše različice pa so postale sintaktično precej bolj zapletene

[1] Jeziki, kot so Lisp in njegove izpeljanke, imajo resda zelo preprosto sintakso, vendar ta ni preprosta za programerja, temveč je preprosta le v formalnem smislu, po številu elementov.

od izvirnika.

Prvi programski jezik, ki je bil razvit predvsem z mislijo na otroke, je bil Logo. Kljub svojemu namenu je bil nenavadno moderen in vpliven. Logo je v osnovi izpeljan iz Lispa in je funkcijski, pomemben pa je njegov vpliv na Smalltalk in prek njega na razvoj objektno usmerjenega programiranja. Za nas je pomemben predvsem zaradi želvje grafike in uporabe egocentričnega koordinatnega sistema. Ta ne uporablja kartezičnih koordinat temveč kote in razdalje glede na trenutno smer in pozicijo objekta, kar je za učenca naravnejše [47]. Minecraft, ki ga uporabljamo v tem magistrskem delu, žal uporablja kartezične koordinate, zato smo mu morali dodati nekaj funkcij, ki v osnovi pretvarjajo iz egocentričnih koordinat, "dve polji pred mano", v kartezične.

Jezik, ki smo ga izbrali za to delo, je Python. Razvil ga je nizozemski programer Guido van Rossum leta 1990, prva javna različica je bila objavljena februarja 1991. Ime je dobil po priljubljeni angleški televizijski nanizanki Monthy Python's Flying Circus. Razvili so ga kot odprtokodni projekt. V glavnem se uporablja za računalniško analizo in razvijanje internetnih aplikacij. Je visokonivojski jezik, ki ga lahko razširjamo in vključujemo v razne aplikacije. Obstaja veliko paketov in modulov, ki programerju olajšajo delo in predvsem prihranijo čas [48]. Njegova priljubljenost med uporabniki se povečuje, IEEE Spectrum ga je na svoji lestvici top programskih jezikov v letu 2018 postavil na prvo mesto [35].

Python je skriptni jezik. Programi, ki so pisani v njem, so lepo berljivi, zato je primeren tudi za začetnike, saj se jim poleg samega učenja programiranja ni potrebno toliko ukvarjati s sintakso, tako kot v drugih jezikih, na primer v C-ju, C#-u in podobnh. Jezik je usmerjen predmetno, tudi modul, funkcija in tip so objekti. Te usmerjenosti ne vsiljuje, zato se nam ni potrebno ukvarjati z razredi, razen če tega sami ne želimo [48, 10].

Na izbor jezika se, kot smo že omenili, navezuje izbor okolja. Določen jezik je lahko v smislu sintakse in izrazne moči primeren za otroke, vendar zanj ne obstaja takšno okolje, da bi lahko učenci v tem jeziku reševali zanje

zanimive probleme. Ta problem je do določene mere prisoten pri Pythonu, saj programi, ki izpisujejo Fibonaccijeva števila, niso posebej privlačni, igre, kakršne lahko delamo v Scratchu, pa je v Pythonu začetniku težko narediti.

2.2 Okolja za poučevanje programiranja

Z okolji za poučevanje programiranja tu ne mislimo na razvojna okolja za programerje, temveč na to, kakšne tipe projektov lahko učenec izdeluje.

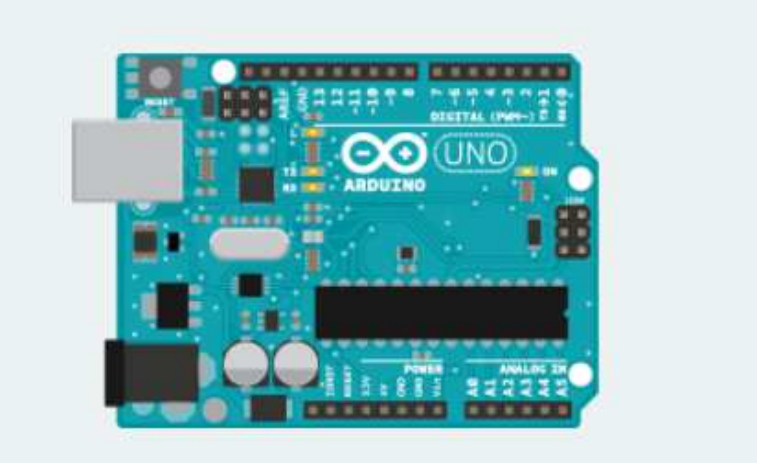
Igre. Privlačen tip projekta je očitno igranje ali izdelovanje iger. Z igranjem iger mislimo na okolja, v katerih mora učenec napisati program, s katerim reši določeno nalogo, na primer pripelje figuro čez labirint. Takih okolij je kar nekaj, verjetno tudi zato, ker jih je s sodobnimi orodji preprosto narediti. Pogosto pa so v didaktičnem smislu vprašljiva, saj učencu ne puščajo svobode, njihovi avtorji pa z dodatnimi omejitvami poskušajo vsiliti točno določeno rešitev problema - na primer z največjim številom dovoljenih ukazov ali z ožanjem nabora ukazov. Takšne naloge lahko sicer razumemo kot reševanje ugank, kar pa ni skladno s konstrukcionistično pedagogiko.

Popularno okolje za izdelovanje iger lahko vidimo kar v Scratchu ali izpeljankah, kot je AppInventor. Ta omogočajo ustvarjalnost, ki je hkrati motivacija za to, da se je učenec pripravljen potruditi s programom. Če si je učenec sam zamislil, da bodo v njegovem labirintu tudi pošasti, se bo za to, da jih sprogramira, pripravljen bolj potruditi. Čisto drugače bi bilo, če bi si pošasti izmislil učitelj (ali pa mu, po slabem zgledu iz prejšnjega odstavka, predpisal, katere ukaze sme in katerih ne sme uporabiti).

Fizično računalništvo. Privlačen tip projekta je tudi fizično računalništvo, saj je otipljivo. Med fizično račnalništvo spada programiranje raznih mikrokrmilnikov, kot so Arduino [5], Raspberry Pi [32] in NodeMCU [24]. Mikrokrmilnik je majhen računalnik na integriranemu vezju, ki vsebuje enega ali več procesorjev (procesorskih jeder) skupaj s pomnilnikom in programabilnimi vhodi in izhodi. Učenci torej s programiranjem upravljajo izhode, na

katere se lahko priklopi zunanje naprave (senzorji, led lučke itd.) ali pa celo naredijo neko "pametno" igračo [15].

Arduino je proizvajalec mikrokrmilnikov, katerega strojna in programska oprema sta dovolj preprosti za začetnike. Na mikrokrmilnik Arduino (slika 2.4) se lahko priklopijo razni senzorji, kot so foto-senzor, senzor temperature in podobno. Njegove izhodne parametre lahko preberemo in glede na prebrano stanje, sprogramiramo izhode - na primer aktiviramo motor, vklopimo LED lučko ali sprožimo nek zvok. Za programiranje se tipično uporabljata programska jezika C in C++ v okolju Arduino, ki že vsebuje knjižnice z definicijami nekaterih funkcij in jih lahko uporabimo za programiranje. Lahko jih programiramo tudi z grafičnimi jeziki, vendar se program v tem primeru izvaja na računalniku, ki komunicira z Arduinom [6]. Takšno računalništvo je še vedno dovolj "fizično" konkretno in učencem zanimivo.



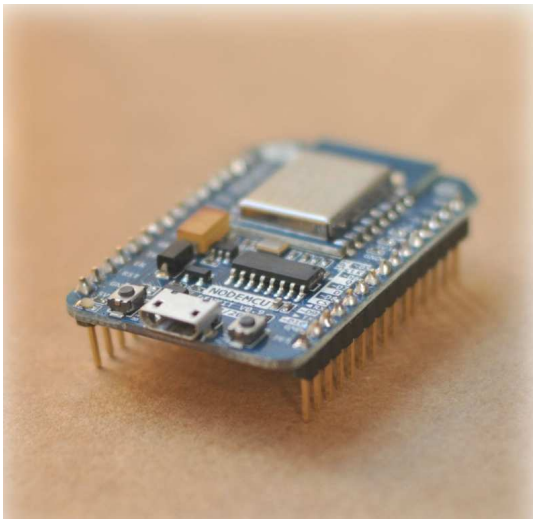
Slika 2.4: Arduino *(vir: https://www.arduino.cc/)*.

Raspberry Pi (slika 2.5) je serija mikrokrmilnikov, ki že mejijo na računalnik, saj imajo ti mikrokrmilniki že vgrajene grafične procesne enote, grafične izhode, zmogljivejšo procesno enoto ter več notranjega pomnilnika in RAM-a. Podobnost med računalnikom in Raspberry Pi se odraža tudi v uporabi operacijskih sistemov, večinoma distribucije Linux operacijskega sistema. Za programiranje z Raspberry Pi lahko uporabljamo poljuben jezik, kot na pri-

mer C, Python, C++, Scratch, Bash, Java itd. Pri izbiri programskega jezika je mikrokrmilnik bolj zapleten in zato bolj omejen glede komunikacije s senzorji in posledično fizično slabše otipljiv.

Slika 2.5: RaspberryPi *(vir: https://www.raspberrypi.org/products/)*.

Raspberry Pi je dražji od preprostejših mikrokrmilnikov, kot je Arduino [11], zato je za projekt iz fizičnega računalništva, ki zahteva močnejše okolje, pogosto bolj smiselno uporabiti kar običajni računalnik z ustreznim vmesnikom. Druga možnost je NodeMCU (slika 2.6), ki je podoben Arduinu. Je cenovno ugoden (na spletni strani Amazon ga najdemo za manj kot deset dolarjev [3]) in ga lahko programiramo tudi v Lui in MicroPythonu.



Slika 2.6: NodeMCU *(vir: http://nodemcu.com/index_en.html/)*.

Programiranje spletnih strani. Otroci se s spletom srečajo zelo zgodaj. To je eden izmed razlogov, da je želje po znanju za programiranje spletnih strani vedno več. Že samo dejstvo, da bo učenec po koncu učenega postopka znal narediti svojo spletno stran, je dovolj velika motivacija.

Spletna stran je dokument, ki ga prikaže spletni brskalnik. Strukturo in vsebino spletne strani sestavimo z jezikom HTML, ki ni programski ampak opisni oz. označevalni jezik [28]. Izgled spletnih strani navadno določimo s pravili v datoteki CSS. Za izdelovanje interaktivnih spletnih strani potrebujemo tudi znanje programskega jezika JavaScript. Ta je objektni skriptni jezik, s katerim lahko na spletni strani implementiramo bolj kompleksne stvari, kot so interaktivni zemljevidi, animirane slike in podobno [19].

Učenja izdelovanja spletnih strani se ponavadi lotimo v določenem vrstnem redu. Učenci najprej spoznajo jezik HTML, ki predstavlja osnovo spletnega dokumenta. Nato spoznajo CSS, ki skrbi za oblikovanje. Za izdelavo dokumentov HTML in CSS ne potrebujemo nobenega posebnega programa, izdelamo jih lahko tudi v navadni beležnici, kar je prednost za uporabo pri učenju. Problem lahko nastane, ko se začne učenje jezika JavaScript. Starejše verzije programskega jezika imajo kar nekaj napak, pri novejših pa se srečamo s problemom, da je potrebno nastaviti razne prevajalnike. Razvoj takoj postane veliko bolj zapleten. Če bi hoteli, da del aplikacije teče na strežniku (ne samo na odjemalcu), bi se morali naučiti še kakšnega drugega jezika, kot na primer Python, C#, Node, Ruby, ali pa celo tudi kakšna ogrodja, kot na primer Django, Flask itd. Lahko bi prišli tudi do točke, kjer bi se morali spoznati s podatkovnimi bazami, kar zna biti za učence že preobsežno in prezahtevno.

2.3 Izbira programskega okolja in jezika za učno gradivo

Na spletu torej najdemo kar nekaj jezikov in okolij, s katerimi lahko otroke navdušimo nad programiranjem [27]. Želeli smo najti nekaj, kar bi učence

navdušilo in pritegnilo v svet programiranja. Izbire in gradiv v tujih jezikih je bilo kar nekaj. Ker pozornost otrok najbolj pritegnemo z raznimi igrami, smo se odločili za spletno okolje Minecraft. Na spletu je kar nekaj gradiva in pogovorov na forumih o učenju Pythona s pomočjo Minecraft-a, zato smo se odločili uporabiti programski jezik Python.

Minecraft je učencem zanimiv in radi ustvarjajo v njem, zato se že uporablja v izobraževalne namene. Laboratorij za grafiko in multimedije na Fakulteti za računalništvo in informatiko je imel letos poletno šolo na temo Minecrafta [12]. Uporabljali so Minecraft for education, za programiranje pa so uporabili MakeCode, ki je jezik, ustvarjen za programiranje v Minecraftu. MakeCode omogoča programiranje z vizualnimi bloki in tudi z besedilom, če uporabimo vmesnik za JavaScript. Na poletni šoli so uporabljali vizualne bloke, saj so sodelovali mlajši učenci [1].

Nekatere učence pri začetnem učenju programiranja moti množica različnih ukazov in sintaktičnih vzorcev (oklepaji za imenom funkcije, dvopičja, vejice, zamiki itd.), ki si jih je potrebno zapomniti. S kombinacijo Minecrafta in Pythona lahko hitro vidimo rezultate svoje kode in spremembe v Minecraft svetu, kar je dodatna prednost pri poučevanju otrok. S tem lažje pridobimo njihovo pozornost in zanimanje. Na voljo je tudi nekaj gradiv za učenje Pythona v slovenskem jeziku (S. Lajovic: Python za otroke [36], P. Krebelj: Spoznavamo programski jezik Python [29] itd.).

Poglavje 3

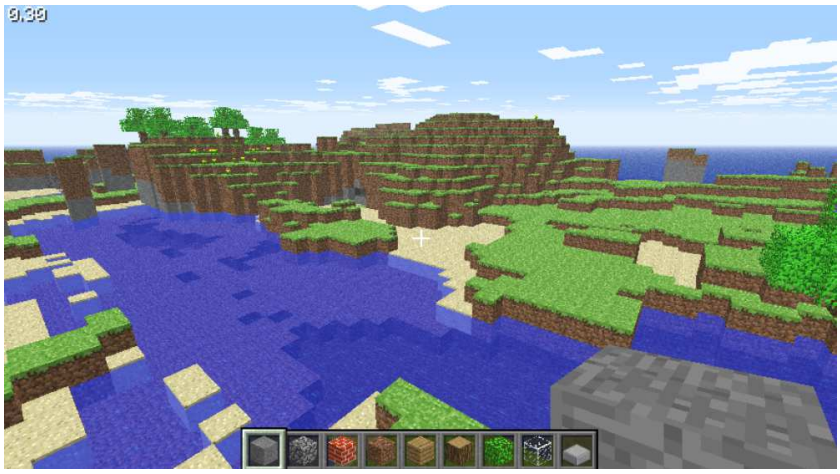
Računalniška igra Minecraft

V prvem delu bomo najprej opisali računalniško igro Minecraft. Opisan bo sam potek igre, možnosti izbire načina igranja in opis samega izgleda. V drugem delu bomo opisali uporabljen Minecraft Pi API, strežnik Spigot ter način komunikacije strežnika s Python-om. V tretjem delu se bomo osredotočili na pripravo okolja za programiranje v Minecraftu.

3.1 Opis igre

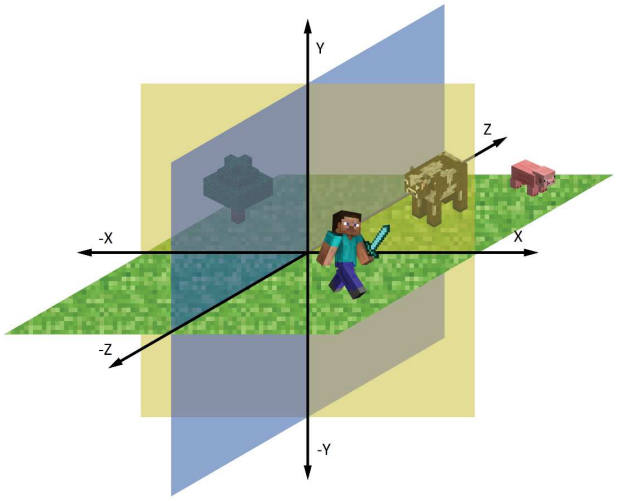
Minecraft je ustvaril švedski razvijalec iger Markus Persson, ustanovitelj podjetja Mojang AB. Razvoj se je začel leta 2009, uradna izdaja je bila 2011, 2014 pa je postala najbolje prodajana računalniška igra vseh časov. Do marca 2018 je bila prodana v 28 milijonov izvodov [21].

Minecraft je računalniška igra, v kateri je igralec postavljen v naključno generiran svet (slika 3.1). Vsak element tega sveta (voda, puščava, drevesa itd.) je sestavljen iz kock, ki predstavljajo različne materiale - vodo, različne kamnine, lavo, travo, ogenj itd. V igri lahko igralec pridobiva kocke tako, da seka drevesa, koplje rudo, pobira predmete in podobno. Kocke lahko postavlja, razbija ali pa jih spreminja v drugo snov. Izdela lahko razna orodja in orožja. Če želi nabirati zemljo, si izdela lopato, če želi razbijati kamen ali rudo, pa kramp.



Slika 3.1: Igralčev pogled na svet
(vir: https://minecraft.gamepedia.com/File:Classic.png).

Svet je razsežen, koordinatni sistem pa je predstavljen tako, da sta `x` in `z` vodoravni poziciji, `y` pa je navpična (slika 3.2).



Slika 3.2: Koordinatni sistem *(avtor: Ivana Bajec).*

Igra ima več igralnih načinov: klasični, preživetveni, ustvarjalni, težki, pustolovski in opazovalni. Med seboj se razlikujejo po tem, ali je možno predelovati elemente, ali se pojavljajo pošasti, ali so zaloge elementov omejene

itd. Klasični način je bil iz novejših različic odstranjen, vendar ga omenjamo, ker je še vedno dostopen, če v Minecraftu ustvarimo novo možnost zagona in izberemo ustrezno različico igre [22]. Obstajata dva načina igranja, ki nekoliko odstopata. To sta pustolovski in opazovalni, saj lahko v pustolovskem samo raziskujemo, v opazovalnem pa lahko samo opazujemo. Običajni načini igranja so tako način preživetja, ustvarjalni in težki.

3.1.1 Klasični način

Način, ki je bil namenjen grajenju brez omejitev se imenuje klasični (angl. *Classic mode*). Igralec je postavljen v "peskovnik" - svet, kjer ima neomejene količine kock in funkcijo hitrega uničevanja kock. V tem načinu niso podprte možnosti, kot so izdelovanje novih elementov iz zbirk elementov (angl. *crafting*), meni s prikazom zaloge vseh elementov v lasti (angl. *inventory*), različni prikazi svetlosti (angl. *dynamic lightning*), zato je ta način zastarel. V tem načinu so začeli z razvojem načina preživetja, ki je bil v začetku poimenovan test preživetja [22].

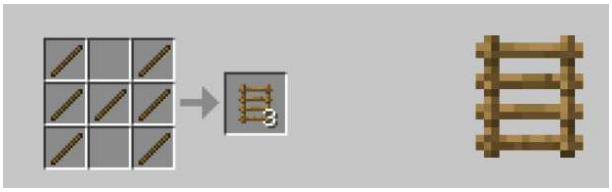
3.1.2 Način preživetja

V načinu preživetja (angl. *Survival mode*) je igralčev glavni cilj preživeti. Ko v igri pade noč, se začnejo pojavljati razne pošasti, ki želijo igralca ubiti. Igralec gradi zavetišča za obrambo pred pošastmi, načrtuje nove napade, obrambo ter si povečuje svojo zmogljivost. Ima omejeno zdravje, ki se mu zmanjšuje z napadi pošasti, padci, utapljanjem, dušenjem in drugimi dogodki. V težji stopnji mora spremljati tudi svojo lakoto, ki vpliva na hitrost zdravljenja in premikanja. Igralec izgublja "kose mesa" (s katerimi je predstavljeno igralčevo stanje), ko se mu povečuje lakota. Ta narašča s hojo, plavanjem, skakanjem, podiranjem kock in podobno. Ko igralec ni lačen (ima vse kose mesa), se začne obnavljati zdravje. Lakota in zdravje sta prikazana v spodnjem delu zaslona. Kot smo omenili zgoraj, je lakota prikazana s kosi mesa, zdravje pa s srčki (slika 3.3). Če igralcu zmanjka zdravja, umre

in se vrne na točko, kjer se je pojavil na začetku igre. Igra v osnovi teče brez konca, vendar si lahko igralec kot cilj zastavi borbo z zmajem. Igra se tudi po tem, ko je zmaj ubit, ne zaključi, igralec pridobi zmagovalni element, nato pa lahko raziskuje naprej.

Slika 3.3: Pogled igralca v načinu preživetje *(vir: https://minecraft.gamepedia.com/File:Survival1.13.png).*

V tem načinu so igralcu prvič omogočili, da kombinira elemente in tako ustvarja nove predmete. Za predelovanje obstaja kar nekaj "receptov" - navodil, kateri elementi ustvarijo katerega [20]. Tako bi za izdelavo treh lestev potreboval sedem lesenih palic (kombinacijo si lahko ogledate na sliki 3.4). Igralec v tem načinu nima neomejene izbire kock in elementov, ampak te tekom igre zbira in ustvarja nove.



Slika 3.4: Prikaz "recepta" za lestev, v oknu za ustvarjanje novih elementov *(vir: https://www.minecraft-crafting.net/).*

3.1.3 Ustvarjalni način

Ustvarjalni način (angl. *Creative mode*) je zamenjal klasičnega. V njem je igralcu omogočeno enostavno ustvarjanje in uničevanje. Ima neomejeno število vseh predmetov in lahko ustvarja brez omejitev. Meni s prikazom zaloge (angl. `invetory`) se spremeni v meni za izbiro elementa iz seznama, ki vsebuje skoraj vse bloke in elemente. Nekateri elementi so dostopni le v tem načinu igranja.

Slika 3.5: Zgrajeni izdelki v ustvarjalnem načinu *(vir: https://minecraft.gamepedia.com/File:Creative.png).*

V tem načinu ima igralec možnost letenja in tako lahko z višine opazuje svoje izdelke in pokrajino (primer lahko vidimo na sliki 3.5). Pošasti v tem načinu niso napadalne, igralcu ni potrebno skrbeti za zdravje in lakoto, umre samo v primeru, če pade iz sveta (se spusti pod $y = -64$).

3.1.4 Težki način

Težki način igranja (angl. *Hardcore mode*) je različica načina preživetja, le da se igralcu ob izgubi življenja (glej sliko 3.6) odstrani ves svet in mora začeti znova. Težavnost je vedno nastavljena na težko, kar pomeni, da je hitreje noč, pojavlja se več pošasti itd. Prav tako so onemogočeni vsi bo-

nusi in možne goljufije (omogočajo, da blokiramo poljubno mesto, prikličemo sovražne pošasti in prijazna bitja, dobimo močno orodje ali celo ustvarimo brezplačne vire preživetja [42]).

Slika 3.6: Pogled ob izgubi življenja
(vir: https://minecraft.gamepedia.com/File:Hardcore_Death.png).

3.1.5 Pustolovski način

Pustolovski način (angl. *Adventure mode*) je bil ustvarjen za raziskovanje zemljevidov, ki so jih ustvarili igralci in za obvarovanje zemljevidov na strežnikih. V tem načinu igralec ne mora postavljati ali uničiti kock. Ta način igre je podoben načinu preživetja, saj velja večina pravil, ki veljajo v njem. Omenjena načina se razlikujeta v tem, da se v pustolovskem načinu igralec, ko izgubi življenje, vedno vrne na isto točko, ki je že vnaprej določena na zemljevidu. Igralec še vedno lahko komunicira z ostalimi elementi in liki iz okolja.

3.1.6 Opazovalni način

Ta način (angl. *Spectator mode*) je namenjen samo opazovanju. Igralec ne more graditi ali podirati, prav tako pa nima nobenega inventarja. Lahko leti, hodi čez kocke in like. Lahko tudi označi kakšnega od igralcev, živali ali

pošasti ter opazuje skozi njihove oči. Opazovalci so drugim igralcem nevidni, lahko pa jih vidijo drugi opazovalci.

3.2 Minecraft Python API in SpigotMc

Da lahko programiramo s programskim jezikom Python in uporabljamo narejene funkcije v igri Minecraft, smo uporabili Minecraft Python API in strežnik SpigotMc. V tem poglavju ju na kratko opišemo in razložimo zakaj ju potrebujemo.

3.2.1 Minecraft Pi API

Igra Minecraft deluje tako, da so podatki o svetu (položaji kock, položaj in smer igralca in podobno) in akterjih, ki "živijo" v tem svetu (zombiji, pajki, vaščani, itd.), shranjeni na strežniku. Do tega strežnika en ali več igralcev dostopa s klientom. Strežnik je lahko na istem računalniku kot klient, vendar je igra v tem primeru omejena na enega igralca. Če bi želeli spreminjati Minecraft svet, lahko do tega strežnika dostopamo. To naredimo tako, da do njega ne dostopamo s klientom kot običajni igralec, ampak to naredimo preko vmesnika.

API je kratica za Application Programming Interface, vmesnik za namensko programiranje ali aplikacijski programski vmesnik. Na splošno je to skupek jasno opredeljenih načinov komuniciranja med različnimi komponentami. API je lahko za spletni sistem, operacijski sistem, sistem zbirk podatkov, računalniško strojno opremo ali knjižnico programske opreme. Je gradnik, ki olajša komunikacijo dveh programov [45].

API lahko uporabljamo tako, da dostopamo do strežnika neposredno preko vtičnice (programska osnova medprocesnim komunikacijam [14]) in mu preko njega pošiljamo sporočila oz. ukaze. Ker je to precej nerodno, so razvijalci naredili knjižnice za Python in Javo.

3.2.2 Strežnik SpigotMc

Standardni strežnik za Minecraft ne podpira interakcije z igro, kot je na primer spreminjanje kock, pridobitev igralčeve pozicije, spremembe pogleda itd., zato razvijalci razvijajo prilagojene strežnike, ki to podpirajo. Ti so narejeni kot alternativa prvotnemu strežniku in omogočajo dodatne funkcije, spremembe in optimizacije igre. Protokol se lahko spremeni z vsako izdajo. Ustvarjanje in vzdrževanje takih strežnikov prinaša dodaten izziv, saj je potrebno slediti vsem tem spremembam [25].

V tem delu uporabljamo strežnik SpigotMC, ki komunicira s strežnikom za Minecraft in nam omogoča naslednje možnosti:

- pridobitev igralčeve pozicije,
- spreminjanje ali nastavljanje pozicije igralca
- pridobitev nekega tipa kocke,
- spreminjanje kock,
- spremembo pogleda sveta (pogled na igro iz drugega kota),
- objavljanje sporočil v klepetu.

Spigot je veja modificiranega strežnika Bukkit in vsebuje dodatne optimizacije zmogljivosti, možnosti konfiguracije in funkcije. Obenem je še vedno združljiv z vsemi obstoječimi vtičniki in skladen z igro Vanilla Minecraft (Vanilija pomeni različico programa Minecraft brez načinov - ti so spremenjene datoteke s končnico `.jar`, ki pogosto spreminjajo način igranja). V Spigot-u je na voljo več kot 150 dodatkov [40].

Uporabili smo tudi vtičnik RaspberryJuice. Ta sprejema ukaze na vratih 4711 in kliče ustrezne funkcije strežnika. RaspberryJuice implementira spremembe API Minecraft Pi za strežnike Bukkit (prilagojene strežnike za namizje). Razvijalci API-ja so implementirali naslednje metode:

- world.getBlock,

- world.getBlockWithData,
- world.setBlock,
- world.setBlocks,
- world.getPlayerIds,
- chat.post,
- events.clear,
- events.block.hits,
- player.getTile,
- player.setTile,
- player.getPos,
- player.setPos.

Vtičnik je še vedno v postopku razvoja. V verziji 1.11 je implementiranih še nekaj dodatnih ukazov, ki omogočajo tudi klicanje entitet ter dobivanje in nastavljanje njihovega položaja. Tu jih nismo uporabili, saj ni zagotovljeno, da bodo vzdrževane v nadaljnjih verzijah in spremembah Minecraft Pi API-ja [50].

Poleg RaspberryJuice smo uporabili tudi druge vtičnike. Eden izmed njih je PluginMetrics, ki je pomembno orodje za lastnike strežnikov in razvijalce vtičnikov. PluginMetrics zbira anonimne podatke o igralnem strežniku in jih pošlje v osrednjo bazo podatkov. S tem lahko razvijalci vtičnikov pridejo do pomembnih informacij za nadaljni razvoj. Na podlagi teh se odločajo, katere Java platforme bi bilo potrebno podpreti, katere različice Minecrafta bi morali podpreti, je še smiselno ohranjati združljivost s starejšimi verzijami Minecrafta in podobno. Brez teh meritev bi lahko strežniki postali neuporabni, saj bi njihova različica izgubila podporo vtičnika [7]. Če onemogočimo meritve, potem razvijalci ne dobijo meritev od našega strežnika in

če nihče drug ne uporablja iste verzije, potem se bodo raje posvetili drugim različicam, platformam, sistemom dovoljenj, itd. To pomeni, da nastavitev našega strežnika ne bo podprta. Uporaba tega vtičnika bo torej izboljšala uporabljene vtičnike in uporabljen server [7].

3.3 Priprava okolja

Okolje Minecraft Python API smo spremenili tako, da je njegova uporaba kar se da prijazna slovenskim otrokom. Najprej smo poskrbeli, da lahko kličemo razreda `Block` in `Blocks` s slovenskima izrazoma, in sicer `kocka` in `kocke`. `Kocka` vsebuje dve metodi, prva sprejme koordinate in vrne kocko, ki se nahaja na podanih koordinatah, druga pa sprejme material in ustvari kocko. Razred `kocke` je podoben. Prav tako vsebuje dve metodi - prva sprejme pravokotno regijo in vrne kocke, ki se nahajajo v njej, druga metoda pa podano regijo napolni s kockami izbranega materiala. Regijo podajamo tako, da povemo spodnjo in zgornjo mejo vsake koordinate, pri čemer sta vključeni obe meji. Tako ukaz `kocke[5:10, 5:10, 5:10]` predstavlja kocko s stranico 6.

Za potrebe igralca v Minecraftu bi bila preprostejša oblika `kocke[x:dolzina, y:visina, z:sirina]`. Tako bi zid višine 5 postavili s `kocke[x, y:5, z]`. Vendar bi s tem, ko bi ga navadili na to obliko, otroku povzročili težave pri morebitnem kasnejšem učenju Python-a ali kakšnega druga jezika, saj imajo intervali tam tipično obliko `spodnjaMeja:zgornjaMeja` in ne `spodnjaMeja:dolžina`. Prav tako je podajanje intervalov z mejami tipično v matematiki.

Kocke so predstavljene z materiali in imena teh smo prevedli v slovenski jezik. Tako lahko učenci uporabijo ukaz `kocka[x, y, z] = Kamen`, kar povzroči, da se kocka na koordinatah `x, y, z` spremeni v kocko, ki predstavlja material `Kamen`. Namesto imena materiala lahko v tem ukazu uporabljamo tudi številske id-je posameznih materialov.

Za lažjo razlago smo naredili nekaj funkcij, ki jih uporabljamo v učnem

gradivu in jih lahko učenci kličejo v svojih funkcijah. Modul vsebuje naslednje funkcije:

- `izpisi(s)`, ki izpiše podani niz `s`,.
- `premakni_se(x, y, z)` premakne igralca na koordinate `x, y, z`,.
- `pred_mano(x)`, ki vrne koordinate kocke, ki je za `x` kock pred igralcem, v smeri proti kateri igralec gleda,.
- `ali_lahko_sadim(x, y, z)`, ki vrne true, če je na koordinatah `x, y, z` ena izmed kock materiala `Zemlja`, `Trava`, `VisokaTrava` ali `ObdelanaZemlja` (to so materiali kock, na katerih lahko sadimo rastline),.
- `ali_lahko_kopljem(x, y, z)` vrne true, če je na koordinatah `x, y, z` ena izmed kock materiala `Zemlja`, `Trava`, `Kamen`, `NeobdelanKamen` ali `Pesek` (to so materiali kock, na katerih lahko kopljemo),.
- `ali_lahko_gradim_most(x, y, z)` vrne true, če je na koordinatah `x, y, z` ena izmed kock materiala `Zrak`, `Voda`, `StojecaVoda`, `Lava` ali `StojecaLava` (to so materiali kock, na katerih lahko gradimo most).

Osnovna ideja našega dela (in glavna motivacija za učenca) je, da si bo lahko sprogramiral ukaze, ki jih bo klical iz okolja Minecraft. Novi ukazi bi lahko bili definirani kot funkcije v Pythonu, ki bi jih igralec shranil v ustrezno datoteko oz. ustrezno mapo. Naš pilotni poskus je pokazal, da tak način ni preveč posrečen. Učencu je potrebno za to prezgodaj razložiti preveč konceptov, ki bi se jim raje izognili, ali pa od njega zahtevati, da v program dodaja "magične besede", ki jih ne razume, kot so različni ukazi `import` na začetku modula.

V končni različici, ki jo opisujemo v tem delu, učenec definira nov ukaz tako, da v ustreznem direktoriju ustvari datoteko s končnico `.py`, katere ime je enako imenu novega ukaza. Datoteka vsebuje zgolj telo funkcije, za vse ostalo pa poskrbi naš vmesnik.

Če ukaz prejema tudi argumente, učenec v prvo vrstico zapiše komentar, ki vsebuje seznam formalnih imen argumentov. Primer kaže slika 4.1.

```
# dolzina, visina, sirina, material

x, y, z = pred_mano(2)
kocke[x : x + dolzina, y : y + visina - 1, z : z] = material
kocke[x : x, y : y + visina - 1, z : z + sirina - 1] = material
kocke[x : x + dolzina, y : y + visina - 1, z : z + sirina] = material
kocke[x : x + dolzina, y : y + visina - 1, z : z + sirina] = Les
```

Slika 3.7: Vsebina datoteke, ki definira funkcijo `hisa`.

Poglavje 4

Učno gradivo

Pisanja učnega gradiva smo se lotili tako, da smo najprej zbrali ideje za različne naloge, ki bi bile primerne za posamezen sklop snovi, ki jo želimo naučiti. Med pisanjem gradiva smo nekatere naloge uporabili, nekaj pa smo naredili novih, saj smo potrebovali smiselno zaporedje poglavij in nalog. Naloge se skozi poglavja nadgrajujejo ali pa prikažejo drugačen pristop reševanja problema (različna koda za nalogo, ki nam da isti rezultat). Ko smo zaključili s pisanjem, smo rezultat dela tudi testirali. V tem poglavju bomo najprej opisali končno učno gradivo, nato njegovo pripravo in na koncu sledi še testiranje.

4.1 Opis

Gradivo je razdeljeno na dva sklopa. V prvem je opisano, kako namestimo vso potrebno programsko opremo, v drugem pa je učno gradivo. Gradivu je priložena datoteka s končnico `.zip`, kjer se nahaja potrebna programska oprema in nekaj narejenih programov, ki jih bodo učenci uporabljali, ko se bodo preko gradiva učili programirati.

4.1.1 Navodila za začetek

Vse, kar moramo namestiti, razen igre Minecraft, se nahaja v priloženi datoteki s končnico `.zip`:

- Python 3.5.0,
- Java (verzija 8),
- Minecraft Python API, z vsemi potrebnimi vtičniki, in Minecraft strežnik - Spigot

V navodilih za začetek je opisano, kje lahko najdemo igro Minecraft, če je še nimamo na svojem računalniku, in kako jo namestimo. Nato sledijo navodila za namestitev druge programske opreme iz priložene datoteke s končnico `.zip` ter navodila za zagon strežnika Spigot.

4.1.2 Učenje Pythona

Gradivo je namenjeno otrokom, zato je napisano v poljudnem jeziku in obliki. Poglavja v učnem gradivu si sledijo tako, da učenci po vrsti spoznavajo nove pojme in koncepte. V kasnejših poglavjih smo želeli uporabljati snov iz prejšnjih poglavij, saj jo tako uporabljajo in ponavljajo. Naslovi poglavij in sama vsebina je prilagojena tako, da so uporabljeni otrokom razumljivi izrazi. Med poglavji se nahajajo tudi naloge, v katerih morajo uporabiti do tistega trenutka pridobljeno znanje. Rešitve teh nalog se nahajajo v mapi Samostojne naloge, v priloženi datoteki s končnico `.zip`.

V tem sklopu učnega gradiva najdemo naslednja poglavja:

Klepet. V tem poglavju je opisana uporaba klepeta in nekateri ukazi, s katerimi lahko spreminjamo vreme ali menjamo dan in noč, ki jih lahko uporabijo znotraj njega.

Ukaz. Tu se nahaja razlaga ukaza ter navodila za uporabo enega izmed narejenih ukazov `skoci`, ki povzroči, da igralec v igri skoči. Priložen je v mapi z nalogami v datoteki s končnico `.zip`.

Klic ukaza s podatkom. Tu je opisano, kako pokličemo ukaz skoci s podatkom. Nato bralci spoznajo ukaz `izpisi`, ki povzroči izpis v klepetu ter kako računalniku povemo, da bi želeli, da podatek prepozna kot besedilo. Med razlagami se nahajata tudi dve nalogi, ki se navezujeta na pridobljeno znanje.

Nov ukaz. Sledi opis, kako naredimo nov ukaz, ki ga bomo lahko uporabljali v igri. Igralec naredi svoj prvi ukaz, ki izpiše v klepet: "Minecraft je zakon".

Zaporedje ukazov. V tem sklopu se naučijo, kako lahko uporabljajo zaporedje ukazov in opis samega pojma. Najprej, podobno kot v prejšnjem poglavju, izpišejo v klepet: "Minecraft je zakon", le da pri tej nalogi izpišejo vsako besedo v novo vrsto. Nato naredijo nov ukaz izračunaj, kjer se igrajo z izpisovanjem besedila in števil. Na koncu poglavja je še naloga, kjer morajo ponovno uporabiti pridobljeno znanje.

Nov ukaz s podatkom. Za tem, ko so se učenci naučili, kaj je ukaz, kako ga naredimo ter kaj je zaporedje ukazov, se naučijo narediti ukaz s podatkom. Najprej razložimo, kako naredimo ukaz, ki sprejme argumente in kako te argumente uporabljamo pri pisanju ukaza. Naredijo nov ukaz `izracunaj_enacbo`, ki sprejme neko enačbo in njen rezultat izpiše v klepetu. Na koncu poglavja zopet sledi naloga.

Koordinatni sistem. Z ukazom `premakni_se` smo pojasnili, kako se premikamo po svetu Minecrafta. Za tem bralci navodil spoznajo še, kako lahko pridobijo koordinate pozicije igralca. Tudi temu sklopu sledi naloga.

Spremenljivke. Opis izraza in načina shranjevanja vrednosti.

Kocka. Tu se učenci začnejo učiti spreminjanja okolja v igri. Spoznajo, kaj je kocka ter kako jo lahko naredijo, uničijo ali spremenijo. Najprej razložimo ukaz `kocka`, nato pa sledijo navodila, kako lahko uporabijo

kocke v igri. Spoznajo tudi materiale, ki jih lahko uporabljajo v igri. Pred igralca postavijo kocko in zid, za kar uporabijo material `Kamen`. Med razlago se nahajata dve nalogi.

Več kock. Za tem, ko so se naučili postavljati eno kocko, se naučijo še, kako lahko na enkrat postavijo cel blok kock. Tu je razlaga in uporaba ukaza `kocke` za grajenje zidu in hiše. Da bi lahko videli uporabo ukaza `kocke` še za kaj drugega kot gradnjo, se ta sklop nadaljuje z razlago kako narediti ukaz, ki postavi drevo. Tu se naučijo tudi kako "saditi", to znanje pa nato uporabimo in nadgradimo v naslednjem poglavju. Na koncu poglavja je nova naloga.

Pogojni stavek. V prejšnjem poglavju so se otroci naučili, kako sadijo, tu pa razširimo znanje in jih naučimo, kako pri sajenju upoštevajo različne materiale kock, kjer želijo saditi in se glede na material odločijo, kaj bodo posadili. To naredijo z uporabo pogojnega stavka, ki ga zapišemo s `if`. Najprej razložimo pojem ter opisan ukaz, ki posadi rožo, če je kocka, kjer želi saditi materiala `Trava`. To nalogo nadgradimo tako, da spoznajo še drugi del pogojnega stavka `else`, ki ga zapišemo, če želimo izvesti kakšen drug stavek, ko pogoj ni izpolnjen. Ta v primeru, da material kocke ni `Trava`, izpiše opozorilo. Nalogo nato še enkrat nadgradimo in razložimo še uporabo `elif`, ki v primeru, da material kocke ni `Trava`, preveri še, če je material kocke morda `Pesek` in v tem primeru posadi `Kaktus`. Nato ga seznanimo še z nekaterimi posebnimi ukazi, ki preverjajo določene pogoje, in jih lahko uporabijo pri pisanju pogojnih stavkov. Ti ukazi preverjajo, če se na podani točki lahko sadi, koplje ali pa gradi most. Sledi razlaga uporabe enega izmed teh ukazov, `ali_lahko_gradim_most`. Otroci naredijo nov ukaz, ki gradi most in premika igralca po njem. Sledi še ena naloga.

Zanke. Za razlago zank učenci nadgradijo nalogo iz prejšnjega poglavja, `most`. Otroci spoznajo hitrejši način postavljanja mostu, pri tem pa uporabijo kar nekaj znanja, ki so ga pridobili skozi prejšnja poglavja.

4.2 Priprava gradiva

Pri pripravi smo morali biti pozorni na več stvari. Kot je omenjeno že v poglavju 4.1, smo morali uporabljati otrokom prijazne izraze. Otrokom je precej besed še neznanih in bi jih lahko z njihovo uporabo zbegali in izgubili njihovo zanimanje (funkcija, parameter in podobno). Tudi razlaga novih pojmov mora biti zelo preprosta. Kar nekaj novih pojmov je razloženih preko nalog in igre Minecraft. S tem smo želeli poskrbeti, da bi jim bilo učenje čim bolj zanimivo in da bi lahko sproti videli, kaj so se novega naučili in kako bodo lahko novo pridobljeno znanje uporabljali v sami igri. Kar nekaj problemov smo imeli tudi z odločanjem, kako si bodo poglavja sledila. Med samim sestavljanjem nalog smo morali biti pozorni, da se nikjer ne pojavi še nerazložena snov, zato je bilo sestavljanje nalog precej zahtevno. Naloge morajo biti zanimive ter se lepo povezovati med seboj, znanje pa se mora počasi nadgrajevati.

4.3 Testiranje

S testiranjem smo želeli najti morebitne napake v učnem gradivu in preizkusiti, če učenec dosega želene rezultate in pridobi ustrezno znanje. Učno gradivo smo želeli narediti primerno za učence vseh triad osnovne šole, zato je tudi testiranje potekalo z otroci različnih starosti.

4.3.1 Načrtovanje

Želeli smo izvesti več testov našega učnega gradiva. Prvi testiranec nam bo pomagal odkriti morebitne napake in podal predloge za izboljšave le-tega. Po tem, ko bomo popravili ter dodali nove razlage in naloge, bomo ponovno izvedli teste. Testiranje bo potekalo v sproščenem okolju, kjer se otroci ne bodo počutili pod pritiskom. Razložili jim bomo, da bodo igrali računalniško igro Minecraft in se pri tem še nekaj naučili. Po končanih testiranjih bomo s pomočjo učencev razmislili še o možnih izboljšavah.

4.3.2 Izvedba

Pri prvem testiranju nam je pomagal Žan (v nadaljevanju učenec), ki ima 14 let in je v času testiranja obiskoval osmi razred osnovne šole. Z učencem smo preživeli dobrih deset ur, imeli pa smo več premorov. Učencu smo dali naše učno gradivo in ga spremljali ob učenju. Večino časa je delal samostojno, če je potreboval pomoč, smo mu nudili dodatno razlago. Celoten proces smo spremljali ter si beležili, kaj mu je povzročalo težave. Po vsakem predelanem poglavju smo mu postavili nekaj vprašanj, s čimer smo preverili, če je pridobil znanje, ki je bilo predvideno za obdelano poglavje.

Prve težave so se pojavile pri namestitvi potrebne programske opreme. Kljub temu, da je pri tem delu pri mlajših učencih predvidena pomoč staršev ali učitelja, smo želeli narediti razumljiva navodila tudi mlajšim učencem. Navodilom smo zato dodali več slik (izgled ikon, izgled oken pri nameščanju programov in podobno) in podrobnejšo razlago.

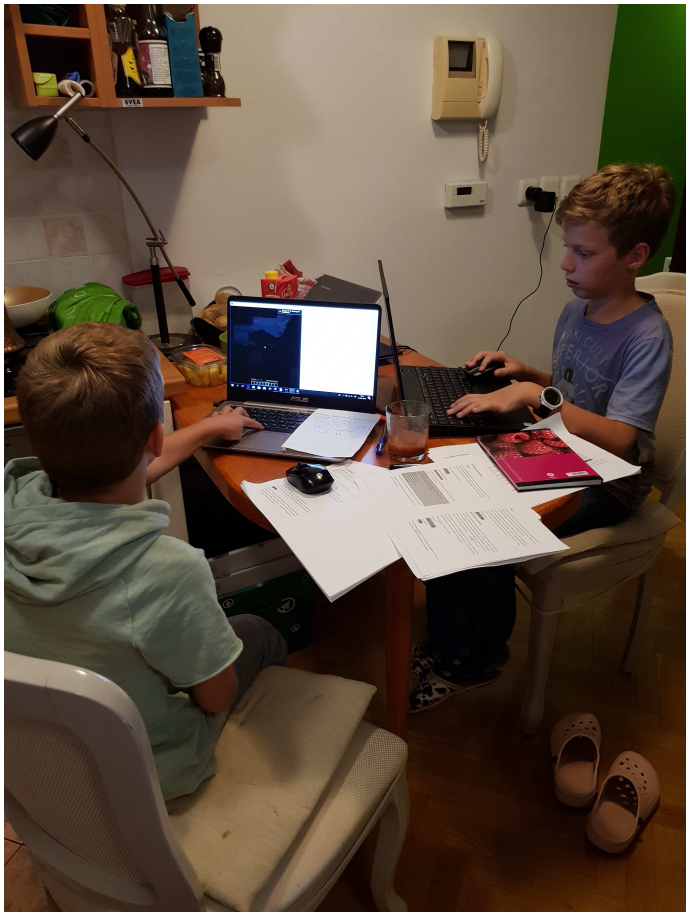
Po namestitvi potrebne programske opreme je sledilo spoznavanje z igro Minecraft. Učenec igre še ni poznal, tako da je ta del trajal malo dlje. Premikanja in uporabe klepeta se je hitro navadil (ukaza za spreminjanje noči v dan ter za izboljšanje vremena sta mu bila najljubša).

Pri ustvarjanju novih ukazov se je pojavil problem pri ustvarjanju novih datotek. Ni razumel, kako narediti ustrezen tip datoteke in je končnico `.py` zapisal datoteke s končnico `.txt`. Ko smo prišli do poglavja `Koordinatni sistem` je učenec postavil nekaj dodatnih vprašanj, s katerimi se je želel prepričati o tem ali je snov prav razumel. Kljub temu, da smo pričakovali, da se bodo pojavili problemi pri razumevanju pojma spremenljivke, ni bilo tako, vse mu je bilo hitro razumljivo. Podobno je bilo s poglavjem `Kocka`. Pri poglavju `Več kock` je imel precej težav z razumevanjem, kako določamo območje postavitve kock oz. kako določimo zgornjo in spodnjo mejo. Pri poglavju `Pogojni stavek` je potreboval dodatno razlago, ko je moral za pogoj uporabiti funkcijo, ki vrača logično vrednost. Pri poglavju Zanke pa ga je zbegala zadnja naloga, kjer je bilo prav tako potrebno uporabiti tako funkcijo. V zadnjih dveh poglavjih je delal tudi napake pri uporabi sintaktičnih

vzorcev; izpuščal je znak dvopičje, pozabljal presledke, zamenjeval oklepaje z oglatimi oklepaji in obratno.

S prvim testiranjem smo preizkušali uspešnost našega gradiva in našli nekaj pomanjkljivosti. Po končanem testiranju smo učnemu gradivu dodali dodatno razlago in slike izvedbe postopkov, ki so bili nerazumljivi.

Pri drugem testiranju sta nam pomagala Mark in Gašper. Mark ima 7 let in obiskuje drugi razred, Gašper pa ima 10 let in obiskuje četrti razred. Testiranje z obema skupaj je potekalo dvakrat po tri ure, prvič s tremi in drugič z dvema premoroma. Nato smo izvedli še eno dodatno testiranje z Gašperjem.



Slika 4.1: Testiranje učnega gradiva z Markom in Gašperjem *(avtor: Jasmina Satler).*

Potrebno programsko opremo smo Marku namestili mi, saj je za mlajše učence to prezahtevno. Gašper jo je želel namestiti sam, tako da smo mu le pomagali, kadar je naredil kakšno napako (ni natančno sledil navodilom).

Ko smo prišli do učenja Pythona, je Mark imel probleme, saj še ne zna hitro brati. Besedilo smo mu zato pomagali prebrati in z njim počasi predelovali snov. Potreboval je veliko dodatne razlage in pomoči.

Pri poglavjih Ukaz in Ukaz s podatkom je imel težave z narekovaji, oklepaji in poševnicami, saj ni vedel, kje na tipkovnici se nahajajo, oziroma katera kombinacija tipk povzroči izpis. Kombinacije tipk si je izpisal na list, na katerega smo ga spomnili, ko se je težava ponovila. Pri poglavju Ukaz je velikokrat pozabil uporabiti znaka oklepaj in zaklepaj, vendar se je kasneje navadil. Ko smo prišli do poglavja Nov ukaz, smo mu navodila za ustvarjanje novega ukaza napisali na list. Vsakič, ko je bilo potrebno narediti nov ukaz, smo šli skupaj po napisanih korakih in ga ob tem spraševali še dodatna vprašanja, s katerimi je ponovil pridobljeno znanje iz prejšnjih poglavij. Z dodatno pomočjo mu je uspelo osvojiti predvideno znanje do zaključka poglavja Zaporedje ukazov. Pri poglavju Nov ukaz s podatkom je imel učenec problem, ko je bilo potrebno narediti ukaz, ki sprejme več kot en podatek. Ko mu je uspelo napisati ukaz za nalogo Uspelo mi je, smo zaključili z učenjem, saj je nadaljnja snov prezahtevna za to starostno skupino.

Gašperju smo na začetku naročili, naj dela samostojno, saj je v Scratchu že programiral. Prva poglavja mu niso povzročala večjih težav, le tu in tam je pozabil kakšne oklepaje. Imel je veliko idej, kaj vse bi si želel sprogramirati, tako da nas je kar pogosto spraševal, če je njegova ideja izvedljiva. Za kar nekaj idej smo mu povedali, da se jih bo naučil v kasnejših poglavjih. Učencu se je malo zataknilo pri ustvarjanju novega ukaza. Nudili smo mu ponovno razlago in pomagali ustvariti prvi ukaz.

V poglavju `Koordinatni sistem` je imel učenec nekaj težav, saj se koordinatnih sistemov še niso učili v šoli. Manjše probleme mu je povzročalo tudi določanje mej pri ukazu `kocke`, s kasnejšimi poglavji ni imel večjega problema. Kot smo omenili zgoraj, je imel že nekaj izkušenj z delom v Scratch-u,

tako da je že vedel, kaj je spremenljivka ter kako delujejo pogojni stavki in zanke.

Poglavje 5

Sklepne ugotovitve

Glavni namen magistrskega dela je bil pripraviti učno gradivo, kjer bi se učenci s pomočjo računalniške igre Minecraft naučili osnov programiranja in algoritmičnega razmišljanja. Učenci se tekom učenja spoznajo s funkcijami (ukazi), pogojnimi stavki in zankami. Iskali smo takšno okolje, da bi bilo otrokom zanimivo in da bi imeli hitro vidne rezultate dela, zato smo se odločili za igro Minecraft.

Učno gradivo smo želeli narediti primerno za vse triade osnovne šole, zato vsebuje tudi veliko slik in preprostih nalog, ki bi bile zanimive in ne prezahtevne za mlajše učence. Tekom testiranja smo ugotovili, da je določena snov vseeno prezahtevna za učence prve triade, saj nekateri še ne znajo hitro in tekoče brati. Imeli pa so tudi težave s tipkanjem (iskanjem črk, kombinacijami tipk za določene znake itd.). To snov se učenci lahko naučijo, vendar z veliko pomoči in dodatne razlage učitelja. Potrebno bi jo bilo tudi prilagoditi za to starostno skupino, tako da bi poglavja razširili in dodali več nalog. S tem bi učenci večkrat ponovili določena pravila in bi si jih lažje zapomnili. Samo gradivo bi moralo počasneje prehajati na nova poglavja in postopno uvajati nove pojme.

Testi so pokazali določene pomanjkljivosti v učnem gradivu, ki bi jih lahko še popravili. Učenci so imeli največ problemov z ustvarjanjem novih ukazov v tekstovni datoteki. Ta del bi bilo potrebno preizkusiti in razložiti še na drug

način oz. pokazati odpiranje datotek tudi v Python IDLE-u. Pri določenih poglavjih bi lahko dodali še nekaj nalog, da bi se pridobljeno znanje še bolj utrdilo. Gradivo bi lahko razširili in dodali spoznavanje novih znanj.

Učence je sama tematika zelo pritegnila. Igra je bila nekajkrat kar sama po sebi velika distrakcija, otroci so imeli namreč preveč motečih faktorjev (spreminjanje dneva v noč, dež, živali, itd.). Kljub omenjenim težavam menim, da učno gradivo dosega svoj namen. Učenci so si želeli pridobiti nova znanja in še naprej raziskovati in ustvarjati v igri Minecraft. Osvojili so predvideno znanje in se pri tem zabavali. Po končanih testiranjih so imeli veliko idej za nove ukaze, najmlajši testiranec pa si kljub težavam in zahtevni snovi, želi nadaljnjega učenja z nami.

Dodatek A

Navodila za začetek

V tej prilogi najdete prvi del učnega gradiva, ki je opisan v poglavju 4.1.1. Vsebuje navodila za namestitev potrebne programske opreme, ki jo potrebujemo za učenje.

NAVODILA ZA ZAČETEK

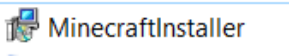
Kaj potrebujete:

- Minecraft
- Python 3.5
- Java (verzija 8)
- Minecraft Python API
- Minecraft server - Spigot

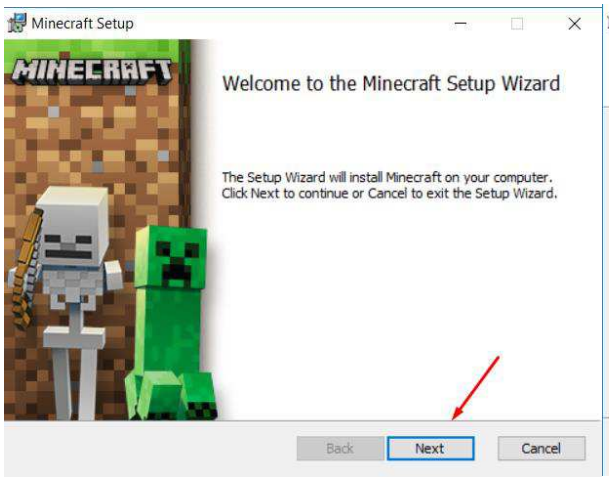
Namestitev Minecrafta

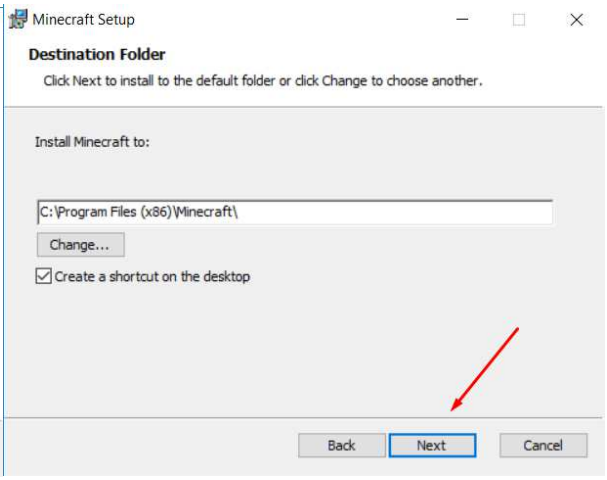
Če že imate nameščeno igro Minecraft, potem lahko ta del navodila preskočite in nadaljujete z namestitvijo Pythona.

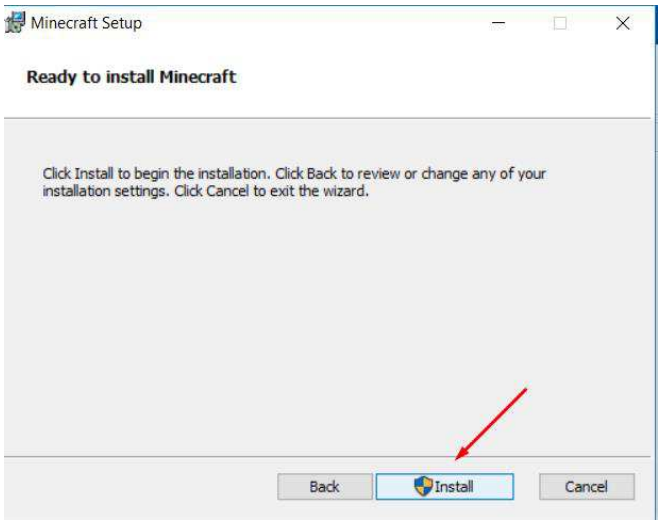
Če igre še nimate, potem obiščite uradno stran Minecraft (https://minecraft.net/en-us/store/?ref=m), kjer lahko kupite uradno različico – s klikom na gumb »download« se vam bo prenesla datoteka MinecraftInstaller.msi.

Izgled datoteke v priloženem paketu: 

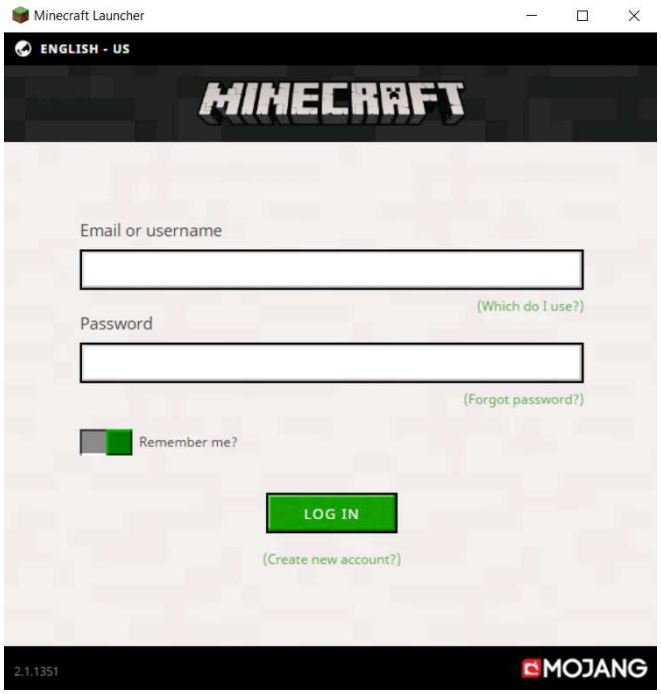
Ko boste zagnali datoteko (dvojni klik nanjo), sledite navodilom za namestitev.







Po namestitvi najdite ikono Minecraft in odprla se vam bo stran za vpis uporabniškega imena in gesla – vnesite podatke, ki ste jih uporabili pri nakupu igre.

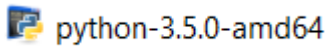


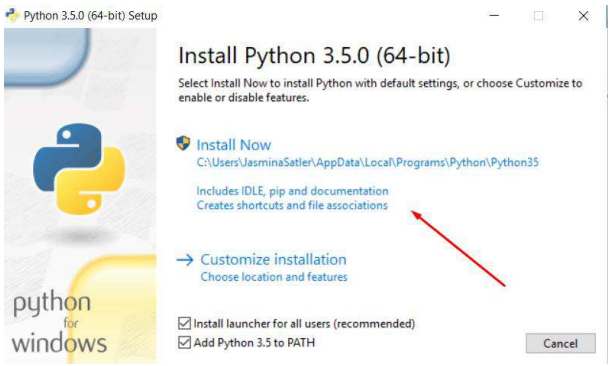
Ko se boste vpisali, vstopite v igro. Na začetku se vam bodo morda namestile še nadgradnje, nato pa se vam bo odprla stran, kjer boste lahko izbirali med načini igre. Na začetku izberite »Single player«, kar pomeni, da boste v igri sami. Tako se lahko spoznate z igro in se malo pozabavate.

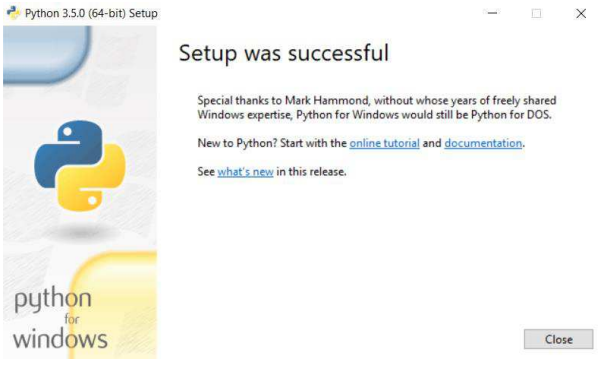
Namestitev Pythona

Python je programski jezik, s pomočjo katerega se bomo učili programirati s tem priročnikom. V paketu, ki je bil priložen tem navodilom, lahko najdete Python 3.5.0, ki je bil v času pisanja navodil najnovejša različica.

Lahko si naložite tudi najnovejšo verzijo, ki jo najdete na strani https://www.python.org/downloads/ - s klikom na gumb »Download Python« se vam bo prenesla datoteka s končnico .exe. Ko boste zagnali datoteko, sledite navodilom za namestitev.

Izgled datoteke v priloženem paketu: 





Namestitev Jave

Igra Minecraft je napisana v programskem jeziku Java, zato je najboljše, da imate nameščeno zadnjo verzijo. V priloženem paketu najdete Javo verzije 8.

Lahko pa si naložite najnovejšo različico preko uradne spletne strani - https://java.com/en/download/. Poiščite gumb »Free Java Download«; po kliku nanj se bo pojavil gumb »Agree and start free download«. Z njim potrdite, da sprejemate pogoje uporabe, ki si jih lahko ogledate na povezavi s klikom na »Terms of use«, takoj pod gumbom.

Shranite datoteko in jo po prenosu odprite ter sledite navodilom za namestitev.

Izgled datoteke v priloženem paketu: jre-8u181-windows-i586

Namestitev Mincraft Python API-ja in Minecraft serverja

Naslednji korak je namestitev Minecraft Python API-ja in strežnika za Minecraft na vaš računalnik.

Minecraft Python API je vmesnik, ki skrbi da lahko s Pythonom povemo računalniku, kaj naj počne v igri Minecraft. S pomočjo API-ja se bo naš program povezal z strežnikom za Minecraft , na katerem je shranjen Minecraftov svet. Minecraftovi strežniki so po večini uporabljeni za igranje preko interneta, tako da lahko igra več igralcev hkrati, lahko pa ga poganjate tudi na računalniku in igrate sami. Mi bomo uporabili strežnik za Minecraft z možnostjo igranja za enega igralca – Spigot.

Navodila za namestitev:

Na namizju naredite novo mapo z imenom Minecraft Python in vanjo razpakirajte vsebino mape Minecraft Python, ki jo najdete v priloženem paketu datotek.

Odprite mapo Minecraft Python in poiščite datoteko Install_API.

Izgled datoteke v priloženem paketu: Install_API

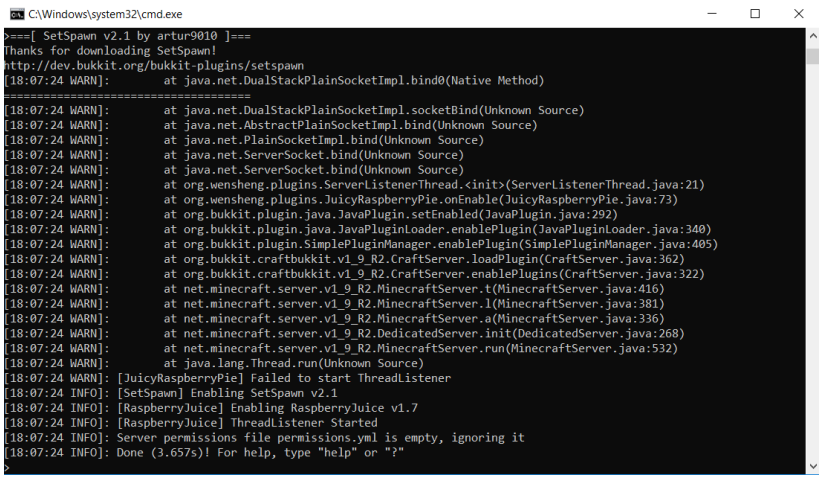
Ko kliknete na to datoteko, se bo odprlo novo okno in namestil se bo Minecraft Python API. Sedaj imate nameščen Minecraft Python API.

Zagon Minecraft serverja - Spigot

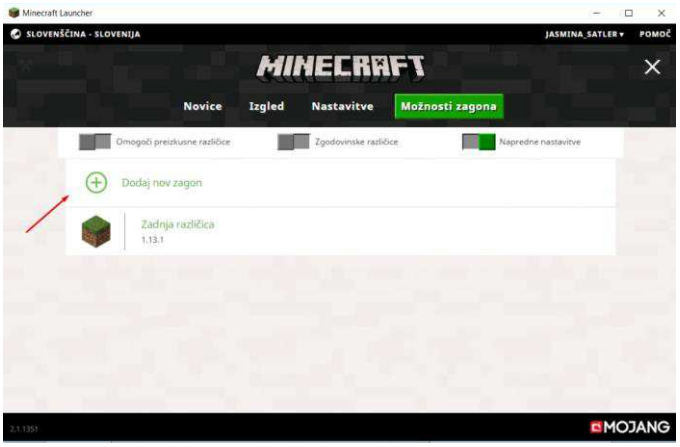
Ko se server prvič zažene, ustvarite nov svet v Minecraftu.

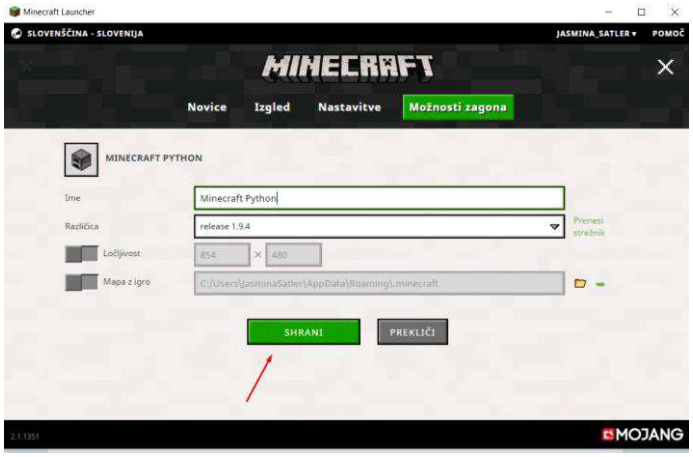
Pojdite v mapo Minecraft Python, ki ste jo v prejšnjem koraku razpakirali na namizje in najdite mapo server. Odprite jo in zaženite datoteko start.bat.

Po zagonu se vam bo odprlo črno okno, ki ga pustite odprtega.

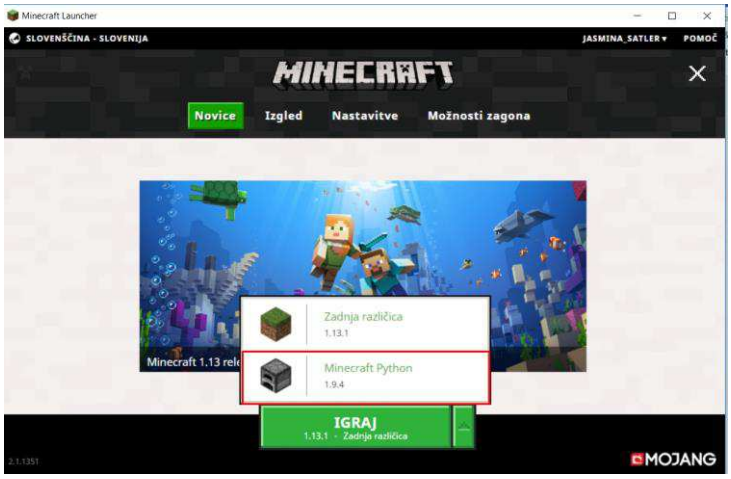


Sedaj zaženite igro Minecraft in izberite možnosti zagona. Tu morate dodati nov način zagona za verzijo 1.9.4..



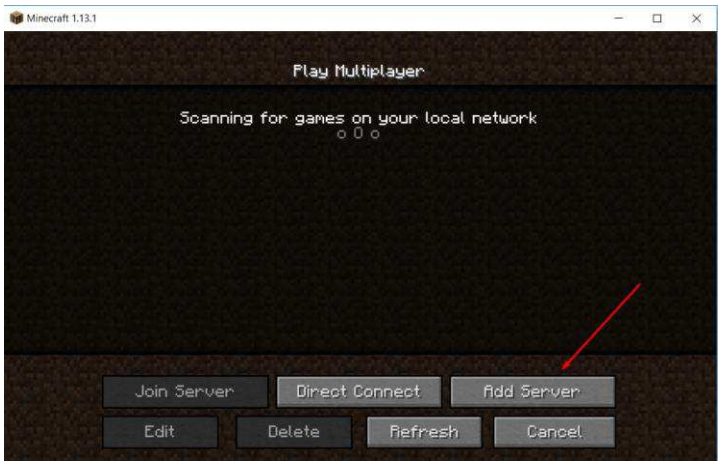


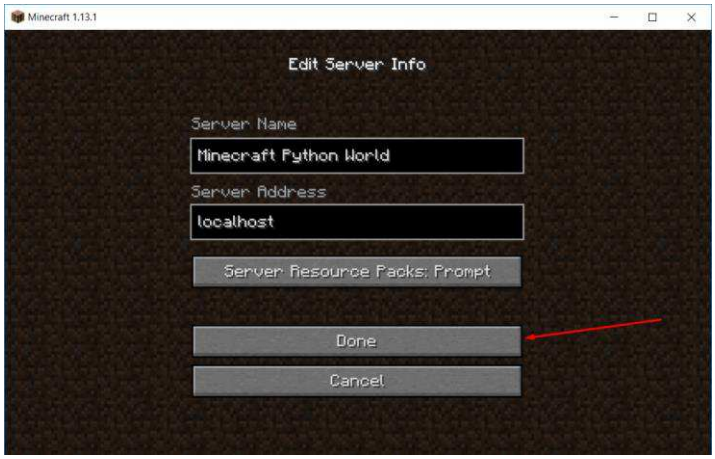
Nato nastavite igro na novo narejen način zagona in pritisnite na gumb »IGRAJ«.



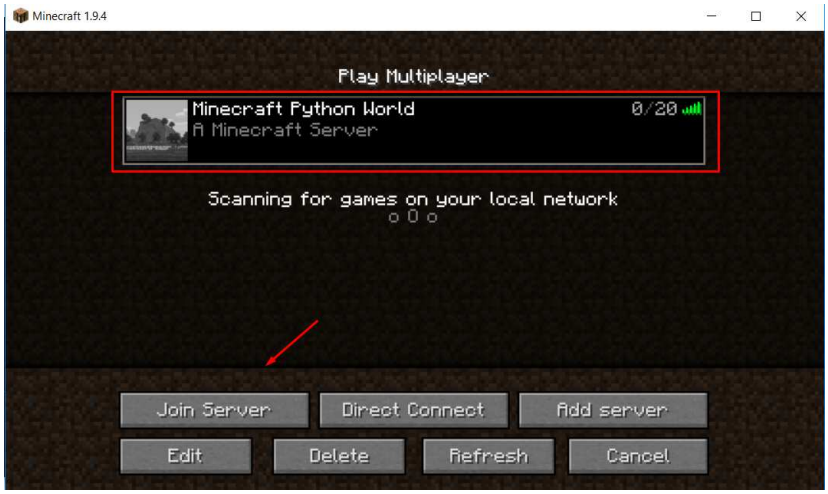
Ko pridete do dela, ko morate izbrati način igre, izberite »Multiplayer«.

Kliknite gumb »Add Server« in pod Server Name napišite »Minecraft Python World«, pod Server Address pa »localhost«. Nato kliknite »Done«.

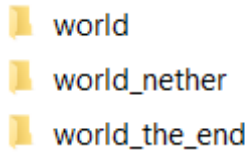




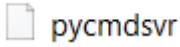
Sedaj lahko z dvojnim klikom na Minecraft Python World odprete svoj svet in vstopite v igro. Če je server označen kot neaktiven, preverite, če imate izbrano različico igre 1.9.4..

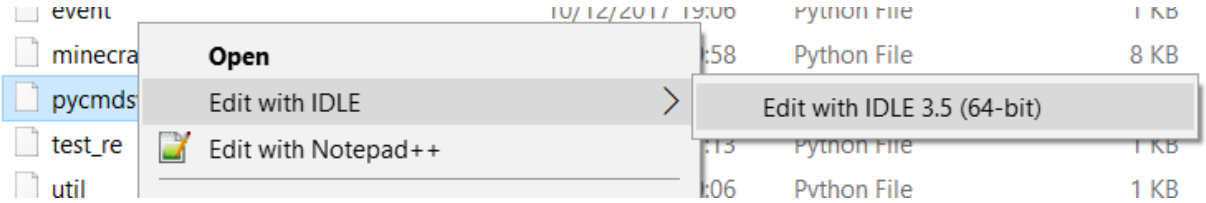


Če bi želeli ustvariti nov svet, pojdite nazaj v mapo server in pobrišite mape word, word_nether in word_the_end.

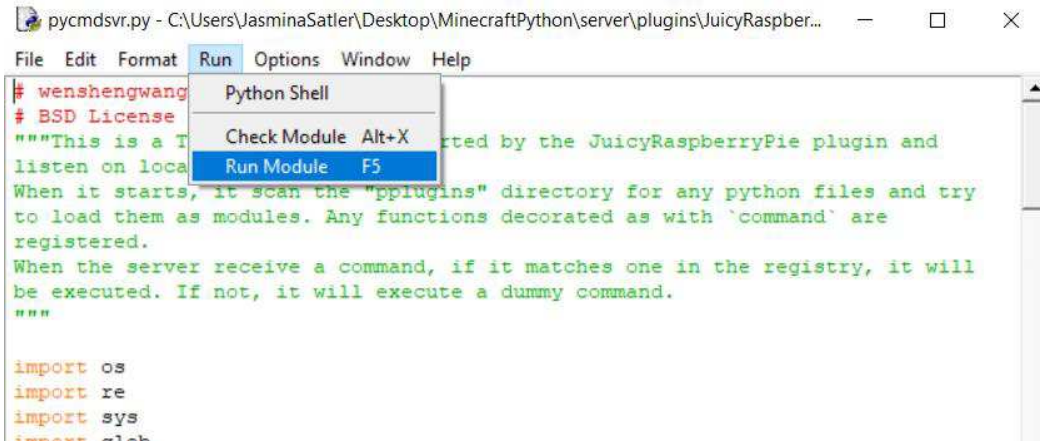


Ko boste želeli začeti programirati, morate imeti zagnan program pycmdsrv. To lahko naredite tako, da odprete datoteko pycmdsrv.py (desni klik in odprite z IDLE) – datoteko najdete v mapi MinecraftPython – server – plugins – JuicyRaspberryPie – mcpi (v prejšnjem koraku ste jo prekopirali na namizje).

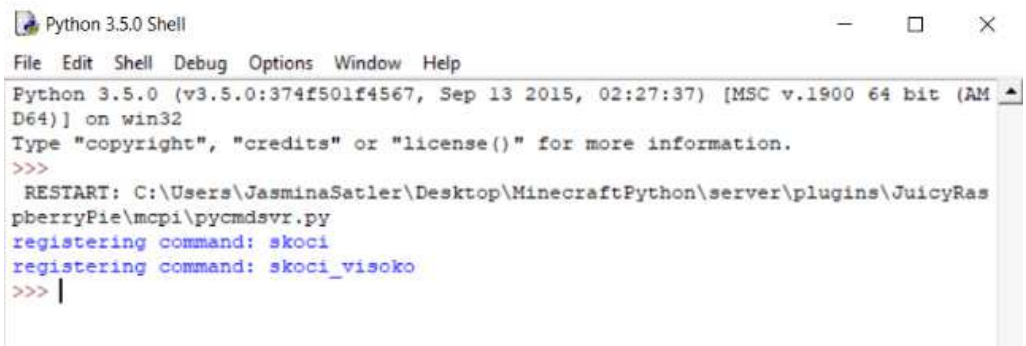
Izgled datoteke v mapi mcpi : 



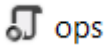
Po tem, ko ste odprli datoteko, jo zaženite preko orodne vrstice, kot je prikazano na sliki ali s pritiskom na tipko F5.



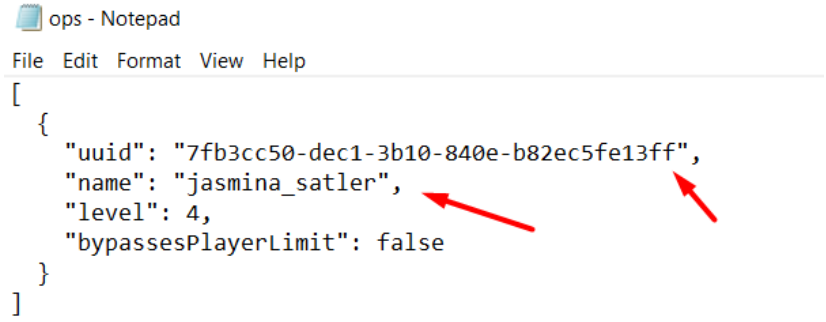
Po zagonu se bo odprlo novo okno, ki ga pustite odprtega.



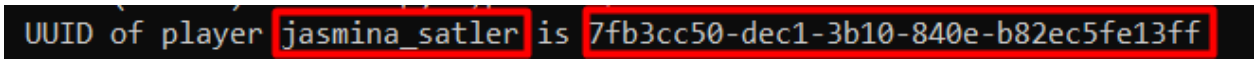
Če vam po uspešni nastavitvi in natančnem sledenju navodil ukazi v klepetu v igri ne delujejo, potem poiščite datoteko ops v mapi server in jo odprite v Notepadu.

Izgled datoteke v mapi server: 

V datoteko zapišite svoje uporabniško ime in UUID, ki predstavlja kodo, pod katero je igralec zapisan v igro, kot sta prikazana na sliki. S tem ste svojemu igralcu dodali dovoljenje za dostop do strežnika.



Uporabniško ime in UUID sta se vam izpisala v oknu serverja, ko ste vstopili v igro Minecraft.



Dodatek B

Učenje Pythona

Drugi in glaven del učnega gradiva, se nahaja na tem mestu. Sestavljen je iz nalog in napotkov, s pomočjo katerih se učimo programskega jezika Python. Podrobneje ga opišemo v poglavju 4.1.1.

UČENJE PYTHONA

Klepet

Vsi, ki smo kdaj igrali igro Minecraft, zagotovo poznamo tudi Minecraftov klepet. Če želimo nekaj sporočiti preostalim igralcem, to naredimo s pritiskom na tipko t, vpišemo želeno besedilo in ga pošljemo s tipko enter. Besedilo se prikaže v klepetu vseh, ki z nami igrajo igro.

Če v klepet napišemo besedo »zdravo«,

se ta izpiše tudi ostalim igralcem.

Prav tako imamo prek klepeta možnost spreminjanja vremena, časa itd. Če zapišemo ukaz */weather clear*, se bo vreme zjasnilo.

Pomoč: spremembe

V igri lahko spremenimo vreme ali menjamo med dnevom in nočjo. To storimo z vnosom kateregakoli ukaza iz spodnjega seznama v Minecraft klepet.

Seznam ukazov:
/weather rain – začne deževati
/weather thunder – začne se neurje
/weather clear – vreme se zjasni
/summon lightning_bolt – naredi se strela
/time set day – postane dan
/time set night – postane noč

Na enak način bomo klicali tudi naše ukaze iz Pythona, le da jih bomo začeli z zapisom *'/p'*.

Do Minecraftovega klepeta lahko dostopamo tudi s pomočjo Pythona. Kako, si bomo ogledali v naslednjem poglavju.

Ukaz

Ko želimo, da računalnik izvede določene naloge, to dosežemo s podajanjem ukazov. V mapi, ki je priložena tem navodilom, imamo že podanih nekaj ukazov. Najprej se bomo naučili, kako jih kličemo, nato pa se jih bomo naučili še ustvarjati.

Med že vnaprej pripravljenimi ukazi najdemo ukaz »skoci«. Kot lahko razberemo že iz imena, z ukazom poskrbimo, da naš igralec skoči. Če bomo v klepetu začeli naše besedilo s »*/p*«, bomo računalniku povedali, da bomo za tem poklicali nek ukaz. Tako z vnosom besedila *»/p skoci()«* računalniku ukažemo, naj naš igralec skoči.

Klic ukaza skoci:

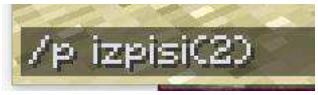
Klic ukaza s podatkom

Zgoraj smo se naučili klica preprostega ukaza »skoci« z uporabo klepeta. Ukaz smo poklicali tako, da smo zapisali ime ukaza ter dodali oklepaja – (). Vsak ukaz ima svoje ime, prav tako pa lahko v oklepaju vsebuje določen podatek. Tako bi lahko naredili ukaz, ki bi sprejel še podatek o višini skoka. Računalniku lahko na primer ukažemo, naj igralec skoči dve kocki visoko. Ukaz bi poklicali s */p skoci_visok(2)*.

En izmed ukazov, ki jih prav tako lahko uporabljamo, je »izpisi«. Ta ukaže računalniku naj piše v Minecraftov klepet. Kot smo omenili v prejšnjem poglavju, je to še en način dostopa do Minecraft klepeta, le da do tega dostopamo s pomočjo Pythona.

Zapišemo */p izpisi(2)* in v klepetu se prikaže 2.

Klic ukaza:



Izpis na ekranu:

Kaj pa, če zapišemo */p izpisi(2+2)*? V našem klepetu se nam prikaže število 4. Juhu! To pomeni, da lahko Minecraftov klepet uporabljamo tudi za računanje.

Klic ukaza:

Izpis na ekranu:

1. naloga: Račun

Hitro skoči do mamice in jo prosi za en račun iz trgovine. Izračunaj, ali so v trgovini pravilno sešteli končni znesek nakupa.

Na enak način lahko uporabimo tudi druge računske operacije. Zapišimo */p izpisi(2+4*2)* in v klepetu se bo izpisalo število 10. Kar naenkrat imamo svoje računalo.

Kaj pa v primeru, da računalniku naročimo, naj izpiše */p izpisi("2+2")?*

Klic ukaza:

Izpis na ekranu:

V klepetu se nam izpiše 2+2.

Pa smo se naučili nečesa novega. Če želimo računalniku povedati, da je nekaj besedilo, to postavimo med narekovaje.

2. naloga: Izpis v klepet

V tem poglavju smo se naučili, kaj je ukaz in kako ga pokličemo. Videli smo, da obstajajo različni tipi podatkov, ki jih lahko uporabljamo (število, besedilo itd.) in da lahko ukaze kličemo s takšnimi podatki.

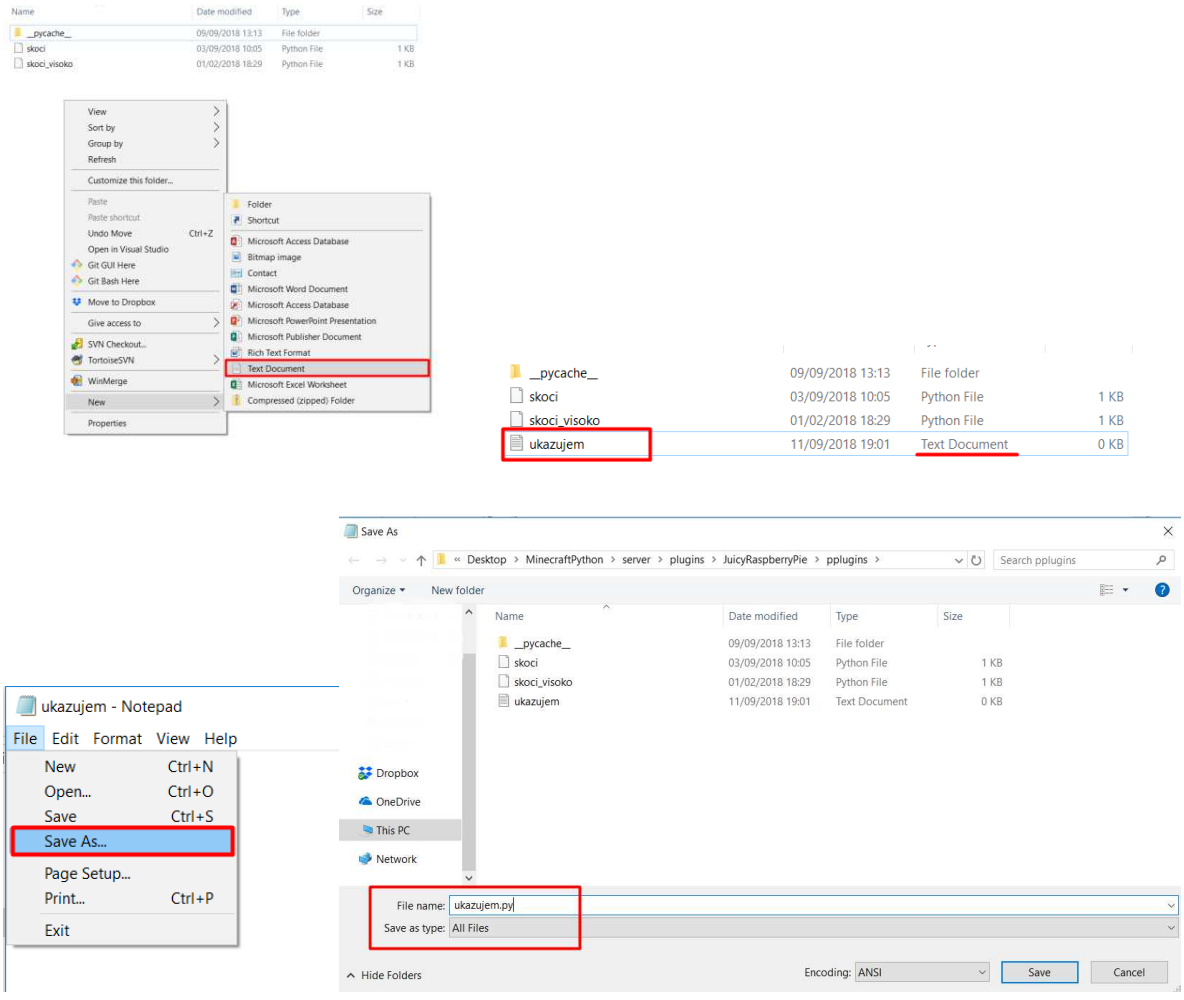
Ko smo izpisovali v klepet, smo izvedeli, kako lahko to naredimo s pomočjo Pythona. Uporabi ukaz »izpisi«, ki bo v klepet izpisal “Živjo, stari, kako si?”.

Namig: ukaz kot podatek sprejme neko podano besedilo, zato bodi pozoren na narekovaje.

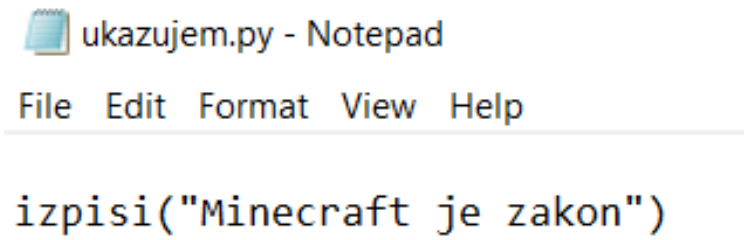
Nov ukaz

Ukaze sedaj znamo uporabljati, ne znamo pa jih še ustvarjati. No, pa se tega naučimo.

Nov ukaz naredimo tako, da v mapi, ki je priložena tem navodilom, najdemo mapo pplugins, ki se nahaja v mapi MinecraftPython, ki je bila v priloženemu paketu (Namizje – MinecraftPython – server – plugins – JuicyRaspberryPie – pplugins). V njej naredimo novo datoteko, ki jo poimenujemo z želenim imenom ukaza, ki ga bo vsebovala, ter shranimo s končnico '.py' (primer: ukazujem.py). To lahko naredimo s pomočjo urejevalnikov besedil. En tak urejevalnik je Notepad.



V Notepadu nato odpremo datoteko ukazujem.py in vanjo vpišemo:

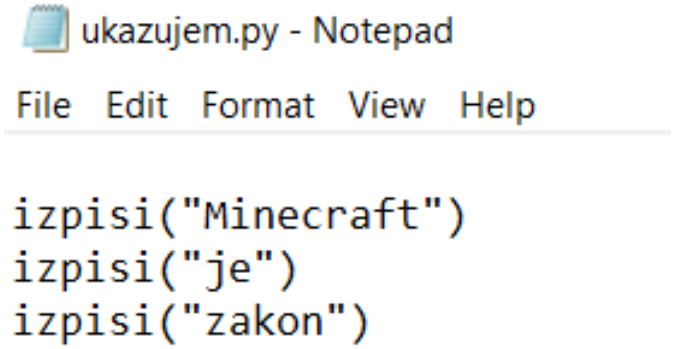


Datoteko shranimo v mapo pplugins. Sedaj moramo ponovno zagnati datoteko pycmdsrv.py (postopek je opisan v navodilih za začetek). V klepetu napiši */p ukazujem()*. Kaj se je zgodilo? Minecraft je to datoteko našel in naredil to, kar smo v datoteki zapisali. Prebral bo vsebino datoteke in v klepetu izpisal:

Vse datoteke, ki bodo vsebovale ukaze, se morajo nujno nahajati v mapi pplugins, sicer jih Minecraft ne bo našel.

Zaporedje ukazov

Do sedaj smo se naučili, kaj je ukaz in kako ga naredimo. Sedaj si bomo pogledali še, kako sestaviti zaporedje ukazov. Ukaz »ukazujem« nam je izpisal “Minecraft je zakon”. Kaj pa, če bi želeli, da nam izpiše vsako besedo v novo vrsto? Ali lahko v datoteko zapišemo več ukazov? Lahko, vendar mora biti vsak ukaz v svoji vrstici. Pa spremenimo ukaz »ukazujem« tako, da bo zapisal vsako besedo v novo vrsto. Vsebina datoteke ukazujem.py:



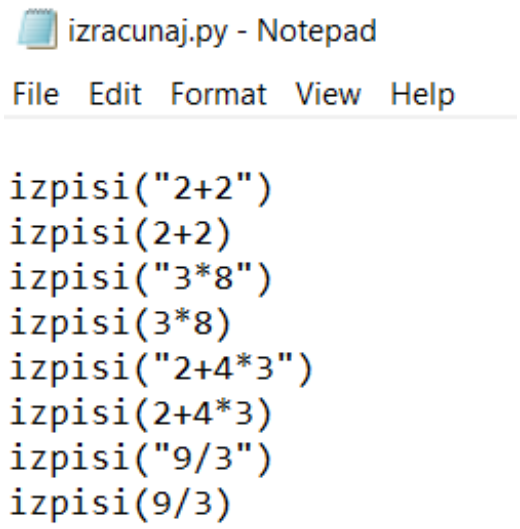
Datoteko shranimo v mapo pplugins in v klepet napišemo */p ukazujem().* Izpis v Minecraftu izgleda takole:

Tako kot prej Minecraft najde datoteko, prebere vsako vrstico posebej in naredi, kar mu je bilo ukazano. Izpisal je vsako vrstico posebej, tako kot smo si želeli.

Pa naredimo še en nov ukaz s pomočjo zaporedja ukazov. Prej smo videli, da lahko s pomočjo Minecrafta tudi računamo. Sedaj bi radi izračunali nekaj računov, ki smo jih dobili pri nalogi iz matematike (hitro skoči po svojo domačo nalogo in najdi nekaj enačb). Naredimo nov ukaz *izracunaj()* s pomočjo zaporedja ukazov.

Ponovimo zgornji postopek. Ustvarimo datoteko izracunaj.py, jo shranimo v mapo pplugins in vanjo zapišemo želene ukaze. V našem primeru bodo ukazi *izpisi(),* ki bodo imeli v oklepajih zapisane račune, ki jih želimo izračunati. Računi so najprej zapisani v narekovajih, nato pa še enkrat brez narekovajev. Se še spomniš, kaj pomenijo narekovaji?

Vsebina datoteke izracunaj.py:



Sedaj lahko že uporabljamo naš novi ukaz. Če v klepet zapišemo */p izracunaj()*, bo izpis izgledal takole:

Najprej se bo izpisal račun, nato pa njegov rezultat.

Zaporedje ukazov imenujemo program. Program si najlažje predstavljamo kot nekakšen recept. Recept vsebuje navodila za pripravo neke jedi, program pa je zaporedje navodil za računalnik.

Tako kot moramo biti previdni pri pisanju receptov, moramo tudi pri pisanju programov paziti na določena pravila. Zelo pomembno je zaporedje, v katerem podamo ukaze, saj jih računalnik prebere v takšnem vrstnem redu, kot jih zapišemo. Pa si poglejmo recept za jabolčni zavitek. Zagotovo si že kdaj opazoval mamo, kako dela štrudelj. Najprej je pripravila testo, ga razvaljala, nato pa nanesla nadev. Kaj bi se zgodilo, če bi v receptu pisalo, naj testo najprej razvalja, ga nato naredi in zatem nanese nadev? Lahko jo greš vprašat in skupaj poskusita v tem zaporedju. Zagotovo bosta prišla do zaključka, da tako ne bo šlo. Če recept ni natančen, hitro pridemo do napačnih rezultatov, prav tako pa je pri sestavljanju programov.

3. naloga: Športnik

S pomočjo vsega, kar si se do sedaj naučil, ustvari ukaz sportnik. Ko boš ukaz priklical, naj igralec skoči v zrak v višini 3 kocke, nato pa naj se v klepetu izpiše »juhu!!«. Zatem naj igralec zopet skoči, tokrat še za 2 kocki, nato pa naj se v klepetu izpiše »Sedaj sem pa utrujen!«.

Klic ukaza: */p sportnik()*.

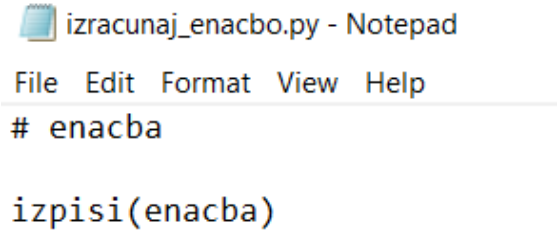
Namig: uporabi ukaza »izpisi« in »skoci« ter zaporedje ukazov.

Nov ukaz s podatkom

Kaj pa, če želimo, da zgornji ukaz »ukazujem« vedno izpiše nekaj novega?

Želimo, da računalnik izračuna nek račun, v Minecraftu pa naj se izpiše rezultat. Ker nočemo vedno znova spreminjati vsebine ukaza oz. datoteke, si poglejmo, kako ustvarimo ukaz, ki sprejme nek podatek. Ukaz poimenujmo *izracunaj_enacbo*.

Vsebina datoteke izracunaj_enacbo.py izgleda takole:



Kako kličemo ukaz s podatkom, smo se že naučili. V klepet napišemo */p izracunaj_enacbo(2+2).* V našem primeru bo računalnik prepoznal 2+2 kot račun oziroma enačbo in jo uporabil v ukazu »izpisi«. V klepetu se nam bo tako izpisal izračun naše enačbe:

Če želimo ukazu poleg tega pošiljati še kakšne podatke, to naredimo tako, da v prvo vrstico datoteke zapišemo *# ime_podatka.* V našem primeru je bila to “enačba”. Če želimo ukazu poslati več podatkov, te ločimo z vejico. Tako bi prva vrstica datoteke izgledala takole *# ime_podatka_1, ime_podatka_2, ime_podatka_3.*

4. naloga: Uspelo mi je

Naredi program z imenom »uspelo_mi_je«. S klicem programa igralec skoči za določeno višino, v klepetu se izpiše besedilo, nato ponovno skoči ter se še enkrat v klepetu izpiše “uspelo mi je!!”. Višina in besedilo sta podatka s katerima kličemo ukaz.

Klic ukaza: */p uspelo_mi_je(3, “uspelo mi je!!”)*

Namig: program lahko kot podatka dobi višino in besedilo – kako naredimo ukaz z več podatki, pa smo se naučili v tem poglavju.

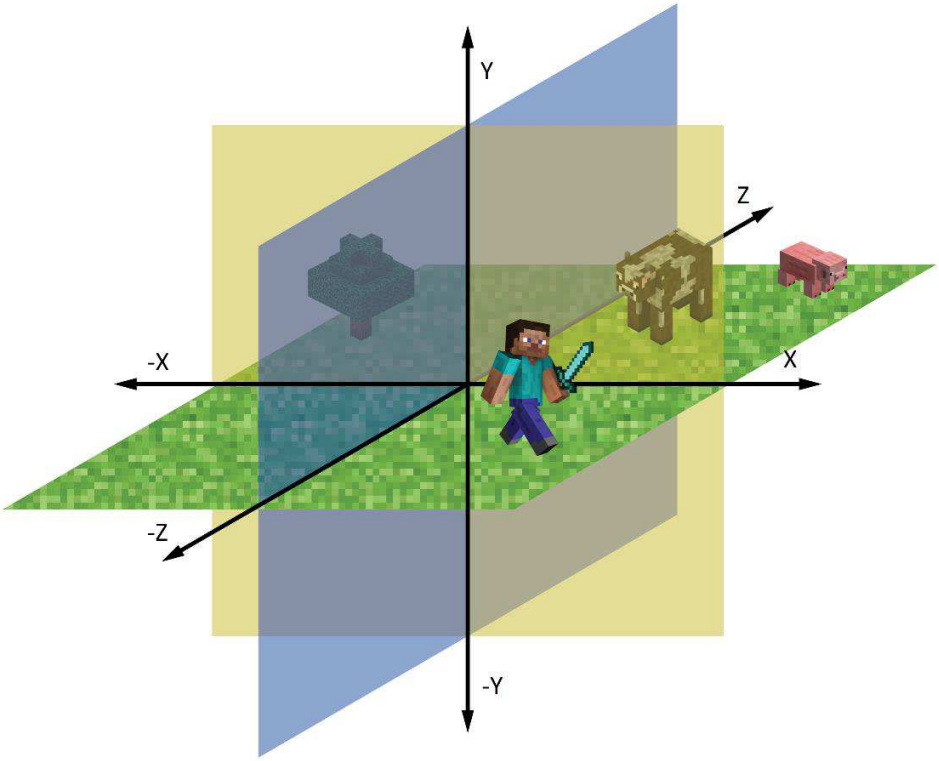
Koordinatni sistem

Ker pa bi bilo le izpisovanje v klepet in skakanje z našim igralcem čisto preveč dolgočasno, si poglejmo še en ukaz – »premakni_se«. Ta ukaz kličemo tako, da mu kot podatke podamo tri števila, na primer */p premakni_se(1,2,3).*

Sedaj pojdi v klepet in pokliči ukaz s tremi poljubnimi števili. Ko to narediš, boš opazil, da igralca nekam odnese. Poskusi se malo poigrati in uganiti, kaj pomenijo ta tri števila. (Nasvet: spreminjaj samo eno in opazuj, kaj se dogaja z igralcem.)

Ta tri števila predstavljajo določeno točko, kamor prestavimo igralca. S klicem tega ukaza računalniku povemo, da želimo premakniti igralca na točko (x, y, z). S tremi števili lahko povemo, kje v prostoru naj se nahaja naš igralec. Potrebujemo točno tri, tem trem številom pa pravimo koordinate.

Igralec se nahaja v Minecraftovem svetu, ta virtualni svet pa si lahko predstavljamo kot koordinatni sistem.



Na naši sliki imamo igralca, medveda, pujsa in drevo. Vsaka točka v Minecraft svetu je predstavljena s tremi koordinatami: x, y in z. Koordinate likov so:

- Igralec (1, 0, -1)
- Medved (1, 0, 2)
- Pujs (2, 0, 3)
- Drevo (-2, 0, 1)

Da ti ne bo potrebno ugibati, kje se igralec nahaja v tem trenutku, lahko vedno dostopaš tudi do njegovih trenutnih koordinat. Imamo tri magične besede, ki nam predstavljajo koordinate točke, na kateri se trenutno nahaja igralec:

- **pozicija.x** – x koordinata
- **pozicija.y** – y koordinata
- **pozicija.z** – z koordinata

Kadarkoli boš vpisal »pozicija.x«, ti bo računalnik podal določeno številko. Če bomo torej klicali ukaz *izpis(pozicija.x),* se nam bo izpisala x koordinata točke, kjer se trenutno nahaja naš igralec. Sedaj poskusi še sam.

4. naloga: Skoči

Na začetku smo za prikaz klica ukaza uporabili ukaz »skoci_visoko«. Sedaj za nalogo naredi ukaz, ki bo povzročil, da igralec skoči za podano višino.

Klic ukaza: */p skoci_visoko(3)*

Namig: ukaz kot podatek sprejme višino ter uporablja ukaz *premakni_se.*

Spremenljivke

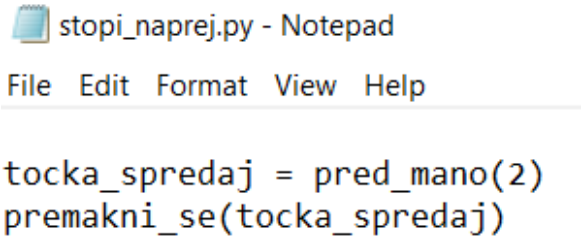
Do sedaj smo spoznali že kar nekaj ukazov, s katerimi povzročimo določen dogodek v igri. Lahko pa imamo tudi takšne ukaze, ki dosežejo, da računalnik nekaj izračuna. Ko boš uporabil takšen ukaz, bo isto, kot če bi napisal tisto številko, ki jo ta ukaz izračuna. Za takšne ukaze lahko rečemo, da nam vrnejo neko vrednost.

Primer enega izmed takšnih ukazov je *pred_mano(stevilo_kock).* Ukaz nam vrne koordinate točke, ki je za število kock pred igralcem, v smeri, v katero je obrnjen. Če želimo na te koordinate kaj postaviti ali se nanje premakniti, pa jih moramo nekam shraniti. Shranili jih bomo s pomočjo spremenljivk.

Spremenljivko si najlažje predstavljamo kot škatlo, kamor pospravimo neko svojo igračo. Škatla ima svoje ime, saj le tako lahko nekomu povemo iz katere škatle želimo, da nam da igračo. V programiranju takim škatlam rečemo spremenljivke, vrednosti, ki lahko shranjujemo vanjo, pa so lahko števila, besedila, točke itd. Ko bomo potrebovali vrednost, ki smo jo shranili v spremenljivko, bomo uporabili njeno ime in računalnik bo vedel kaj želimo, da nam vrne.

Pa si poglejmo na primeru. Naredimo program *stopi_naprej,* v katerem bomo ustvarili spremenljivko *tocka_spredaj*, v katero bomo kot vrednost shranili točko, ki se nahaja dve kocki pred našim igralcem, nato pa igralca premaknili na to točko.

Vsebina datoteke stopi_naprej.py:



Kot smo povedali zgoraj, lahko v spremenljivko shranimo našo točko. Prav tako pa lahko točko zapišemo v več spremenljivkah, ki predstavljajo koordinate točke. Tako lahko v zgornji nalogi shranimo točko kot:

- **eno spremenljivko** - *sprednja_tocka = pred_mano(2)***.** V tem primeru je spremenljivka točka in jo lahko uporabimo, kot smo jo v zgornjem primeru.

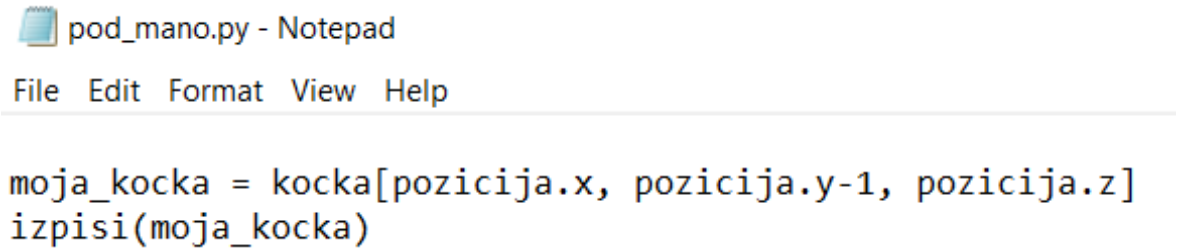
- **tri spremenljivke**, ki predstavljajo koordinate točke - *x, y, z = pred_mano(2)***.** V tem primeru lahko direktno dostopamo do koordinat točke, se pravi lahko kasneje uporabimo samo eno koordinato.

Pozicija je prav tako primer ene izmed spremenljivk in jo lahko uporabimo kadarkoli. Ime te spremenljivke je »pozicija«, vrednost pa je točka, na kateri se igralec nahaja.

Kocka

Ker vemo, da je Minecraftov svet sestavljen iz kock, bi se radi s kockami tudi igrali. Pogledali si bomo, kako lahko kaj zgradimo, spremenimo, razdremo itd. Spoznali bomo ukaz *kocka[x, y, z]*, kjer so x, y in z koordinate točke. Naredi ukaz *pod_mano()*, ki bo v klepet izpisal, kar je vrnil ukaz *kocka[pozicija.x, pozicija.y, pozicija.z]*.

Vsebina datoteke *pod_mano.py*:



Izpis v klepetu:

Pri tebi je izpis lahko drugačen. Veš, zakaj? Če ne veš, potem dobro razmisli, kaj naredi koda v tem ukazu.

Najprej pokliči ukaz *pod_mano()*. No, sedaj pa se malo prestavi in še enkrat pokliči ukaz. Postopek ponovi še enkrat. Če si se dovolj premikal in imel srečo, ti je vsakič izpisalo nekaj drugega. Verjetno si že sam ugotovil, čemu služi ta ukaz. Tako je, izpiše nam material, iz katerega je sestavljena kocka, katere koordinate si podal. Ker pa ne želimo vedeti le, kateri material se kje nahaja, se bomo naučili še, kako lahko s pomočjo kock tudi gradimo.

Preprosto, če želimo postaviti neko kocko, uporabimo ukaz **kocka[x, y, z] = material**. V ukazu potrebujemo koordinate točke, kamor želimo kocko postaviti ter material, ki predstavlja material kocke, ki jo postavljamo. Kateri materiali se pojavljajo v Minecraftovem svetu, si lahko pogledaš na spodnjem seznamu.

Materiali

Material je določen z imenom in številom. V programih in igri lahko uporabimo katerikoli material ali število iz spodnjega seznama. Pri imenih bodi pozoren na podčrtaje ter velike in male črke.

Zrak = 0	VisokaTrava =31	Tracnice = 65
Kamen = 1	PosusenGrm = 32	KamniteStopnice = 67
Trava = 2	Volna = 35	ZeleznaVrata = 71
Zemlja = 3	RumenaRoza = 37	RudaRdecegaKamna = 73
NeobdelanKamen = 4	ModraRoza = 38	Sneg = 78
LeseneDeske = 5	RjavaGoba = 39	Led = 79
Sadika = 6	RdecaRoza = 40	KockaSnega = 80
TemeljniKamen = 7	KockaZlata = 41	Kaktus = 81
Voda = 8	KockaZeleza = 42	KockaGline = 82
StojecaVoda = 9	KamnitaPlosca = 44	SladkornaTrs = 83
Lava = 10	Opeka = 45	Ograja = 85
StojecaLava = 11	TNT = 46	PeklenskiKamen = 87
Pesek = 12	KnjiznaPolica = 47	PesekDus = 88
Gramoz = 13	MahovnatKamen = 48	SvetleciKamen = 89
ZlataRuda = 14	Obsidian = 49	Loputa = 96
ZeleznaRuda = 15	Bakla = 50	KamniteOpeke = 98
Les = 17	Ogenj = 51	SteklenaPlosca = 102
Listje = 18	LeseneStopnice = 53	Melona = 103
Spuzva = 19	Skrinja = 54	MeloninaVitica = 105
Steklo = 20	DiamantnaRuda = 56	OgrajnaVrata = 107
RudaLapisLazulija = 21	DiamantnaKocka = 57	StopniceIzOpek = 108
KockaLapisLazulija = 22	DelovniPult = 58	StopniceIzKamnitihOpek = 108
Izstreljevalec = 23	Psenica = 59	PeklenskaOpeka = 112
Pescenjak = 24	ObdelanaZemlja = 60	StopniceIzPeklenskihOpek = 114

Postelja = 26	PecicaNeaktivna = 61	StopniceIzPescnjaka = 128
ElektricneTracnice = 27	PecicaAktivna = 62	KockaKvarca = 155
ZaznavneTracnice = 28	Napis = 63	StopniceIzKvarca = 156
LepljiviBat = 29	LesenaVrata = 64	SvetlecObsidian = 246
Pajcevina = 30	Lestev = 65	

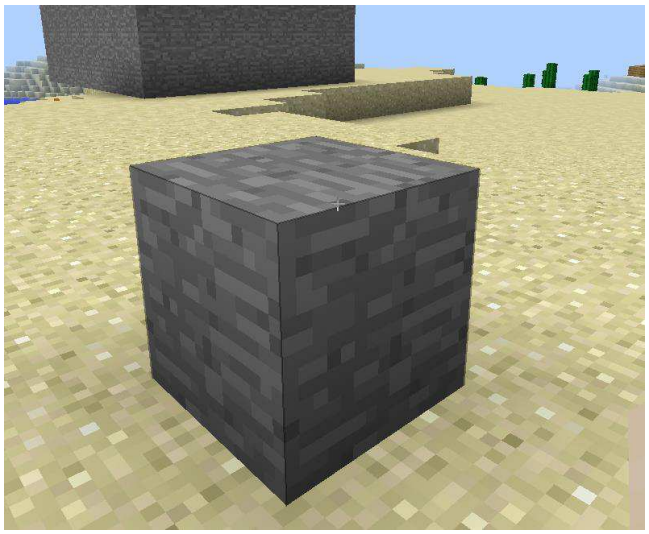
Naredimo program, ki bo dve kocki pred igralca postavil kocko iz materiala Kamen.

Vsebina datoteke *postavi_kamen.py*:

postavi_kamen.py - Notepad

File Edit Format View Help

```
x, y, z = pred_mano(2)
kocka[x, y, z] = Kamen
```



5. naloga: Obzidje

Naredi program z imenom *obzidje*. Program kot podatek sprejme material in naredi zid okoli igralca. Zid naj bo visok dve kocki ter sestavljen iz kock podanega materiala.

Klic ukaza: */p obzidje(Kamen).*

Namig: pomagaj si s pozicijo.

Če znamo postaviti kocko, znamo postaviti tudi zid. Pa ga zdaj postavimo. Spoznali smo ukaz »kocka«, ki ga bomo uporabili večkrat zapored. Vsakič bomo poiskali koordinate točk na katere želimo postaviti kocko in jih v smiselnem zaporedju sestavili v zid.

Vsebina datoteke zid.py:

zid.py - Notepad

File Edit Format View Help

```
x, y, z = pred_mano(5)
kocka[x, y, z] = Kamen
kocka[x, y + 1, z] = Kamen
kocka[x, y + 2, z] = Kamen
kocka[x + 1, y, z] = Kamen
kocka[x + 1, y + 1, z] = Kamen
kocka[x + 1, y + 2, z] = Kamen
kocka[x + 2, y, z] = Kamen
kocka[x + 2, y + 1, z] = Kamen
kocka[x + 2, y + 2, z] = Kamen
kocka[x - 1, y, z] = Kamen
kocka[x - 1, y + 1, z] = Kamen
kocka[x - 1, y + 2, z] = Kamen
kocka[x - 2, y, z] = Kamen
kocka[x - 2, y + 1, z] = Kamen
kocka[x - 2, y + 2, z] = Kamen
```

Več kock

Zgoraj smo postavljali zid, kjer smo večkrat klicali ukaz »kocka«. Če bi želeli spremeniti material zidu in narediti lesen zid, bi morali spremeniti kar nekaj programa. Kako zoprno! Mi se še vedno počutimo lene in bomo zato spoznali še en ukaz - **kocke[x: x + dolzina, y + visina, z: z + sirina] = material**. Ta deluje na podoben način kot kocka, le da moramo navesti dve koordinati in jih ločiti z dvopičjem. S tem ukazom bomo v območju koordinat (spodnja koordinata je vključena, zgornja ne) postavili kocke izbranega materiala. Se pravi, če bi za koordinate x, y in z vzeli dve kocki pred igralcem in uporabili *kocke[x : x, y : y + 2, z : z] = Kamen*, bi dve kocki pred igralcem postavili kocke materiala Kamen, in sicer višine treh kock (za navpično koordinato računalnik vzame y, y+1 in y+2).

Pa se vrnimo na prejšnjo nalogo, kjer smo sestavljali zid. V nalogi smo postavljali kocke eno zraven druge, eno na drugo itd., sedaj pa bomo postavili zid še s pomočjo večih kock. Poimenujmo program kar *zid_kocke*.

Vsebina datoteke zid_kocke.py:

zid_kocke - Notepad

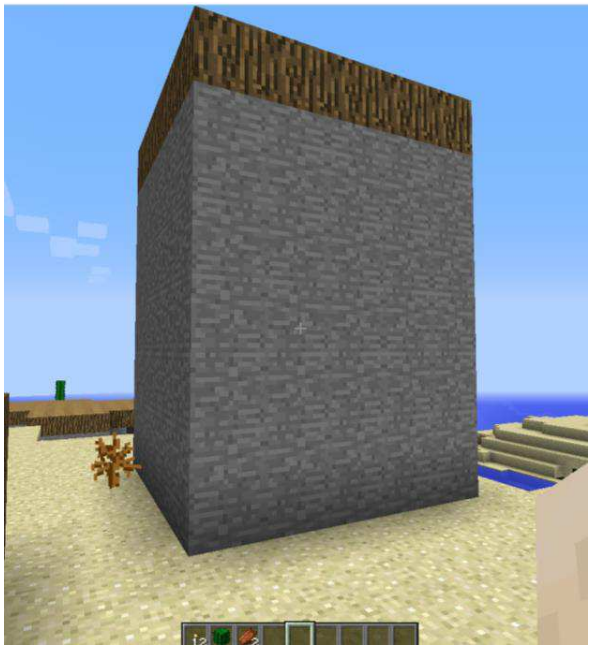
File Edit Format View Help

```
# dolzina, visina, sirina, material

x, y, z = pred_mano(2)
kocke[x : x + dolzina, y : y + visina, z : z + sirina] = material
```

Program postavi zid , ki je iz želenega materiala in je podane dolžine, višine in širine, dve kocki pred igralca.

Sedaj pa se lahko malo poigramo in z nekaj domišljije zgradimo tudi kaj večjega. Pa si poglejmo, kako bi izgledal program, ki zgradi hišo.

```
# dolzina, visina, sirina, material

x, y, z = pred_mano(2)
kocke[x : x + dolzina, y : y + visina - 1, z : z] = material
kocke[x : x, y : y + visina - 1, z : z + sirina - 1] = material
kocke[x : x + dolzina, y : y + visina - 1, z : z + sirina] = material
kocke[x : x + dolzina, y : y + visina - 1, z : z + sirina] = Les
```



Naloge smo se lotili tako, da smo postavili stranice hiše, nato pa na te stranice postavili še streho iz lesa. Lahko pa se tega lotimo tudi na drugačen način. Ustvarimo najprej eno veliko (polno!) kocko, v kateri bomo nato naredili luknjo, in sicer tako, da vanjo postavimo manjšo kocko zraka. Nato dodamo streho.

```
# dolzina, visina, sirina, material

x, y, z = pred_mano(2)
kocke[x : x + dolzina, y : y + visina, z : z + sirina] = material
kocke[x + 1 : x, y + 1 : y + visina - 1, z + 1 : z + sirina - 1] = Zrak
kocke[x : x + dolzina, y : y + visina - 1, z : z + sirina] = Les
```

Način, na katerega želimo postavljati razne elemente v igro, si torej izberemo sami. In tako je tudi pri pisanju programov.

Ker pa znamo graditi, znamo tudi saditi. Pa si poglejmo še, kako bi posadili drevo.
Najprej ustvarimo deblo, ki ni nič drugega kot le stolp kock, ki so iz materiala Les.

```
#deblo
kocke[x : x, y : y + 4, z : z] = Les
```

Na drevo postavimo še krošnjo. S krošnjo se lahko malo poigramo in pomešamo kocke materiala Listje in Zrak. Najprej ustvarimo dva velika bloka kock iz materiala Listje, nato pa vmes postavljamo kocke iz materiala Zrak.

```
 #krosnja
kocke[x - 2 : x + 2, y + 4 : y + 5, z - 2 : z + 2] = Listje
kocke[x - 1 : x + 1, y + 6 : y + 7, z - 1 : z + 1] = Listje
kocke[x - 1 : x - 1, y + 7 : y + 7, z - 1 : z - 1] = Zrak
kocke[x + 2 : x + 2, y + 5 : y + 5, z - 2 : z - 2] = Zrak
kocke[x + 2 : x + 2, y + 4 : y + 4, z + 2 : z + 2] = Zrak
kocke[x + 1 : x + 1, y + 6 : y + 7, z - 1 : z - 1] = Zrak
kocke[x + 1 : x + 1, y + 6 : y + 7, z + 1 : z + 1] = Zrak
kocke[x - 1 : x - 1, y + 6 : y + 7, z + 1 : z + 1] = Zrak
```

Vsebina datoteke drevo.py:

drevo - Notepad

File Edit Format View Help

```
x, y, z = pred_mano(2)

#deblo
kocke[x : x, y : y + 4, z : z] = Les

#krosnja
kocke[x - 2 : x + 2, y + 4 : y + 5, z - 2 : z + 2] = Listje
kocke[x - 1 : x + 1, y + 6 : y + 7, z - 1 : z + 1] = Listje
kocke[x - 1 : x - 1, y + 7 : y + 7, z - 1 : z - 1] = Zrak
kocke[x + 2 : x + 2, y + 5 : y + 5, z - 2 : z - 2] = Zrak
kocke[x + 2 : x + 2, y + 4 : y + 4, z + 2 : z + 2] = Zrak
kocke[x + 1 : x + 1, y + 6 : y + 7, z - 1 : z - 1] = Zrak
kocke[x + 1 : x + 1, y + 6 : y + 7, z + 1 : z + 1] = Zrak
kocke[x - 1 : x - 1, y + 6 : y + 7, z + 1 : z + 1] = Zrak
```

Moje drevo izgleda takole:

Lahko pa se še sam preizkusiš v ustvarjanju svojih oblik krošenj.

6. naloga: Hiša z vrtom

V tem poglavju smo spoznali, kako lahko uporabljamo kocke in v kakšne namene bi nam lahko prišle prav. Pokazali smo tudi dva različna načina, kako s pomočjo Pythona postavimo hišo in se naučili, da se lahko pisanja programa lotimo na več različnih načinov.

Za nalogo poskusi narediti hišo z vrtom. Napiši program, ki bo 4 kocke pred igralcem postavil hišo, nato pa za hišo postavil 3 drevesa. Drevesa lahko postaviš enega pred drugim, enega zraven drugega ali pa jih le naključno razporediš okoli hiše. Zraven postavi še kakšno rožico. Poizkusi se malo poigrati s postavitvami elementov v Minecraftovem svetu.

Klic ukaza: */p hisa_z_vrtom().*

Namig: roža ni nič drugega kot kocka materiala roza. Tako je na primer Rumena roža kocka iz materiala RumenaRoza oziroma 37.

Pogojni stavek

V prejšnjem poglavju smo se naučili saditi. Verjetno pa si, če si se igral z nalogo iz prejšnjega poglavja (hiša z vrtom), ugotovil, da tla v Minecraftovem svetu niso ravna. Hiš in dreves seveda ne želimo v zraku.

Kaj, če želimo posaditi rožico? Če bi jo hoteli posaditi v puščavi, bi bilo to precej neumno. Zato bomo najprej preverili, če se nahajamo na kocki iz materiala Trava in če to drži, potem bomo rožo posadili. Ukaz bomo poimenovali kar *posadi_rozo(),* posadili bomo pa rožo rumene barve (material RumenaRoza). Po zaključenem pogoju moramo zapisati še znak dvopičje (:) in vsem vrsticam kode, ki se bo izvedla ob izpolnjenem pogoju, spredaj dodati presledek (koda bo delovala tudi brez, vendar so presledki pomembni, če boš še kdaj programiral v Pythonu).

Vsebina datoteke *posadi_rozo.py*:

```
# stevilo_kock

x, y, z = pred_mano(stevilo_kock)
if kocka[x, y, z] == Trava:
        kocka[x, y + 1, z] = RumenaRoza
```

Tukaj smo se srečali s pogojnim stavkom - IF. S pogojnim stavkom računalniku ukažemo, naj izvede določeno dejanje, če je izpolnjen nek pogoj. V našem primeru je ta pogoj izpolnjen, če je kocka pred igralcem iz materiala Trava. Če je kocka iz kateregakoli drugega materiala, se ne zgodi nič.

Zgornji program posadi rožo le v primeru, da je material kocke, na kateri želimo saditi, Trava. Če ni, pa ne želimo, da bi uporabnik mislil, da se mu je pokvaril računalnik ali da ukaz ne dela, ampak bi mu želeli sporočiti, da to ne gre. To pa lahko naredimo tako, da po pogojnem stavku (stavku IF) povemo še, kaj naj stori, če ni mogel saditi, torej, če pogoj ni bil izpolnjen.

Pa nadgradimo naš program, ki posadi rožo. Če ne bomo mogli posaditi rože, želimo to izpisati v klepet.

Vsebina datoteke *posadi_rozo.py*:

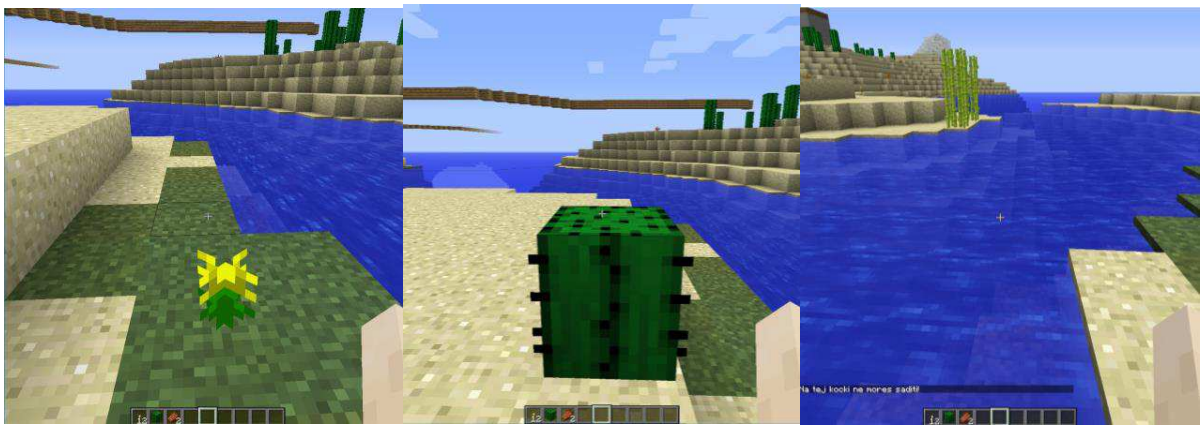
```
 # stevilo_kock

x, y, z = pred_mano(stevilo_kock)
if kocka[x, y, z] == Trava:
        kocka[x, y + 1, z] = RumenaRoza
else:
        izpisi("Na tej kocki ne mores saditi!")
```

Kaj pa, če od programa želimo še več? V primeru, da je material Trava, želimo posaditi rožo, v primeru, da je material Pesek, želimo posaditi kaktus, drugače pa izpisati, da to ne bo šlo. Tudi to lahko naredimo. Preverjamo lahko več različnih pogojev.

Vsebina datoteke *posadi_rozo.py*:

```
 # stevilo_kock

x, y, z = pred_mano(stevilo_kock)
if kocka[x, y, z] == Trava:
        kocka[x, y + 1, z] = RumenaRoza
elif kocka[x, y, z] == Pesek:
        kocka[x, y + 1, z] = Kaktus
else:
        izpisi("Na tej kocki ne mores saditi!")
```

Kot lahko vidimo, na travi zraste roža, na pesku kaktus, če pa želimo saditi na vodi, dobimo obvestilo, da to žal ne gre.



Zaporednih pogojnih stavkov imamo lahko več. Vedno pa moramo pri programiranju paziti, da so zapisani v pravilnem vrstnem redu. Če je pogoj izpolnjen, se vse znotraj bloka izvede, v nasprotnem primeru program naprej preveri naslednji pogoj.

Pomoč: posebni ukazi

Poznamo tudi nekaj posebnih ukazov za IF. Ti ukazi nam povedo, če lahko nekaj naredimo, prav tako pa jih lahko uporabimo kot pogoj v IF stavku:

ali_lahko_sadim(x, y, z)
ali_lahko_kopljem(x, y, z)
ali_lahko_gradim_most(x, y, z)

Pa poskusimo še nekaj novega. Sprehodimo se in se ustavimo pred vodo. Najprej bomo preverili, če na kocki pred igralcem res lahko gradimo most. Most lahko gradimo, če je kocka pred igralcem materiala Zrak. Da nam tega ne bo treba vedno preverjati, lahko uporabimo kar ukaz ali_lahko_gradim_most(x, y, z), ki se spremeni v »da«, če je površina primerna, ali v »ne«, če površina ni primerna. Če most lahko gradimo, potem na točko pred igralca postavimo lesene deske, drugače pa se prestavimo naprej. Ukaz bomo poimenovali »most«.

Vsebina datoteke *most.py*:

```
 x, y, z = pred_mano(1)
if ali_lahko_gradim_most(x, y, z):
        kocka[x, y - 1, z] = LeseneDeske
else:
        premakni_se(x, y, z)
```

7. naloga: Hiša z vrtom 2

V šesti nalogi si postavil hišo z vrtom. Ko si postavljal rože, drevesa ali karkoli drugega, si zagotovo nekaj postavil v zraku. Popravi svojo nalogo tako, da boš dodal pogojne stavke. Preden nekaj postaviš, preveri ali to lahko res narediš na želeni površini.

Klic ukaza: */p hisa_z_vrtom().*

Zanke

Poskusimo ustvariti most. Most ni sestavljen le iz ene kocke in zato bi morali, če bi želeli postaviti most, program zagnati večkrat ali pa isti sklop kode uporabiti večkrat.

Najprej spremenimo zgornji program tako, da se bo igralec premaknil na lesene deske, če bodo te že postavljene.

```
x, y, z = pred_mano(1)
if ali_lahko_gradim_most(x, y, z):
        kocka[x, y - 1, z] = LeseneDeske
        premakni_se(x, y, z)
```

Če sedaj želimo, da igralec nad vodo postavi lesene deske in se premakne nanje, nato pa postopek ponovi, imamo za izvedbo tega zopet več načinov. Prvi način, na katerega pomislimo, je, da program zaženemo še enkrat. Program seveda lahko zaganjamo večkrat. Lahko pa preprosto uporabimo isti sklop kode večkrat. Tako bi naš program izgledal takole:

```
x, y, z = pred_mano(1)
if ali_lahko_gradim_most(x, y, z):
        kocka[x, y - 1, z] = LeseneDeske
        premakni_se(x, y, z)

x, y, z = pred_mano(1)
if ali_lahko_gradim_most(x, y, z):
        kocka[x, y - 1, z] = LeseneDeske
        premakni_se(x, y, z)

x, y, z = pred_mano(1)
if ali_lahko_gradim_most(x, y, z):
        kocka[x, y - 1, z] = LeseneDeske
        premakni_se(x, y, z)
```

Če pogledamo prejšnji ukaz, vidimo, da se nam zgornja koda ponovi trikrat. Ali lahko ustvarimo ukaz, ki bi nam klical isti blok kode večkrat? Seveda, spoznali bomo zanke. Zanka je ponavljanje nekega ukaza ali zaporedja ukazov in ravno to potrebujemo za naš program.

Napišimo zgornji program z uporabo zanke:

```
 stevilo_ponovitev = 3
while stevilo_ponovitev > 0:
        x, y, z = pred_mano(1)
        if ali_lahko_gradim_most(x, y, z):
                kocka[x, y - 1, z] = LeseneDeske
                premakni_se(x, y, z)
        stevilo_ponovitev = stevilo_ponovitev - 1
```

Določili smo število ponovitev, nato izvedli naš blok kode in, ker smo eno ponovitev že naredili, smo število ponovitev zmanjšali za eno. Tako smo napisali program, ki naredi enako kot zgornji, s to razliko, da smo zanj potrebovali dosti manj pisanja.

Kaj pa, če ne vemo, kako daleč moramo iti, da pridemo do kopnega in želimo ponavljati celotno zaporedje ukazov, dokler ga ne dosežemo? V ta namen le malo spremenimo kodo. Najprej potrebujemo točko pred igralcem. Dokler lahko gradimo most na točki pred igralcem, želimo postavljati Lesene deske in premikati igralca naprej. Pozorni moramo biti, da na koncu zaporedja ukazov ne pozabimo določiti novih koordinat, saj se točka pred igralcem spreminja s premikanjem igralca. Če tega ne naredimo, se zanka ne prekine in računalnik vedno postavlja le eno kocko na eno in isto točko.

```
 x, y, z = pred_mano(1)
while ali_lahko_gradim_most(x, y, z):
        kocka[x, y - 1, z] = LeseneDeske
        premakni_se(x, y, z)
        x, y, z = pred_mano(1)
```

Zaključek

Prišel si do konca našega učnega gradiva. Naučil si se delati ukaze, ukaze s podatki, uporabljati pogojne stavke in zanke. Uporabi na novo pridobljeno znanje in ideje, ki si jih dobil, medtem ko si se učil s pomočjo tega učnega gradiva, in naredi še svoje ukaze. S Pythonom lahko v igri ustvariš še veliko več, vendar moraš od tu naprej raziskovati sam.

V priloženi datoteki imaš tudi mapo Dodatni ukazi, kjer najdeš nekaj dodatnih ukazov. Preden jih preizkusiš v igri, jih odpri in razmisli, kaj naredijo.

Dodatek C

Dodatni ukazi

Dodali smo še nekaj kratkih ukazov, s katerimi bi učenci ponovili snov in dobili dodatne ideje za nove funkcije.

drevesna_pot

```
# steviloDreves

x, y, z = pred_mano(3)
i = 0
while i < steviloDreves:
   if ali_lahko_sadim(x-1, y, z) and ali_lahko_sadim(x+1, y, z):
      drevo(x-1, y, z)
      drevo(x+1, y, z)
   z = z + 3  #naslednje drevo odmaknjeno za 3 kocke
   i = i + 1  #eno drevo vec v vrsti
```

stopi_na_kocko

```
# stevilo_kock, material

sprednja_kocka = pred_mano(stevilo_kock)
kocka[pozicija.x, pozicija.y, pozicija.z] = material
premakni_se(pozicija.x, pozicija.y + 1, pozicija.z)
```

skoplji_tunel

```
x, y, z = pred_mano(1)
while ali_lahko_kopljem(x, y, z) or ali_lahko_kopljem(x, y + 1, z):
   kocka[x, y, z] = Zrak
   kocka[x, y + 1, z] = Zrak
   premakni_se(x, y, z)
        x, y, z = pred_mano(1)
```

tunel_z_baklami

```
x, y, z = pred_mano(1)
while ali_lahko_kopljem(x, y, z) or ali_lahko_kopljem(x, y + 1, z):
   kocke[x:x, y:y + 1, z:z] = Zrak
   kocka[x, y + 1, z + 1] = Bakla
   kocka[x, y + 1, z - 1] = Bakla
        premakni_se(x, y, z)
   x, y, z = pred_mano(1)
```

zgradi_most_v_oblakih

```
''' namig: postavi se nekam na hrib, pred kakšen prepad '''
x, y, z = pred_mano(1)
while ali_lahko_gradim_most(x, y, z):
   kocka[x, y - 1, z] = Lesene_deske
   premakni_se(x, y, z)
   x, y, z = pred_mano(1)
```

zagradi_se

```
# material

x,y,z = pozicija
kocke[x + 1:x + 1, y:y + 2, z:z] = material
kocke[x - 1:x - 1, y:y + 2, z:z] = material
kocke[x - 1:x + 1, y:y + 2, z + 1:z + 1] = material
kocke[x - 1:x + 1, y:y + 2, z -1:z - 1] = material
kocke[x:x, y + 2:y + 2, z:z] = material
```

Literatura

[1] A. Isaković, Minecraft: Education Edition in Code Builder — vodič za učitelje, 2017, https://github.com/ialja/minecraftedu-navodila (pridobljeno: 18.9.2018).

[2] A. Ronacher, Dealing with JavaScript's Automatic Semicolon Insertion, Lucumr, 2011, http://lucumr.pocoo.org/2011/2/6/automatic-semicolon-insertion/ (pridobljeno: 20.6.2018).

[3] Amazon, NodeMCU, https://www.amazon.com/s/ref=nb_sb_noss_2?url=search-alias%3Delectronics&field-keywords=nodeMCU (pridobljeno: 23.9.2018).

[4] AppInventor, Dostopno na: http://www.appinventor.org/ (pridobljeno: 9.8.2018).

[5] Arduino, https://www.arduino.cc/ (pridobljeno: 9.8.2018).

[6] Arduino, Introduction, https://www.arduino.cc/en/Guide/Introduction (pridobljeno: 6.8.2018).

[7] Bukkit, Why Plugin Metrics?, https://dev.bukkit.org/projects/communitybridge-fm/pages/why-plugin-metrics (pridobljeno: 6.7.2018).

[8] C. Mason, Computer skills a must in today's workforce, 2017, https://wcfcourier.com/computer-skills-a-must-in-today-s-workforce/article_

df9f77ca-e91c-5bc8-a7a3-f4d319c8df74.html (pridobljeno: 22.9.2018).

[9] CS Unplugged, https://csunplugged.org/ (pridobljeno: 9.8.2018).

[10] D. Prevc, diplomsko delo: Učenje programiranja v Pythonu v osnovni šoli, Pedagoška fakulteta, 2015, https://repozitorij.uni-lj.si/IzpisGradiva.php?id=72693&lang=slv&prip=dkum:8774761:d4 (pridobljeno: 9.8.2018).

[11] D. Albright, Arduino vs Raspberry Pi: A Detailed Comparison, 2016, https://beebom.com/arduino-vs-raspberry-pi (pridobljeno: 6.9.2018).

[12] Fakulteta za računalništvo in informatiko, Poletna šola FRI 2018, https://fri.uni-lj.si/sl/dogodek/poletna-sola-fri-2018 (pridobljeno: 18.9.2018).

[13] Google For Education, Blockly, https://developers.google.com/blockly/ (pridobljeno: 6.7.2018).

[14] Islovar, Socket, http://islovar.org/islovar (pridobljeno: 9.8.2018).

[15] J. Lazar, Arduino and LEGO Projects, Apress, 2013.

[16] Kickstarter, Robot Turtles: The Board Game for Little Programmers, Dostopno na: https://www.kickstarter.com/projects/danshapiro/robot-turtles-the-board-game-for-little-programmer (pridobljeno: 18.6.2018).

[17] LittleBits, https://www.littlebits.com/ (pridobljeno: 6.8.2018).

[18] M. Ropret, diplomsko delo: Učenje programiranja z okoljem Scratch, 2014, Dostopno na: https://repozitorij.uni-lj.si/IzpisGradiva.php?id=96269&lang=slv&prip=dkum:9161472:d1 (pridobljeno: 20.6.2018).

[19] M. Zaveršnik, JavaScript, http://zaversnik.fmf.uni-lj.si/gradiva/javascript/ (pridobljeno: 23.9.2018).

[20] Minecraft Crafting, ACTdesign, https://www.minecraft-crafting.net/ (pridobljeno: 9.8.2018).

[21] Minecraft gamepedia, Abouth Minecraft, https://minecraft.gamepedia.com/Minecraft_Wiki (pridobljeno: 20.6.2018).

[22] Minecraft gamepedia, Classic, https://minecraft.gamepedia.com/Classic (pridobljeno: 20.6.2018).

[23] N. Tosuna in M. F. Baris, Procedia - Social and Behavioral Sciences 28, 2011, str. 530 – 535.

[24] NodeMCU, Dostopno na: http://nodemcu.com/ (pridobljeno: 9.8.2018).

[25] Minecraft gamepedia, Custom servers, https://minecraft.gamepedia.com/Custom_servers (pridobljeno: 20.6.2018).

[26] P. Blikstein, Seymour Papert's Legacy: Thinking About Learning, and Learning About Thinking, https://tltl.stanford.edu/content/seymour-papert-s-legacy-thinking-about-learning-and-learn\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{z\global\mathchardef\accent@spacefactor\spacefactor}\accent20z\egroup\spacefactor\accent@spacefactor-ing-about-thinking (pridobljeno: 22.9.2018).

[27] P. Gabrijelčič, Prvi koraki v programiranje, 31.1.2012, https://www.monitor.si/clanek/prvi-koraki-v-programiranje2/124892/ (pridobljeno: 14.7.2018)

[28] P. Krebelj, HTML in CSS za začetnike, Založba Atelje Doria, 2015, str. 17–23.

[29] P. Krebelj, Spoznavamo programski jezik Python, Založba Atelje Doria, 2016.

[30] P. Mihalič, diplomsko delo: Učenje programiranja z okoljem Scratch, Fakulteta za računalništvo in informatiko, 2014, https://repozitorij.uni-lj.si/IzpisGradiva.php?id=68775 (pridobljeno: 20.6.2018).

[31] Ministrstvo za izobraževanje, znanost in šport, Zavod RS za šolstvo, UČNI načrt. Program osnovna šola. Računalništvo: neobvezni izbirni predmet, 2013, http://www.mizs.gov.si/fileadmin/mizs.gov.si/pageuploads/podrocje/os/devetletka/program_razsirjeni/Racunalnistvo_izbirni_neobvezni.pdf (pridobljeno: 22.9.2018).

[32] Raspberry Pi, https://www.raspberrypi.org/ (pridobljeno: 9.8.2018).

[33] Raspberry Pi, Downloads, https://www.raspberrypi.org/downloads/ (pridobljeno: 9.8.2018).

[34] Robot turtles, http://www.robotturtles.com/ (pridobljeno: 9.8.2018).

[35] S. Cass and P. BulusuIEEE, Interactive: The Top Programming Languages 2018, Spectrum, 2018, https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018 (pridobljeno: 6.8.2018).

[36] S. Lajovic, Python za otroke, Založba Pasadena, maj 2018.

[37] Scratch, https://scratch.mit.edu/ (pridobljeno: 9.8.2018).

[38] ScratchJr, http://www.scratchjr.org/ (pridobljeno: 9.8.2018).

[39] ScratchWiki, Scratch, https://en.scratch-wiki.info/wiki/Scratch (pridobljeno: 20.6.2018).

[40] SpigotMC, About Spigot, 2017, https://www.spigotmc.org/wiki/about-spigot/ (pridobljeno: 6.7.2018).

[41] T. C. Bell, J. Alexander, I. Freeman in M. Grimley, Computer Science Unplugged: school students doing real computing without computers, 2009

[42] Tiger soft solutions, Minecraft Goljufije, Goljufija In Koraki, https://sl.tigersoftwaresolutions.com/gaming/minecraft-cheats-cheat-codes-and-walkthroughs-10853, (pridobljeno: 18.9.2018).

[43] Vidra, http://vidra.si/ (pridobljeno: 9.8.2018).

[44] Vidra, Risanje po navodilih, http://vidra.si/risanje-po-navodilih/ (pridobljeno: 18.6.2018).

[45] Wikipedia, Application programming interface, https://en.wikipedia.org/wiki/Application_programming_interface (pridobljeno: 6.7.2018).

[46] Wikipedia, Blocky, https://en.wikipedia.org/wiki/Blockly (pridobljeno: 6.7.2018).

[47] Wikipedia, Logo, https://en.wikipedia.org/wiki/Logo_(programming_language) (pridobljeno: 9.8.2018).

[48] Wikipedia, Python, https://sl.wikipedia.org/wiki/Python_(programski_jezik) (pridobljeno: 6.8.2018).

[49] Z. Romano, There‘s a new Arduino At Heart: littleBits Arduino Module!, Arduino, 2014, https://blog.arduino.cc/2014/05/15/littlebits-arduino-module/ (pridobljeno: 6.7.2018).

[50] Zhuowei, Github, RaspberryJuice, https://github.com/zhuowei/RaspberryJuice (pridobljeno: 6.7.2018).

[51] H. Tsukamoto, Y. Takemura, H. Nagumo, I. Ikeda, A. Monden in K. Matsumoto, Programming education for primary school children using a textual programming language, IEEE Frontiers in Education Conference (FIE), 2015.