NASTAVNI PROGRAM ZA NASTAVU PROGRAMIRANJA PRATEĆI OSNOVNE OOP PRINCIPE

Co-funded by the
Erasmus+ Programme
of the European Union

Projekat	Object Oriented Programming for Fun
Akronim projekta	OOP4FUN
Broj ugovora	2021-1-SK01-KA220-SCH-00027903
Koordinator projekta	Žilinska univerzita v Žiline (Slovakia)
Partneri na projektu	Sveučilište u Zagrebu (Croatia) Srednja škola Ivanec (Croatia) Univerzita Pardubice (Czech Republic) Gymnazium Pardubice (Czech Republic) Obchodna akademia Povazska Bystrica (Slovakia) Hochschule fuer Technik und Wirtschaft Dresden (Germany) Gymnasium Dresden-Plauen (Germany) Univerzitet u Beogradu (Serbia) Gimnazija Ivanjica (Serbia)
Godina izdanja	2023

Sadržaj

Lista slika

Lista tabela

1.Uvod

1.1. Cilj predmeta

Cilj predmeta je da nauči studente da rešavaju programske zadatke koristeći osnove objektno orijentisanog programiranja (OOP) prateći osnovnu OOP paradigmu. Učenici će naučiti da podele date zadatke na saradničke objekte; da utvrđuju svoje nadležnosti; i da implementira projektovani model. Kurs ne zahteva prethodne veštine programiranja. Predaje se u programskom jeziku Java. Kurs objašnjava osnovne OOP koncepte (kao što su inkapsulacija, nasleđivanje ili asocijacija) o stvaranju kompjuterskih igara, gde se ovi koncepti jednostavno i intuitivno koriste. Proces kreiranja kompjuterske igre zasniva se na timskom radu i praktično koristi znanja i veštine iz drugih oblasti informatike i sa njom srodnih predmeta (rad sa multimedijalnim i kancelarijskim softverom). Dizajn svake kompjuterske igrice je dovoljno otvoren da učenici mogu individualno i kreativno proširiti igru. Štaviše, dizajn vodi ka pravilnom korišćenju stečenog znanja

1.2. Karakteristike predmeta

Predmet je usmeren na uvođenje inovativnog pristupa u nastavi programiranja, zasnovanog na rešavanju zadataka korišćenjem paradigme objektno orijentisanog programiranja (OOP). OOP je danas dominantna paradigma za razvoj aplikacija. Stoga je ispravno da učenici poseduju znanja i veštine iz ove oblasti. Predmet predstavlja razvojno okruženje koje koristi različite oblike uređivanja izvornog koda (uređivanje zasnovano na okvirima korišćenjem pojednostavljene forme kao i pisanje stvarnog izvornog koda) što omogućava podučavanje učenika na različitim nivoima prethodnog tehničkog znanja i aktivnosti. Svojom jednostavnošću i jasnoćom ovaj alat podržava brzo i intuitivno razumevanje tema koje se predaju što pozitivno utiče na učenike i njihovu motivaciju.

1.3. Cilj predmeta

- Kroz programiranje interaktivnih igara u grafičkom okruženju, učenik će steći znanja i veštine, tako da će učenik moći da:
- identifikuje problem,
- identifikuje odgovarajuće objekte za rešavanje identifikovanog problema (dekompozicija objekata),
- dizajnirati klase objekata, kao i njihove atribute i metode,
  identifikuju i pravilno koriste odnose objekata (pridruživanje, nasleđivanje),
- dizajnirati algoritam za rešavanje problema i distribuirati ga među kooperativnim objektima,
- koristiti elemente izvornog koda (grananje, petlje) za implementaciju dizajniranog algoritma,
  efikasno koristiti sredstva za otklanjanje grešaka u izvornom kodu,
- kreirajte jednostavnu aplikaciju sa grafičkim interfejsom u Greenfoot okruženju..

1.4. Ishodi ucenja

Ishodi učenja predmeta sumirani su na sledeći način:

• razumevanje osnovnih principa objektno orijentisanog programiranja,

• razumevanje osnova algoritamizacije,

• razumevanje sintakse programskog jezika Java,

• analizira izvršavanje programa na osnovu izvornog koda,

• mogućnost kreiranja sopstvenih programa uz korišćenje OOP-a.

1.5. Materijalno-tehnički zahtevi

PC učionica koja sadrži poseban radni prostor za svakog učenika plus radni prostor za nastavnika. Radnim prostorom se smatra sto, stolica, personalni računar (PC). Svi radni prostori treba da budu povezani na LAN mrežu sa pristupom internetu (preporučeno).

PC treba da ispunjava sledeće minimalne zahteve:

• operativni sistem (Microsoft Vindovs 7 ili noviji, Linuk (Debian), Mac OS 10.10 ili noviji),

• kancelarijski softver sa uređivačem teksta, tabela i prezentacija (npr. Microsoft Office, Libre Office, Open Office),

• java SE Development Kit (JDK),

• Greenfoot okruženje (verzija 3.8 ili novija),

• jednostavan grafički softver,

• veb pregledač (npr. Edge, Google Chrome, Mozilla Firefok, Opera),

• relevantni softver za drugi hardver računara

2. Principi nastavnog plana i programa

Predloženi nastavni plan je osmišljen tako da se bavi problemima identifikovanim u PR1 i PR2 (pogledajte poglavlje „Usklađivanje rezultata sa rezultatima PR1" u izveštaju PR2). U sledećoj tabeli predstavljamo perspektivu razvoja kurikuluma, ishoda učenja, nastavnih materijala i nastavnih aktivnosti kako je predloženo u PR2. Prateći ove nalaze, uspeli smo da formulišemo principe nastavnog plana i programa.

PR2 rezultati	PR3 principi nastavnog programa
U srednjim školama OOP bi trebalo uvesti temae koje pokrivaju osnovne koncepte programiranja na početku, a uže teme vezane za OOP bi bile prikladnije u posebnim predmetima. Ključno je povezati i podsticati razmenu informacija između školskih i univerzitetskih nastavnika, uz uključivanje kreatora politike koji definišu nastavne planove i programe koji se odnose na veštine programiranja na svim nivoima obrazovanja. Iz perspektive kursa/instruktora i iz perspektive dizajna kursa, trebalo bi da se koriste sledeći inovativni oblici nastave/prenosa znanja: kombinovano učenje, učenje kroz rad, rešavanje problema, problem saradnje, timski rad, učenje zasnovano na problemu, aktivno učenje , praktično učenje . Pored toga, na predavanjima, seminarima i laboratorijskim vežbama treba primeniti različite oblike inovativnih pristupa. Igre isu se često koristile za motivisanje učenika da programiraju. Učenicima se dopala prilika da budu kreativni ili da se takmiče za znanje kada ih podržava odgovarajuća postavka. Iz rezultata pregleda literature predložena su tri različita tipa učenja kroz igru: učenje kroz igru, učenje kroz kreiranje igara, učenje korišćenjem alata vezanih za igru i učenje uz gamifikaciju.	Nastavni plan i program mora pravilno koristiti OOP od samog početka nakon pristupa prvom objektu. Odgovarajući nivo OOP-a za srednje škole je identifikovan i formulisan kao losnove OOP. Osnove OOP može se koristiti sa prednostima prilikom kreiranja igara, jer je lako identifikovati objekte igre, kao i kompetencije i svojstva objekata. Za motivisanje učenika korišćenjem igara važno je zainteresovati učenike za igre. Trebalo bi kreirati nekoliko projekata zasnovanih na igrici sa različitim mehanikama da bi se ispunio ovaj cilj. Štaviše, nekoliko igara koje će biti pripremljene omogućiće: • kreirati različite nastavne materijale za različite uslove nastave (online/onsite, intenzivni/celogodišnji kurs, početnici/napredni studenti). Ovo je ključna osobina za predstojeće rezultate projekta, • napravite jednu igru uz prisustvo nastavnika i druge igre kao domaće zadatke (nastavnik će moći da vidi da li učenici mogu primeniti znanje u različitim kontekstima), • napravite igru prema datim uputstvima uz minimalnu interakciju nastavnika, tako da će učenici razumeti važnost dobro napisanog tehničkog opisa i primeniće naučene veštine da kreiraju igru kako je navedeno, • predstavite novi osnovni OOP koncept dok igra uvodi novu tehniku. Ovo će omogućiti da se zaustavi razvoj projekta ako nastavnik odlučuje da pokrije podskup osnova OOP-a zbog specifičnosti nacije.
Za učenje i podučavanje OOP koncepata, učenje kroz kreiranje igara pokazalo je značajne efekte za poboljšanje veština učenika u rešavanju	

problema i njihovo angažovanje u zabavnom i zabavnom okruženju.	
Kao što je istaknuto nekoliko rezultata PR2, glavni cilj bi trebalo da bude inkorporiranje zadataka učenja i podučavanja u stimulativne i zabavne aktivnosti koje će imati pozitivan uticaj na veću posećenost i stopu završetka. Ovo bi povećalo interesovanje srednjoškolaca za programiranje uopšte i na kraju dovelo do boljeg razumevanja programiranja i OOP koncepata. U tom slučaju studenti ne bi bili „izgubljeni“ kada se suoče sa univerzitetskim nastavnim planovima i programima.	Projekti će se graditi po principu učenja kroz rad. Trudimo se da minimiziramo teorijsko objašnjenje i podržimo istražne, prakse i proizvodne aspekte učenja. Rad u grupama ohrabruje učenike koji se u početku mogu mučiti. Ako se projekat razvija u grupi, mogu se koristiti agilne metode razvoja kao i nastave. Pre realizacije, nastavnik može odlučiti da koristi analitičku i dizajnersku fazu projekta. Ovo će omogućiti korišćenje znanja i veština studenata stečenih na prethodnim predavanjima/kursevima, kao i angažovanje studenata u razvoju projekata. Od učenika se može tražiti da: • radite sa izvorima informacija kako biste pronašli odgovarajuću igru, • formulisati pravila igre i/ili zahteve za njihovu primenu (bilo u usmenoj ili pismenoj formi), • pripremite multimediju (slike/zvukove).
Opšti cilj PR2 bio je pronalaženje odgovarajućih i inovativnih ideja i pristupa učenja i podučavanja koji bi rešili ova pitanja. Kao što je pomenuto u prethodnim poglavljima, postoji nekoliko identifikovanih dobrih praksi koje bi se mogle koristiti za poboljšanje postizanja ishoda učenja tokom srednjoškolskog obrazovanja. Međutim, takođe treba napomenuti da redizajn kurikuluma treba da rezultira uvođenjem OOP tema i postavljanjem ciljeva u postizanju ishoda učenja vezanih za OOP. Takođe je važno odabrati odgovarajuću vrstu ocenjivanja: (onlajn) upitnici su jedini prihvaćeni metod za procenu zadovoljstva, korisnosti, interesovanja, angažovanja i pojednostavljenja programskih i OOP koncepata koji će biti definisani u srednjoj školi i ne na univerzitetskom nivou.	Fokusirajući se na princip učenja kroz rad, pokušali smo da svedemo na minimum aktivnosti sticanja i da pojačamo istraživački deo. Projekti će biti izgrađeni na način koji omogućava jednostavno proširenje, tako da će motivisani studenti biti sposobni da sami rade na projektima. Da bismo potvrdili predloženi nastavni plan i program, pripremićemo dizajn učenja za svaki projekat. Zatim ćemo uporediti analitičke rezultate novih projekata koji koriste osnove OOP sa analitičkim rezultatom dizajna učenja napravljenog za projekat koji je razvijen u prošlosti i koji je već primenjen u praksi u Slovačkoj (među ostalim školama u zemlji, koristi ga partner na projektu Obchodna akademia Povazska Bistrica) i Češka (koristi ga partner na projektu Gimnazija Pardubice). Pozitivne povratne informacije za ovaj validirani projekat objavljene su u PR2, stoga pretpostavljamo da ćemo, slijedeći iste najbolje principe i prakse, prenijeti pozitivno prihvatanje predloženog nastavnog plana i programa.
Korišćenjem timskog rada u OOP zadacima, studenti bi imali priliku da podele svoje znanje i	Kada smo dizajnirali ovde predložene projekte zasnovane na igricama, imali smo na umu korišćenje različitih tehnika, kao što je

prenesu implementaciju osnovnih programskih koncepata na druge učenike (peer-to-peer learning).	EduScrum. Iz tog razloga, mi isporučujemo svaki projekat u obliku GIT repozitorija i pružamo odgovarajuće znanje nastavnicima. Iako nije ograničeno na korišćenje ovog pristupa i nastavnici i dalje mogu da koriste tradicionalni pristup, korišćenje agilnih tehnika će dovesti do podučavanja: • Osnove sistema za verzionisanje - predlažemo GIT kao najkorišćeniji sistem za verzionisanje, korišćenje sistema za verzionisanje olakšava o delite izvorne kodove i o razviti nove karakteristike igara, bez uticaja na tok projekta (npr. u posebnoj grani), što će motivisati ambiciozne učenike. • Timski rad – svaki učenik će biti odgovoran za specifičan aspekt igre koji vodi ka neophodnosti efikasne komunikacije između članova tima. • Upravljanje vremenom – pojedini delovi projekata će morati da budu isporučeni na vreme kako bi se spojili i nastavili sa drugim poslom, međutim, pravilna upotreba sistema za verzionisanje i OOP otvaraju mnoge mogućnosti za suočavanje sa vremenskim borbama što može dovesti do pozitivne motivacije učenika čak i u teškim vremenima. Sa nastavnicima ćemo realizovati multiplikativne događaje koji pokrivaju kako uvod u GIT, tako i korišćenje principa agilnog razvoja softvera u nastavi. Učenici će formirati projektne timove, sa specifičnim ulogama, deliće ideje o stand-up-u, zadavati i zadavati sebi ciljeve, realizovaće sprintove, kreirati dokumentaciju i druge artefakte i prezentovati svoja rešenja.
Rezultati ovog projekta će dati set materijala koji će dati priliku visoko motivisanim učenicima sa solidnim prethodnim znanjem da to znanje uvećaju kroz različite aktivnosti i uloge. Takvi učenici bi mogli značajno da poboljšaju ukupna postignuća cele grupe ako im se pruži prilika da podele znanje ili da vode timove.	
Treba podsticati upotrebu više različitih alata i programskih jezika u ranijim godinama studija. Koncepti programiranja mogu se pojednostaviti korišćenjem alata za vizuelizaciju. Iako su neke zemlje već uvele upotrebu nekih alata kao što su Logo ili Scratch, oni su zanimljivi osnovnim školama, ali ne i srednjim školama. Stoga bi trebalo koristiti naprednije alate koji su dizajnirani da podrže OOP. Prepoznali smo da se Alis i Greenfoot ističu među ostalim alatima.	Koristimo Greenfoot okruženje koje koristi Java programski jezik. Java je trenutno veoma popularna iu praksi se široko koristi programski jezik. Štaviše, Greenfoot predstavlja uređivač izvornog koda zasnovan na okvirima koristeći Stride jezik. Ovo otvara mogućnosti za nastavnike koji će želeti da u ovom nastavnom programu koriste tehnike predstavljene sa učenicima mlađeg uzrasta. Greenfoot je veoma vizuelan i od početka omogućava kreiranje vizuelizovanog objekta, koji je „živ” i sa kojim se može komunicirati. Dakle, teorijski uvod je minimiziran, a studenti će početi da rade od samog početka.

Kako su učenici izjavili, za nastavu programiranja važno je da nastavnici koriste nove i najnovije nastavne materijale i da koriste kreativne nastavne metode. Takođe, nastavnikova dostupnost i fleksibilnost za rad sa učenicima van učionice neophodna je za motivisanje učenika i stvaranje većeg interesovanja za predmet.	Koristeći predstavljene principe, kreiramo moderan nastavni plan i program za nastavnike koji će pokrivati lake OOP teme i zasnovan je na projektnom radu i istinski koristi pristup na prvom mestu. Predlažemo nekoliko projekata zasnovanih na igricama, od kojih se svaki isporučuje u obliku GIT repozitorija. Za svaki projekat kreiran je dizajn učenja koji je omogućio validaciju ovih projekata sa već u praksi korišćenim i pozitivno prihvaćenim pristupom.

Sadržaj i obim obrazovnog programa se razlikuju u odnosu na projekat igre koji se razvija tokom nastave. Za detaljnu analizu pogledajte odgovarajući dizajn učenja i priložene datoteke analize.

3.Projekti

Što se tiče principa nastavnog plana i programa, kreirana su dva projekta igre koji pravilno koriste principe prvi objekt i učenje kroz rad. Takođe, obrađivali smo projekat Bomberman, koji je bio okosnica projekta nacionalnog projekta IT akademije, koji se koristi u praksi u Slovačkoj. Ovaj projekat koristimo za validaciju predloženih projekata. Organizacija svih poglavlja projekta je sledeća.

• Opis projekta – osnovni opis igre sa snimkom ekrana gotove aplikacije i sažetim pravilima igre. Opis projekta takođe predstavlja vezu sa OOP temama.

• Veza sa izvornim kodovima – izvorni kodovi su organizovani u obliku GIT [1] spremišta. Korišćenje pravilno vođenog spremišta upoznaje nastavnika sa savremenim pristupom upravljanja izvornim kodom. Koristili smo GIT kao sistem za upravljanje verzijama koji je među najpopularnijim poslednjih godina [2]. GIT nam takođe omogućava da koristimo riznice zasnovane na oblaku, kao što su GitHub [3] ili GitLab [4], koje su besplatne i nude korišćenje mnogih alata za poboljšanje timske saradnje. Svako spremište je organizovano na sledeći način:

• Ogranak po temi – zadaci svake teme iz nastavnog plana se razvijaju u namenskoj grani. Glavna grana sadrži samo urezivanje inicijalizacije i spajanja grana teme.

• Urezivanje po zadatku – svaki zadatak koji je orijentisan na proizvodnju/modifikaciju izvornog koda je u obliku urezivanja. Opis urezivanja odgovara broju odgovarajućeg zadatka.

• Veza sa dizajnom učenja – nastavni plan i program je izgrađen u obliku dizajna učenja. Dizajn učenja nam omogućava da definišemo ishode učenja (koji pokrivaju neophodne kompetencije identifikovane u PR1 i PR2) i da ih povežemo sa temama (pogledajte vezu sa granama odgovarajućeg GIT repozitorija). Teme su organizovane u jedinice koje se sastoje od TLA (pogledajte vezu sa urezivanja u odgovarajućem GIT spremištu). Korišćenje dizajna učenja omogućava analizu raspodele vremena što je direktno povezano sa validacijom deklarisanog principa učenja kroz rad. Budući da je dizajn učenja obrađen i za već uspostavljeni i u pedagoškoj praksi korišćeni projekat (Bomberman), to omogućava identifikaciju potencijalnih problema u dizajnu. Da bi se mogla izvršiti validacija, projekti su organizovani u teme na isti način.

• Obrađene teme osnova OOP-a.

• Sadržaj i obim obrazovnog programa – pregled opterećenja učenika u određenom tipu učenja, kao i pregled doprinosa tema pojedinačnim ishodima učenja.

• Spisak tema. Svaka tema sadrži:

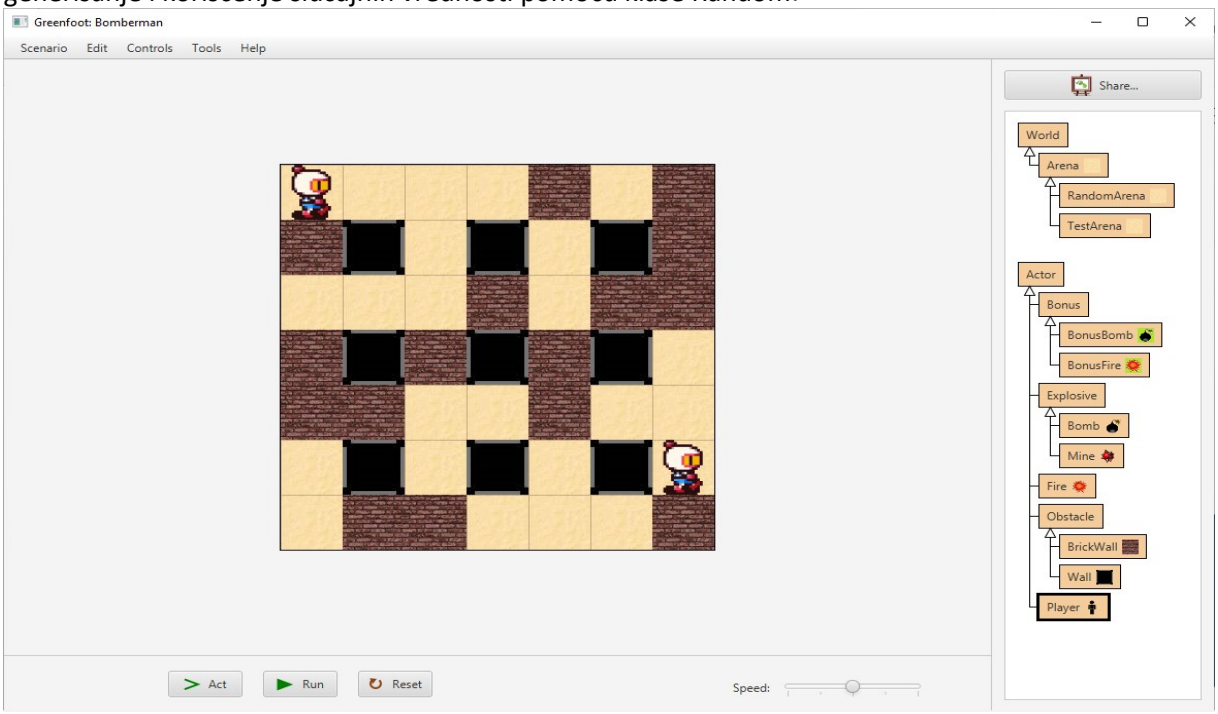
• kratak opis,

• poređenje dizajna učenja,

• spisak zadataka.

3.1.Bomberman

Bomberman je prilično poznata igra za više igrača. Igra se odvija u areni koja sadrži igrače, kao i neke fiksne prepreke. Igrač može postaviti bombe koje eksplodiraju nakon određenog vremena. Cilj igre je eliminisanje protivnika upotrebom bombi. Neke od prepreka u areni mogu se uništiti bombama. Nakon uništavanja prepreke, u igri se može pojaviti nasumični bonus, koji na primer povećava brzinu ili snagu bombi igrača. Igra se završava ako je u igri ostao samo jedan igrač, u kom slučaju taj igrač pobeđuje, ili ako nema preostalih igrača u igri, u kom slučaju se igra završava nerešeno.
Projekat Bomberman pokriva većinu tema osnova OOP-a. Fokusira se na glavne aspekte OOP-a koji su sažeti u temama u nastavku. Štaviše, uvodi neke teme koje sežu dalje od osnova OOP-a, kao što je generisanje i korišćenje slučajnih vrednosti pomoću klase Random.



Slika 1: Greenfoot okruženje sa konačnim stanjem projekta Bomberman

Izvorni kodovi su dostupni na:
https://gitlab.kicon.fri.uniza.sk/oop4fun/project-bomberman

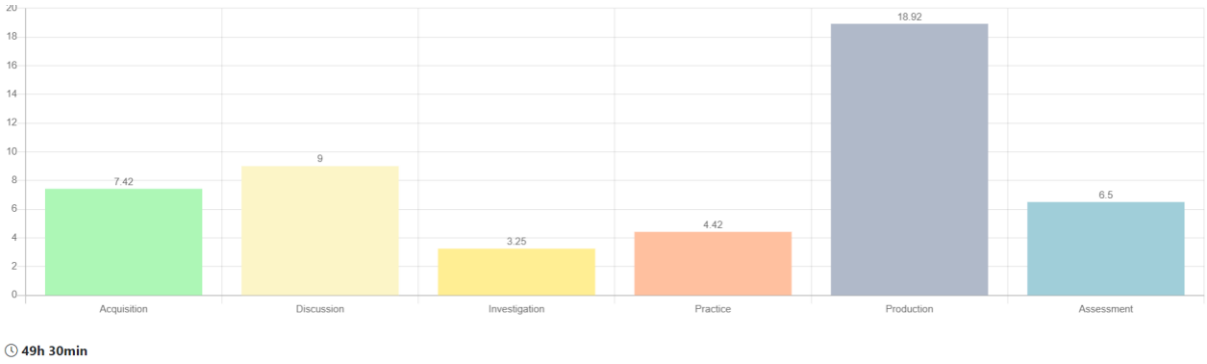
Bomberman projekat je specifičan po tome što sadrži obaveze koje uključuju rad koji treba da se obavi između dva uzastopna zadatka. Takva urezivanja se opisuju pomoću poruka oblika „Šifra Ks.I“, gde je Ks broj teme, a I broj srednjeg zadatka unutar date teme.

Dizajn učenja je dostupan na:

https://learning-design.eu/en/preview/70bcf65d805b0603f6c1aeab/details

3.1.1. Sadržaj i obim obrazovnog programa

Ukupno opterećenje učenika je 49 sati i 30 minuta i raspoređuje se na sledeći način:



Slika 2: Radno opterećenje učenika kada se koristi projekat Bomberman

Konstruktivno usklađivanje je sažeto u tabeli ispod:

Tabela 1: Konstruktivna usklađenost projekta Bomberman

Topic	Assessment		Understanding the basic principles of object-orien... (25)	The ability of creating own programs with the use ... (20)	Understanding the syntax of the Java programming l... (10)	Understanding the basics of algorithmisation (25)	Analysing program execution based on the source co... (20)
	Formative	Summative					
Greenfoot environment	0	0		80%	20%		
Class definition	0	35	60%	20%	20%		
Algorithm	0	20		10%	10%	60%	20%
Branching	0	20		10%	10%	70%	10%
Variables and expressions	0	5		10%	10%	70%	10%
Association	0	10	60%	10%	10%	10%	10%
Inheritance	0	0	50%	30%	10%		10%
Loops	0	40		40%	10%	40%	10%
Lists	0	0		50%	10%	30%	10%
Encapsulation	0	15	50%	30%	10%		10%
Polymorphism	0	15	50%	20%	10%	10%	10%
Random numbers	0	20		30%	10%	50%	10%
Total	**0**	**180**	**270%**	**340%**	**140%**	**340%**	**110%**
	180						

Za detaljan plan pogledajte prilog 5.1.

3.1.2.Topics

Projekat Bomberman je podeljen na deset tema:

Pokrivene teme svetlosnog OOP-a su:

• klase, objekti, instance
• metode, prosleđivanje argumenata metodama
• konstruktori
• atributi
• inkapsulacija
• nasleđe
• apstraktni časovi
•interfejs
• životni ciklus objekta

1 Uvod u Greenfoot okruženje

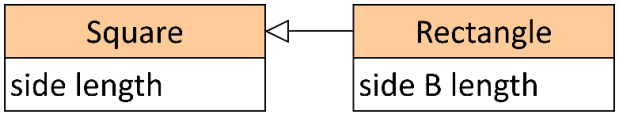
Tema je posvećena osnovnoj postavci projekta. Učenici će naučiti kako da podese dimenzije i izgled okruženja, kreiraju klasu (kao podklasu klase Actor), kreiraju njenu instancu, pošalju joj poruku i posmatraju njeno unutrašnje stanje.

1.1 Identifikujte objekte

Identifikujte objekte u svom okruženju i navedite njihova svojstva i radnje koje mogu da izvrše. Možete li da identifikujete objekte koji nemaju svojstva? Možete li da identifikujete objekte koji ne mogu ništa? Možete li da identifikujete nematerijalne predmete (one koje fizički ne možemo da dodirnemo)?

1.2 Potvrdite hijerarhiju objekata

Sledeća slika prikazuje hijerarhiju klasa Skuare i Rectangle. Da li je ovo dobra hijerarhija?



1.3 Napravite jednostavnu hijerarhiju

Napravite hijerarhiju klasa od:
a) prevozno sredstvo,
b) životinje,
v) i računarskih delova.
Navedite svojstva koja svaka klasa definiše. Koje klase su apstraktne u dizajnu?

1.4 Napravite pločicu

U grafičkom uređivaču napravite pločicu koja će grafički predstavljati ćeliju sveta. Izaberite kvadratnu reprezentaciju, idealno 60k60 piksela. Uvezite ili sačuvajte sliku u folderu projekta u folderu slike. Postavite sliku kao sliku sveta. Imajte na umu da je klasa MiVorld dobila malu ikonu u dijagramu klasa koja predstavlja njen grafički oblik.

1.5 Kreirajte svet

Izmenite konstruktor klase MiVorld da biste kreirali svet ćelija veličine 25k15, pri čemu je svaka ćelija veličine 60 piksela. Kako bi sliku trebalo modifikovati da bi svet izgledao kao šahovska tabla (tj. sa naizmeničnim poljima različitih boja)?

1.6 Pregledajte stanje igrača

Zgrabite kreiranu instancu klase Plaier pomoću miša i pomerite je na drugu poziciju u svetu. Gledajte uživo prikaz unutrašnjeg stanja – šta vidite? Napravite još jednu instancu klase Plaier i pogledajte njeno unutrašnje stanje. Opet, prevucite mišem jednu od dve instance – koje unutrašnje stanje se promenilo?

1.7 Interakcija sa igračima

Pozovite metode koje obezbeđuje Greenfoot okruženje preko različitih instanci klase Plaier. Posmatrajte kako se menja unutrašnje stanje instance.

2 Algoritam, kontrole aplikacije, kreiranje metoda

Ova tema pokriva stvaranje javnih metoda koje pokreću igrača u svetu. Takođe uvodi Greenfoot alate okruženja koji kontrolišu izvršenje scenarija.

2.1 Napišite jednostavan algoritam

Zapišite proceduru, kako se priprema kafa, kako se putuje u školu i kako se kuva ručak.

2.2 Napišite opštiji algoritam

Napravite opšti algoritam za pripremu toplog napitka. Razmislite o tome koji ulazi u takav algoritam treba da budu da bi bio opšti.

2.3 Pregledajte instancu klase

Istražite metode instance klase Plaier. Šta posmatrate?

2.4 Implementacija metoda

Dodajte izjavu u telo metode makeLongStep() tako da se instanca klase Plaier pomera za dve ćelije u trenutnom pravcu. Zatim kreirajte više instanci klase Plaier i pozovite ovaj metod na svakoj instanci. Da li je ponašanje očekivano?

2.5 Dodajte dokumentaciju

Dodajte komentar dokumentacije za metodu makeLongStep().

2.6 Dodajte još dokumentacije

Uredite komentar dokumentacije klase Plaier. Dodajte verziju klase i njenog autora.,

2.7 Pročitajte dokumentaciju

Istražite prozor sa dokumentacijom.

2.8 Dodajte akciju igrača

Izmenite telo act() metode klase Plaier da pozove metod makeLongStep().

2.9 Istražite kontrole aplikacija

Isprobajte dugmad koja kontroliše aplikaciju. Kreirajte više instanci klase Plaier. Pritisnite dugme Act – šta se dešava? Pritisnite dugme Run - šta se dešava? Nakon prvog pritiska na dugme Run pritisnite dugme Pauza, šta se dešava? Kakav efekat ima klizač za brzinu na poziv metode act() nakon što se pritisne dugme Run?

2.10 Dodajte radnju drugog igrača

Dodajte metod u klasu Plaier koji će prošetati instancu klase Plaier u kvadratu. Dokumentujte svoj metod. Koristite odgovarajuće metode iz osnovne klase Actor za pomeranje i rotiranje. Izmenite metod act() tako da instanca klase Actor hoda po kvadratu kada se pozove. Zatim potvrdite svoje rešenje pokretanjem aplikacije.

3. Grananje i kontrola igrača

Ova tema upoznaje učenike sa grananjem u obliku if-else iskaza i svitch iskaza. Grananje se koristi u otkrivanju ivica sveta i sudara sa zidovima – što su novi objekti dodati ovoj temi

.

3.1 Pomerite igrača

Izmenite kod metode act() u klasi Plaier tako da se igrač kreće samo kada se pritisne M taster (M kao potez). Neka kod bude odgovoran za okretanje igrača kada stigne na ivicu sveta, ali razmislite o njegovoj lokaciji. Kada se može izvesti okret igrača?

3.2 Posmatrajte stanje igrača

Napravite instancu klase Plaier i postavite je u centar table. Otvorite prozor sa unutrašnjim stanjem instance i postavite ga tako da bude vidljiv dok je aplikacija pokrenuta. Zatim pokrenite aplikaciju i posmatrajte kako se menjaju vrednosti atributa k i i u klasi Plaier. Kako se ove vrednosti menjaju dok se krećete gore, dole, levo i desno?

3.3 Dodajte detekciju ivica sveta

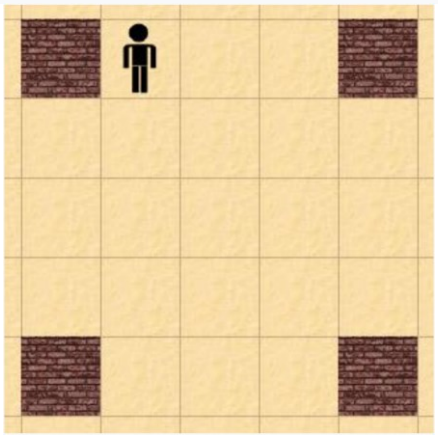
Dodajte kod u telo metode act() da biste pravilno rotirali plejer nakon što dođete do donje i leve ivice sveta.

3.4 Dodajte zidove

Napravite dve nove klase kao podklase klase Actor. Prva klasa će biti BrickVall klasa, a druga klasa će biti Vall. Pripremite odgovarajuće slike od 60k60 piksela u grafičkom uređivaču. Zatim dodelite ove slike novostvorenim klasama.

3.5 Posmatrajte kretanje igrača

Napravite četiri instance klase BrickVall i jednu instancu klase Plaier kao što je prikazano na slici ispod. Pogodite kako će se igrač kretati? Pokrenite aplikaciju. Da li se vaše predviđanje poklapa sa onim što posmatrate?



3.6 Dodajte detekciju sudara zida

Dodajte kod metodi act() klase Plaier da biste osigurali da se igrač okrene za 90° u smeru suprotnom od kazaljke na satu kada uđe u ćeliju koja sadrži instancu klase Vall

3.7 Predvidite kretanje igrača

Predvidite kako će se kretati instanca klase Plaier. Da li se rezultat slaže sa vašim predviđanjem?

3.8 Posmatrajte detekciju ivica

Postavite jednu instancu klase Plaier u uglove sveta. Predvidite kako će se ova instanca kretati kada se aplikacija pokrene. Da li se rezultat slaže sa vašim predviđanjem?

3.9 Završite rukovanje sudarom zida

Dovršite kaskadu uslova tako da dodirivanje instance klase BrickVall i klase Vall bude provereno samo ako igrač nije na ivici sveta. Prvo proverite instancu klase BrickVall.

3.10 Automatski pomerajte plejer

Kreirajte metod za automatsko premeštanje instance klase Plaier. Premestite sav kod iz metode act() u novu metodu. Identifikator metode može biti, na primer, moveAutomaticalli.

3.11 Pomerajte igrača pomoću strelica

Kreirajte metodu moveUsingArrovs() u klasi Plaier. Programirajte ovaj metod tako da se igrač kreće samo kada se pritisne strelica. Pomeraće se u pravcu pritisnute strelice. Vodite računa da kod bude efikasan. U metodi act() pozovite novi metod.

3.12 Prepare images

Pripremite četiri slike da se igrač pomera gore, dole, levo i desno. Dimenzije slika ne smeju biti veće od dimenzija ćelije, u našem slučaju 60k60 piksela. Postavite pripremljene slike u direktorijum slika.

3.13 Koristite slike

Kreirajte metod updateImage() koji menja sliku igrača u skladu sa njegovom trenutnom rotacijom. Dodajte poziv ovoj metodi u telo metode act().

3.14 Pokrenite aplikaciju

Pokrenite aplikaciju. Primetite da se slike prilagođavaju, ali se rotiraju u skladu sa načinom na koji se plejer rotira. Reši ovaj problem.

3.15 Koristite više igrača

Pokušajte da napravite više igrača i da ih kontrolišete pomoću tastature. Šta posmatrate?

4. Promenljive, izrazi i napredna kontrola igrača

Ova tema se bavi uvođenjem promenljivih u obliku atributa klase i parametara metoda. Učenici će koristiti atribute za kontrolu brzine igrača, kao i za postavljanje specifičnih tastera koji kontrolišu kretanje igrača. Ovo će omogućiti igri da ima više igrača koji kontrolišu različiti ključevi.

4.1 Primetiti razliku

Koja je razlika između sledećih algoritama?

1.
```
int a;
boolean c;
...
if(a > 0){c = true;}
if(a < 0){c = false;}
```

2.
```
int a;
boolean c;
...
if(a > 0){c = true;}
else {c = false;}
```

4.1 Koja je razlika između sledećih algoritama?

Napišite izraz koristeći Bulove i relacione operatore da izrazite da promenljiva tipa int a ima vrednost koja pripada sledećim intervalima : **<-10,10), (5,142), (-11,-3) OR (1,25>**.

4.3 Ocenite izraze

Izaberite vrednosti promenljivih k i i i dodajte ih izrazima. Koje će biti vrednosti promenljivih b1 i b2 u svakom slučaju (1 i 2)?

1. **int x, y;**
   ...
   boolean b1 = x > 0 && y == 1;
   boolean b2 = x <= 0 || y <= 0;

2. **int x, y;**
   ...
   boolean b1 = (x > 0) && (y == 1);
   boolean b2 = (x <= 0) || (y <= 0);

4.4 Testirajte konstruktor

Testirajte konstruktor i modifikovani metod za kontrolu kretanja igrača umetanjem dve instance klase Plaier u svet. Za svaku instancu u dijalogu, postavite različite tastere za kontrolu njenog kretanja. Testirajte da li se umetnuti plejeri mogu kontrolisati nezavisno y.

4.5 Preimenujte klasu

Preimenujte klasu MiVorld u klasu Arena.

4.6. Dodajte još jednog igrača

Dodajte još jednog igrača u svet, na primer pomoću atributa reference plaier2, koji će se kontrolisati pomoću tastera v – gore, s – dole, d – desno i a – levo. Postavite igrača na koordinate [24,14]. Kada dodate, testirajte kontrole.

4.7. Reference attributes

Šta bi se desilo ako bismo dodelili ovaj.plaier1 = this.plaier2; nakon kreiranja oba igrača? Da li bi to bio problem?

4.8 Proširite klasu igrača

Proširite klasu Plaier sa dodatnim atributom tipa int koji predstavlja veličinu koraka igrača.

4.9 Integrišite veličinu koraka

Izmenite metod moveUsingKeis() tako da poštuje povećanu veličinu koraka. Kreirajte novu instancu klase Plaier i testirajte funkcionalnost programa.

4.10 Neka se igrači kreću različitim brzinama

Vaš zadatak je da osigurate da se igrač pomera jednu po jednu ćeliju, ali različitim brzinama. Svaki igrač može imati različitu brzinu. Kao nagoveštaj, možete programirati različite brzine kretanja tako što, na primer, ne pomerate plejera svaki put kada se detektuje pritisak na taster (mi detekciju vršimo u metodi act(), tako da ako držite taster, otkrićete njegovu pritisnite svaki put kada se izvrši), ali samo svaki N-ti put kada se izvrši. Što je N veće, to će igračeva brzina biti manja. Kako se upisuje N? Gde ga čuvate? Kako ćete znati koliko je pritisaka na tastere prošlo? (Savet: potreban je i brojač).

5 Predmetna i klasna saradnja

Glavni fokus ove teme je saradnja objekata. U ovoj temi, učenici će projektu dodati interakciju između objekata u areni – na primer, pazeći da igrač ne može da prođe kroz zidove. Štaviše, u ovoj temi učenici će projektu dodati i objekat bombe – jedan od glavnih delova igre Bomberman.

5.1 *Dodajte kod za vertikalno kretanje*

Analogno tome, dodajte kod za pravce gore i dole na isti način (tako ćete promeniti vrednost lokalne promenljive i).

5.2 Proverite mogućnost kretanja

Izmenite metod koji obezbeđuje kretanje igrača (moveUsingArrovs) da biste proverili mogućnost ulaska u ciljnu ćeliju pre promene položaja igrača.

5.3 Razmotrite zidove od cigle

Izmenite metod canEnter() tako da igrač takođe reaguje na zidove od cigle i ne može da prođe kroz njih.

5.4 Dodajte bombu

Kreirajte novu klasu bombe i dizajnirajte njene atribute da predstavljaju snagu eksplozije. Kreirajte parametarski konstruktor i inicijalizujte atribute objekta.

5.5 Proverite mogućnost postavljanja bombe

Dodajte metodu canPlantBomb() u klasu Plaier sa povratnom vrednošću tipa boolean, koja vraća zastavicu koja govori da li je moguće postaviti bombu na ćeliju u kojoj igrač trenutno stoji. Bomba se može postaviti kada se pritisne odgovarajući taster i na ćeliji nema druge bombe.

5.6 Dodajte zvučne efekte

Produžite igru tako da eksploziju bombe prati zvučni efekat. Zvuk se može snimiti ili preuzeti sa interneta. Pronađite komandu za reprodukciju zvuka u dokumentaciji klase Greenfoot.

6 Nasleđivanje i for petlja

Ova tema se bavi uvođenjem nasleđa. Učenici kreiraju superrazred za časove Vall i BrickVall. Zatim će napraviti probnu arenu kao podklasu klase Arena. Konačno, ovaj odeljak se fokusira na petlje sa fiksnim brojem

6.1 Dodajte klasu pretka

Napravite klasnu prepreku. Koja klasa je nadklasa klase Prepreke? Uredite zaglavlja klasa BrickVall i Vall tako da budu podklase klase Obstacle.

6.2 Pojednostavite test zauzetosti

Nakon dodavanja superklase Prepreke, lakše je testirati da li igrač može da uđe u datu ćeliju. U svojoj metodi getObjectsAt(), svet zahteva kao treći parametar klasu koju treba tražiti u datoj ćeliji. Pošto su i BrickVall i Vall prepreka, moguće ih je tretirati ujednačeno. Izmenite metod canEnter() u klasi Plaier da koristi samo jednu listu prepreka.

6.3 zmenite klasu arene

Izmenite klasu Arena tako da njen konstruktor ima dva parametra koja predstavljaju širinu i visinu. Izmenite poziv konstruktoru superklase u klasi Arena da uzme ove parametre. Imajte na umu da Greenfoot ne može automatski da napravi Arenu, pošto su joj potrebni parametri za konstruktor. Uklonite iz ovog konstruktora kod koji je odgovoran za kreiranje i postavljanje igrača u arenu, to će učiniti podklase. Takođe možete ukloniti deklaraciju atributa tipa Plaier iz klase Arena.

6.4 Popravite klasu TestArena

Izmenite konstruktor klase TestArena da biste napravili praznu arenu veličine 7k7 ćelija. Da biste proverili, kreirajte instancu klase TestArena – iz kontekstnog menija klase TestArena izaberite stavku nev TestArena().

6.5 Dodajte upit za dimenziju

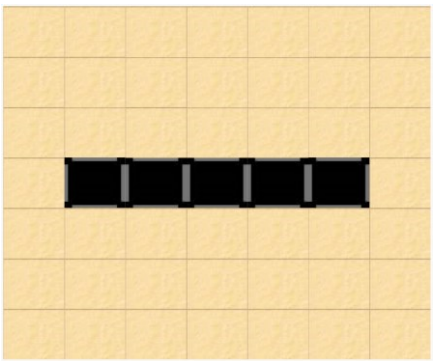
Dodajte metod shovDimensions() klasi TestArena koja štampa dimenzije arene na ekranu.

6.6 Dodajte još jedan upit za dimenziju

Dodajte metod shovDimensions() u klasu Plaier koja će prikazati dimenzije arene ako se nalazi u test areni. Ako jeste, koristite metod shovDimensions() klase TestArena.

6. 7 Dodajte zidove u test arenu

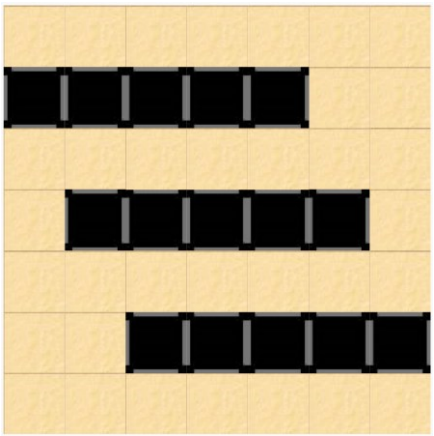
Izmenite konstruktor klase TestArena da biste napravili arenu 7k7 ćelija sa zidovima postavljeni sledeći način:



Podsetimo se da možemo da koristimo metod addObject() klase Vorld, koja ima tri parametra, da ubacimo instancu klase Actor u svet (tj. u potomke klase Vorld i, u našem slučaju, klase Arena) :
• glumac koji treba da se ubaci,
• k-koordinata ćelije koja se ubacuje (tj. indeks kolone numerisan od 0),
• i-koordinata ćelije koja se ubacuje (tj. indeks reda numerisan od 0)
[0;0] pozicija u svetu je u gornjem levom uglu.

6.8 Dodajte još više zidova

Izmenite konstruktor klase TestArena tako da ćelije budu raspoređene kao što je prikazano na slici.



6.9 Razmislite o tome kako da predstavite niz zidova

Hajde da razmislimo o tome koliko nam je informacija potrebno da bismo mogli da kreiramo bilo koju seriju uzastopnih zidova.

6.10 Dodajte metodu za stvaranje niza zidova

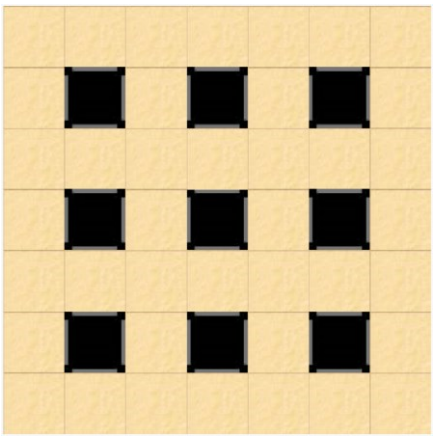
Napravite metodu createRovOfValls() u superklasi Arena, koja će imati tri parametra:
• red (gornji red ima indeks 0) na kojem se započinju pravljenje zidova,
• kolona (leva kolona ima indeks 0) od koje se počinje pravljenje zidova,
• broj koji pokazuje koliko uzastopnih zidova treba da se napravi.
Metod nema povratnu vrednost (zato koristimo ključnu reč void).

6.11 Koristite novi metod

Izmenite kod u konstruktoru klase TestArena da biste koristili metod createRovOfValls() iz njegove superklase.

6.12 Promenite raspored zidova

Izmenite konstruktor klase TestArena da biste kreirali arenu prikazanu na slici ispod. Da biste to uradili, modifikujte metod createRovOfValls() tako da ima četvrti parametar koji definiše razmak između zidova.



6.13 Razmislite o tome kako da predstavite pravougaonik zidova

Razmislimo o tome koliko i kakve informacije su nam potrebne da bismo mogli da kreiramo zidove u pravougaonom rasporedu čija se polazna tačka može odrediti i za koje se može podesiti razmak između

6.14 Dodajte metodu stvaranja pravougaonika zidova

Deklarisajte metod createRectangleOfValls() u pretku Arene, koji će imati sledeće parametre:

• red (najgornji red ima indeks 0) od koga se započinju pravljenje zidova,

• kolona (leva kolona ima indeks 0) od koje se počinje pravljenje zidova,

• broj redova koji će se kreirati,

• broj uzastopnih zidova koji će se napraviti u nizu,

• broj praznih ćelija (redova) između redova,

• broj praznih ćelija između zidova u redu.

Metoda nema povratnu vrednost.

6.15 Koristite novi metod

Izmenite konstruktor klase TestArena tako da koristi metod createRectangleOfValls() za postavljanje arene kao što je prikazano u zadatku 6.12.

6.16 Testirajte svoju arenu

Napravite nekoliko zidova u svojoj areni i dodajte dva igrača. Testirajte funkcionalnost igre, odnosno da li igrači neće ući u BrickVall ili u zid.

7 Lista i za svaku petlju

Ovo poglavlje se fokusira na liste detaljnije koristeći ih za praćenje drugih objekata u areni. Uvodi osnovne metode za rad sa listom (kreiranje, dodavanje elementa, uklanjanje elementa, pristupanje elementu) i takođe uči kako da koristite za svaku petlju za lak pristup svim elementima liste.

7.1 Proverite kraj igre

Kreirajte metod isGameEnded() bez parametara u klasi Arena koji detektuje da li je igra završena (preostao je samo jedan ili nijedan igrač) i govori da li se to dogodilo u povratnoj vrednosti logičkog tipa. Za sada, pretpostavimo da se kraj igre nikada ne dogodi.

7.2 Završi igru

Dodajte metodu act() klasi Arena. U metodi proverite da li je igra završena (pomoću metode isGameEnded()) i ako jeste, zaustavite igru. Da biste zaustavili Greenfoot okruženje, koristite Greenfoot.stop(); komanda.

7.3 Dodajte listu igrača

Dodajte atribut listOfPlaiers tipa LinkedList<Plaier> klasi Arena. Ne zaboravite da morate da uvezete paket sa klasom LinkedList. Inicijalizujte atribut u konstruktoru klase Arena.

7.4 Registrujte igrače

Dodajte metod registerPlaier() u klasu Arena koja uzima jedan parametar tipa Plaier i umeće ga na kraj liste listOfPlaiers koristeći metod add(). Izmenite podklase klase Arena da biste registrovali igrača u superklasi (Arena) kada je igrač ubačen u svet na tačnoj lokaciji.

7.5 Odjavite se i uklonite igrača

Dodajte unregisterAndRemovePlaier() klasi Arena koja uzima jedan parametar tipa Hrac. Metoda uklanja igrača sa liste igrača, a zatim ga uklanja sa sveta.

7.6 Ispravno završite igru

Implementirajte telo metode isGameEnded() tako da metoda vraća true kada u igri ostane jedan ili nijedan igrač. Koristite odgovarajuće metode liste.

7.7 Učinite bombe opasnim

Izmenite kod u metodi act() klase Bomb tako da pre nego što bomba bude uklonjena iz sveta, ona dobije sve igrače koji su udaljeni najviše jednu moć od nje. Metod getObjectsInRange() vraća listu tipa List. Njegov generički parametar je identičan klasi koja se šalje kao drugi parametar.

7.8 Uklonite pogođene igrače

Koristite for petlju da pređete preko svih igrača na listi igrača pogođenih bombom. Odjaviti takve igrače u klasi Arene.

7.9 Uredite klasu igrača

Kreirajte metod hit() u klasi Plaier koji će biti pozvan od strane bombe igrača nakon što ga bomba pogodi. Poništite registraciju igrača iz sveta na ovaj način. Uredite kod u metodi act() klase Bomb tako da odražava novu funkcionalnost.

7.10 Uklonite vlasnika

Napravite metodu removeOvner() u klasi Bomb koja postavlja svoj atribut vlasnika na null.

7.11 Dodajte listu bombi

Kreirajte atribut listOfActiveBombs tipa LinkedList<Bomb> u klasi Plaier. Inicijalizujte ga u ispravnom konstruktoru. Izmenite tela metoda prema sledećim pravilima:

• u metodi act() registrovati novokreiranu bombu na listuActiveBombs;

• u metodi bombEkploded(), uklonite bombu koja je došla kao parametar (onu koja je eksplodirala) sa listeActiveBombs;

u metodi hit(), koristite for svaku petlju da uklonite vlasnika iz svih bombi na listi ActiveBombs.

8 Privatne metode i vhile petlja

Ova tema uvodi vhile petlju – petlju koja se ponavlja dok se neki uslov definisan na početku petlje procenjuje na tačno. Štaviše, on takođe uči učenike kako da kreiraju privatne metode – metode koje se mogu pozvati samo od strane instance klase u kojoj je metod definisan.

8.1 Kreirajte klasu vatre

Kreirajte klasu Fire. Izaberite odgovarajući grafički prikaz. Konstruktor ove klase ima jedan parametar koji određuje koliko dugo vatra gori na mestu. Postarajte se da vatra nestane sa sveta nakon određenog vremena.

8.2 Zapali vatru

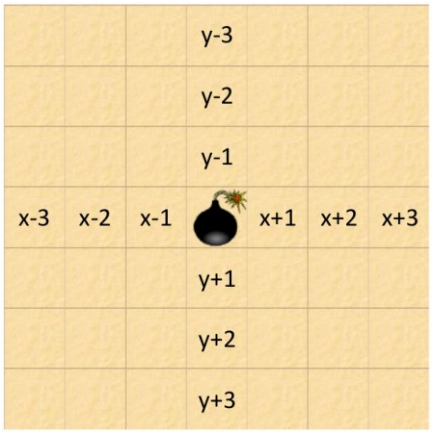
Izmenite postojeći kod eksplozije bombe da biste ostavili instancu klase Fire na mestu bombe. Testirajte svoje rešenje.

8.3 Raširite vatru

Koristite for petlju da proširite eksploziju bombe (napravite instancu klase Fire) u pravom smeru od bombe. Proširite eksploziju na onoliko ćelija koliko je naznačeno atributom snage bombe.

8.4 Širite vatru u svim pravcima

Podesite eksploziju bombe tako da stvara vatru u svim pravcima. Pomozite sebi tako što ćete promeniti koordinate kao što je prikazano na slici ispod.



8.5 Prepišite petlje

Prepišite sve for petlje za širenje vatre koristeći vhile petlju. Za sada izostavite drugi deo uslova (moguće je staviti vatru u sledeću ćeliju).

8.6 Koristite privatnu metodu

Koristite privatni metod spreadFire() da proširite vatru nakon eksplozije bombe.Add another private method

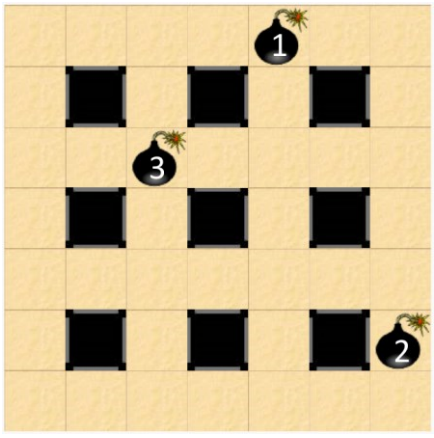
8.7 Dodajte još jedan privatni metod

Kreirajte privatnu metodu canCellEkplode() u klasi Bomb koja uzima koordinate reda i kolone kao parametre i vraća true ako može doći do eksplozije u toj ćeliji i false u suprotnom. Požar se ne može nastaviti ako:

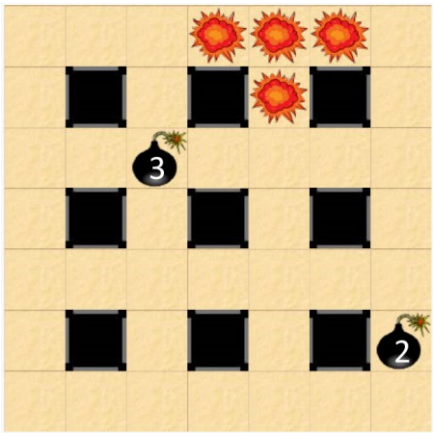
- stigao je na ivicu sveta,
- ako ćelija sadrži zid.

8.8 Koristite privatne metode

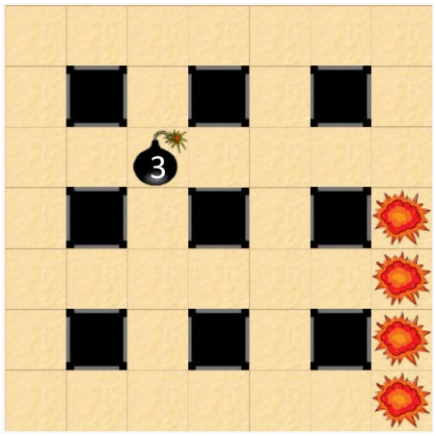
Koristeći metodu canCellEkplode() sada možete da izmenite uslov u vhile petlji u metodi spreadFire() u klasi Bomb. Izmenite uslov u petlji da poštuje rezultat provere iz metode canBombEkplode(). Testirajte funkcionalnost rešenja sa bombama različite jačine između zidova. Testovi različitih eksplozija su prikazani na sledećim slikama:



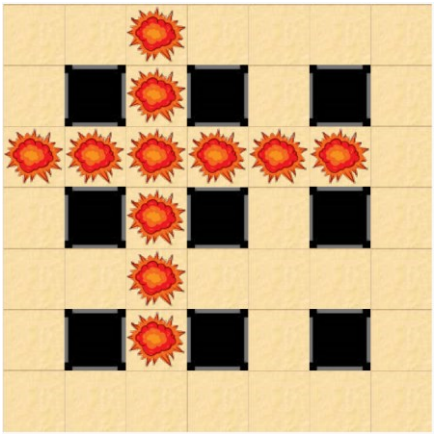
a



b



c



d

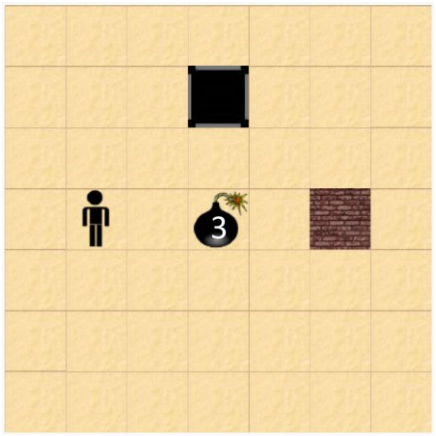
U delu (a) možemo videti originalnu distribuciju, u delu (b) je eksplodirala bomba snage 1, u delu (c) eksplodirala je bomba snage 2 i u delu (d) je eksplodirala bomba snage 3.

8.9 Proverite prepreku eksplozije

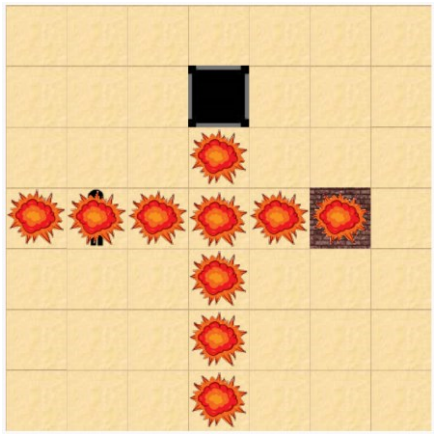
Dodajte privatnu metodu canEkplosionContinue() u klasu Bomb koja uzima koordinate reda i kolone u parametrima i vraća true ako ćelija nije zaustavila eksploziju na datim koordinatama. Ako je ćelija zaustavila eksploziju, metoda vraća netačno. Eksplozija se ne može nastaviti ako udari u zid.

8.10 Ograničite eksploziju

Koristeći metodu canEkplosionContinue(), izmenite metod spreadFire() u klasi Bomb. Ako se eksplozija može nastaviti iz date ćelije, povećajte vrednost promenljive i za 1 i ponovo izračunajte koordinate novog reda i kolone eksplozije. U suprotnom, veštački povećajte vrednost promenljive i na vrednost veću od sile bombe, što zaustavlja petlju. Testirajte svoje rešenje na situaciju sa sledeće slike:



A



b

8.11 Izmenite proveru prisutnosti vatre

Izmenite ponašanje instance klase Bomb tako da ne poziva igračev metod hit() u svom dosegu. Umesto toga, sam igrač će proveriti da li se vatra preklapa. Izmenite ponašanje metode act() igrača tako da prvo proveri da li se preklapa sa instancom klase Fire. Ako jeste, pozvaće sam svoj metod hit(). Ovo će osigurati da igrača takođe pogodi vatra koja gori nakon što bomba eksplodira.

8.12 Dodajte lanac eksplozija

Izmenite metod act() klase Bomb tako da bomba eksplodira čak i ako se nalazi u istoj ćeliji kao i vatra. Proverite rešenje tako što ćete izvršiti lančanu reakciju od nekoliko bombi.

9 Polimorfizam

Cilj ove teme je da nauči učenike kako da kreiraju virtuelne metode i preklapaju ih po potrebi u podrazredima. Takođe, studenti će biti upoznati sa novom vidljivošću atributa i metoda – modifikatorom zaštićenog pristupa. Učenici će koristiti polimorfizam da pojednostave postojeći kod.

9.1 Dodajte mine

Počnimo dodavanjem rudnika. Mina eksplodira baš kada je igrač nagazi. Mina će takođe uvek eksplodirati kada je pogodi vatra (npr. od bombe koja je eksplodirala u blizini). Ostavlja vatru na svom mestu (i samo tamo). Dodajte mogućnost da igrač postavlja mine (poput bombi) kada se pritisne taster (npr. kontrola ili promena). Slično kao kod bombi, igrač takođe ima ograničen broj mina (tj. ako postavi sve mine, može postaviti još jednu minu samo ako jedna od prethodno postavljenih mina eksplodira). Početni broj mina je postavljen parametrom konstruktora igrača. Da biste registrovali mine i reagovali na njihovu eksploziju u klasi Plaier, pratite istu proceduru kao i za bombe (napravite listu mina, dodajte metode mineEkploded(), canPlantMine(), itd.).

9.2 Dodajte superklasu

Napravite zajedničku superklasu za klase Bomb i Mine – klasu Ekplosive. Koje atribute i metode treba premestiti u superklasu, a koje treba da ostanu u potklasama? Izmenite postojeće klase prema svom dizajnu.

9.3 Podesite vidljivost atributa

Izmenite vidljivost atributa vlasnika u klasi Ekplosive u zaštićenu.

9.4 Dodajte tekstualni izlaz

Kreirajte metod printVhoIouAre() bez parametara u klasi Ekplosive koji nema povratnu vrednost. Metoda štampa tekst „EKSPLOZIV“ na ekran gde se eksploziv trenutno nalazi. Kreirajte instancu klase Mine i pozovite metod printVhoIouAre(). Šta se dešava? Napravite instancu klase Bomb i pozovite metod printVhoIouAre(). Šta se dešava u ovom slučaju?

9.5 Poboljšajte izlaz teksta

Kreirajte metod printVhoIouAre() u rudniku sa istim zaglavljem kao u klasi Ekplosive (tj. metoda će imati isto ime, iste parametre i isti tip povratne vrednosti). Metod će odštampati tekst „MINE“ na ekran. Opet, kreirajte instancu klase Mine i klase Bomb. Pokušajte da pogodite šta se dešava kada pozovete test metod u instanci klase Mine i instanci klase Bomb. Nakon toga, pozovite metode. Da li se vaša pretpostavka poklapa sa rezultatom?

9.6 Završite izlaz teksta

Zaobiđite metod printVhoIouAre() u klasi Bomb tako da zapiše tekst „BOMB“ na ekran. Proverite ispravnost svog rešenja.

9.7 Dodajte rukovanje eksplozijom

Kreirajte metode shouldEkplode() i ekplosion() u klasi Ekplosive i ekplosiveEkploded() u klasi Plaier kao što je gore opisano. Još uvek nemojte implementirati tela metoda. Ako je potrebna povratna vrednost, vratite false.

9.8 Neka eksploziv eksplodira

Napišite telo metode act() u klasi Ekplosive.

9.9 Neka bomba eksplodira

Zaobiđite metode shouldEkplode() i ekplosion() u klasi Bomb. Koristite odgovarajući kod iz metode act(). Primetite da je moguće jednostavno napisati tela metoda pošto nema potrebe da se razmatraju uslovi (superklasa je to uradila).

9.10 Neka mina eksplodira

Zaobiđite metode shouldEkplode() i ekplosion() u klasi Mine. Koristite odgovarajući kod iz metode act(). Zašto je na kraju potrebno ukloniti metod act()?

9.11 Dodajte interakciju sa vatrom

Izmenite telo metode shouldEkplode() u klasi Ekplosive tako da metoda vrati true ako instanca dodirne instancu klase Fire. Izmenite zamenjene metode shouldEkplode() u klasi Bomb i Mine da biste koristili funkcionalnost metode pretka.

9.12 Pojednostavite atribute igrača

Uklonite atribute listOfActiveBombs i listOfActiveMines iz klase Plaier. Dodajte jedan atribut tipa listOfActiveEkplosives tipa LinkedList<Ekplosive> u klasu Plaier. Inicijalizujte ga u konstruktoru i uklonite inicijalizaciju originalnih atributa iz konstruktora.

9.13 Pojednostavite rukovanje eksplozijom

Implementirajte telo ekplosiveEkploded() metode. Korišćenje operatora instanceof za određivanje da li je eksploziv bomba ili je eksploziv mina. Na osnovu njenog stvarnog tipa, povećajte brojač dostupnih bombi ili brojač dostupnih mina. Obavezno uklonite eksploziv sa liste aktivnih eksploziva. Na kraju, uklonite nepotrebne metode bombEkploded() i mineEkploded().

10 Slučajni brojevi

Ova tema je posvećena slučajnosti. Učenici će učiti o Random času. Uz pomoć njegovih instanci, oni će generisati nasumične brojeve. Tema takođe pokazuje način generisanja slučajnih brojeva bez upotrebe klase Random direktnim korišćenjem Greenfoot okruženja. Učenici će koristiti nasumične brojeve kako bi nasumično podesili raspored arene i dodali bonuse svetu – specijalne elemente koji se stvaraju nakon što zid od cigle eksplodira i koji poboljšavaju svojstva izabranih igrača.

10.1 Razmislite o slučajnosti

Razmislite o tome šta je slučajnost, kako možemo dobiti neki slučajni rezultat iz eksperimenta i koje slučajne pojave posmatramo u svetu oko nas.

10.2 Razmislite o generisanju slučajnih vrednosti

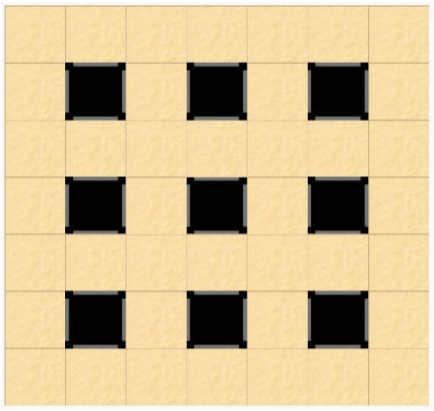
Hajde da razmotrimo klasičnu kocku sa šest strana. Možemo li ga koristiti da generišemo slučajnu poziciju na šahovskoj tabli sa 6k6 polja? Šta je sa šahovskom tablom sa kvadratima 3k3? Kako bi se promenio način generisanja ako bismo koristili novčić? Predložite takve algoritme za generisanje položaja.

10.3 Posmatrajte slučajnost

Koristeći algoritam iz prethodnog zadatka i uz pomoć kockice generiše nasumične pozicije na šahovskoj tabli. Zabeležite svoje rezultate na šahovskoj tabli. Može li se uočiti bilo kakva pravilnost u rezultatima?

10.4 Pripremite slučajnu arenu

Pripremite arenu. Napravite podklasu klase Arena, koju imenujete npr. RandomArena. Podesite odgovarajuću veličinu sveta u konstruktoru. Preporučujemo redovan raspored zida sa jednim praznim poljem između zidova kao što je prikazano na sledećoj slici:



10.5 Dodajte generator slučajnih brojeva

Dodajte referentni atribut tipa Random klasi RandomArena. Zapamtite da je klasa Random definisana u paketu java.util.Random. Inicijalizujte atribut u konstruktoru.

10.6 roverite popunjenost ćelija

Dodajte privatnu metodu isCellFree() klasi RandomArena koja uzima dva parametra – kolonu i red. Metod će vratiti tačno ako je ćelija slobodna u svetu (ne sadrži aktera), u suprotnom će vratiti netačno.

10.7 Generišite nasumične zidove

Izmenite konstruktor klase RandomArena da nasumično generiše zidove u trećinu svih ćelija u areni.

10.8 remestite slučajnost na pretka

Premestite metode createRandomVall(), isCellFree() i atribut generatora (uključujući njegovu inicijalizaciju u konstruktoru) u nadklasu Arena. Ne zaboravite da pomerite i linije za uvoz.

10.9 eneralizujte nasumično generisanje

Dodajte parametar tipa Actor metodi createRandomVall() – to će biti akter, kojeg ćemo ubaciti na nasumične koordinate. Promenite ime metode (npr. insertActorRandomli()) i ažurirajte poziv metode iz klase RandomArena.

10.10 Dodajte bonuse

Kreirajte Bonus klasu kao podklasu klase Actor. Napravite dve podklase klase Bonus – klasu BonusFire i klasu BonusBomb. Postavite odgovarajuće slike za časove.

10.11 Kreirajte bonuse nasumično

Uredite kod u metodi act() klase BrickVall. Sa verovatnoćom od 10% generiše na njeno mesto bonus vatru, a sa verovatnoćom od 10% generiše bonus bombu na njenom mestu. U 80% slučajeva ništa se ne generiše nakon uništenja.

10.12 Primenite bonus

Pripremite bonus klasu. Kreirajte zaštićeni virtuelni applilourself() metod bez povratne vrednosti, koji uzima jedan parametar tipa Plaier. Ostavite ovaj metod prazan u klasi Bonus.

Zatim definišite akciju u metodi act() da biste prvo otkrili da li je igrač stao na bonus (metod (Plaier)this.getOneIntersectingObject(Plaier.class)) i ako jeste, primenite je (pozivanjem virtuelne metode) i na kraju uklonite bonus sa sveta.

10.13 P ovećajte broj bombi

Dodajte javnu metodu povećanjaBombCount() bez parametra u klasu Plaier koja nema povratnu vrednost i koja će povećati broj bombi koje igrač može da postavi za jednu. Zatim nadjačajte metod applilourself() u klasi BonusBomb. Primena bonusa znači povećanje broja bombi koje igrač ima za jednu (pozivanje metode povećanjaBombCount()).

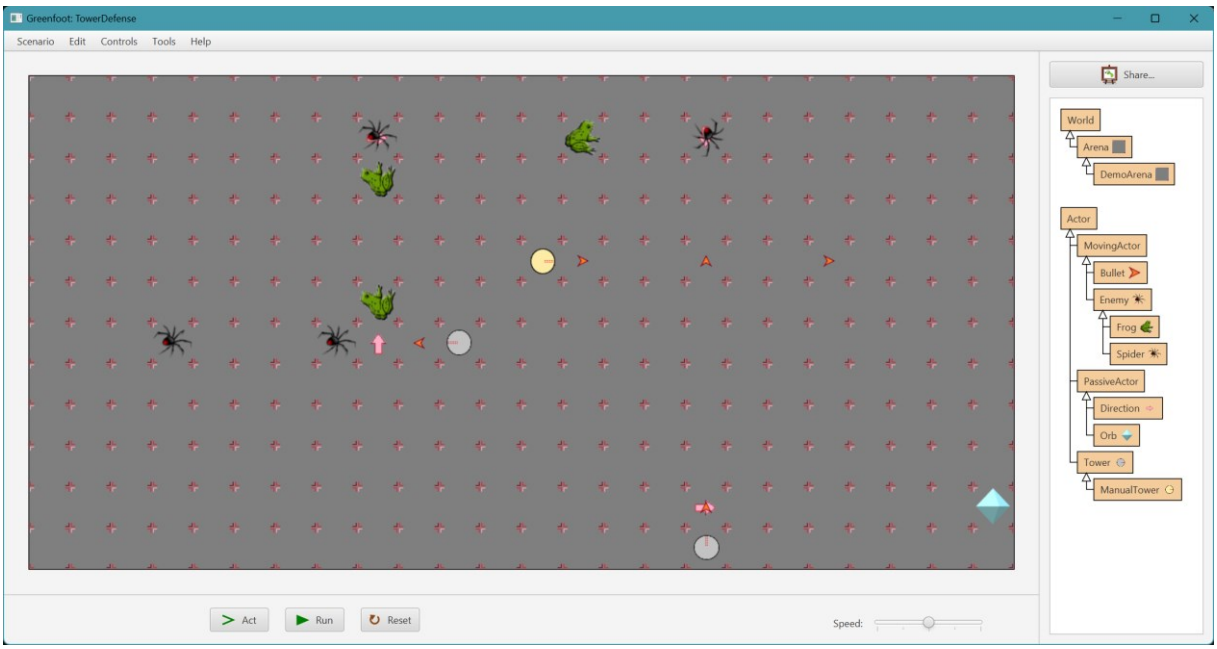
10.14 Povećajte snagu bombe

Dodajte javni metod povećanjaBombPover() bez parametra u klasu Plaier koja nema povratnu vrednost i koja povećava vrednost atributa bombPover za jedan. Zatim nadjačajte metod applilourself() u klasi BonusFire i povećajte snagu bombe igrača za jedan.

3.2 Toranj odbrana

U odbrani tornja poput igrica, igrač koristi kule koje ispaljuju neku vrstu metaka kako bi sprečile neprijatelje da dosegnu i unište kuglu. Neprijatelji uvek idu istim putem, međutim kako se igra nastavlja, neprijatelji postaju jači i rađaju se u većim grupama. Igrač mora da postavi kule na strateška mesta kako bi ih zaustavio u nadolazećim talasima. Postoji mnogo verzija igre koje se odvijaju u različitim svetovima koristeći različite entitete (od balona do orka, odbrane koristeći životinjske kule do magičnih bazena).

Ovaj projekat će uvesti jednu vrstu kule koju igrač može ili ne može ručno kontrolisati. Neprijatelji će biti različitih tipova sa razlikama u HP i brzini. Uvedeni dizajn igre je lako proširiti, što ostavlja dovoljno prostora za kreativnost učenika, kao i za zadatke nastavnika. Iz tog razloga smo pokušali da minimiziramo aktivnosti tipa procene u prvim poglavljima. Štaviše, dizajn ostavlja dovoljno prostora za prirodno uvođenje tema koje su van okvira lakog OOP-a (kao što je polimorfizam) sa lakoćom. Projekat u svom konačnom stanju tokom igranja prikazan je na slici 3



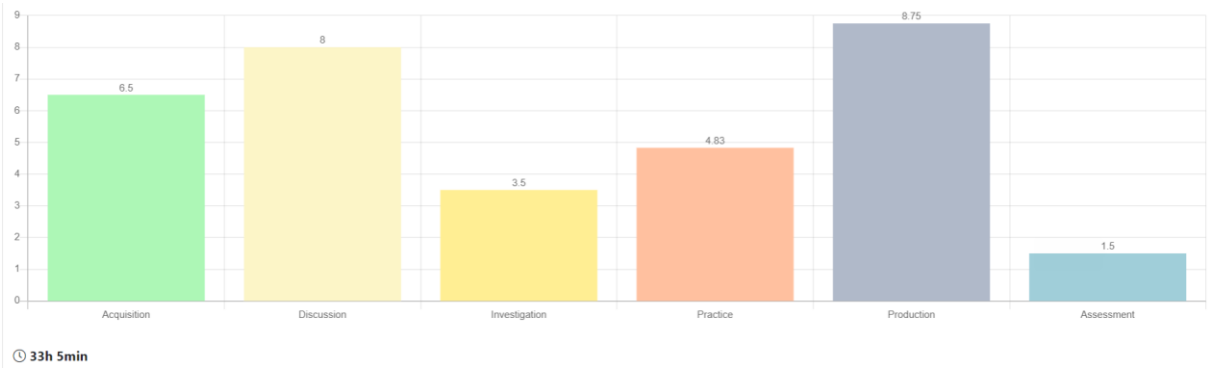
Slika 3: Greenfoot okruženje sa konačnim stanjem projekta Tover defense

Izvorni kodovi su dostupni na:
https://gitlab.kicon.fri.uniza.sk/oop4fun/project-tower-defense

Izvorni kodovi su dostupni na:
http://learning-design.eu/en/preview/452257b563cbf14b6f06acfd/details

3.2.1 Sadržaj i obim obrazovnog programa

Ukupno opterećenje učenika je 33 sata i 5 minuta i raspoređuje se na sledeći način:



Slika 4: Radno opterećenje učenika kada se koristi projekat Tover defense

Konstruktivno usklađivanje je sažeto u tabeli ispod:

Tabela 2: Konstruktivna usklađenost projekta Tover defense

Topic	Assessment		Understanding the basic principles of object-orien... (25)	Understanding the basics of algorithmisation (25)	Understanding the syntax of the Java programming l... (10)	Analysing program execution based on the source co... (20)	The ability of creating own programs with the use ... (20)
	Formative	Summative					
Greenfoot environment	0	0					100%
Class definition	0	0	60%		20%		20%
Algorithm	0	0		60%	10%	20%	10%
Branching	0	0	10%	60%	10%	10%	10%
Variables and expressions	0	0	40%	30%	20%		10%
Association	0	60	30%	30%	10%		30%
Inheritance	0	30	40%	20%	10%		30%
Encapsulation	0	0	50%	10%	20%		20%
Total	**0**	**90**	**230%**	**210%**	**100%**	**30%**	**230%**
	90						

Za detaljan plan pogledajte prilog 5.2.

3.2.2 Teme

Projekat Tover Defense je podeljen na sedam tema:

Pokrivene teme svetlosnog OOP-a su:

- klase, objekti, instanca
- metode, prosleđivanje argumenata metodama
- konstruktori
- atributi
- statičke varijable i metode
- inkapsulacija

• nasleđe
• apstraktni časovi
•interfejs
• životni ciklus objekta

1 Uvod u Greenfoot okruženje

Tema je posvećena kreiranju projekata. Učenici će biti sposobni da kreiraju novi projekat u Greenfoot okruženju, kreiraju klasu (kao podklasu Actor), izaberu sliku za novokreiranu klasu, kreiraju njenu instancu i pošalju joj poruku.

Kreirajte novi projekat. Dajte mu pravo ime (npr. ToverDefense) i sačuvajte ga na odgovarajućoj lokaciji.

Tabela 3 sumira poređenje opterećenja teme Greenfoot okruženje između projekata Bomberman i Tover defense. Nema razlike u dizajnu tema.

Tabela 3: Poređenje opterećenja teme Greenfoot okruženje između projekata Bomberman i Tover defense

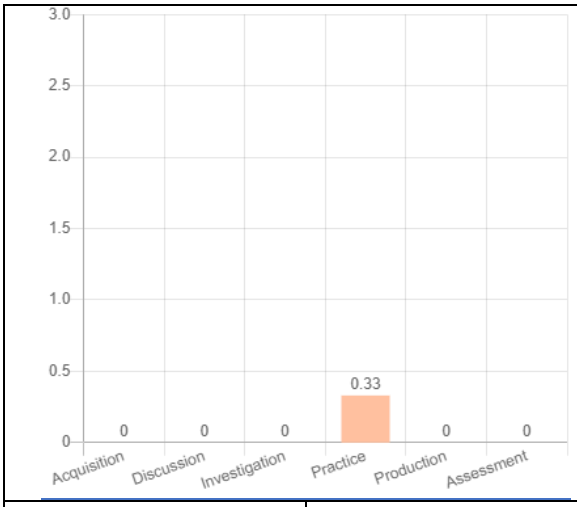
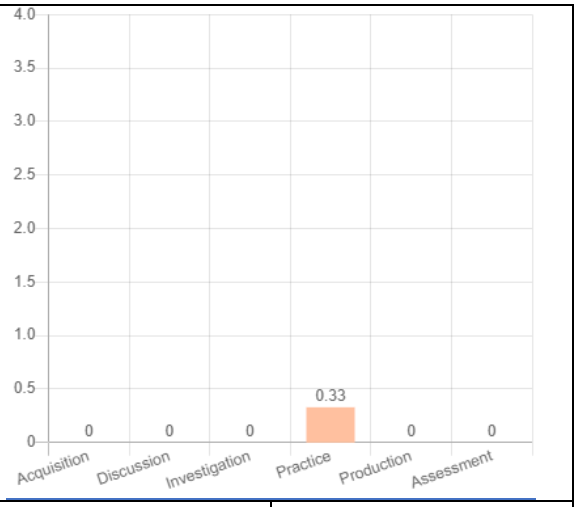
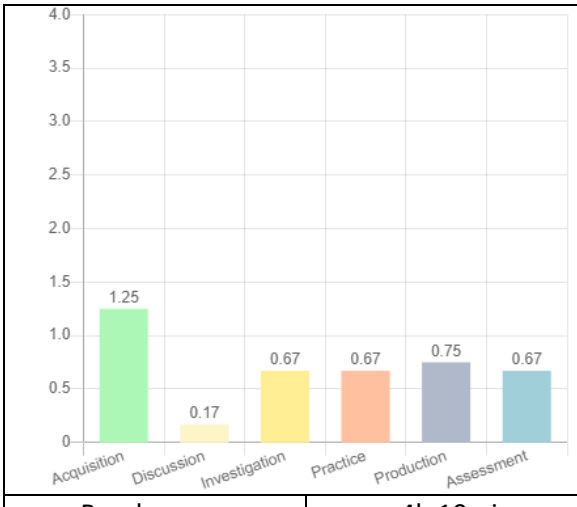
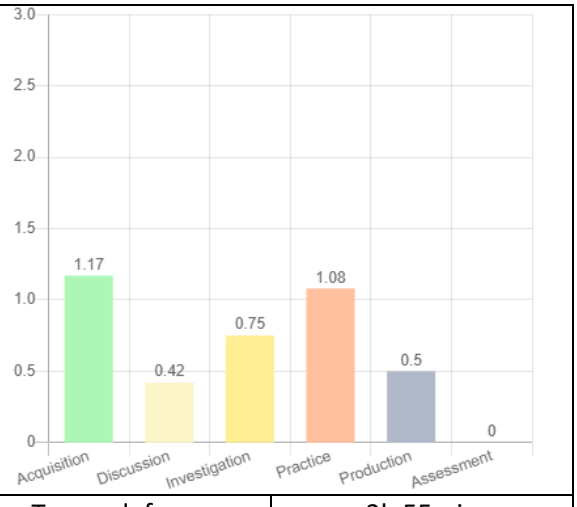
	Bomberman	0h 20min		Tower defense	0h 20min

Tabela 4 sumira poređenje opterećenja teme Definicija klase između projekata Bomberman i Tover defense. Obim posla projekta Tover Defense je manji, praktičnije orijentisan sa akcentom na istraživanje i praksu.

Tabela 4: Poređenje opterećenja teme Definicija klase između projekata Bomberman i Tover defense

	Bomberman	4h 10min		Tower defense	3h 55min

Task 0

1.1 Identifikujte objekte iz projekta Bomberman.

1.2 Pripremite svet

Uredite izvorni kod klase MiVorld (dvaput kliknite na njega) da biste kreirali svet veličine ćelija 24k12. Svaka ćelija treba da bude veličine 50 piksela.

1.3 Pripremite svetsku grafiku

Pronađite ili napravite odgovarajuću sliku za pozadinu sveta. Možete koristiti ili pripremljene slike (izaberite stavku Postavi sliku... iz kontekstnog menija klase MiVorld) ili prilagođenu sliku (kopirajte sliku u slike podfoldera vašeg projektnog foldera i izaberite je na isti način kao što je prethodno opisano).
Kao pozadinu možete koristiti jedinu sliku koja će pokrivati celu svetsku oblast (izračunajte potrebnu veličinu slike u odnosu na veličinu sveta) ili manju, koja će se više puta kopirati (koristite kvadratnu sliku sa veličinom ćelije).

1.4 Napravite klasnog neprijatelja

Stvorite neprijatelja. Neprijatelj će marširati prema kugli igrača da bi je oštetio i na kraju uništio. Kreirajte novu podklasu klase Actor (izaberite stavku Nova potklasa... iz kontekstnog menija klase Actor). Dajte mu pravo ime (Neprijatelj) i sliku.

1.5 Kreirajte instancu klase Neprijatelj

Kreirajte instancu klase Enemi (izaberite stavku nev Enemi() iz kontekstnog menija klase Enemi, stavite instancu u svet levim klikom miša na željenu poziciju). Istražite njegovo unutrašnje stanje (izaberite stavku Inspect iz kontekstnog menija kreirane instance).
Napravite još jednu instancu klase Enemi i stavite je na drugu poziciju. Uporedite unutrašnja stanja dve kreirane instance.

1.6 Šaljite poruke instanci

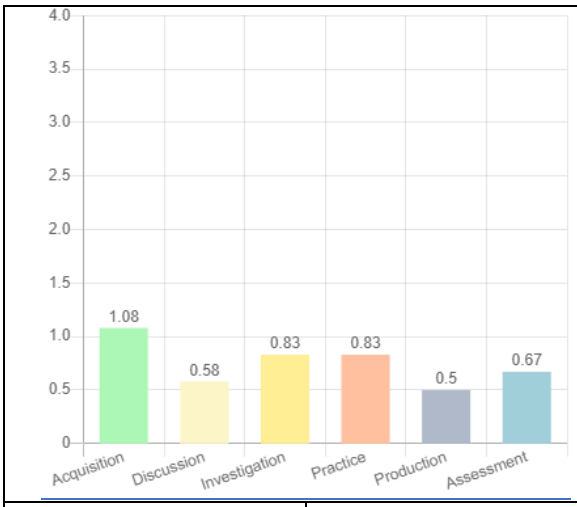
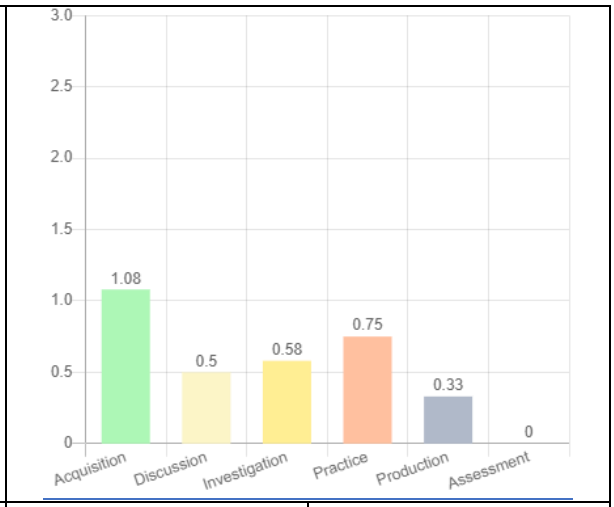
Pošaljite poruku instanci klase Neprijatelj (izaberite stavku nasleđenu od Actor iz kontekstnog menija izabrane instance, a zatim izaberite željenu stavku). Šta se desilo? Kako je uticalo na unutrašnje stanje dotične instance?
Send messages to the instance of class Enemy so it will move into position [12, 6] and it will be facing down. Write the sequence of sent messages onto paper.

2 Algoritam, kontrole aplikacije, kreiranje metoda

Tema se bavi osnovama algoritmizacije i uvodi rad sa dokumentacijom od samog početka. Studenti će biti sposobni da pozivaju metod u izvornom kodu, pišu i pozivaju dokumentaciju.

Tabela 5 sumira poređenje opterećenja teme Algoritam između projekata Bomberman i Tover defense. Dizajn Tover defence je sličan Bombermanu, ali sa znatno manjim brojem TLA istražnog tipa. Vidite, da su mnogi TLA isti kao u Bomberman projektu. To je zato što u ovakvom stanju projekata postoji veliko preklapanje onoga što se može uraditi sa tim. Inspiraciju je moguće pronaći u zadacima u projektu Bomberman, ako bude bilo potrebe da se pojača istraživački deo nastavnog plana. Međutim, radi podučavanja svetlosnog OOP-a korišćenjem projekta Tover Defense, smatramo da je predložena količina TLA-a tipa istraživanja dovoljna.

Tabela 5: Poređenje opterećenja teme Algoritam između projekata Bomberman i Tover defense

	Bomberman	4h 30min		Tower defense	3h 15min

2.1 Zadatak 2.1 Napisati jednostavan algoritam iz projekta Bomberman.

2.2 Zadatak 2.2 Napisati opštiji algoritam iz projekta Bomberman.

2.3 Pozovite metod

Dodajte izjavu u telo metode act() tako da instanca klase Enemi pomeri dve ćelije u trenutnom pravcu. Zatim kreirajte više instanci klase Enemi i pozovite metod na svakoj instanci. Da li je ponašanje očekivano?

2.4 Dodajte dokumentaciju

Dodajte komentar dokumentacije za metodu act().

2.5 Dodajte još dokumentacije

Uredite komentar dokumentacije klase Enemi. Dodajte verziju klase i njenog autora.

2.6 Zadatak 2.7 Pročitajte dokumentaciju iz projekta Bomberman

2.7 Zadatak 2.9 Istražite kontrole aplikacije iz projekta Bomberman

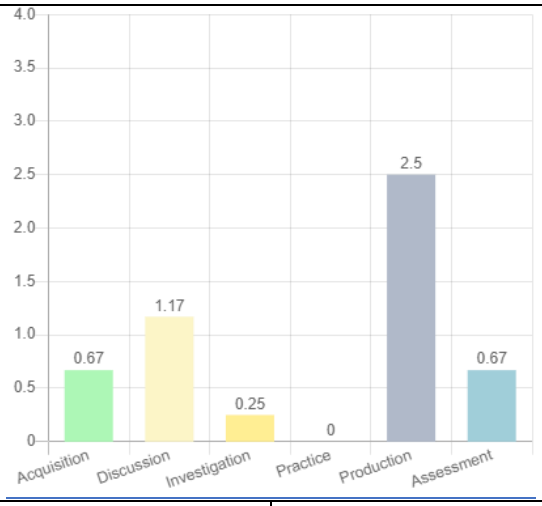
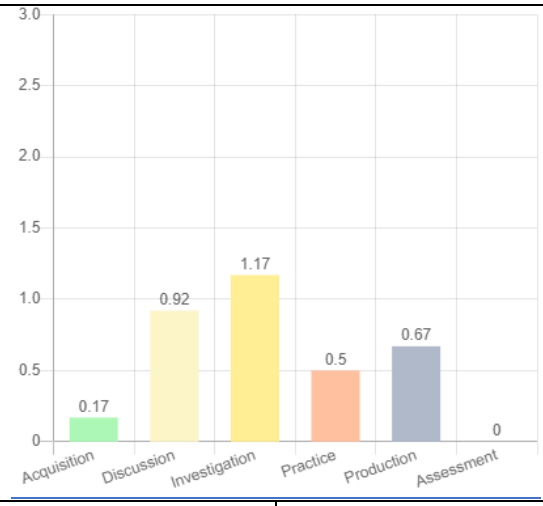
3 Grananje i neprijateljska kontrola

Tema pokriva nepotpuno i potpuno grananje (višestruko grananje je namerno izostavljeno). Uvedene su osnove glumačke percepcije sveta. Učenici će biti sposobni da pišu kod koristeći uslove.

Stanje projekta u ovom poglavlju otvara mogućnosti za nastavnika da dodeli zadatke kao što su „koristite instance klase Direction i Orb za navigaciju neprijatelja tako da se kreće po željenom obrascu“, sa ograničenim uslovima kao što je korišćenje maksimalnog broja instanci određene klase ili zadaci poput „predvidi kretanje u željenoj postavci sveta“.

Tabela 6 sumira poređenje opterećenja teme Grananje između projekata Bomberman i Tover defense. Postoji jasna razlika između dizajna. Odbrana tornja deli TLA-ove proizvodnog tipa da bi se ojačala istraga i praksa. Ovo omogućava više eksperimentisanja i ostavlja otvorena vrata za kreativnost učenika. Obratite pažnju na mali broj TLA-a za akviziciju. To je zato što a) nije uvedeno višestruko grananje i b) su naglašeni TLA-i istraživanja.

Tabela 6: Poređenje opterećenja teme Algoritam između projekata Bomberman i Tover defense

	Bomberman	5h 15min		Tower defense	3h 25min

3.1 Posmatrajte stanje igrača

Napravite instancu klase Neprijatelj i postavite je u centar table. Otvorite prozor sa unutrašnjim stanjem instance i postavite ga tako da bude vidljiv dok je aplikacija pokrenuta. Zatim pokrenite aplikaciju i posmatrajte kako se menjaju vrednosti atributa k, i i rotacije u klasi Enemi. Kako se ove vrednosti menjaju dok se krećete (gore, dole, levo i desno) i okrećete?

3.2 Dodajte detekciju ivica sveta

Dodajte kod u telo metode act() da biste rotirali neprijatelja za 180° nakon što stignete do ivice sveta.

3.3 Dodajte klase Direction i Orb

Napravite dve nove klase, potomke klase Glumac. Prva klasa će biti klasa smera, a druga klasa će biti orb. Pripremite odgovarajuće (maks. 50k50 piksela) slike u grafičkom uređivaču. Zatim dodelite ove slike novostvorenim klasama.

3.4 Dodajte detekciju sudara

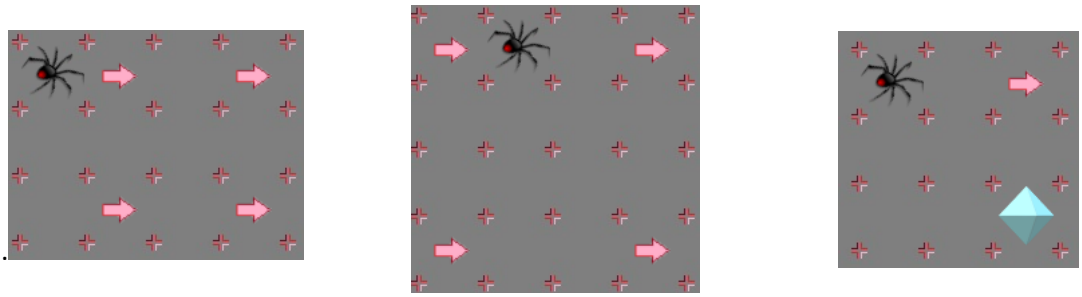
Dodajte kod metodi act() klase Enemi da biste osigurali da:

• igrač se okreće za 90° u smeru kazaljke na satu kada uđe u ćeliju koja sadrži instancu klase Direction,

• igrač se okreće za 90° u smeru suprotnom od kazaljke na satu kada uđe u ćeliju koja sadrži instancu klase Orb.

3.5 Predvidite kretanje neprijatelja na prilagođenom podešavanju

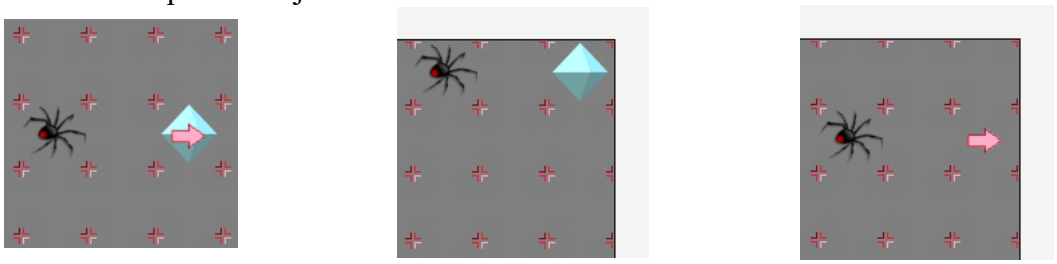
Pripremite različite konfiguracije, inspiraciju možete pronaći na slikama ispod. Pogodite kako će se neprijatelj kretati? Pokrenite aplikaciju. Da li se vaše predviđanje poklapa sa onim što posmatrate? Šta je izazvalo razlike u predviđanju i stvarnosti?



Slika 5: Konfiguracije prilagođenih podešavanja instanci za predviđanje kretanja instance klase Enemi

3.6 Predvidite kretanje neprijatelja na lukavom podešavanju

Pripremite situaciju kao što je prikazano na slici ispod. Pogodite kako će se neprijatelj kretati? Pokrenite aplikaciju. Da li se vaše predviđanje poklapa sa onim što posmatrate? Šta je izazvalo razlike u predviđanju i stvarnosti?



Slika 6: Konfiguracije lukavih podešavanja instanci za predviđanje kretanja instance klase Enemi

3.7 istite potpuno grananje sa detekcijom kolizije

Izmenite kod metode act() klase Enemi tako da koristi potpuno grananje. Stvorićete kaskadu uslova. Neka detekcija ivice bude najvažnija provera, zatim proverite dodir instance klase Direction i na kraju proverite dodir instance klase Orb.

3.8 Predvidite kretanje neprijatelja na prethodnim podešavanjima

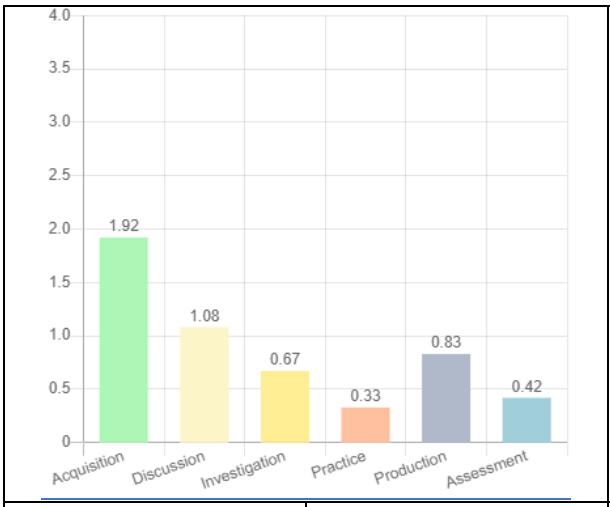
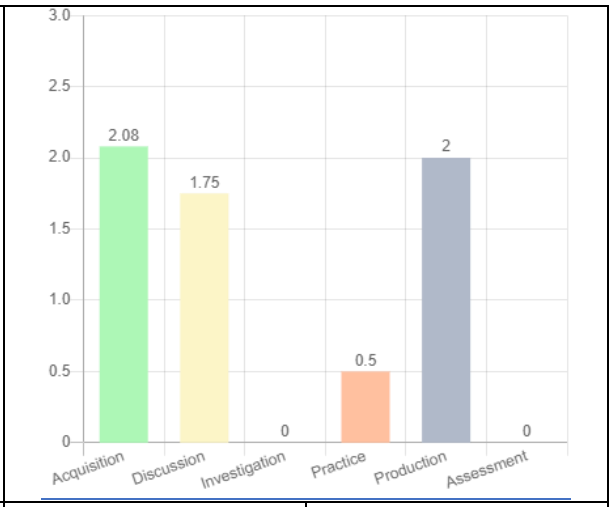
Ponovo prođite kroz zadatke 3.5 i 3.6. Šta se promenilo?

4 Promenljive i izrazi

Ovo poglavlje uvodi promenljive i izraze.

Stanje projekta u ovom poglavlju otvara slične mogućnosti kao u prošlom poglavlju. Uz malo kreativnosti može se dodati još klasa kao što je klasa Direction, koja će uzrokovati drugačije ponašanje kada stupite na (npr. teleporti, tuneli, itd.) od strane instance klase Neprijatelj. O ovim časovima se može razgovarati sa učenicima i odgovarajuća implementacija se može dodeliti kao domaći zadaci. Tabela 7 sumira poređenje opterećenja teme Varijable i izrazi između projekata Bomberman i Tover defense. Pogledajte sličnost između dizajna prethodnih i trenutnih tema između dva projekta. Ova tema je više orijentisana na produkciju (slično prethodnoj temi u Bombermanu) i obrnuto (imajte na umu da je Bomberman drugačiji tip projekta, gde je kreativnost učenika bila sa prednostima korišćenim u ovoj temi).

Tabela 7: Poređenje opterećenja teme Varijable i izrazi između projekata Bomberman i Tover defense

	Bomberman	5h 15min		Tower defense	6h 20min

4.1 Turn in direction

Promenite kod u Enemi's act() metodi tako da će se okrenuti u istom pravcu kao instanca klase Direction (imaće istu rotaciju). Koristite getOneIntersectingObject(_cls_) metod da skladištite instancu u odgovarajuću lokalnu promenljivu (Pravac smera - moraćete da koristite tipecast, pošto je povratna vrednost tipa Actor, napišite (Direction) ispred poziva getOneIntersectingObject metode, kasnije ćemo doći do tipecast) . Ako je dobijena bilo koja instanca odgovarajuće klase, izdvojite rotaciju pravca pomoću metode getDirection() (sačuvajte je ako želite) i zatim je postavite u neprijatelj (ovo) pomoću metode setDirection(int). Testirajte svoje rešenje.

4.2 Preimenujte klasu MiVorld u Arena

Dajte klasi MiVorld bolje ime. Preimenujte ga u Arena. Ne zaboravite da u skladu sa tim preimenujete konstruktor.

4.3 Napravite izgled Arene

Napravite prilagođeni izgled Arene. Popunite konstruktor klase. Dodajte jednu instancu Enemi-a, jednu instancu Orb-a i najmanje jednu instancu Direction-a. Da biste dodali (podklasu) glumca, možete koristiti sledeći šablon:

1. Deklarisajte i inicijalizujte promenljivu potrebnog tipa (podklasa Actor)
   Enemi e = nev Enemi();
2. Dodelite svojstva koristeći odgovarajuće metode
   e.setRotation(90);
3. Stavite ga u svet (Arena) koristeći metod addObject(Actor).
   this.addObject(e, 6, 0);

Testirajte svoje rešenje.

4.4 Identifikujte problem sa kretanjem i predložite rešenje

Identifikujte šta uzrokuje probleme sa kretanjem. Kako se ovi problemi mogu rešiti?
Neprijatelj trenutno pomera 2 ćelije odjednom, što izaziva probleme sa kretanjem. Možemo drugačije modelirati brzinu neprijatelja. Instanca neprijatelja će uvek pomeriti 1 ćeliju odjednom. Međutim, uvodimo odlaganje pokretanja – instanca Enemi-a će se pomeriti nakon što prođu pozivi odlaganja metode act().

4.5 Atribut Enemi.moveDelai

Dodajte novi atribut tipa int pod nazivom moveDelai u klasu Enemi. Kreirajte parametarski konstruktor sa parametrom da biste inicijalizovali ovaj atribut. Inicijalizujte atribut parametrom. U skladu sa tim prilagodite kod u Areni.

4.6 Kretanje neprijatelja uz poštovanje kašnjenja

Ažurirajte metod act() klase Enemi, tako da se kreće nakon moveDelai poziva metode. Uvesti novi atribut nektMoveCounter. Inicijalizirajte ga na 0. Izmenite metod act() tako da poziva this.move(1) samo ako nektMoveCounter dostigne 0. Nakon pokreta, resetujte nektMoveCounter na vrednost moveDelai. Ako se instanca Enemi ne može pomeriti (jer nektMoveCounter nije dostigao 0), smanjite nektMoveCounter za 1.

4.7 Parametarski konstruktor klase Direction

Dodajte parametarski konstruktor u klasu Direction sa rotacijom jedinog parametra tipa int. Rotirajte kreiranu instancu u telu konstruktora prema parametru. U skladu sa tim prilagodite kod u Areni.

4.8 Preopterećenje konstruktora u klasi Direction

Preopteretiti konstruktore u klasi Direction dodavanjem neparametarskog konstruktora. U telu neparametarskog konstruktora, pozovite parametarski konstruktor sa smerom argumenta jednakim 0. Prilagodite kod u Areni u skladu sa tim – gde je moguće, pozovite neparametarsku verziju konstruktora klase Direction.

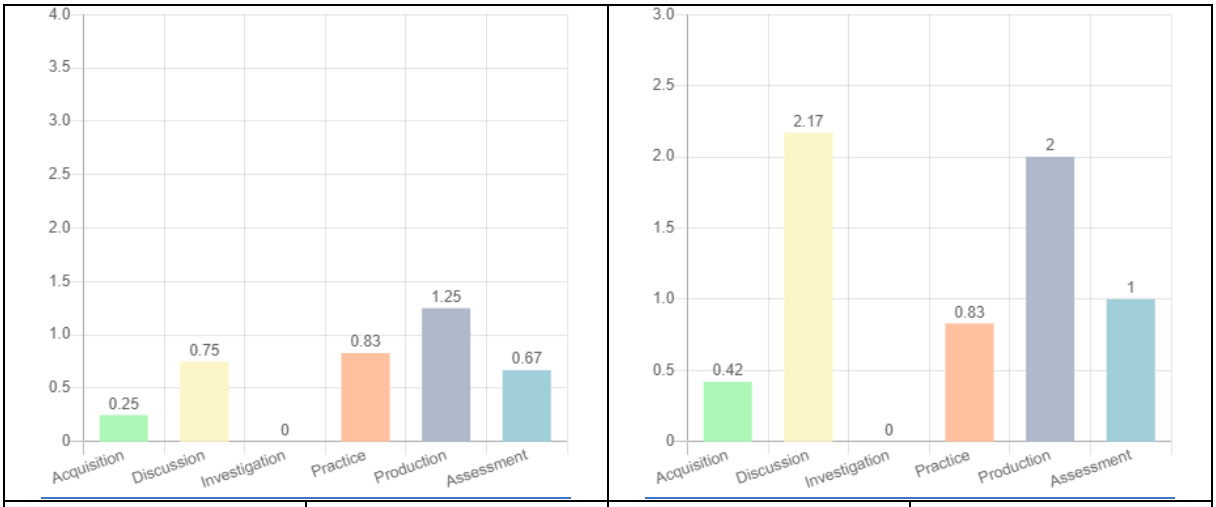
5 Asocijacija

Najvažnija tema ovog projekta je usmerena na udruženje. Diskusija se koristi da bi se otkrilo kako saradnja različitih objekata može dovesti do složenog ponašanja, iako kodovi u objekti su laki za razumevanje i održavanje. UML dijagrami sekvence se koriste da ilustruju saradnju objekata i širenje algoritma među kooperativnim objektima. Ovaj dijagram se može napraviti tokom diskusije sa razredom.

Projekat se može naći završenim nakon ove teme. Sledeća poglavlja uvode više varijabilnosti u aplikaciju sa fokusom na prednosti OOP-a kada se pravilno koristi.

Tabela 8 sumira poređenje opterećenja teme Povezivanje između projekata Bomberman i Tover defense. Smatramo da je razumevanje asocijacije najvažnija kompetencija kada se koristi ovaj projekat za podučavanje OOP-a. Stoga smo značajno ojačali TLA za proizvodnju i diskusiju. Imajte na umu da postoje i TLA za procene. Oni su dizajnirani na način da koriste prethodno urađene TLA u nešto drugačijem kontekstu.

Table 1: Comparision of workloads of topic Association between projects Bomberman and Tower defense



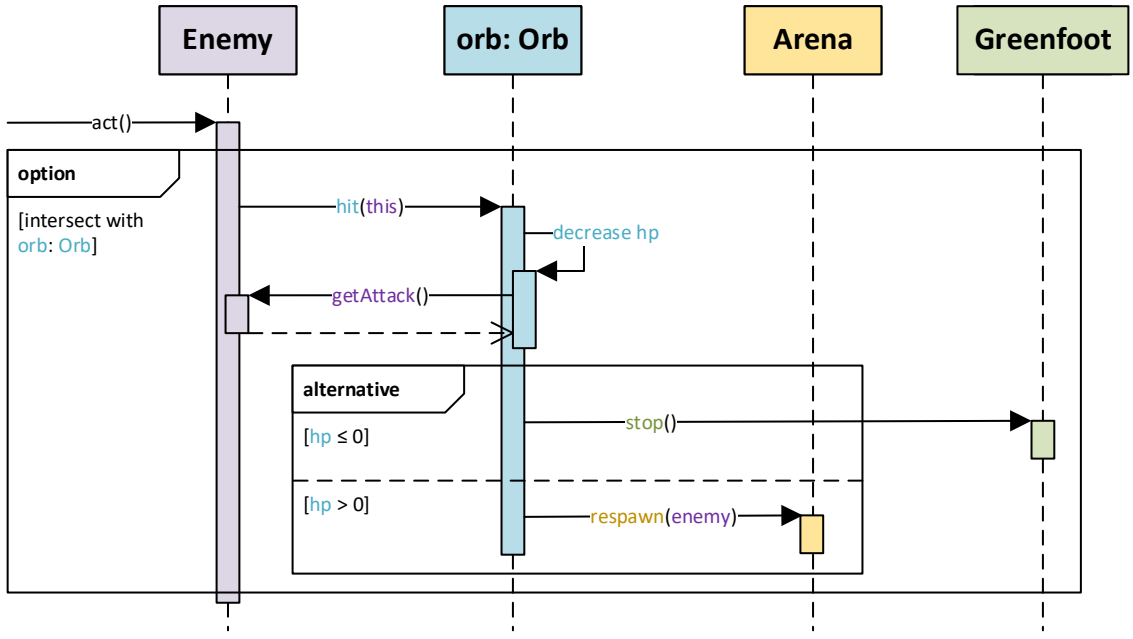
Bomberman	3h 45min	Tower defense	6h 25min

5.1 Razgovarajte o tome šta bi trebalo da se desi kada neprijatelj stigne do kugle

Nakon što neprijatelj stigne do kugle, kugla smanjuje HP. Ako je HP = 0, igra se završava, inače se neprijatelj ponovo pojavljuje u areni.

5.2 Razgovarajte o tome kako instanca klase Enemi treba da stupi u interakciju sa relevantnim objektima koristeći poruke kada udarite instancu klase Orb

Algoritam se širi među saradničkim objektima.



Slika 7: UML dijagram sekvence instance klase Enemi u interakciji sa drugim objektima kada udari instancu klase Orb

5.3 Atributi Enemi.attack i Orb.hp

Dodajte novi atribut tipa int koji se zove napad u klasu Enemi. Dodajte parametar u da biste inicijalizovali ovaj atribut. Inicijalizujte atribut parametrom.

Dodajte novi atribut tipa int koji se zove hp u klasu Orb. Dodajte parametarski konstruktor sa parametrom da biste inicijalizovali ovaj atribut. Inicijalizujte atribut parametrom.

U skladu sa tim prilagodite kod u Areni.

5.4 Dobavljač atributa Enemi.attack

Kreirajte napadač atributa u klasi Enemi.

5.5 Kreirajte i testirajte metod Arena.respavn(Enemi)

Dodajte metod respavn bez povratne vrednosti i sa jedinim parametrom tipa Enemi u klasu Arena. U metodi postavite lokaciju i rotaciju neprijatelja na iste vrednosti kao kada je kreiran u konstruktoru.
Testirajte metodu. Nakon kreiranja instance Arene, nemojte pokretati aplikaciju. Umesto toga, prevucite instancu Enemi. Zatim pozovite kontekstni meni instance Arena i izaberite stavku metoda respavn. Da biste popunili parametar, uverite se da je aplikacija pauzirana i da je parametar aktivan (sa kursorom koji treperi unutra). Ako jeste, kliknite levim tasterom miša na instancu Enemi. Posmatrajte koji izraz je izgrađen u prozoru. Zatim kliknite na dugme OK i pogledajte šta se dešava.

5.6 Kreirajte i testirajte metod Orb.hit(Enemi)

Dodaj metod hit bez povratne vrednosti i sa jedinim parametrom tipa Enemi u klasu Orb. Neka se telo isprazni.
Testirajte poziv metode. Koristite slične korake kao gore, međutim, pozovite kontekstni meni instance klase Orb. Posmatrajte koji izraz je izgrađen u prozoru.

5.7 Pozovite metod Orb.hit(Enemi) iz Enemi

Izmenite kod u metodi act() klase Enemi tako da će metod biti pozvan kada instanca Enemi (ovo) pogodi instancu Orb.
Uklonite stare kodove zbog kojih se neprijatelj rotirao kada je kugla pogođena i zbog kojih je neprijatelj odskočio sa ivice sveta.

5.8 Primeni metod Orb.hit(Neprijatelj)

Implementirati telo metode Orb.hit(Enemi) u odnosu na analizu urađenu u zadatku 5.2. Testirajte svoju aplikaciju.

5.9 Dodajte klase Bullet i Tover

Dodajte klase Bullet i Tover. Koristite iste principe kao u zadatku 3.3.

5.10 Razgovarajte o tome kako instanca klase Bullet treba da se kreće i šta treba da se desi kada stigne do instance klase Neprijatelj ili ivice arene.

Metak treba da se kreće dok ne stigne do neprijatelja ili do ivice sveta. Metak ne menja pravac kretanja. Brzinom metka se može upravljati korišćenjem istog mehanizma kao u zadatku 4.6.

5.11 mplementirati kretanje instance klase Bullet

Primena znanja obuhvaćena zadacima 3.2, 3.4 i 4.6. Postavite komentar dokumentacije u kod, gde bi trebalo da se desi interakcija sa instancom klase Enemi.

5.12 Razgovarajte o tome kako će instanca klase Tover pucati u instancu klase Bullet

Imajte na umu da ta instanca Tover-a ne bi trebalo da se aktivira u svakom pozivu metode act(). Inspirišite se mehanizmom koji se koristi u 4.6. Odvojite relevantne korake u metode klase Tover.

5.13 Razgovarajte o tome kako instanca klase Tover treba da komunicira sa relevantnim objektima koristeći poruke prilikom snimanja

Koristite analogne principe kao u 5.2.

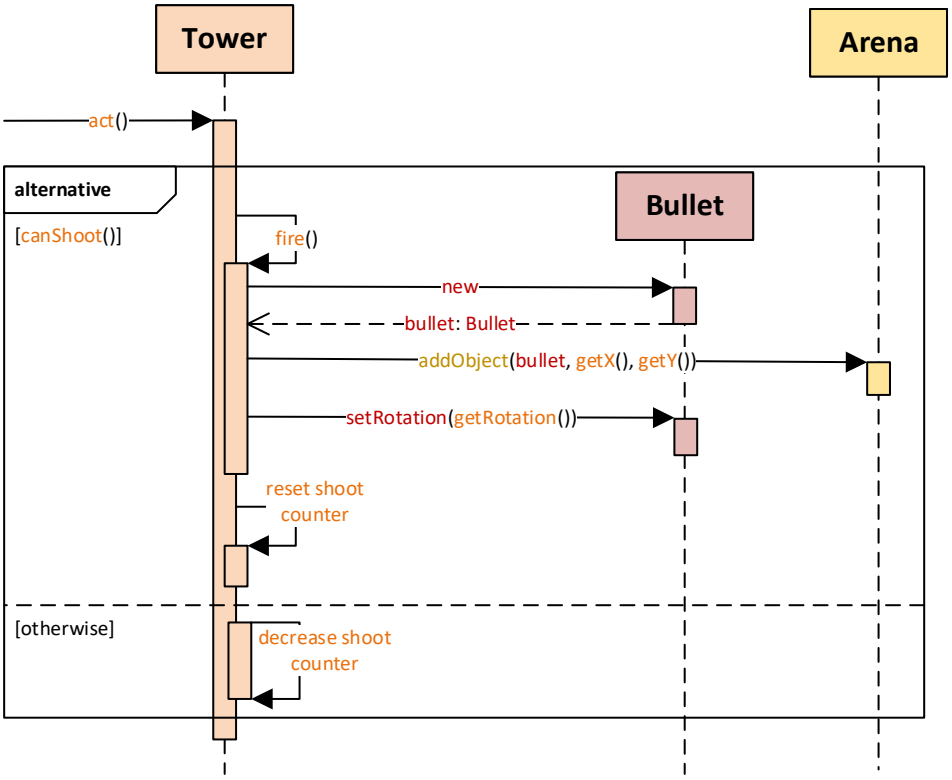


Figure 1: UML sequence diagram of instance of class Tower interacting with other objects when creating instances of class Bullet

5.14 Sprovedite snimanje instance klase Tover

Pratite rezultat zadatka 5.13:

• Prvo pripremite potrebne atribute i konstruktor, zatim

• kreirajte metode boolean Tover.canShoot() i void Tover.fire() (prva neka vrati false, druga neka ne radi ništa, tako da ih možete koristiti u metodi act()), a zatim

• implementirati telo metode act().

Implementirajte metod canShoot() da biste vratili tačno ako brojač snimanja dostigne 0.

Implementirajte metod fire() na sledeći način:

• pozvati konstruktor klase Bullet i sačuvati kreiranu instancu u lokalnoj promenljivoj (Bullet bullet),

• dodati kreirani metak u arenu na istim koordinatama kao instanca klase Tover (ovo) i

• postavite istu rotaciju metka kao streljački toranj.

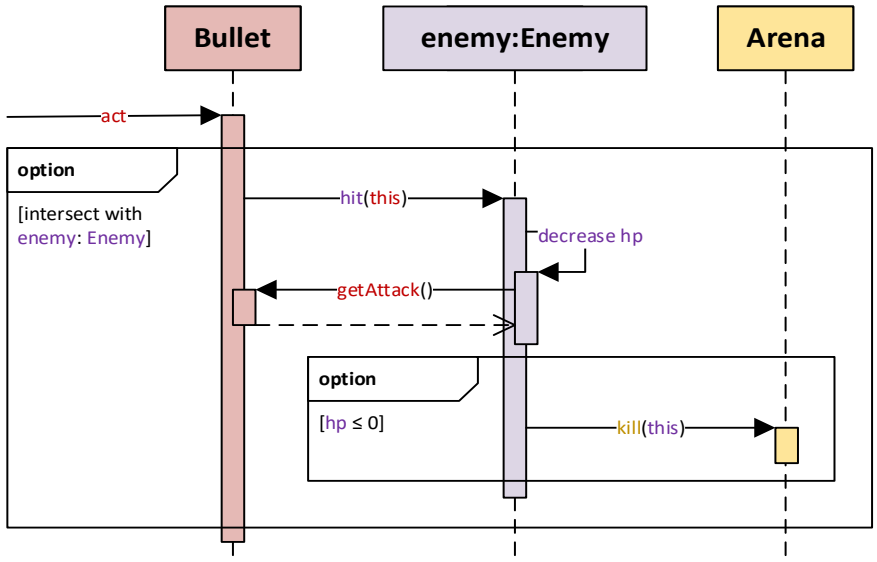
Testirajte svoje rešenje.

5.15 Kule u Areni

Preopteretiti konstruktor klase Tover tako da prihvata parametar i int rotaciju (analogno 4.8). Ažurirajte konstruktor klase Arena da biste postavili instance klase Tover po želji. Koristite odgovarajući konstruktor klase Tover.

5.16 Razgovarajte o tome kako instanca klase Bullet treba da komunicira sa relevantnim objektima koristeći poruke

Koristite analogne principe kao u



Slika 9: UML dijagram sekvence instance klase Bullet u interakciji sa drugim objektima kada udari instancu klase Enemi

5.17 Implementirajte instancu klase Metak koji pogađa instancu klase Neprijatelj

Pratite rezultat zadatka 5.17:

• prvo pripremi atribut i metode (analogno zadacima 5.3, 5.4, 5.5 i 5.6),

• zatim pozovite metod Enemi.hit(Bullet) iz instance klase Bullet (analogno 5.7) gde je komentar ostavljen iz 5.11 i,

• na kraju, implementirajte metod Enemi.hit(Bullet) (analogno 5.8).

Testirajte svoje rešenje.

5.18 Pojava neprijatelja i kraj igre

Koristite metod Arena.act() da pozovete stvaranje neprijatelja. Pravilno primenite kašnjenje između pojavljivanja neprijatelja. Spavning proces (kreirajte instancu klase Enemi, dodelite njena svojstva, dodajte je u arenu) implementirajte u metodu Arena.spavn(). Broj kreiranih instanci klase Neprijatelj u atributu klase Arena (inicijalizovano na 0, povećava se kada se pojavi, smanjuje kada se ubije). Izmeni metod Arena.kill(Enemi) – ako je poslednji neprijatelj ubijen, igrač je pobedio u igri – zaustavite Greenfoot-a i napišite poruku na ekranu.

5.19 Nasleđe

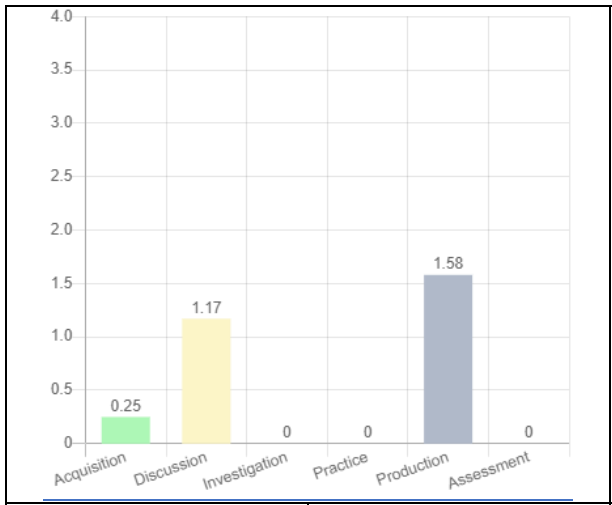
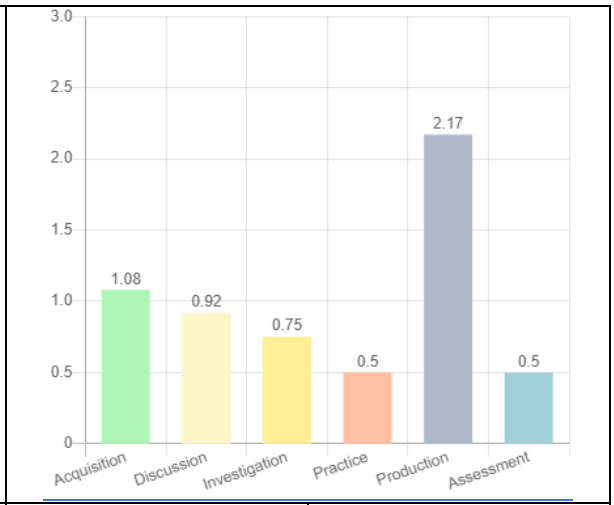
Ova tema uvodi varijabilnost u projekat putem nasleđivanja. Uvodimo princip zamene Liskova da pokažemo prednosti OOP-a. Snažno predlažemo da pustite učenicima da eksperimentišu i osmisle prilagođene podklase neprijatelja i arena.

Pošto će oni imati zajednički interfejs, biće lako sve spojiti. Slično kao kod teme 4, može biti napravljeno mnogo domaćih zadataka.

Kao što je ranije pomenuto, može se zaključiti da je projekat završen nakon prethodne teme. Stoga, ako je potrebno, nastavnik može da prilagodi ovu temu kako bi pokazao prednosti nasleđa i povezane sa njim univerzalnost samo na ovde predloženim hijerarhijama klase Arena ili Enemi. Ovo će dovesti do smanjenja sati povezanih sa ovom temom.

Tabela 9 sumira poređenje opterećenja teme Nasleđivanje između projekata Bomberman i Tover defense. Sugestija u eksperimentisanju je projektovana u više TLA-ova za istraživanje, praksu i proizvodnju. Više teorije je pokriveno u ovoj temi sa fokusom na princip zamene Liskova.

Tabela 9: Poređenje opterećenja teme Nasleđivanje između projekata Bomberman i Tover defense

			
Bomberman	3h 0min	Tower defense	5h 55min

6.1 Identifikujte zajednička svojstva klasa Orb i Direction

Primeri klasa Orb i Direction ne deluju tokom života. Oni samo reaguju na poruke. Možemo uvesti zajedničkog pretka koji će metod act() držati praznim i učiniti podklase transparentnijim.

6.2 Dodajte klasu PassiveActor kao pretka klasa Orb i Direction

Kreirajte novu klasu PassiveActor. Promenite kodove klasa Orb i Direction da budu potomci PassiveActor. Uklonite metod act() iz klasa Orb i Direction – nasleđen je od PassiveActor.

6.3 činite klasu PassiveActor apstraktnom

6.4 Identifikujte zajednička svojstva klasa Bullet i Enemi

Slučajevi klasa Bullet i Enemi se ponašaju slično tokom života. Kreću se na isti način i nakon toga reaguju na okolinu. Možemo uvesti zajedničkog pretka, koji će implementirati metod act() da se kreće na isti način i da se podklase fokusiraju na njihovu specifičnu svrhu.

6.5 Dodajte apstraktnu klasu MovingActor kao pretka klasa Bullet i Enemi

Koristite sličan pristup kao u 6.2.

6.6 Identifikujte atribute klasa Bullet i Enemi potrebne za kretanje

Istražite metod act() odgovarajućih klasa. Identifikujte atribute moveDelai i nektMoveCounter. Zapazite, taj kod metode act() odgovoran za kretanje je isti.

6.7 Premesti kod odgovoran za kretanje u klasu MovingActor

• Premestite atribute identifikovane u 6.6 iz podklasa Bullet i Enemi u MovingActor (uklonite ih iz podklasa).
• Dodajte parametarski konstruktor u klasu MovingActor da biste inicijalizovali ove atribute.
• Pozovite roditeljski konstruktor sa odgovarajućim parametrima iz podklasa Bullet i Enemi.
• Premestiti kod odgovoran za kretanje u metodi act() podklasa Bullet i Enemi u MovingActor (ukloniti kod iz podklasa, zadržati ostatak metode).
• Pozovite nadređenu verziju metode act() kao prvu liniju metode act() u podklasama Bullet i Enemi.

6.8 Kreirajte prilagođene neprijatelje

• Dodajte podklase klase Enemi koje će predstavljati različite neprijatelje (npr. Žaba i Pauk). Uverite se da slike ne prelaze veličinu ćelije.
• Dodajte u klase konstruktor bez parametara, koji će pozvati roditeljski (neprijateljski) konstruktor sa parametrima specifičnim za svaku vrstu neprijatelja.
• Uklonite metod act() (ili dodajte poziv super.add()).

6.9 Postavite prilagođene neprijatelje

Metod ažuriranja Arena.spavn(). Napravite instancu žabe ili pauka i sačuvajte je u promenljivoj tipa Enemi. Koristite bilo koju vrstu odluke da odlučite koju instancu treba kreirati (može biti slučajna, može se precizno brojati, itd.). Uverite se da nijedan drugi kod u aplikaciji ne sme da se menja.

6.10 Razgovarajte o hijerarhiji Arenasa

Diskutujte i dizajnirajte hijerarhiju časova u Areni. Podklase Arene će biti odgovorne za prilagođeni raspored – položaj instanci Orb i Direction, veličina arene. Ovi zadaci će se obavljati u konstruktoru potklase. Šta će biti potrebno da se prođu parametri konstruktora roditeljske klase (Arena)? Imajte na umu da će se sve ostalo (mriješćenje, ponovno pojavljivanje i ubijanje neprijatelja) izvoditi u klasi Arene.
Trebalo bi da identifikujete potrebu za postavljanjem i čuvanjem pozicije i rotacije mrijesta. Ovo će biti urađeno korišćenjem atributa i relevantnih parametara konstruktora. Štaviše, konstruktor treba da prihvati i dimenzije površine.

6.11 Napravite univerzalnu arenu

Uvedite atribute int spavnPositionKs , int spavnPositionI i int spavnRotation i koristite ih u metodama spavn() i respavn(Enemi). Dodajte parametre u konstruktor klase Arena da biste ih inicijalizovali.
Dodajte još dva parametra u konstruktor klase Arena – int širinu i int visinu. Prosledite ove parametre roditeljskom konstruktoru.
Primetite da Greenfoot ne može automatski da napravi Arenu, pošto su joj potrebni parametri za konstruktor. Neka bude apstraktno.

6.12 Napravite DemoArenu

Dodajte podklasu DemoArena klase Arena. Pozovite roditeljskog konstruktora klase DemoArena sa parametrima koji će obezbediti stvaranje arene istih dimenzija i stvaranje neprijatelja na isti način kao i ranije.
Premesti kod koji je odgovoran za raspored uputstava, kugle i kula sa konstruktora Arene na konstruktor DemoArene.

Create instance of **DemoArena** – from context menu of **class DemoArena** select item **new DemoArena()**.

6.13 Napravite prilagođene arene

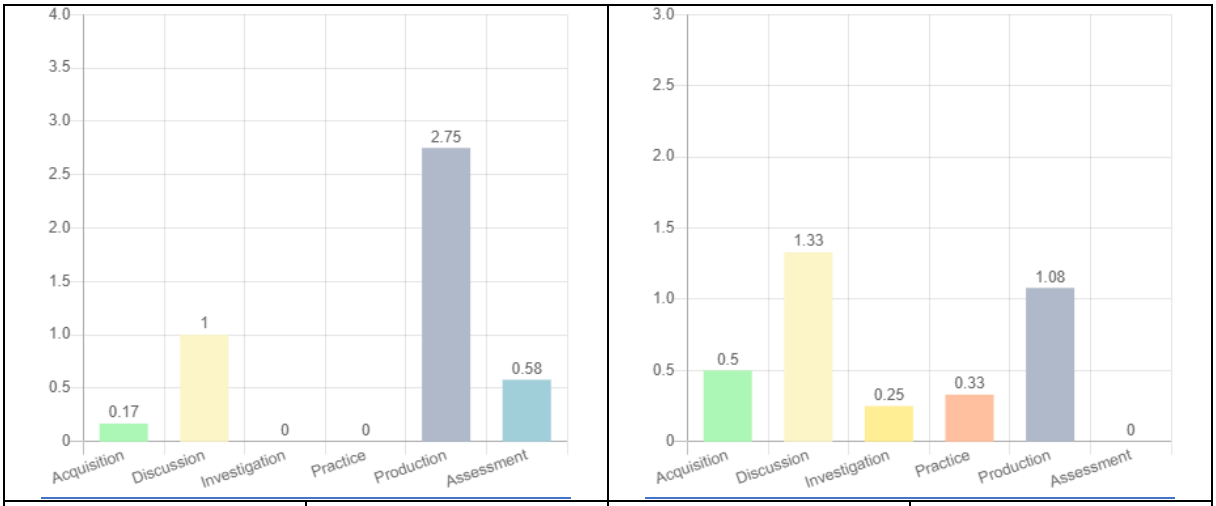
Koristeći sličan pristup kao u 6.12 kreirajte druge inovativne podklase klase Arena. Možete da delite kod sa drugim učenicima u vašoj grupi.

7 Enkapsulacija

Poslednja tema je fokusirana na pravilno korišćenje privatnih metoda i metoda i promenljivih klasa. Upotreba metoda i promenljivih klase može se zameniti (ne-klasnim) atributima i metodama u klasi Arena (u kontekstu našeg projekta postoji samo jedna instanca Arene). Ovo će omogućiti implementaciju zadataka iz ove teme, ali koristeći već poznate koncepte.

Tabela 10 sumira poređenje opterećenja teme Enkapsulacija između projekata Bomberman i Tover defense. Slično prethodnoj temi, proizvodnja je ravnomernije raspoređena među ostalim tipovima TLA. Veća količina TLA-a za akviziciju dolazi od uvođenja metoda i varijabli klase. Kao što je predloženo, postoji mogućnost da se izbegnu ovi koncepti, šta će dovesti do smanjenja tih TLA, šta će dovesti do sličnog dizajna kao kod projekta Bomberman

Table 2: Comparision of workloads of topic Encapsulation between projects Bomberman and Tower defense



Bomberman	4h 30min	Tower defense	3h 30min

7.1 Kreirajte klasu ManualTover

Kreirajte klasu ManualTover kao potomka klase Tover. Pripremite slike ove klase kada su pod kontrolom, a kada ne. Dodajte dva konstruktora sa istim potpisom kao i roditeljski konstruktori i osigurajte pozivanje roditeljskog konstruktora. Neka metod act() pozove roditeljsku verziju sebe.

Dodajte instance ove klase u raspored odabrane Arene.

7.2 Promena upravljanja ručnim tornjem

Dodajte atribut boolean isManualliControlled i inicijalizirajte ga na false. Kreirajte metod void changeControl(boolean) i promenite atribut i promenite sliku u skladu sa tim.

7.3 Pozovite promenu ručne kontrole

Ručno pozovite promenu ručne kontrole izabrane instance klase ManualTover. Posmatrajte promene u unutrašnjem stanju slično kao 1.6.

7.4 Kontrola korisnika procesa

Kreirajte privatni metod void processUserControl(). Prvo otkrijte da li je kliknut na ovoj instanci. Ako jeste, pređite na ručnu kontrolu. Nakon toga implementirajte samu ručnu kontrolu. Testirajte da li je instanca u ručnom režimu i ako jeste, nabavite MouseInfo objekat. Ako je objekat dobijen, okrenite se prema poziciji kursora.

Pozovite pripremljeni metod iz metode act() pre nego što izvršite prosledenje roditelju. Proverite u metodi da li je kliknut na ovoj instanci. Ako je tako, pozovite metod changeControl sa odgovarajućom vrednošću.

Testirajte svoje rešenje tako što ćete ponovo izvršiti 7.3.

7.5 dentifikujte problem sa kontrolom korisnika i predložite rešenje

Identifikujte šta je problematično sa kontrolom korisnika. Kako se ovi problemi mogu rešiti?

Trenutno nije moguće poništiti izbor tornja. Trebalo bi da postoji dokaz za trenutno kontrolisanu instancu, koji će biti deaktiviran kada se izabere neka druga instanca. Dodajte dokaz o ručno kontrolisanom tornju

7.6 Dodajte dokaz o ručno kontrolisanom tornju

Dodajte atribut u klasu ManualTover tipa ManualTover da biste predstavljali referencu na ručno kontrolisanu instancu i inicijalizujte je na nultu vrednost. Proverite unutrašnje stanje klase (iz kontekstnog menija klase ManualTover izaberite stavku menija Inspect). Šta je dodato?

7.7 Promena ručno upravljanog tornja sa centralizovanog mesta

Dodajte metod changeControlledInstance da promenite ručno kontrolisan toranj kao metod klase klase ManualTover. Parametar metode treba da bude referenca na instancu ManualTover-a, koja će se ručno kontrolisati.

Ako se dragocjeno i novo proslijeđene instance razlikuju, pošaljite changeControll poruku sa odgovarajućim parametrima na prethodno, kao i na novo sačuvanu referencu na ManualTover instancu. Ne zaboravite da rukujete nultim referencama i sačuvate nove reference!

Testirajte svoje rešenje. Iz kontekstnog menija klase Tover izaberite stavku menija sa novokreiranim metodom. Za popunjavanje parametra možete koristiti isti princip kao u 5.5.

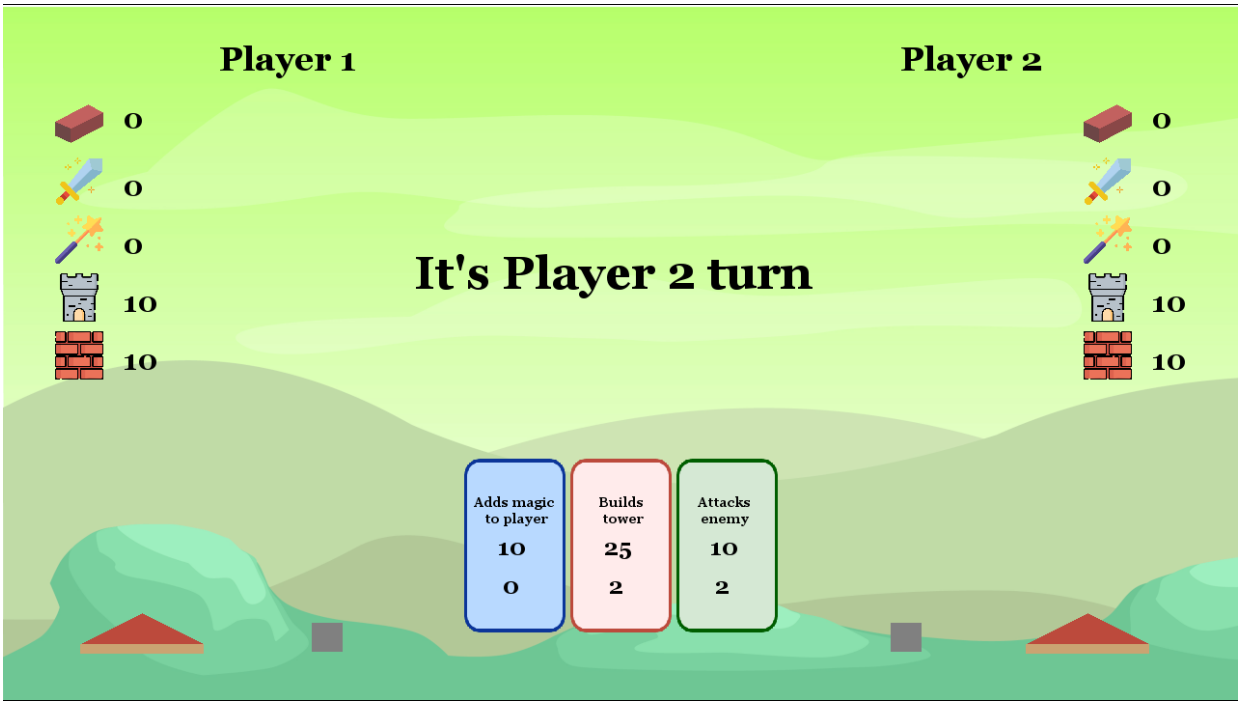
Treba da primetite da se novokreirana metoda klase ne poziva dosledno, instance ManualTover-a zaobilaze dokaze prilikom obrade unosa, što izaziva probleme.

7.8 Pozovite promenu ručno kontrolisanog tornja

Pozovite ManualTover.changeControlledInstance(ManualTover) sa relevantnih mesta. Na kraju, učinite metodu ManualTover.changeControl(boolean) privatnim. Posmatrajte promene u interfejsu instance klase ManualTover slično kao u 1.6.

3.3 Projekt mravi

Ants je kartaška igra za dva igrača. Svaki igrač ima svoju kulu i zid i resurse kao što su cigle, mačevi i magija. Jedan okret igre sastoji se od akcije, gde igra nudi igraču nasumično 3 karte, a on bira jednu. Postoje tri vrste karata – karte za izgradnju, borbene karte i magične karte. Karte za izgradnju se mogu koristiti za povećanje sopstvenog tornja ili zida, borbene karte za napad na neprijateljskog igrača i magične karte za povećanje broja sopstvenih resursa ili krađu neprijateljskih.



Izvorni kodovi su dostupni na:
https://gitlab.kicon.fri.uniza.sk/oop4fun/project-ants

Dizajn učenja je dostupan na:
http://learning-design.eu/en/preview/67aa1d089763d07f29809d42/details

3.3.1 Teme

Projekat Mravi je podeljen na 6 tema:

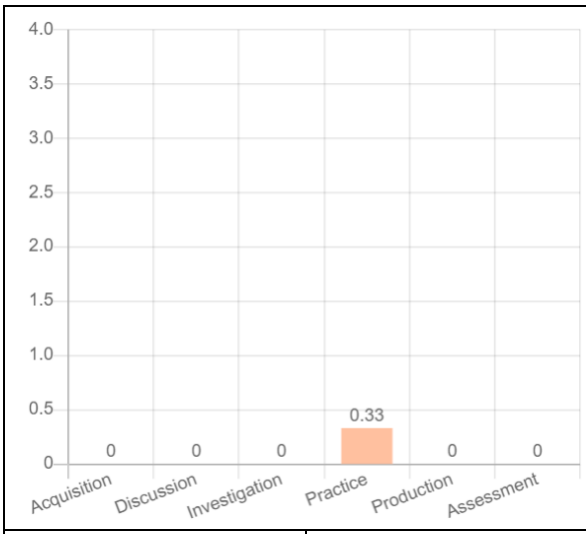
Pokrivene teme svetlosnog OOP-a su:
- klase, objekti, instanca
- metode, prosleđivanje argumenata metodama
- konstruktori

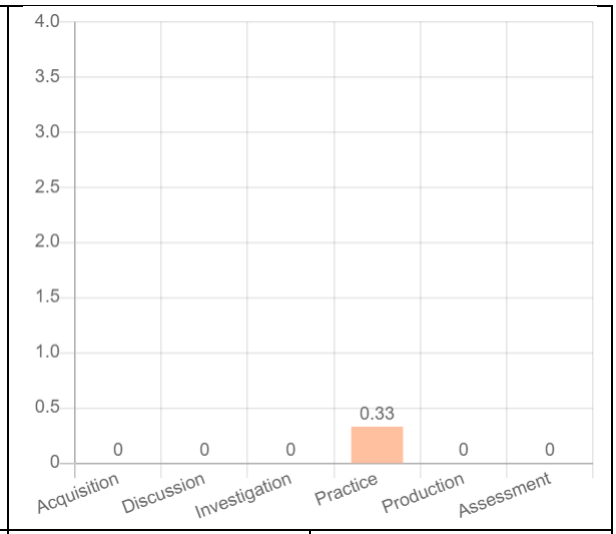
- atributi
- statičke varijable i metode
- inkapsulacija

1. Greenfoot okruženje, Definicija klase, osnovni rad sa klasama

Tema je posvećena kreiranju projekata. Učenici će biti sposobni da kreiraju novi projekat u Greenfoot okruženju, kreiraju klasu (kao podklasu Actor), izaberu pozadinsku sliku, kreiraju instancu kreirane klase i pošalju joj poruku.
Kreirajte novi projekat. Dajte mu pravo ime (npr. Mravi) i sačuvajte ga na odgovarajućoj lokaciji.
Tabela 11 sumira poređenje opterećenja teme 1 između projekata Bomberman i Ants. Ukupno opterećenje oba projekta je isto u ovom delu

Tabela 11: Poređenje opterećenja teme 1 između projekata Bomberman i Ants

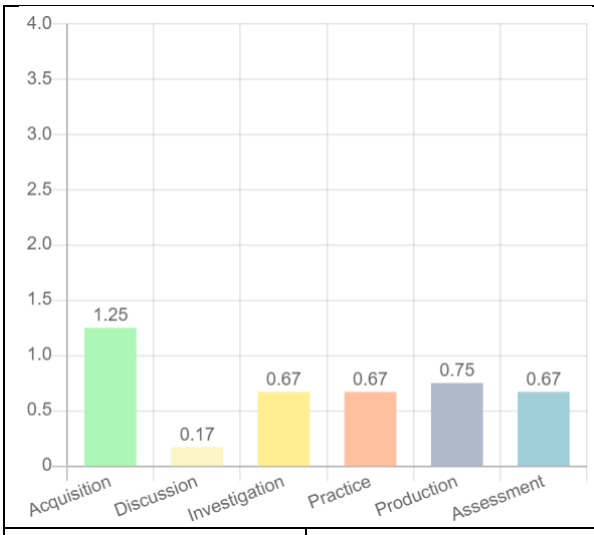


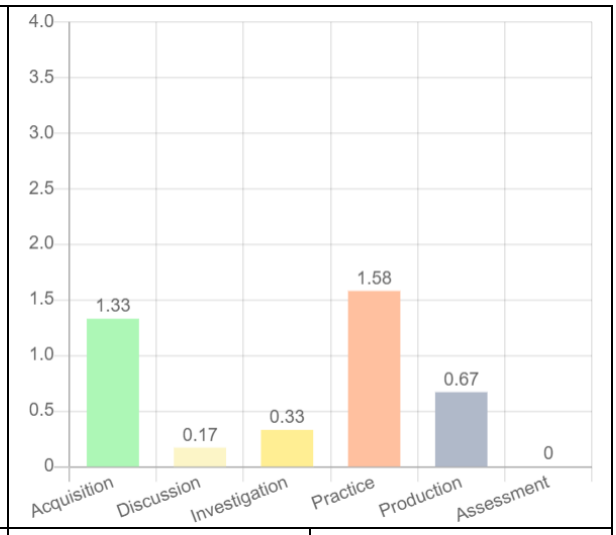


Bomberman	0h 20min	Ants	0h 20min

Tabela 10 sumira poređenje opterećenja teme Definicija klase i osnovnog rada sa klasama između projekata Bomberman i Ants. Ukupno opterećenje oba projekta je slično. Glavna razlika je u praksi i proceni. Međutim, kao što će biti navedeno u sledećem odeljku, neki od zadataka za vežbu mogu se dati i kao procene, što ih može još više uravnotežiti.

Tabela 12: Poređenje opterećenja teme Definicija klase i osnovni rad sa klasama između projekata Bomberman i Ants





Bomberman	4h 10min	Ants	4h 05min

1.1 Uvod u Greenfoot

Kreirajte novi projekat u Greenfoot-u i upoznajte studente sa osnovnim elementima, korisničkim interfejsom itd. Početno stanje spremišta sadrži i sredstva koja možete koristiti u projektu.

1.2 Stvaranje klase Vall

Napravite klasni zid kao dete glumca. Predstavite učenicima koncepte kao što su klase, hijerarhija klasa, instance itd.

1.3 Stvaranje klase Tover

Slično, kao i prethodni zadatak, kreirajte klasu Tover. Možete ga ostaviti učenicima kao individualni zadatak.

1.4 Definisanje atributa/polja klase

Upoznati učenike sa pojmovima polje/atribut, primitivni tipovi itd. Pokušajte da identifikujete polja u našim razredima (navedite učenike posebno do visine) i definišete ih u razredu Vall.

1.5 Dodeljivanje vrednosti atributu/polju

Razgovarajte o vrednostima polja i zadacima i dodelite visini zida vrednost 10.

1.6 Definisanje i dodeljivanje vrednosti atributu/polju za klasu Tover

Kao samostalni rad ostavite učenicima da ponove isto za razred Tover.

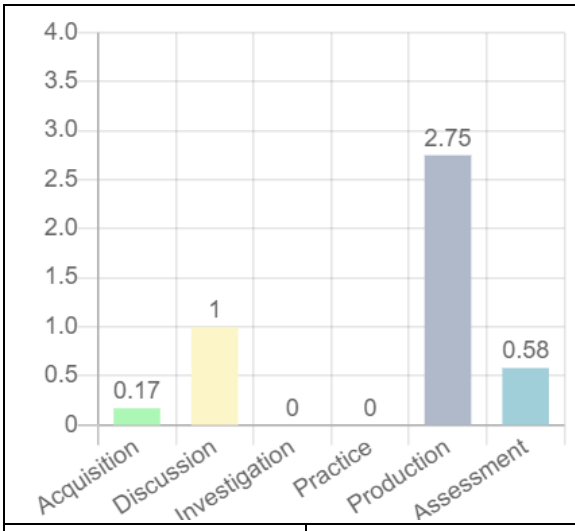
1.7 Konstruktori klasa

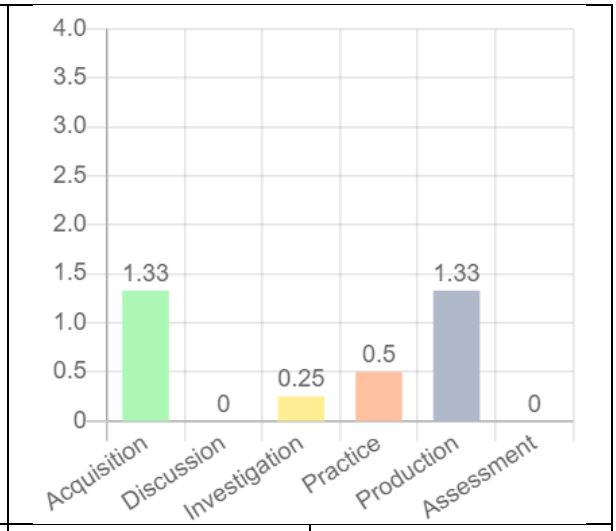
Razgovarajte o instanciranju klase i konstruktorima. Premesti dodelu vrednosti u konstruktor.

2. Enkapsulacija, sastav, metode

U ovom odeljku su dati osnovni principi OOP-a, posebno termini kao što su enkapsulacija, kompozicija i metode. Učenici će naučiti kako i zašto polja/atributi treba da budu inkapsulirani, a ne dati kao javni, kako se objekti sastavljaju sa drugim objektima i kako da kreiraju metode i da ih pozovu. Učenici će kreirati objekat Plaier u ovom odeljku i dati mu polja tipa objekta – zid i kula. Tabela 11 pokazuje razlike u dve slične teme – Enkapsulacija iz Bomberman projekta i Enkapsulacija, sastav i metode u Ants. Kao što se može videti, projekat Ants daje više fokusa na akviziciju, a sa druge strane, Bomberman se više fokusira na produkciju i diskusiju. Ovo je uzrokovano činjenicom da se ova tema u mravima sastoji od više od inkapsulacije, stoga je potrebno i više akvizicije. Ukupno radno opterećenje je jedan sat kraće u Ants nego u Bombermanu, jer su ovi koncepti objašnjeni pliće, ali u narednim odeljcima ćete konsolidovati ovo znanje.

Tabela 13: Poređenje opterećenja teme Enkapsulacija, kompozicija, metode između projekata Ants i slične teme u Bomberman-u - Enkapsulacija





Bomberman	4h 30min	Ants	3h 25min

2.1 Definisanje metoda

Razgovarajte o inkapsulaciji, objasnite učenicima zašto nije dobra praksa da se na primer visina zida učini javnom svojinom i radije je inkapsulira u getter. Zatim kreirajte metod getHeight() u zidu.

2.2 Definisanje metoda sa parametrima

Objasnite parametre metode i definišite metodu povećanje visine u zidu, koja će povećati visinu zida za određeni broj.

2.3 Ponavljanje za klasu Toranj

Kao samostalni rad možete dati učenicima zadatak da ga ponove i za razred Toranj. Ovaj zadatak nema povezano urezivanje jer je posao prilično jednostavan. U sledećem urezivanju takođe postoje izmene za ovaj zadatak, tako da možete da ga uporedite ako niste sigurni u rezultat.

2.4 Kompozicija objekta

Objasniti učenicima šta je kompozicija i zašto je potrebno imati tipove objekata kao polja. Pokušajte da identifikujete takve tipove za svakog igrača u ovoj igri. Zatim kreirajte objekat Plaier i dajte mu polja/atribute Vall i Tover.

2.5 *Kreiranje instance objekta*

Objasnite konstruktore i kreirajte instance zida i kule u konstruktoru Plaier.

2.6 Pozivanje metoda instance

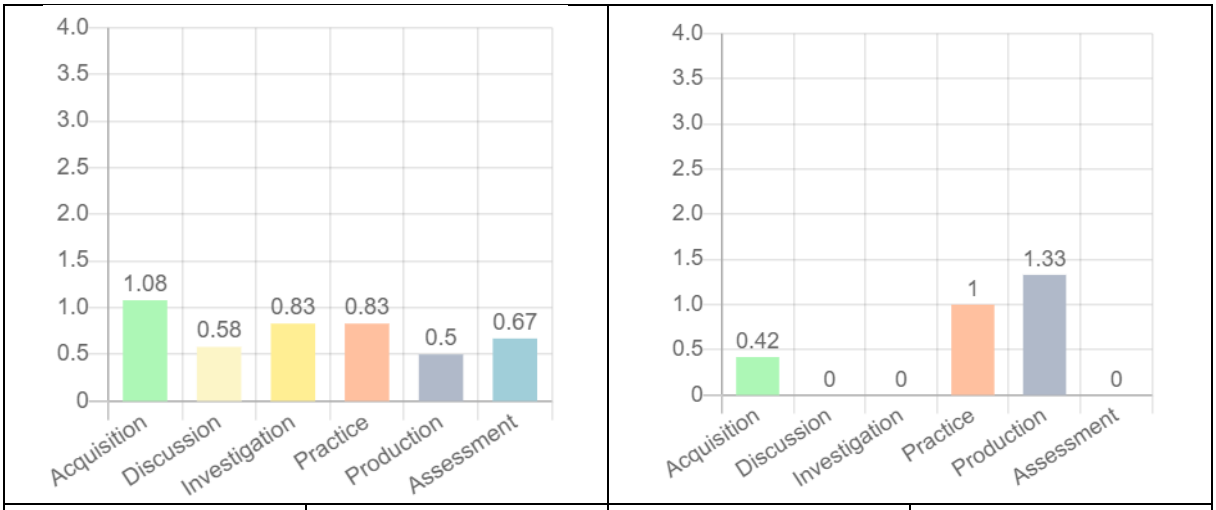
Objasnite učenicima kako možete pozvati metode kreiranih instanci. Zatim pokušajte da ih enkapsulirate tako da možete da ih pozovete izvan Plaiera preko instance igrača. Kreirajte metode getVallHeight, getToverHeight i PovećajteVallHeight, povećajteToverHeight u Plaier objektu.

3 Konstruktori, složeniji pozivi metoda (rad sa grafikom u Greenfootu)

Ovaj odeljak je fokusiran na pozive metoda koji koriste pozivanje Greenfoot objekata za crtanje naših instanci. Učenici će crtati primere zida, tornja i igrača.

U tabeli 12 prikazano je poređenje ove teme u projektu Ants i najsličnije teme u projektu Bomberman. Imajte na umu da su ove teme malo drugačije, jer naša prikazuje osnovni rad sa konstruktorima i uvodi grafiku u Greenfoot, au projektu Bomberman je više fokusirana na algoritam. Stoga se i zbog ovoga mogu ukazati na razlike.

Tabela 14: Poređenje opterećenja teme Konstruktori, složeniji pozivi metoda (rad sa grafikom u Greenfootu) između projekata Ants i slične teme u projektu Bomberman, koja je obrađena u temi Algoritam



Bomberman	4h 30min	Ants	2h 45min

3.1 Crtanje objekata u Greenfoot – Vall

Upoznajte učenike sa konstantama u Javi – definišite vallSizeKs i vallSizeI kao statičke konačne atribute (konstante) sa vrednostima 32 i 3. Zatim implementirajte funkciju drav u Vall, gde se kreira nova slika zadate veličine i popunjava kao pravougaonik.

3.2 Crtanje objekata u Greenfoot – Tover

Isto se ponavlja i za toranj, ali je komplikovanije jer se toranj sastoji i od krova koji je realizovan kao poligon. Poligon zahteva niz tačaka. Ako želite, možete učenicima dati kratko objašnjenje, iako nizovi nisu deo ovog odeljka.

3.3 Definisanje drugih svojstava Plaiera

Pokušajte da identifikujete druga svojstva igrača – posebno broj cigli, mačeva i magije i ime igrača. Zatim implementirajte ova svojstva u Plaier i inicijalizujte ih u konstruktoru

3.4 Draving Plaier

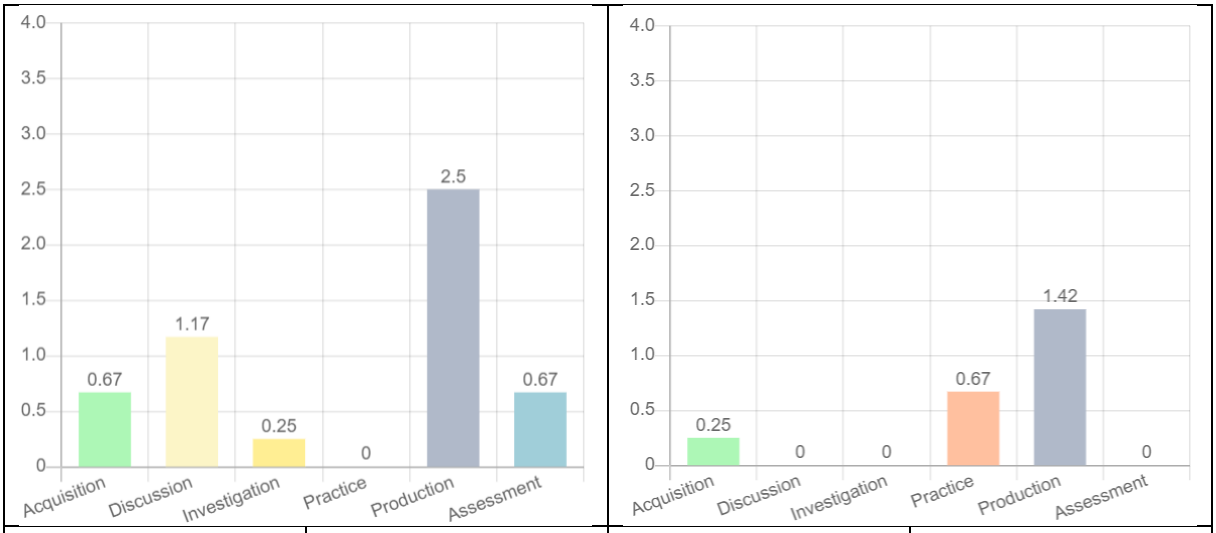
Poslednji zadatak u ovom delu je crtanje igrača. Morate da nacrtate ikone svakog resursa i broj takvog resursa, kao i ime igrača.

4 Grananje, uslovno izvršenje

Ovaj odeljak se fokusira na grananje programa i uslovno izvršavanje delova naše igre. Postoji nekoliko slučajeva kada je to neophodno u ovoj igri, kao što je crtanje prvog igrača u levom delu ekrana, a drugog u desnom itd.

Tabela 13 pokazuje razlike između slične teme u Bomberman i Ants. Kao što se može videti, projekat Bomberman daje veći naglasak na diskusiju kao projekat Ants. Takođe, ukupno radno opterećenje u satima je otprilike upola manje od Bombermana. Ovo je uzrokovano činjenicom da je grananje u ovom projektu podeljeno na više sekcija – ovaj odeljak je više uvod i osnovna upotreba grananja.

Tabela 15: Poređenje opterećenja teme Grananje, uslovno izvršenje između projekata Bomberman i Ants



Bomberman	5h 15min	Ants	2h 20min

4.1 Igra stvaranja predmeta

Pre nego što počnemo da stavljamo logiku grananja u naš kod, moramo da kreiramo objekat Game koji će držati oba igrača i, u budućnosti, upravljati logikom igre, prebacivanjem okreta, izvršavanjem kartice itd. Za sada ćemo tamo staviti svojstva za dva igrača.

4.2 Grananje, uslovljavanje izvršenja koda – igrači su prikazani na odgovarajućim stranama plana igre

Sada bi trebalo da uvedemo novi parametar za naš Plaier – informaciju da li plejer treba da bude prikazan na levoj ili desnoj strani ekrana. Koristićemo ove informacije da bismo podesili odgovarajuća svojstva igrača – položaj zida, kule, „huda“ i imena. Ovaj pomak treba zatim dodati funkciji ponovnog crtanja.

4.3 Dodavanje instanci u svet

U sledećem zadatku, početno izvlačenje igrača se kreira u igri i instanca igre se kreira u svetu.

4.4 Dodavanje instanci u svet 2

Da bi igrač bio ispravno nacrtan u igri, moramo dodati zid i kulu u svet i pozvati funkciju ponovnog crtanja u metodi akta. Ovde možete objasniti izvođenje petlje igre (metoda akta).

4.5 Uslovljavanje koda za izvršavanje samo jednom

Kada pokušate da pokrenete igru nakon poslednjeg zadatka, možete naići na problem – objekti se dodaju svetu više puta u sekundi. Učenicima možete dati zadatke da reše ovaj problem ili ga poprave

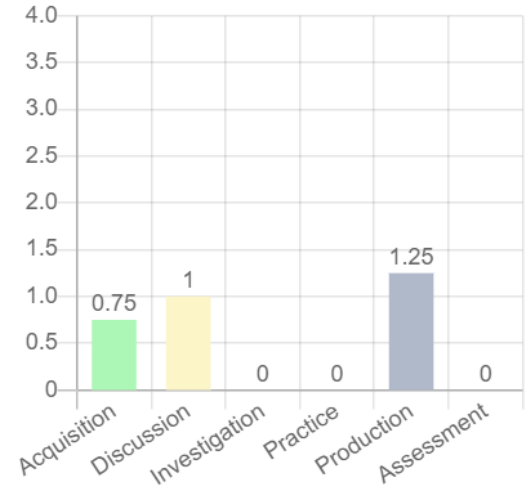
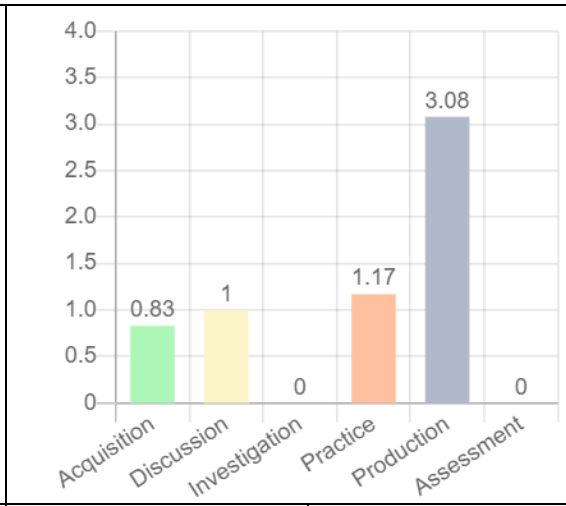
sa njima. Jedno od rešenja je da se uvede novi logički atribut, koji bi čuvao informacije da li je početni objekat dodat u svet izvršen ili ne i nakon prvog izvršavanja act metode ovo svojstvo se postavlja na true.

5 Algoritam, nabrajanja, nizovi

Ovaj odeljak govori o sledećim konceptima, kao što su algoritam, enumeracije, nizovi i petlja preko niza elemenata. U ovoj sekciji učenici će implementirati kartice koje će se koristiti za igru.

Tabela 14 sadrži poređenje sličnih tema u projektima Ants i Bomberman. Kao što se može videti, projekat Ants daje više fokusa na proizvodnju i praksu i nešto više na nabavku. Takođe, ukupno opterećenje je oko dva puta veće nego u projektu Bomberman. Ovo je uzrokovano činjenicom da se ova tema sastoji od više od rada sa listama, već uvodi i koncepte kao što su nabrajanja itd.

Tabela 16: Poređenje opterećenja teme Algoritam, enumeracije, nizovi u projektu Ants i slična tema u projektima Bomberman - List

			
Bomberman	3h 00min	Ants	6h 05min

5.1 Implementacija klase kartice

Prvo, možemo razgovarati sa učenicima o tome kako bi konačna igra funkcionisala i dizajnirali objekat kartice. Trebalo bi da dođete do rešenja koje će sadržati informacije o tipu kartice, njenim zahtevima, efektu i nekom opisu. Tada možete kreirati takav objekat kao dete Actor.

5.2 Enums

U prethodnom zadatku ste kreirali tip kartice kao polje u objektu kartice. Razgovarajte sa učenicima koji tip ovo polje treba da bude – string, int, itd. i možete im uvesti enume. Kreirajte CardTipe enum sa njima i navedite njegove pojedinačne vrednosti.

5.3 Branching – switch

Sada treba da primenimo crtanje kartice. Ovo se zasniva na tipu kartice, da bi se kartice vizuelno podelile. Možete pokazati učenicima kako bi se to uradilo korišćenjem if i uporedite sa prekidačem. Postoje tri vrste karata – karte za izgradnju, karte za napad i magične karte. Na osnovu ove kategorije bira se pozadina. U slučaju da ćemo to uraditi koristeći if naredbu, kod bi trebao izgledati ovako:

```
if(type == BuildTower || type == BuildWall || type == IncreaseBricks) {
        background = new GreenfootImage("building-card.png");
```

```
} else if(type == IncreaseSwords || type == Attack)
...
```

Ovo se može zameniti naredbom svitch. Java svitch naredba funkcioniše na način da će se izvršiti ta određena grana, ali neće prestati da izvršava druge grane osim ako ne pozovete naredbu break. U našem slučaju ovo nam omogućava da spojimo više grana i napišemo našu izjavu o dodeli samo za poslednji tip kartice takve kategorije. Dakle, ovaj kod:

```
switch (type) {
	case BuildTower:
		background = new GreenfootImage(“building-card.png”);
		break;
	case BuildWall:
		background = new GreenfootImage(“building-card.png”);
		break;
	case IncreaseBricks:
		background = new GreenfootImage(“building-card.png”);
		break
	...
}
```

može biti napisan i na način koji koristimo u ovom zadatku:

```
switch (type) {
	case BuildTower:
	case BuildWall:
	case IncreaseBricks:
		background = new GreenfootImage(“building-card.png”);
		break
	...
}
```

5.4 Niz

Objekt igre treba da drži karte. Ovo se može uraditi uvođenjem tri polja tipa kartice (možete i da pružite ruku – tj. broj kartica koje igra nudi igraču u jednom potezu – na više njih). Možete im objasniti da bi bilo nemoguće sačuvati još više kartica, kada odlučimo da još više pružimo ruku i objasnite učenicima koncept nizova. Takođe bi trebalo da kreirate instancu niza.

5.5 Pojednostavljivanje instanciranja kartica – CardFactori

Prilikom kreiranja novih kartica, potrebno je dati mnogo informacija. Možete pokušati da pronađete rešenje za ovaj problem. Jedno od rešenja je kreiranje CardFactori koji će držati instance za svaku

karticu i implementirati metod kloniranja na kartici. Stoga se nove kartice mogu kreirati korišćenjem ove CardFactori i kloniranjem postojećih.

5.6 Random – Instanciranje slučajne kartice

Nakon implementacije CardFactori-a, trebalo bi da se implementira i metoda kloniranja, kao što je već rečeno. CardFactori bi takođe trebalo da bude u mogućnosti da daje slučajeve slučajne kartice. Možete objasniti generator slučajnih brojeva i metod implementacije, koji će klonirati nasumične kartice, kao i nasumične osnovne kartice (osnovne karte su karte bez troškova - ovo je implementirano tako da možemo garantovati da igrač uvek može da igra barem jednu od ponuđenih karata).

5.7 Petlja preko niza

Sada moramo da kreiramo instancu CardFactori-a u igri i implementiramo generisanje kartica i njihovo brisanje (uklanjanje iz sveta) - kada objašnjavate ovo, možete uvesti neki oblik petlje.

5.8 Crtanje igre

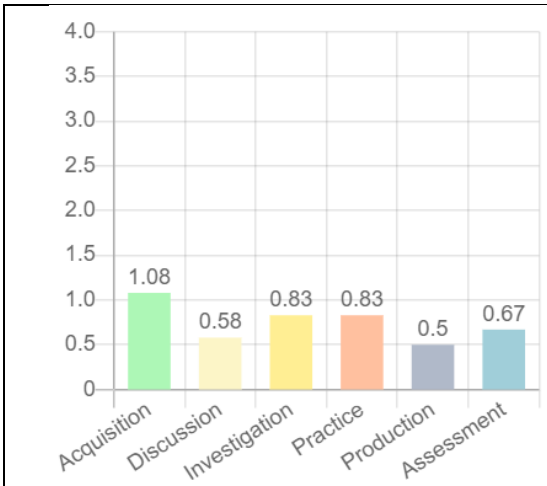
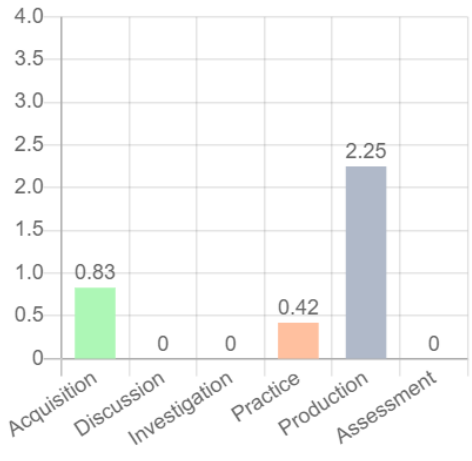
Poslednji zadatak u ovom odeljku je implementacija crteža cele Igre. Tokom ovoga, takođe treba da pripremimo naše karte koristeći prethodno kreiranu metodu i takođe uvedemo informacije da li je prvi igrač aktivan. Izvlačenje igre treba da se sastoji od izvlačenja svih karata i informacija o tome koji igrač je trenutno na potezu

6 Rukovanje korisničkim unosom, logika igre

Ovaj odeljak se fokusira na rukovanje korisničkim unosom – kako da dobijete, na primer, ime igrača od korisnika, kako da rukujete klikom na kartice i kako da završite logiku igre. U ovom odeljku su takođe predstavljeni neki napredni koncepti (singleton).

U tabeli 15 prikazano je poređenje ove teme sa najsličnijom temom u projektu Bomberman – Algoritam. Napominjemo da je ovo poslednja tema ovog projekta, mnogi koncepti uvedeni u projekat Bomberman su već poznati studentima u ovom trenutku ovog projekta. Stoga je znatno manje fokusa na istragu i diskusiju nego u projektu Bomberman i mnogo više produkcije.

Tabela 17: Poređenje opterećenja teme Rukovanje korisničkim unosom, Logika igre u projektu Ants i slična tema u projektu Bomberman - Algoritam

<table>
<tr>
<td> 
</td>
<td>Bomberman</td>
<td>4h 10min</td>
<td> 
</td>
<td>Ants</td>
<td>3h 30min</td>
</tr>
</table>

6.1 Unesite imena od korisnika

U ovom zadatku treba da objasnite učenicima prozore dijaloga (Greenfoot metod pitanja) i podesite imena igrača u skladu sa ovim unosima.

6.2 Statička instanca klase – Igra kao singleton

Objekti igre treba da se konstruišu tačno jednom – o ovom problemu možete razgovarati sa učenicima, a možete dati rešenje u obliku singletona – statičke instance Game objekta i privatnog konstruktora. Ovo je neophodno zato što kada igrač koristi karticu, treba da postoji referenca na igru kako bi se klik na nju mogao pravilno rukovati, o čemu će biti reči u sledećem delu. Drugi način da se to implementira (bez singletona) je da se obezbedi instanca igre konstruktoru kartice i CardFactori.

6.3 Upravljajte unosom kliknite na karticu

Klikom mišem se upravlja pozivanjem Greenfoot.mouseClicked metode u akciji. Kada kliknete, trebalo bi da pozovete metod useCard of Game i pošaljete referencu sebi (ovo).

6.4 Implementacija gettera kartice i igrača

Pre implementacije ostatka logike za igru, potrebno nam je još nekoliko gettera i setera za igrača i karte. Dakle, trebalo bi da ih implementiramo (ovo ne bi trebalo da predstavlja problem za studente jer je to ranije rađeno).

6.5 Implementacija metode podrške za Plaier i popravku u svetu

Pošto je igra sada singleton, moramo je dodati u svet u klasi MiVorld. Sledeći zadatak je da se implementira pomoćni metod za igrača koji će biti korišćen da dobije određenu količinu štete. Igrač treba da smanji zid ili kulu u skladu sa tim.

6.6 Implementacija logike igre

Na kraju, moramo da implementiramo logiku igre – metod okretanja, koji će se sastojati od sledećih koraka:

1. Proverite da li je jedan igrač pobedio – to znači da bilo koji toranj igrača dostigne visinu od 100 ili padne ispod vrednosti 0. Ako je jedan od ovih uslova ispunjen, prikazaćemo pobednički ekran i izaći iz petlje igre – call return izjavu.

2. Podesite aktivni plejer na drugi – samo okrenite vrednost isPlaier1Active atributa.

3. Pripremite karte za sledećeg igrača – pošto smo već napravili metod readiCards, sve što treba da uradimo u ovom koraku je da ga pozovemo.

4. Upravljajte igračima da se okreću – posebno klikom na karte. Kako sama kartica može da sluša klikove na nju, sve što treba da uradimo je da pozovemo igru izvlačenja, koja izvlači pripremljene karte.

5. Ponovno crtanje igrača – ovo se radi korišćenjem metode ponovnog crtanja za svakog igrača.

Zatim moramo da implementiramo metod useCard u igri koristeći prekidač. Ovaj prekidač treba da sadrži granu za svaki tip kartice i da upravlja njenim izvršavanjem. Postoji 7 tipova kartica: BuildTover, BuildVall, IncreaseBricks, IncreaseSvords, Attack, IncreaseMagic i StealBricks. Hajde da pogledamo prvi tip kartice - BuildTover. U ovom slučaju moramo da proverimo da li igrač može da igra ovu kartu (tj. broj njegovih kockica je veći ili jednak zahtevima za kartice), povećamo visinu tornja i smanjimo broj pogodaka cigli prema zahtevima kartice. Dakle, kod može izgledati ovako:

```
if (activePlayer.getBricksNumber() >= card.getRequirements())
{
        activePlayer.increaseTowerHeight(card.getEffect());
    activePlayer.setBricksNumber(
                activePlayer.getBricksNumber() - card.getRequirements()
        );
}
```

Druge vrste kartica su slične:

- **BuildVall** – moramo da proverimo **bricksNumber** igrača i povećaćemo **Wall height**.

- **IncreaseBricks** – ne moramo ništa da proveravamo i povećaćemo **bricksNumber**

- **IncreaseSwords** – ne moramo ništa da proveravamo i povećaćemo **swordsNumber**

- **Attack** – moramo da proverimo da li ima igrača **swordsNumber** i da pozovemo **receiveDamage** neaktivnog igrača

- **IncreaseMagic** – ne moramo ništa da proveravamo i povećaćemo **magicNumber**

- **StealBricks** – moramo da proverimo **magicNumber**, smanjimo **bricksNumber** neaktivnih igrača i povećamo **bricksNumber** aktivnih igrača.

Moguće je kreirati i druge tipove kartica – to je za maštu učenika. Ovo su neke osnovne.

Konačno, moramo da pozovemo metod okretanja nakon igranja karte da bismo obezbedili logiku igre.

Kao poslednji korak, trebalo bi da primenimo pobednički ekran. Ovo je kao i drugi crteži u ovom projektu, tako da je na vama da ga kreirate sa učenicima ili da im ga ostavite.

Da to završimo, postoje i neke popravke u **Player** i **Tower** radi čišćenja koda.

4. Literatura:

[1] „Git,“ 1 10 2023. [Online]. Available: https://git-scm.com. [Cit. 1 10 2023].

[2] „GIT, SVN, mercurial – Google Trends,“ 2 10 2023. [Online]. Available: https://trends.google.com/trends/explore?cat=5&date=today%205-y&q=GIT,SVN,mercurial&hl=sk. [Cit. 2 10 2023].

[3] „GitHub: Let’s build from here · GitHub,“ 1 10 2023. [Online]. Available: https://github.com. [Cit. 1 10 2023].

[4] „The DevSecOps Platform | GitLab,“ 1 10 2023. [Online]. Available: https://about.gitlab.com. [Cit. 1 10 2023].

5. Prilozi

5.1 Izvoz dizajna učenja za projekat Bomberman

See file LD_Bomberman.pdf

5.2 Izvoz dizajna učenja za projekat Tover defense

See file LD_Tower_defense.pdf

5.3 Izvoz dizajna učenja za projekat Ants

See file LD_Ants.pdf