Zvonimir Ivković

Zivkov00@fesb.hr

Seminar iz kolegija Računalna grafika (250)

U Visual Studio Code razvojnom okruženju napisana u Javascript programskom jeziku

A close-up of a ring

Description automatically generated with low confidence

Seminar iz kolegija Računalna grafika (250)

Split, Siječanj 2022.

Sadržaj

[1. Uvod 1](#_Toc93264861)

[1.1. Opis Seminara 1](#_Toc93264862)

[1.2. Visual Studio Code 1](#_Toc93264863)

[1.3. JavaScript 2](#_Toc93264864)

[2. Opis igre 5](#_Toc93264865)

[2.1. Funkcionalni zahtjevi 5](#_Toc93264866)

[2.2. Use-case dijagram 5](#_Toc93264867)

[2.3. Stanja igre 6](#_Toc93264868)

[2.4. Slike objekata 6](#_Toc93264869)

[3. Praktični dio seminara 7](#_Toc93264870)

[3.1. Kratak opis igre 7](#_Toc93264871)

[3.2. Strukturna organizacija igre 7](#_Toc93264872)

[3.3. Main.js 7](#_Toc93264873)

[3.4. Display.js 13](#_Toc93264874)

[3.5. Game.js 15](#_Toc93264875)

[3.5.1. Animacija objekata 15](#_Toc93264876)

[3.5.2. Kolizija na temelju pločica 20](#_Toc93264877)

[Popis Literature 29](#_Toc93264878)

# Uvod

U prvom poglavlju predstavit će se opće informacije vezan uz ovi seminar, 2d igru koja će se izradit, te će se ukratko opisati jezik i okruženje koje će se koristiti u svrhu realizacije seminara.

U drugom poglavlju će se opisati igra, njeni funkcionalni zahtjevi, te će se prikazati i opisati sva stanja i objekti korišteni u realizaciji igre.

Treće i zadnje poglavlje sadrži praktični dio igre, odnosno detaljan uvid i opis programskog koda, povezivanje dijelova igre i njeno funkcioniranje.

## Opis Seminara

Za potrebe realizacije ovoga seminara kreirana je igra Rabbit Ttrap namijenjena operacijskom sustavu Windows pomoću Visual Studio Code okruženja. Kod igre je pisan u Javascript programskom jeziku pomoću kojeg su kreirani i međusobno povezani objekti nužni za realizaciju igre. Također pošto se igra pokreće unutar web preglednika, napravljene su potrebne css i html datoteke.

## Visual Studio Code

Visual Studio Code je uređivač izvornog koda koji je napravio Microsoft za Windows, Linux i macOS. Sadrži sljedeće značajke[1]:

* Može se koristiti s raznim programskim jezicima, uključujući Java, JavaScript, Go, Node.js, Python i C++
* Ugrađena podrška za Git
* Mogućnost otklanjanja pogreški (engl. debugging)
* Isticanje sintakse koje omogućuje lakše utvrđivanje pogrešno napisane sintakse
* Inteligentno dovršavanje koda (engl. IntelliSense) pomoću kojeg se brže i točnije piše kod
* Podrška za kod isječke (engl. code snippets)
* Podrška za refaktoriranje koda
* Mogućnost nadogradnje sučelja pomoću ekstenzija

Visual Studio Code se pokazao kao favorit kod programera. U anketi Developers Survey 2021. Visual Studio Code je rangiran kao broj 1, a koristi ga 70% od 82.000 ispitanika. Pokazao se kao favorit radi svoga elegantnog, jednostavnog sučelja i svojih bogatih mogućnosti. Također promovira dobru praksu pisanja urednog i čitljivog koda pomoću automatskog uvlačenja, zatvaranja zagrada, itd. Također omogućuje suradnju članova tima preko ugrađene Git podrške, te također pomoću LiveShare ekstenzije omogućuje mijenjane i otklanjanja pogreški unutar koda u stvarnom vremenu[2].



Slika 1.1 Logo Visual Studio Code-a

## JavaScript

JavaScript je jednostavan, interpretiran programski jezik namijenjen ponajprije razvoju interaktivnih HTML stranica. Jezgra JavaScript-a uključena je u većinu današnjih preglednika[3]. JavaScript je jezik visoke razine. Ima dinamičko tipkanje, objektnu orijentaciju temeljenu na prototipu i prvorazredne funkcije. Podržava programiranje orijentirano na događaje (engl. event-driven programing), funkcionalno programiranje, i imperativno programiranje. Ima aplikacijsko sučelje (engl. application programing interface, API) za rad s tekstom, datumima, regularnim izrazima, standardnim strukturama podataka, i modelom objekta dokumenta (engl. Document Object Model).

JavaScript sadrži standardnu ​​biblioteku objekata, kao što su **Array**, **Date** i **Math**, te osnovni skup jezičnih elemenata kao što su operatori, kontrolne strukture i iskazi. Jezgra JavaScripta može se proširiti za razne svrhe dopunjavajući ga dodatnim objektima; na primjer[4]:

* *JavaScript na strani klijenta* proširuje osnovni jezik opskrbljujući objekte za kontrolu preglednika i njegovog *modela objekta dokumenta* (DOM). Na primjer, proširenja na strani klijenta omogućuju aplikaciji da postavi elemente na HTML obrazac i odgovori na korisničke događaje kao što su klikovi mišem, unos obrasca i navigacija stranicama.
* *JavaScript na strani poslužitelja* proširuje temeljni jezik isporukom objekata relevantnih za pokretanje JavaScripta na poslužitelju. Na primjer, proširenja na strani poslužitelja omogućuju aplikaciji komunikaciju s bazom podataka, osiguravaju kontinuitet informacija od jednog pozivanja do drugog aplikacije ili obavljaju manipulacije datotekama na poslužitelju.

JavaScript omogućava izvršavanje radnji u inače statičnim HTML dokumentima, npr. omogućava interakciju s korisnikom, promjenu svojstava prozora preglednika, ili dinamičko stvaranje HTML sadržaja.

Logo

Description automatically generated

Slika 1.2 Neslužbeni logo JavaScript

Program koji obrađuje i izvršava skripte zove se interpreter. Interpreter čita kod i prevodi ga u strojni jezik svakog puta kada se pokrene skripta. Svaki jezik koji se interpretira, tj. koji izvršava interpreter, naziva se skriptni jezik. Interpreter za JavaScript ugrađen je u većinu današnjih web preglednika.

Razvoj programa je znatno jednostavniji uz skriptne jezike, zato što se kod skriptnog jezika ne treba prevoditi u strojni jezik.

Koraci kod skriptnih jezika su sljedeći:

1. Napisati ili popraviti skriptu
2. Pokrenuti interpreter
3. Za potrebne popravke ponovno krenuti od 1. koraka

HTML je jezik koji se koristi za opis dokumenata i nema dinamičnih elemenata. Ukazala se potreba za uvođenje dinamičnog načina stvaranja HTML elemenata i stvaranje interaktivnog sadržaja u HTML-u. Danas za to postoji mnogo tehnologija, no u ovome radu je najinteresantnija skupina takozvanih klijentskih jezika.

Kod klijentskih jezika se interpretira na klijentskog strani u web pregledniku. Glavni predstavnik klijentskih jezika je JavaScript. Danas je podrška za JavaScript izvrsna u svim preglednicima, tako da se danas autori preglednika natječu u brzini izvođenja određenih algoritama, dok su se prije natjecali u podršci.

Document Object Model (DOM) je model za prikaz i interakciju s objektima u HTML dokumentu. Omogućava jednoznačan i jednostavan pristup dijelovima HTML dokumenta te rukovanje njegovim dijelovima, npr. elementi u HTML dokumentu. JavaScript definira svoj DOM u obliku hijerarhijske strukture prikazane na

Diagram

Description automatically generated

Slika 1.3 JavaScript DOM

Svakom objektu ili svojstvu pristupa se kroz taj model, tj. dokument je osnovni objekt preko kojeg se pristupa svim drugim objektima dokumenta.

# Opis igre

U ovome poglavlju je detaljno opisana igra, te su dane informacije o funkcionalnim zahtjevima igre te su prikazana stanja i slike objekata koje su korištene u realizaciji igre.

## Funkcionalni zahtjevi

Unutar tablice prikazani su svi funkcionalni zahtjevi postavljeni u dvodimenzionalnoj Rabbit Trap igri.

Tablica 2.1 Funkcionalni zahtjevi

|  |  |  |
| --- | --- | --- |
| ID Zahtjeva | Naziv Zahtjeva | Detaljan opis |
| Z-1 | Pokretanje igre | Igra počinje kada korisnik u svoj web preglednik unese IP adresu servera na kojem je pokrenuta igra |
| Z-2 | Igranje igre | Korisnik koristi tipke na tipkovnici kako bi igrao igru, te igrica se prekida tek kada korisnik dođe predodređenog završetka igre |
| Z-3 | Jednostavnost | Igra treba biti dovoljno jednostavna kako bi je što veći broj korisnika mogao zaigrati, te kako bi što veći broj korisnika mogao doći do završetka igre |
| Z-4 | Uvid u broj sakupljenih objekata | Za vrijeme trajanja igre korisnik ima uvid u trenutni broj objekata koje je prikupio unutar igre |

## Use-case dijagram

Na dijagramu prikazan je use-case dijagram korisnika s radnjama koje može obavljati unutar aplikacije.

Diagram

Description automatically generated

Dijagram 1 Use-case dijagram

## Stanja igre

Igrica posjeduje samo jedno stanje, stanje Game. Kada se igra pokrene, korisnik dobije kratke upute kako igrati igricu. Korisnik se nalazi u Game stanju sve dok ne sakupi sve potrebne mrkve, ili dok igricu ne ugasi. Kada korisnik sakupi sve mrkve, dobije kratku obavijest da je uspješno sakupio sve mrkve i završio igricu, te ako pritisne na tipku OK, ponovno se pokreće stanje Game od početka. Iz igrice se izlazi tako da se ili ugasi web browser u kojem je pokrenuta igra, ili ako se ugasi web kartica u kojoj je igrica pokrenuta.

## Slike objekata

Slika potrebna za realizaciju Rabbit Trap igrice preuzeta je sa stranice [5]. Svi elementi potrebni za realizaciju igrice nalaze se unutar iste .png datoteke. Na slici 2.1 prikazan je sadržaj rabbit-trap.png datoteke. Dimenzije rabit-trap.png slike su 128 x 128. Slika unutar sebe ima osam stupaca, te će se dijelovi slike „isjeći“ u male blokove veličine 16 x 16 piksela. Blokovi će se zatim slagati jedan pokraj drugoga kako bi se formirao svijet igre. Također će se brzom izmjenom ovih isječaka postići animacija igračevog lika, elemenata koje igrač treba skupiti, te animaciju trave koja se nalazi na platformama svijeta.

A picture containing text, electronics, circuit

Description automatically generated

Slika 2.1 Sadržaj datoteke rabbit-trap.png

# Praktični dio seminara

U ovom poglavlju su detaljno opisani, s obzirom na ovaj kolegij, najinteresantniji i najbitniji dijelovi koda platformerske igre u JavaScript-u.

## Kratak opis igre

Rabbit Trap je igrica napravljena pomoću JavaScript programskog jezika. Igrač je predstavljen u obliku zeca čiji je cilj skupit sve mrkve koje se nalaze unutar svih nivoa igre. Kada igrač skupi jednu mrkvu, brojač mrkvi se uveća za jedan. Zec se kreće tako da skače i trči lijevo i desno po raznim platformama. Platforme su promišljeno smještene unutar svijeta, te predstavljaju prepreke koje igrač treba svladati kako bi završio igru. Igra završava tek kada zec sakupi sve mrkve unutar igre, ili dok korisnik ne isključi igru.

## Strukturna organizacija igre

Unutar glavnog direktorija se nalaze sljedeće datoteke:

* rabbit-trap.png
* rabbit-trap-html
* rabbit-trap.css
* server.js
* 8 direktorija unutar kojih su spremljene različite faze razvoja igre

Unutar 8 direktorija koji predstavljaju različite faze razvoja igre se nalazi jedna od, ili više sljedećih datoteka:

* controller.js datoteka, unutar koje je sadržan kod upravljanje igračevim likom
* display.js datoteka, unutar koje je sadržan kod za crtanje svijeta i objekata u njemu
* game.js datoteka, unutar koje su sadržani najbitniji dijelovi igre, kao animacija igre, kreacija svijeta, itd.
* engine.js datoteka, unutar koje je sadržan kod za pokretanje i gašenje ciklusa igre, te prisilno ažuriranje igre
* main.js datoteka, unutar koje je sadržan kod za pozivanje svih drugih datoteka. Preko main.js datoteke, sve druge datoteke komuniciraju

Direktorij imena 08 je finalna verzija igrice, te je ona postavljena kao zadana (engl. default) verzija igre koja se pokreće kada se igra pokrene. Drugim dijelovima je moguće pristupiti kroz meni. Pomoću html datoteke se radi dinamički poziv različitih instanci datoteka koje se nalaze unutar tih 8 direktorija, tj. pozivaju se one instance tih datoteka koje su potrebne da ta verzija igre radi ispravno. Unutar direktorija 06, 07, i 08 se nalaze JSON datoteke iz koji se učitavaju nivoi igrice, jer su se tek od 6 iteracije igrice one uključile.

## Main.js

Unutar main.js funkcije nalazi se klasa AssetManager, koja ima ulogu učitavanja i spremanja rabbit-trap.png slike, i JSON datoteka pomoću kojih će se učitavati nivoi unutar igrice. Ova klasa unutar sebe ima funkcije requestJSON i requestImage. Koristimo JavaScript callback() funkciju kako bi se funkcije za kreiranje svijeta i pokretanje igre tek pokrenule nakon što se uspješno učita JSON i konvertira u JavaScript objekt, i nakon što se rabbit-trap,png uspješno učita. Pomoću eventListener procedure, gore navedene funkcije se tek pokreću kada se datoteke krenu učitavati, te pošto učitavanje traje, s pomoć callback() funkcije smo osigurali da se funkcije izvrše tek kada to učitavanje bude gotovo.

Text

Description automatically generated

A screenshot of a computer screen

Description automatically generated with medium confidence

Resize funkcija unutar main poziva funkciju resize koja se nalazi unutar Display.js. Pokreće se samo kada se izvrši potrebni događaj (engl. event) u ovom slučaju kada se promjeni veličina prozor web preglednika. Unutar nje se mijenja veličina platna, te time i svih elemenata koji se nalaze unutar platna (mrkve, igrač, platforme, itd.). Također se mijenja i veličina brojača sakupljenih mrkva.

Text

Description automatically generated

A screenshot of a video game

Description automatically generated

Slika 3.1 Igrač bez vertikalnog pomaka

Funkcija render zadužena je za pozivanje funkcija definiranih unutar Display.js datoteke. U ovoj funkciji se šalju podatci učitani iz rabbit-trap.png i iz JSON objekata kako bi se generirao svijet. Obje for petlje prolaze kroz JSON datoteku koja definira zonu u kojoj se korisnik trenutno nalazi, čitaju njen sadržaj, te taj sadržaj prosljeđuje funkcijama za crtanje. Može se primijetiti da se svaki objekt (igrač, mrkve, trava) pomiče za neki offset. To se radi kako bi objekti prirodnije i bolje pristajali unutar blokova svijeta. Za primjer na slici 3.1 možemo pogledati šta se dogodi ako izbrišemo vertikalni pomak igračevog objekta. Vidimo da igrač „propada“ kroz svijet igre. To se događa zato što smo iz rabbit-trap.png isjekli pravokutne sličice za igrača, te one nisu same prilagođene da savršeno pristaju generiranom svijetu. Zato ih je potrebno pomaknuti za nekoliko piksela.

Text

Description automatically generated

Text

Description automatically generated

Update funkcija vrši ažuriranje igre. Unutar nje se poziva funkcija za pokretanje novog nivoa ako igrač prođe kroz vrata koja ga vode na drugi nivo. Vidi se da se igrica na tren zaustavi kada igrač prođe kroz vrata. Zaustavi se kako bi igrica imala vremena da uspješno učita JSON datoteku o drugom nivou, te uspješno nacrta sljedeći nivo. Nakon toga se igrica ponovno pokreće. Također se vidi da se igrica ažurira kada korisnik pritisne tipke za pomicanje lika.

Pogledajmo još sadržaj JSON datoteke za prvi nivo igre. Unutar datoteke su zapisane pozicije mrkvi i travki unutar ovoga nivoa, vrata unutar nivoa i na koji drugi nivo ta vrata vode, veličinu nivoa u stupcima i redcima (veličina je konzistentna među nivoima), i neki identifikator nivoa. Kako su nivoi spremljeni unutar JSON datoteke možemo lako objasnit pomoću mrkvi. Naime mrkve su u ovo datoteci jedan veliki niz s više pod nizova. Jedan podniz, npr. [1,3] predstavlja lokaciju jedne mrkve unutar svijeta, tj. ona se nalazi u drugom stupcu svijeta, i u četvrtom retku svijeta. Pošto je riječ o nizu, prvi član niza se predstavlja s 0 umjesto s 1. Ustvari podnizovi predstavljaju lokaciju pojedine mrkve u svijetu, tj. predstavljaju u kojem stupcu i kojem retku u svijetu se nalazi pojedina mrkva.

Text

Description automatically generated

Razmotrimo još graphical\_map i collision\_map vrijednosti. graphical\_map vrijednosti predstavljaju isječak iz rabbit-trap png koji želimo isjeći, a njegova pozicija unutar niza predstavlja gdje ćemo tu sličicu nacrtati u svijetu.

rabbit-trap png unutar sebe ima 8 stupaca. To znači da svaki redak slike ima u stupaca. Ako želimo isjeći gornje lijevi kut slike, vrijednost unutar graphical\_map ćemo postavit u 1. Vrijednost 2 će isjeći prvi redak, drugi stupac iz slike, dok će vrijednost 9 isjeći drugi redak, prvi stupac iz slike. Svi isjeći će bit iste veličine, 16 x 16 piksela.

Interesantniji dio predstavlja pozicija svake graphical\_map vrijednosti unutar niza. Pošto je zadano da imamo 12 stupaca u svijetu, prvih 12 elemenata graphical\_map niza predstavlja prvi redak svijeta, drugih 12 elemenata drugi redak svijeta, itd. Vrijednostima se ustvari određuje koju sličicu siječemo iz rabbit-trap.png slike, dok njihova pozicija u nizu određuje gdje će se ta sličica nacrtati.

Po sličnom principu, vrijednosti unutar collision\_map niza predstavljaju strane pojedine sličice koje će predstavljati zid za igrača, dok njihova pozicija u nizu predstavlja lokaciju tih zidova. Više o tome će se reći kasnije kada se bude razmatrala kolizija.

graphical\_map i collision\_map su ručno napisani nizovi, te su vrlo usko povezani, jer svijet treba biti igraču intuitivan. Ako nacrtamo platformu na koju igrač intuitivno misli da može skočiti, moramo mu osigurati da se kolizija s tom platformom odvija kao što je i pretpostavio, i jednostavno ne bi imalo smisla da igrač može propadati ili kretati se kroz zidove jer smo krivo postavili koliziju za prikazanu sliku.

## Display.js

Unutar Display.js datoteke se nalaze funkcije za crtanje svijeta i objekata u njemu, te funkcija za promjenu veličine.

Resize funkcija je dosta jednostavna. Svijet igre je smješten unutar platna (engl. canvas), te se veličina platna, a time i veličina svijeta igre, mijenja ovisno o veličini korisnikovog web preglednika. Kada je preglednik maksimalne veličine, platno je također svoje maksimalne veličine. Kako se veličina preglednika smanjuje, tako se smanjuje i veličina platna.

Text

Description automatically generated

Funkcije za crtanje objekata su kompleksnije. rabbit-trap.png ima unutar sebe 8 stupaca, a svijet igre ima 12 stupaca, s tim da je veličina svakog bloka unutar svijeta 16 x 16 piksela. source\_x i source\_y predstavljaju poziciju bloka unutar rabbit-trap.png kojeg ćemo isjeći. Ustvari namještamo x i y poziciju unutar rabbit-trap.png datoteke, te onda siječemo 16 piksela u širinu i 16 piksela u visinu kako bi formirali blok veličine 16 x 16, unutar kojeg se nalazi željeni dio slike. Zatim taj blok crtamo na željenu poziciju unutar svijeta igre. Isti postupak se vrši za svaki redak i stupac svijeta igre.

Text

Description automatically generated

## Game.js

Unutar datoteke Game.js se nalazi najveća količina najbitnijeg koda za kreaciju svijeta i animiranje objekata unutar svijeta. Većina klasa unutar ove datoteke su apstrahirane pomoću prefiksa Game., ustvari koristi se . sintaksa kako bi se te klase dodale unutar glavne Game klase. U principu ova apstrakcija je najbitnija za organizaciju koda, no korisna je i za odvajanje koda naše igrice od globalno koda web stranice (na primjer ako naša igrica završi na web stranici preko koje je moguće pristupiti mnogim drugim igrama, želimo odvojiti naš kod od koda drugih igara).

### Animacija objekata

Pogledajmo dijelove koda za animiranje objekata igre. Za početak najbolje je pogledati Frame i TileSet klasu.

Text

Description automatically generated

Frame klasa definira pravokutnu regiju iz koje ćemo isjeći sličicu iz rabbit-trap.png slike. Offset svojstva služe za pomicanje isječka unutar našeg svijeta.

Graphical user interface

Description automatically generated with medium confidence

Unutar TileSet klase prati broj stupaca i veličinu svakog isječka glavne slike. Animacije objekata unutar svijeta se postižu pomoću frames niza. Promotrimo prvi poziv Frame klase unutar niza. On predstavlja pravokutnik koji siječemo iz slike rabbit-trap.png, te on predstavlja isječak unutar kojeg se igračev lik, bijeli zec, nalazi u svojoj lijevoj mirnoj (engl. idle) poziciji. Grupa isječaka u trećem i šestom redu niza predstavljaju sve isječke koji formiraju lijevu i desnu animaciju kretnje. Same po sebi, ovaj niz neće animirati našeg igrača jer nije definirano kada se će se koji element niza pozvati. Unutar Player klase se definira kada će se pozvati koja sličica iz frames niza, te će se ta klasa detaljno objasniti. Pošto su igračev lik, mrkve i trava u svijetu apstrahirane kao objekti, na sličan način će i klase Carrot (hr. mrkve) i Grass (hr. trava) animirati objekte koje predstavljaju. Programski kod apstrahirao kako bi bilo moguće iskoristiti napisane klase za još mnoge druge objekte koji se potencijalno mogu dodati u igricu.

Text

Description automatically generated

Unutar frame\_sets objekta koji se nalazi u Player klasi su definirani nizovi, te vrijednosti unutar nizova odgovaraju isječcima iz frames niza iz TileSet klase. Unutar Animator klase se vrši animacija lika.

Text

Description automatically generated

Animator klasa je ovisna o frame rate. To znači da brzina sličica po sekundi o potpunosti ovisi o frame rate-u cijele igrice. frame\_set je frame\_set iz Player klase, frame\_index predstavlja gdje se nalazimo unutar frame\_set niza, a frame\_value predstavlja vrijednost trenutnog indeksa unutar frame\_set niza. Sada pogledajmo loop funkciju unutar Animator klase.

Text

Description automatically generated

Za animacije šetanja je definirano kašnjenje (engl. delay) od 10. To znači da svakih 10 frame-ova unutar igre će programski kod uči unutar while petlje. Prvo se brojač frame-ova resetira, a zatim se postavlja vrijednost frame\_index-a na pomoću sljedećeg pitanja: Jeli frame\_index trenutno manje od dužine frame\_set niza?

Najbolje je objasniti način rada kroz primjer. Na slici 3.2 se nalazi frame\_set s vrijednostima za kretnju lika udesno. frame\_index = 0 ima vrijednost sličice na krajnje lijevoj strani. Kada se vrijednost frame\_index-a poveća za 1, također će se i promijeniti vrijednost frame\_value-a za 1, te će to indeks 1 imati vrijednost druge sličice s krajnje lijeve strane. Kada se frame\_index nalazi na krajnjoj vrijednosti frame\_set niza, umjesto da se ponovno vrijednost inkrementima, ona se vraća nazad na početnu vrijednost. Time se postiže animacija konstante kretnje lika.

Qr code

Description automatically generated

Slika 3.2 frame\_set niz za šetanje udesno

Text

Description automatically generated

changeFrameSet funkcija ima svrhu samo promijeniti vrijednost frame\_set niza unutar Player klase u onaj frame\_set koji definira trenutno stanje igrača. Na primjer, ako se igrač pomakne u lijevo, frame\_set se mijenja u vrijednosti u koje su spremljene sličice za animiranje šetanje lika uljevo.

Text

Description automatically generated

Pogledajmo još updateAnimation funkciju koja je zaslužna za određivanje koji frame\_set želimo postavit, tj. koji frame\_set odgovara trenutnoj poziciji ili kretnji igračevog lika. Prva if izjava pita dali igrač skače. Zatim druga if izjava unutar prve if petlje gleda u kojem smjeru se kreće lik. Ako se pritisne tipka za kretanje ulijevo, direction\_x postaje negativna vrijednost, te se frame\_set postavlja u vrijednosti za skakanje uljevo. U suprotnom, ako se uvjet petlje ne zadovolji, frame\_set se postavlja u vrijednosti za skakanje udesno.

Zatim po sličnom postupku se namješta vrijednost frame\_set-a za kretanje uljevo. Ako igrač ima negativnu vrijednost smjera kretanja, i brzinu kretanje manju od 0, frame\_set se postavlja u vrijednosti za kretanje ulijevo. U slučaju da je brzina igrača 0, frame\_set se postavlja u vrijednosti za mirovanje (engl. idle) ulijevo. Isto se radi i za kretanje udesno.

### Kolizija na temelju pločica

Pogledajmo dijelove koda za detekciju kolizije, te kako igrica reagira nakon detekcije. Prvo objasnimo široku fazu (engl. broad phase) koliziju koja se nalazi unutar collideObject funkcije.

Funkcija provjerava svaki različiti kut igračevih sličica. Zato imamo četiri komada koda, jedan za provjeru svakog kuta. Vrijednosti top (hr. gore) i bottom (hr. dolje) predstavljaju redak u kojem se igrač nalazi, dok left (hr. lijevo) i right (hr. desno) predstavljaju stupac u kojem se igrač nalazi. Naravno provjerava se odgovarajući dio igračeve sličice.

Npr. left predstavlja u kojem se stupcu nalazi lijevi dio igračeve sličice. Vrijednosti stupaca u kojem se nalazimo dobijemo tako da y ili x poziciju igrača podijelimo s veličinom pojedine sličice, što je uvijek 16 (jer su sličice 16x16 piksela). Pomoću floor vrijednost indeksa zaokružujemo na nižu vrijednost. Zatim red i stupac konvertiramo u indeks unutar value-a. Indeks ima vrijednost redak \* broj stupaca u svijetu (postavljeno na 12) + stupac.

Text

Description automatically generated

Primjećuje se da se više puta provjerava ista strana igrača. Kako bi igrica ispravno radila, moramo imati dvije vrijednosti za svaku provjeru kako se ne bi došlo do pogreške. Na primjer, kada se igrač sudari s zidom u gornje desnom kutu, igrača će zid pomaknuti, te može doći do pogreške da kolizija za donje desni kut ima istu vrijednost desnog stupca kao i za gornje desni kut, iako to ne bi trebao biti slučaj jer je zid igrača pomaknuo. Zato je bolje osigurati se i provjeravati koliziju svakog kuta igrača zasebno.

Pogledajmo sada funkciju usmjeravanja. Funkcija prima vrijednosti iz collideObject funkcije. Ulazi u switch izjavu pomoću vrijednosti indeksa, koja zatim usmjerava u odgovarajuću usku fazu (engl. narrow phase) koliziju.

Pogledajmo prvi slučaj unutar switch izjave. Kada igrač stane na pločicu s vrijednosti 1, poziva se funkcija collidePlatformTop unutar koje se šalje objekt (igrač) i tile\_y (pločicu y) jer nas za ovaj slučaj ne interesira pločica x. Za kompleksniji slučaj možemo pogledat kada igrač stane na pločicu s vrijednosti 13. Kada se to dogodi, pozivaju se funkcije collidePlatformTop, collidePlatformLeft, i collidePlatformBottom. Odnosno na ovu pločicu igrač se može sudariti u tri različita smjera, ali u samo jednom smjeru u jednom određenom trenutku.

Imamo 16 različitih mogućnosti za koliziju s platformama. Za svaku sličicu unutar svijeta ručno je zadana vrijednost od 0 do 15, koja određuje s koje strane će se nalaziti zidovi na toj platformi. Ove vrijednosti su zapisane unutar JSON objekta za svaki nivo, te svaka sličica svijeta unutar JSON objekta ima svoju odgovarajuću vrijednost za koliziju.

Vrijednosti je najbolje objasniti pomoću slike 3.3. Vrijednost 0 predstavlja da se igrač neće moći sudariti s sadržajem te sličice. Vrijednost 1 predstavlja mogućnost da se igrač sudari samo s gornjim dijelom sličice, tj. gornji dio te sličice će predstavljati zid za igrača. Ove vrijednosti odgovaraju vrijednostima unutar switch izjave koje su prethodno objašnjene. Zato i unutar switch izjave ako se stupimo u kontakt s sličicom s vrijednosti kolizije 1, odmah znamo da moramo gledati samo jeli se dogodila kolizija s gornjim vrhom te sličice (platforme). Ove vrijednosti odgovaraju collision\_map vrijednostima učitanim iz JSON datoteke za pojedini svijet.

A screenshot of a computer

Description automatically generated with medium confidence

Slika 3.3 Vrijednosti za koliziju

Graphical user interface, text

Description automatically generated

Pogledajmo sada usku fazu kolizije. Uska faza je definirana s četiri funkcije: collidePlatformBottom, collidePlatformLeft, collidePlatformRight, i collidePlatformTop. Kako funkcije rade, možemo razmotriti collidePlatformTop funkciju. Iz funkcije usmjeravanja odbijamo objekt i gornji vrh pločice. Zatim se ulazi u if izjavu koja provjerava jeli dno objekta (igrača) veće od gornjeg vrha pločice i jeli dno objekta iz prethodnog ciklusa animacije (engl. previous frame of animation) manje ili jednako gornjem vrhu pločice. Uvjet je tu kako bi se osigurao gladak prijelaz lika i glatka kolizija. Bez drugog dijela petlje gdje se provjerava stara pozicija objekta, kretnja unutar igre bi izgledala kao da se ciklus igre preskače. Trenutna pozicija igrača i pozicija igrača u prethodnom ciklusu se nalaze unutar Player klase, te kada se pozove update funkcija, trenutne vrijednosti se spremaju kao stare za svrhe novog ciklusa.

Text

Description automatically generated

Text

Description automatically generated

Kada se uvjet zadovolji, dno igrača postavljamo na gornji vrh pločice – 0.01. Vertikalna brzina igraća se postavlja na 0 kako bi se prestala povećavati, te da se to ne napravi, brzina igrača bi se nastavila povećavati dok ga ne bi povukla u potpunosti kroz pločicu. Također se postavi da igrač više ne skače. Funkcija vraća istinitu vrijednost u funkciju usmjeravanja, čime njoj govorimo da nije više potrebno provjeravati druge moguće slučaje kolizije s tom platformom.

Sada detaljnije pogledajmo zašto se od vrha pločice oduzima 0.01. Isti problem imamo i s funkcijom za koliziju s lijevom stranom pločice, i istom metodom rješavamo problem, no rješenje problema ćemo objasniti na koliziji s lijevom stranom pločice jer tu će se jednostavnije razumjeti.

Naime problem nastaje što se igračeva sličica može nalaziti malo unutar sličica platformi. Bez da oduzmemo neki mali broj, kada bi se igrač sudario u potpunosti s lijevim zidom, a potom skočio, u nekim slučajevima bi zapeo za gornji dio platforme, jer bi se igrač registrirao kao da se nalazi u drugome stupcu od onoga u kojem je zapravo. Upravo ovi problem je prikazan na slici 3.3. Kada oduzimamo taj neki mali broj od igrača, ustvari ga guramo od platforme nazad u stupac u kojem se zapravo nalazi, čime se garantira dobra funkcionalnost kolizije.

Diagram

Description automatically generated with medium confidence

Slika 3.4 Problem s detekcijom kolizije na lijevoj strani

Pogledajmo sada koliziju igrača i mrkve, tj. koliziju dva objekta. Sve kreće od funkcije za koliziju dva objekta. Ovo je vrlo jednostavna funkcija, ako se pravokutne sličice dva objekta dotaknu, dolazi do kolizije ta dva objekta.

Text

Description automatically generated

Zatim pogledajmo sadržaj update funkcije. Prvo ažuriramo igračevu poziciju, zatim sudarimo igrača s svijetom igre kako bi pokrenuli koliziju s terenom, te zatim dolazimo do kolizije s mrkvama. Carrot niz sadrži sve naše objekte mrkve koje smo kreirali unutar unašeg svijeta. Zatim ažuriramo položaj svih mrkvi i animiramo ih, sve po sličnom principu kao i za igrača.

Zatim se pitamo jeli se mrkva sudara s igračem, te ako je ulazimo u if petlju. Ako se igrač sudario s mrkvom, mrkvu s kojom se sudario brišemo iz niza u kojem su spremljene mrkve, te brojač mrkvi inkrementiramo za 1.

Text

Description automatically generated

Za kraj, pogledajmo koliziju igrača s vratima, tj. lokacije na kojoj igrač mora biti kako bi prešao na drugi nivo. Također unutar update funkcije na sličan način prolazimo kroz sva vrata unutar svijeta, te ćemo testirati jeli igrač došao u koliziju s vratima. Zatim vrijednost vrata, koja je inicijalno ne definirana, mijenjamo u vrijednost vrata s kojima je igrač došao u sudar. Potom unutar main.js datoteke se zaustavlja igrica, učitava se nivo u koji vrata s kojima se igrač sudario vode, te se taj nivo učitava i crta na ekran. Tek nakon što se ispravo narcta svijet, igrica se opet pokreće.

Text

Description automatically generated

Zatim razmotrimo dio setup funkcije unutar Game.js datoteke koji je sadužen za ispravno pozicioniranje lika kada prođe kroz vrata. Ako door.destination za x ili y iznosi -1, to označava da se igračeva pozicija ne mijenja, već da ostaje ista kao i prije nego što je prošao kroz vrata. Da nema ove provjere, igračeva tranzicija kroz nivoe bi bila neprirodna, kao što je prikazano na slici 3.5. Ustvari igrač bi krenuo padata s vrha svijeta, što može uzrokovati niz problema s kolozijom.

Text

Description automatically generated

A picture containing shape

Description automatically generated

Slika 3.5 Neprirodna tranzicija igrača između nivoa

Text

Description automatically generated

Detekcija kolizije s vratima se radi pomoću funkcije collideObjectCenter koja detektira središnju koliziju. Jedina bitna razlika od funkcije za koliziju s objektima je to što je kod kolizije između dva objekta bilo samo bitno da se neki dio njihovih pravokutnih sličica dotakne da bi došlo do kolizije, dok se kod središnje kolizije mora doći do kolizije sredine objekata.

# Popis Literature

1. <https://en.wikipedia.org/wiki/Visual_Studio_Code>
2. <https://code.visualstudio.com/learn>
3. <https://www.srce.unizg.hr/files/srce/docs/edu/osnovni-tecajevi/c501_polaznik.pdf>
4. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction>
5. <https://github.com/pothonprogramming/pothonprogramming.github.io/tree/master/content/rabbit-trap>
6. <https://www.youtube.com/watch?v=w-OKdSHRlfA>