Primjena GameMaker Studio razvojnog okruženja za pravljenje računalnih igara

Mijić, Matej

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni preddiplomski studij računarstva

PRIMJENA GAMEMAKER STUDIO RAZVOJNOG OKRUŽENJA ZA PRAVLJENJE RAČUNALNIH IGARA

Završni rad

Matej Mijić

Osijek, 2018.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 11.09.2018.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Matej Mijić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3808, 25.09.2017.
OIB studenta:	83441250579
Mentor:	Doc.dr.sc. Časlav Livada
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Primjena GameMaker Studio razvojnog okruženja za pravljenje računalnih igara
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene mentora:	11.09.2018.
Datum potvrde ocjene Odbora:	17.09.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 19.09.2018.

Ime i prezime studenta:	Matej Mijić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3808, 25.09.2017.
Ephorus podudaranje [%]:	1

Ovom izjavom izjavljujem da je rad pod nazivom**: Primjena GameMaker Studio razvojnog okruženja za pravljenje računalnih igara**

izrađen pod vodstvom mentora Doc.dr.sc. Časlav Livada

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.

Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Danas postoji puno različitih programa za razvoj računalnih igara. Većina tih programa zahtjeva predznanje kompliciranih programskih jezika i nisu jednostavni za korištenje, pogotovo za one koji su novi u svijetu programiranja i pravljenja računalnih igara. YoYo Games, tim programera iz Engleske, razvili su softversku platformu za razvoj računalnih igara GameMaker Studio 2. To je alat koji teži što lakšem i bržem načinu stvaranja računalnih igara i ostvarivanja različitih ideja [1]. Prvenstveno je dizajniran za stvaranje 2D računalnih igara koje se prenose na više platformi iz istih inicijalnih baza resursa. Za one koji su novi u razvoju računalnih igara i programiranju GameMaker Studio 2 uvodi „Drag and Drop“ sučelje koje omogućuje stvaranje igara vizualnim skriptiranjem. GameMaker Studio 2 uz „Drag and Drop“ metodu koristi i programski jezik Game Maker Language odnosno GML. On je namijenjen korisnicima sa više iskustva i nudi više fleksibilnosti pri izradi računalnih igara pisanjem programskog koda. U glavnom dijelu rada detaljno je opisano GameMaker Studio razvojno okruženje, Game Maker Language koji je korišten pri izradi igre te razvoj same igre. Tema igre je 2D akcijski platformer u kojoj je cilj koristeći različita oružja i mehanike glavnog lika te izbjegavati neprijatelje i prelaziti složene razine.

1.1. Zadatak završnog rada

U radu je potrebno opisati način rada u GameMaker Studio razvojnom okruženju te u istom napraviti 2D računalnu igru. Pri izradi igre korišten je GameMaker Language programski jezik.

2. GAMEMAKER STUDIO RAZVOJNO OKRUŽENJE

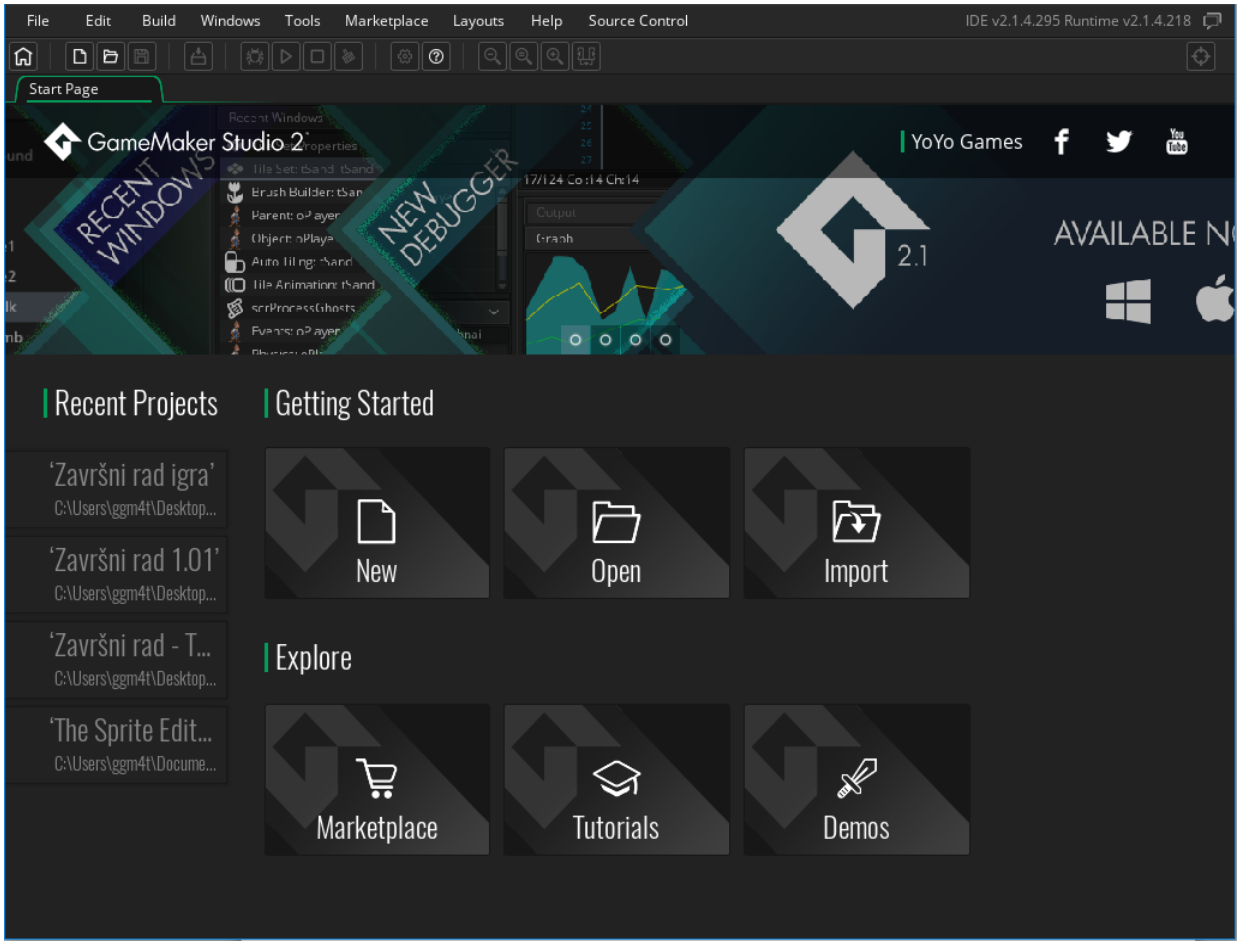
GameMaker Studio 2 (kasnije u tekstu skraćeno GMS) je zadnja i najveća verzija GameMakera. GMS sadrži sve što je potrebno za izradu 2D računalnih igara, iako su i 3D igre savršeno izvedive. Integrirano razvojno okruženje GMS-a je jako fleksibilno i može se prilagoditi našim potrebama. GMS sadrži veliku bazu ugrađenih funkcija koje možemo koristiti za upravljanje resursima, za namještanje kamera, za rad s varijablama i poljima i još puno toga. Također GMS dolazi sa velikim brojem alata koji nam uvelike pomažu pri izradi naših igara. Neki od tih alata su „Image Editor“ za stvaranje ili uređivanje spriteova i setova pločica, „Room Editor“ za uređivanje soba i dizajniranje razina i mnogi drugi. Našu igru možemo izvesti za različite platforme i operacijske sustave kao što su Windows desktop, Mac OS X, Ubuntu, Android, iOS, fireTV, Android TV, Microsoft UWP, HTML5, PlayStation 4, i Xbox One. Na stranici YoYo Games-a postoje forumi i različiti video uradci koji pružaju veliku pomoć pri korištenju GMS-a [1][2].

2.1. Početna stranica

Pri pokretanju GMS-a otvara se početna stranica koja od nas traži prijavljivanje na naš YoYo račun. Lincenca koju posjedujemo vezana je za naš YoYo račun i preko računa možemo pristupati „GameMaker Community“ forumu gdje se nalaze razni materijali za pomoć i konzultacije [2]. Kada smo se prijavili na račun došli smo na početnu stranicu GMS-a. Ona sadrži traku izbornika, nedavno korištene projekte, različite funkcije za pokretanje novih projekata i otvaranje starih te novosti i pristup različitim materijalima.

Na slici 2.1. vidimo početnu stranicu GMS-a. Na vrhu početne stranice nalazi se traka izbornika i alatna traka. File izbornik služi za pokretanje i otvaranje projekata, te uvoz projekata i otvaranje nedavnih projekata. Preko file izbornika možemo i pristupiti postavkama našega računa ili se odjaviti sa istog. Putem build izbornika možemo birati hoćemo li izgraditi naš projekt za testiranje, ispravljanje pogrešaka ili kao konačnu verziju. Izbornik windows sadrži razne funkcije za upravljanje prozorima prilikom rada. Tools izbornikom otvaramo različite alate. Preko izbornika marketplace pristupamo tržištu gdje možemo kupovati razne dodatke i pomagala. Layouts izbornikom spremamo i učitavamo različita korisnička sučelja. Help izbornik nam služi za pomoć pri radu. Na alatnoj traci nalaze se razni prečaci.

Na slici 2.1. lijevo vidimo nedavno korištene projekte. Klikom na jedan od projekata se otvara projekt. U „Getting Started“ odjeljku možemo pokrenuti novi projekt, otvoriti stari projekt te uvesti projekt. „Explore“ odjeljak nam služi za pristup marketu i različitim pomagalima.

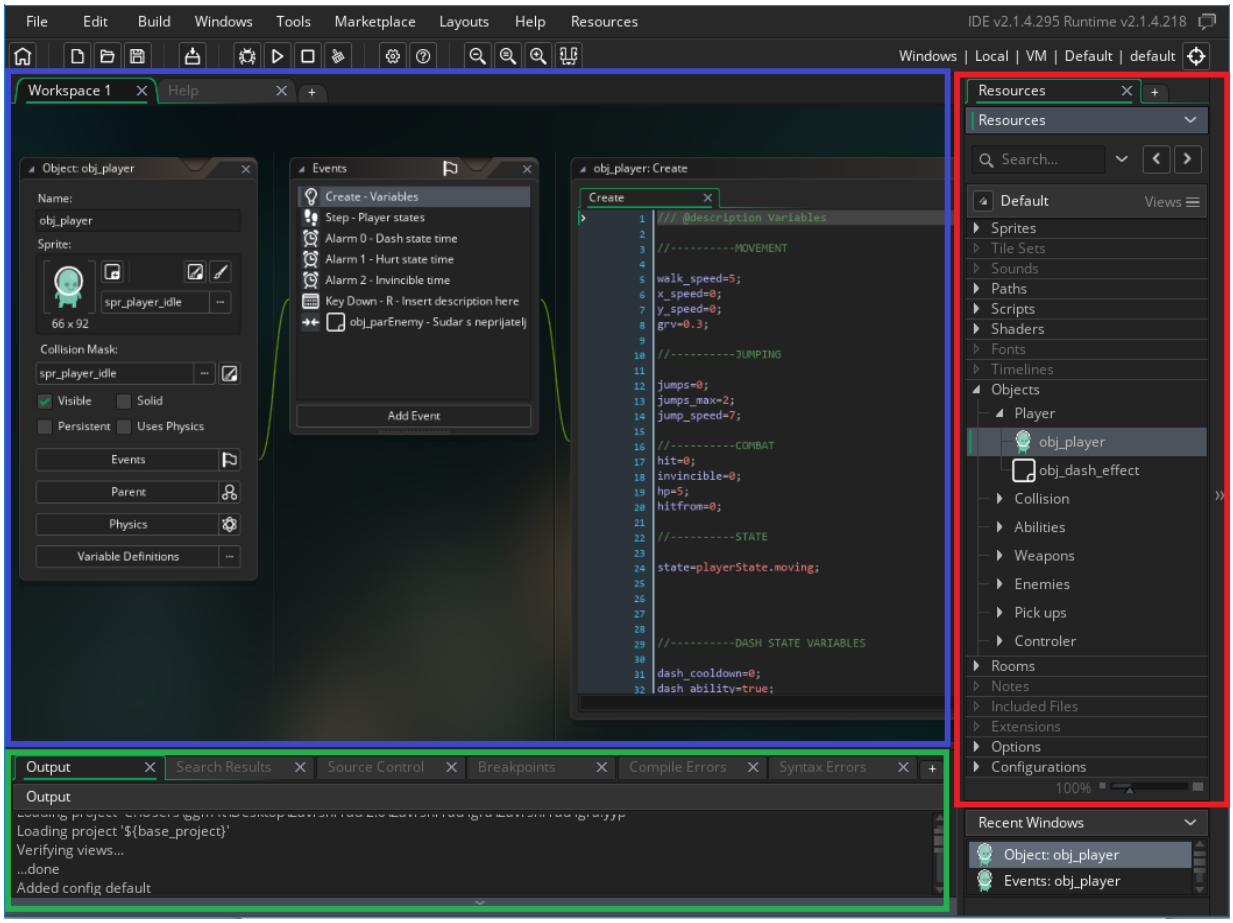


Sl. 2.1. *Početni prozor GMS-a*

2.2. Radni prostor

Prilikom pokretanja procesa GMS otvara radni prostor (Sl.2.2.). Radni prostor je područje u kojem možemo organizirati različite resurse prilikom rada na našoj igri. Možemo stvoriti više radnih prostora kako bi naši resursi bili pregledniji. Različite funkcije kao što su „Go To“, „Search“, spajanje odnosno „Docking“ i druge omogućavaju lagano navigiranje kroz naš projekt i radni prostor. Na alatnoj traci nalaze se razni prečaci za organiziranje radnog prostora i pretraživanje sadržaja.

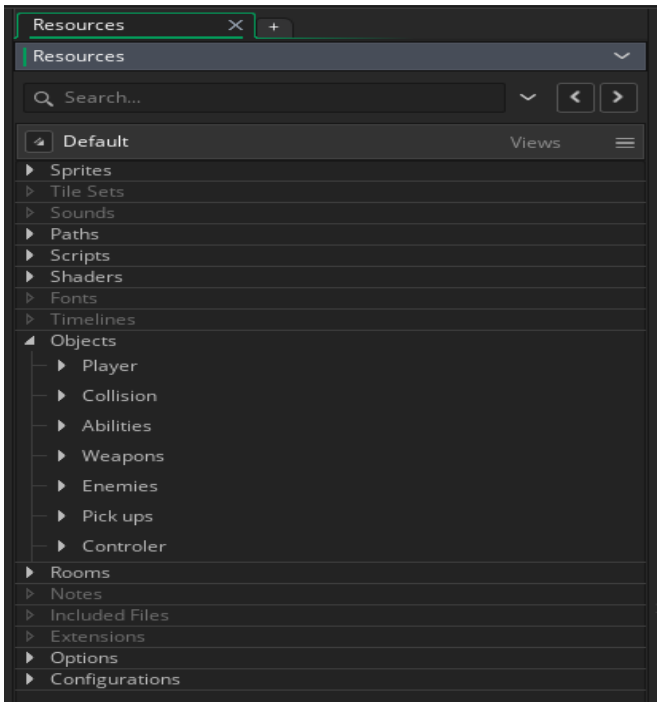
Na slici 2.2. možemo vidjeti dijelove radnog prostora prilikom rada na projektu. On se sastoji od trenutnog radnog prostora, izlaza i stabla resursa. Plavom bojom označen je radni prostor, crvenom resursi i zelenom izlaz.



Sl. 2.2. *Prikaz radnog prostora GMS-a*

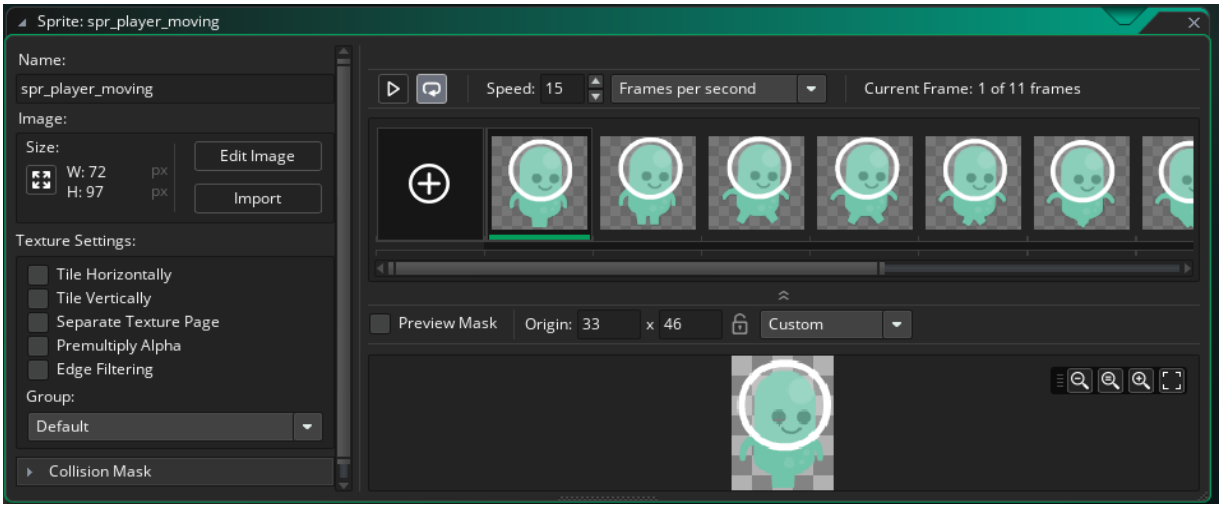
2.3. Resursi

Sva sredstva i atribute koje koristimo prilikom izrade naše igre nalaze se u stablu resursa. U stablo resursa možemo dodavati sve što ćemo koristiti u našem projektu, a i brisati ono što nam ne treba. U njemu se nalaze objekti, spriteovi, sobe, naše skripte, shaderi, fontovi, zvukovi i sve ostalo što nam služi pri izradi naše igre. Resurse možemo organizirati stvaranjem grupa kako bi se lakše snalazili. Resursa uvijek ima jako puno tako da se uvijek treba dobro organizirati.



Sl. 2.3. *Stablo resursa*

Sprite je slika ili niz slika koje dodjeljujete objektima [3]. To su slike koje predstavljaju objekte koje smo mi napravili. Znači kada napravimo objekt pridružujemo mu sprite koji je njegova vizualna reprezentacija [4]. Spriteove možemo napraviti unutar samog GMS-a sa alatom „Image Editor“ ili možemo uvesti našu sliku koju smo napravili u nekom drugom programu. Također sprite nemora biti samo jedna slika, on se može sastojati od niza slika koje stvaraju animaciju (Sl. 2.4.)

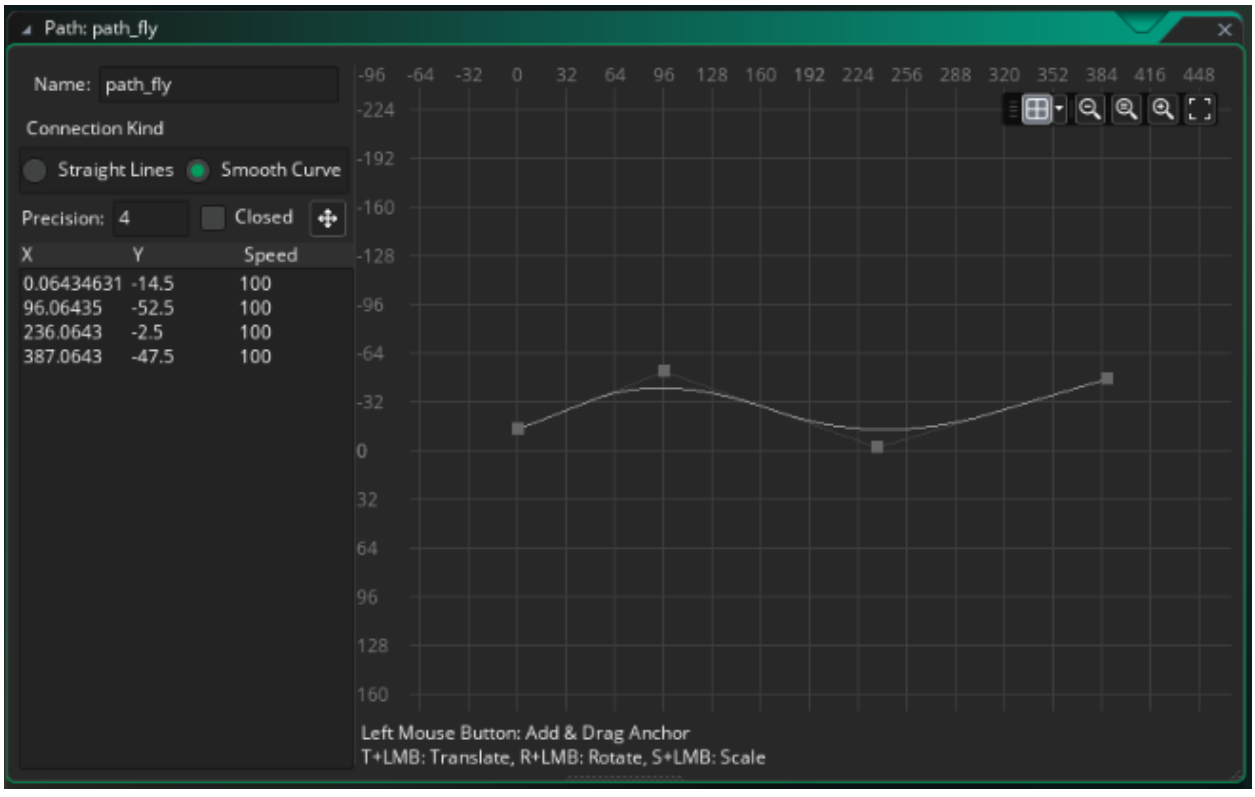


Sl. 2.4. *Primjer niza spriteova i uređivanje spritea*

Set pločica je jedna slika koju će GMS podijeliti u zasebne odjeljke na temelju zadanih vrijednosti (engl. Tile Set). Setovi pločica se uzimaju iz sprite resursa, ali GMS ih gleda kao odvojeni resurs i s njima će upravljati drugačije.Koriste se za dizajniranje bilo kakve statične stvari u našoj igrici kao što su pozadina, teren, zidovi i slično [2].

Zvukovi i zvučni efekti se nalaze u „Sounds“ resursima. Zvukovi su jako bitan aspekt naše igre. GMS prihvaća WAV, OGG i MP3 formate. Zvukovi se također mogu dodatno dorađivati sa ugrađenim alatom „Sound Editor“ a mogu se i uvoziti isto kao i spriteovi.

Putanje odnosno „Paths“ nam koriste da napravimo putanju koju će instanca nekog objekta pratiti kroz igru [3]. Koristeći „Path Editor“ (Sl.2.5.) možemo jednostavno napraviti putanju a zatim je dodijeliti instanci objekta. Putanju možemo nacrtati ili je dobiti pisanjem koordinata.



Sl. 2.5. *Kreiranje putanje neprijatelja „Muha“ u „Path Editoru“*

Skripte su funkcije koje mi sami pišemo. Skripte se najčešće koriste kako bi nam kod bio čistiji i organiziraniji. Kao primjer možemo uzeti situaciju kada ubijemo neprijatelja i on treba izbaciti novce. Umjesto pisanja koda za izbacivanje novaca u svaki objekt koji to treba, lakše nam je to napisati u skriptu, a zatim tu skriptu pozivati iz objekta.

```
case playerState.moving: script_execute(scr_player_moving);
                         break;
```

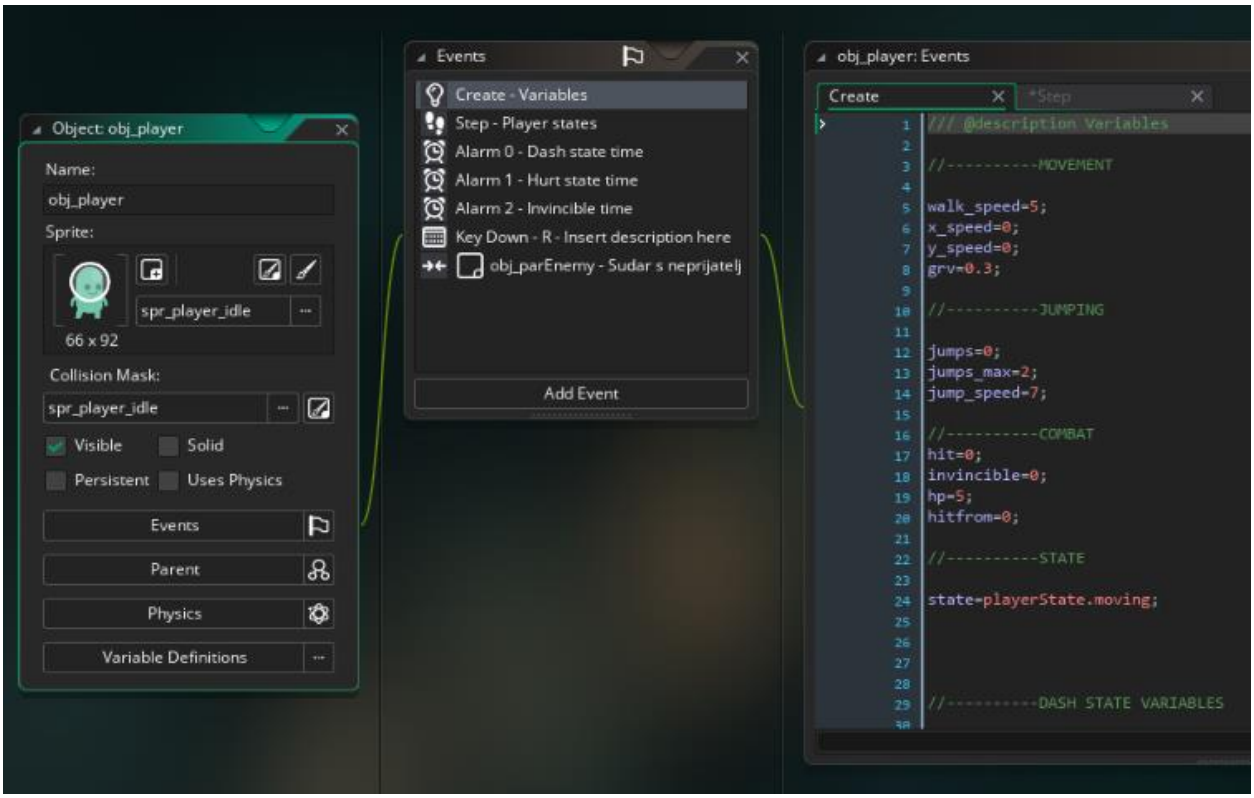
Sl. 2.6. *Primjer pozivanja skripte*

Shaderi su jako dobar alat koji nam služi za manipuliranje spriteovima. Pomoću shadera možemo stvarati efekte [5]. Za primjer možemo uzeti neprijatelja kojega kada pogodimo, dogodi se blijesak. Shaderi se pokreću direktno na grafičkoj kartici. Oni su jako kompliciranite za njihovo korištenje treba puno proučavanja i iskustva.

U font resurse stavljamo naše fontove koji će se koristiti u igrici. Ako nemamo svojih fontova GMS kao standard koristi Arial font. Također imamo i „Font Editor“ koji se koristi za uređivanje fontova.

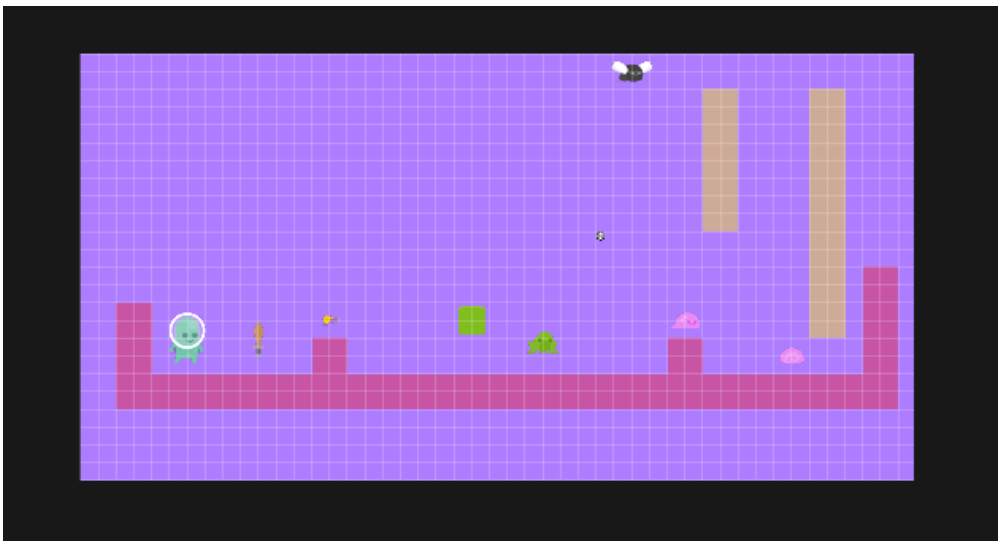
Vremenske trake odnosno „Timelines“ i „Timeline editor“ koristimo za određivanje događaja koji će se dogoditi u nekom specifičnom vremenu u našoj igri [3].

Objekti su posebni resursi kojima kontroliramo našu igru i koji rade specifične stvari. Objekti mogu imati različita ponašanja i mogu reagirati na različite događaje. Na slici 2.7. vidimo primjer objekta te događaje na koje on reagira i njegove varijable. Objektima možemo pridruživati različita svojstva i varijable.



Sl. 2.7. *Objekt glavnog lika i njegovi događaji na koje reagira*

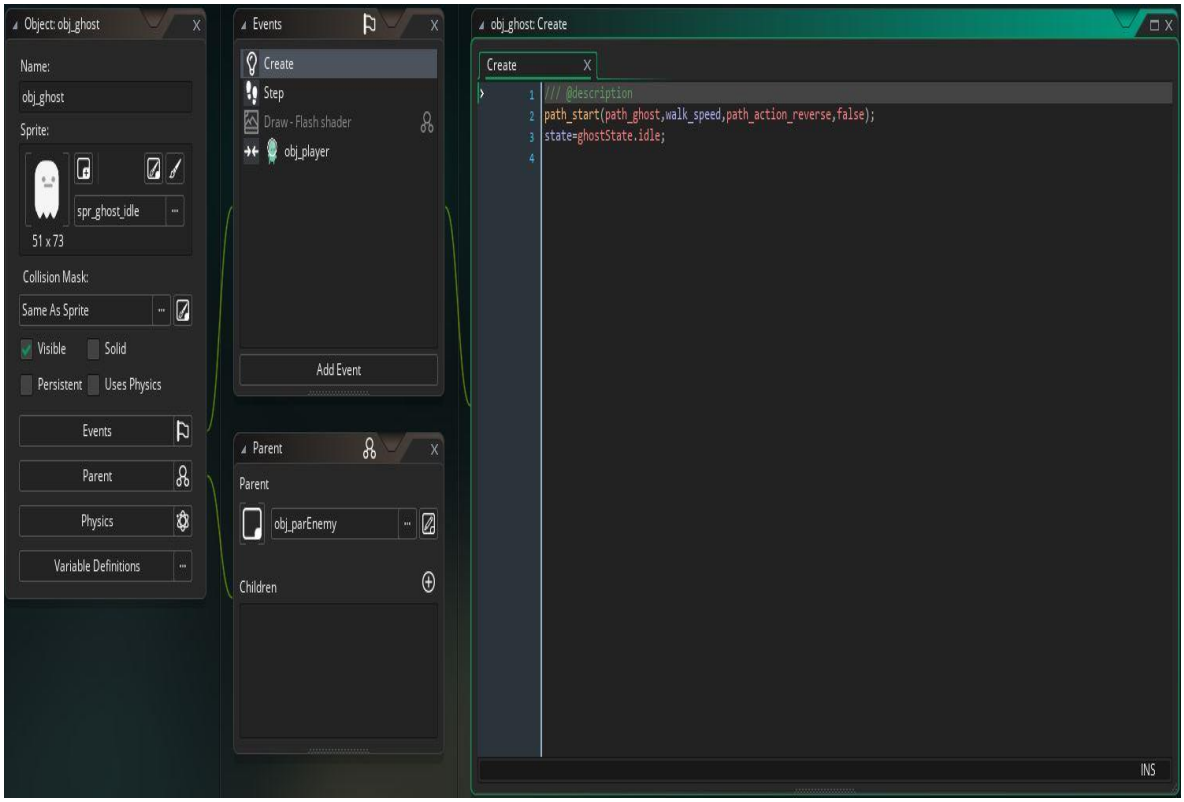
Svaka igra koju razvijamo u GMS-u mora sadržavati barem jednu sobu kako bi je pokrenuli. Soba je zapravo prostor gdje postavljamo instance naših objekata i gdje se odvija akcija u našoj igrici [3]. Sobe uređujemo preko alata „Room Editor“. Preko njega možemo sređivati poglede na sobu, dodavati pozadine, kreirati druge soba i još puno toga. To je jedan od najmoćnijih alata u GMS-u.



Sl. 2.8. *Primjer sobe u GMS-u*

3. GML

Gamemaker Language (kasnije u tekstu skraćeno GML) je programski jezik koji se koristi u GMS-u. Koristi se za razvoj igara te se dodaje objektima unutar „Object Editora“. Namijenjen je korisnicima s više iskustva i nudi više fleksibilnosti pri izradi računalnih igara pisanjem programskog koda. Također GML se može koristiti paralelno sa „Drag and Drop“ metodom. Cijelu logiku napravljenu „Drag and Drop“ metodom GMS može automatski pretvoriti, odnosno prepisati u GML programski kod. Na slici 3.1. se vidi objekt s njegovim događajima i otvorenim uređivačem koda. Svaki događaj ima vlastitu karticu koda u „Object Editoru“ gdje se može dodavati, uređivati ili brisati kod u bilo kojem trenutku. Kod mora imati osnovnu strukturu i može sadržavati različite resurse, varijable, funkcije, izraze i slično.

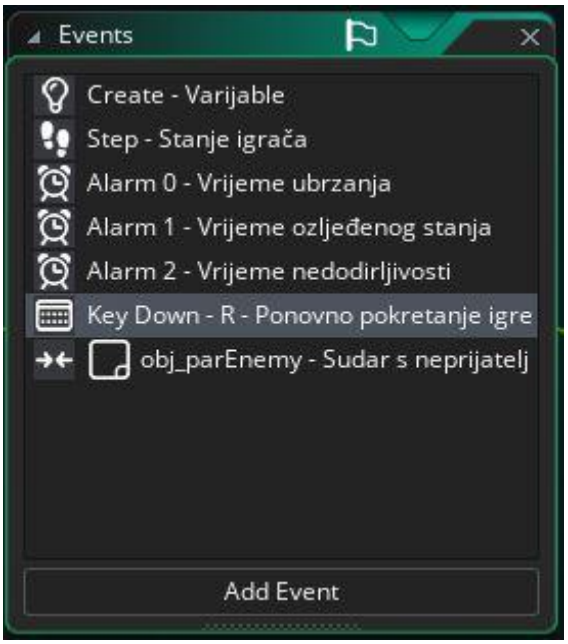


Sl. 3.1. *Objekt duh, njegovi događaji i uređivač koda*

3.1. Događaji

Za korištenje GML-a moraju se razumjeti događaji. Kada se napravi objekt, njemu se dodjeljuju događaji. Svaki događaj koji se doda objektu ima vlastitu karticu koda gdje se može dodavati, uređivati ili brisati kod u bilo kojem trenutku. Događaji se međusobno razlikuju te pokreću kod u različitim slučajevima zbog čega je potrebno biti dobro upoznat s događajima. Događaja u GMS-u ima puno zato su opisani samo oni najvažniji te oni koji su korišteni u izradi igre.

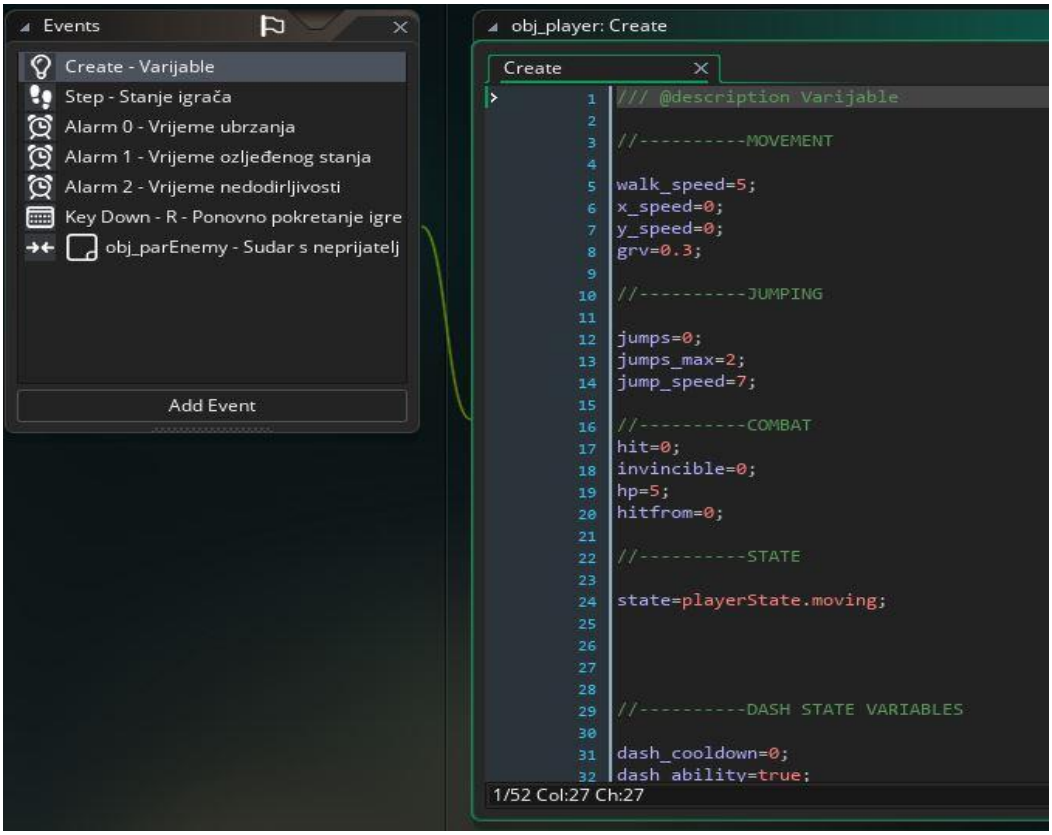
GMS ima dvije glavne kategorije događaja. Prva kategorija događaja izvodi kod svaki pojedini korak igre, a u drugoj kategoriji kod je aktiviran događajima u igri kao što su klik mišem ili instanca koja se sudarila s drugom instancom. Mogu se i imenovati događaji, odnosno dodavati komentari koji će se prikazivati u „Event Editoru“. To služi za lakše snalaženje u kodu jer se vrlo lako može izgubiti među događajima. Na slici 3.2. prikazan je „Event Editor“ u kojem su imenovani pojedini događaji.



Sl. 3.2. *„Event Editor“ sa imenovanim događajima*

Također bitno je znati da se događaji u GMS-u izvode različitim redoslijedom. Kod događaja koji se izvode svaki korak igre nemoguće je točno znati redoslijed izvođenja jer ovisi o unutarnjem radu GMS-a koji se mijenja ovisno o tome kako se softver razvija. Međutim, postoje određeni događaji koji će se uvijek izvoditi istim redoslijedom.

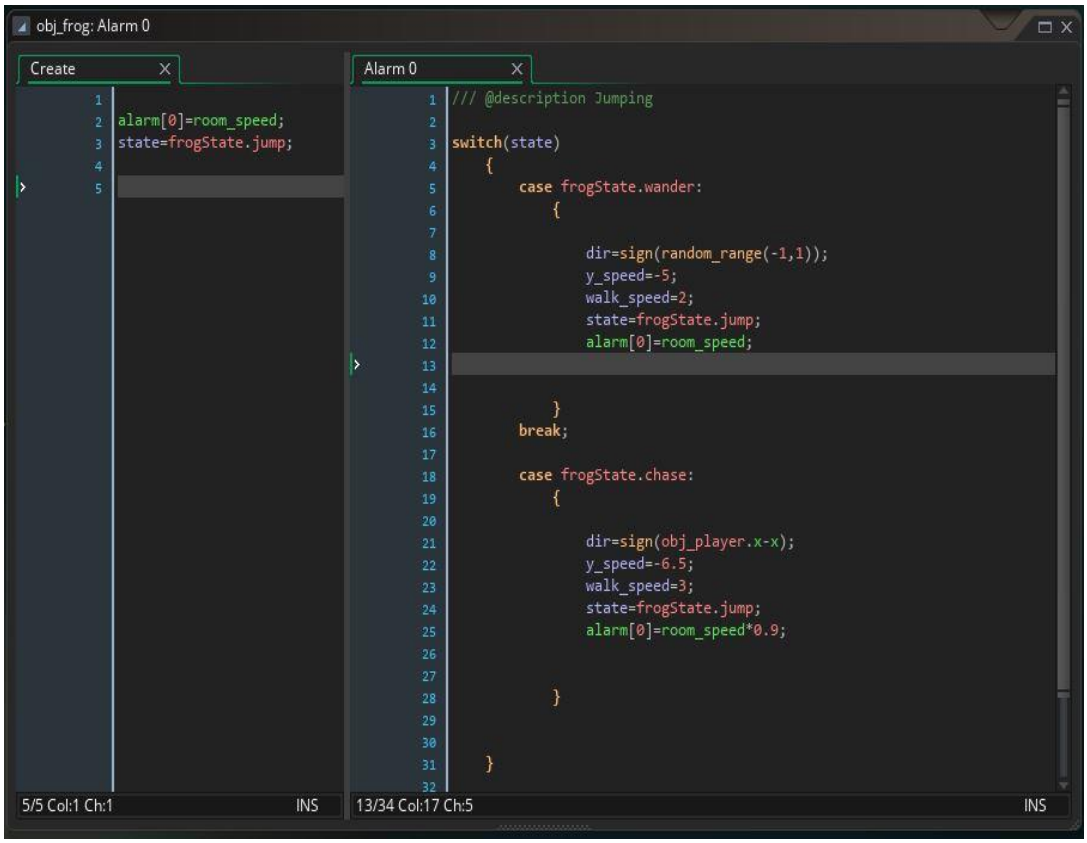
„Create Event“ je događaj koji se pokreće kada je stvorena instanca objekta [3]. To je prvi događaj koji se izvodi unutar instance kada se uđe u sobu u kojoj se nalazi ta instanca. U „Create“ događaju se inicijaliziraju varijable koje definiraju objekt, postavljaju putanje, pokreću vremenske trake itd. U „Create Event“ se zapravo stavlja samo ono što se treba dogoditi jednom nakon što je instanca objekta stvorena u sobi.



Sl. 3.3. *Primjer „Create“ događaja i njegovog koda*

„Destroy“ je događaj koji izvodi svoj kod nakon što je instanca objekta uništena. Na primjer nakon uništavanja instance glavnog lika, igra se ponovno pokreće.

Alarmi su posebni događaji koji ne čine ništa, osim ako je alarm prethodno bio postavljen. Kada je alarm postavljen, on će se odbrojavati do nule a zatim će se pokrenuti kod unutar tog alarma. Svaki objekt može imati do dvanaest alarma. Kada alarm dođe do nule i kod se izvede, alarm se postavlja na minus jedan i ostat će na toj vrijednosti sve dok ponovno ne postavimo alarm. Alarmi su jako korisni za događaje koji se ponavljaju. Na primjer u igri postoji neprijatelj koji ispucava metke svake sekunde. Tada je u „Create“ događaju postavljen alarm, a u samom alarmu je kod za ispucavanje metaka i ponovno postavljanje alarma na željenu vrijednost. Na slici 3.4. prikazan je primjer korištenja alarma. U „Create“ događaju alarm nula postavljen je na brzinu sobe, a u alarmu je kod za skakanje objekta žabe te ponovno postavljanje alarma na brzinu sobe. U ovom slučaju žaba će skočiti svake sekunde.



Sl. 3.4. *Primjer korištenja alarma unutar objekta žabe*

„Step Event“ je događaj koji se izvodi svaki korak igre sve dok instanca postoji [3]. GMS dijeli vrijeme u korake zajedno s brzinom sobe te tako određuje koliko koraka treba biti u sekundi. Taj događaj je jako koristan za bilo kakve petlje ili ako je potrebno kontinuirano izvođenje koda u igri. „Step“ događaj se sastoji od tri poddogađaja: „Begin Step“, „Step“ i

„End Step“. Ti poddogađaji omogućuju više kontrole nad time koji će se kod kada izvesti. Međutim, standardni „Step“ događaj je dovoljan za većinu slučajeva. Potrebno je obratiti pozornost kada koristimo „Step“ događaj zato što velika količina koda u „Step“ događaju može znatno usporiti igru pogotovo ako postoji više instanca objekata u sobi koje izvode „Step“ događaje. Na slici 3.5. nalazi se primjer „Step“ događaja u kojem se kontroliraju stanja glavnog lika, te se prema tome pozivaju skripte s kodom koje opisuju to stanje. Takvim načinom korištenja „Step“ događaja igra neće biti usporena zato što je kod stanja glavnog lika raspodijeljen u skripte umjesto da je napisan u „Step“ događaju što bi možda utjecalo na kvalitetu izvođenja igre. GMS-u je lakše svaki korak provjeravati deset linija koda te pozivati potrebne skripte, nego da svaki korak igre provjerava tisuće linije koda za svaku instancu objekta u sobi.

obj_player: Step

Step

```
1  /// @description Stanje igrača
2
3  switch(state)
4      {
5          case playerState.moving: script_execute(scr_player_moving);
6                                   break;
7
8          case playerState.duck: script_execute(scr_player_duck);
9                                 break;
10
11         case playerState.dash: script_execute(scr_player_dash);
12                                break;
13
14         case playerState.onWall: script_execute(scr_player_onWall);
15                                  break;
16
17         case playerState.gun: script_execute(scr_player_gun);
18                               break;
19
20         case playerState.sword: script_execute(scr_player_sword);
21                                 break;
22
23         case playerState.hurt: script_execute(scr_player_hurt);
24                                break;
25
26
27     }
28
29
30
```

INS

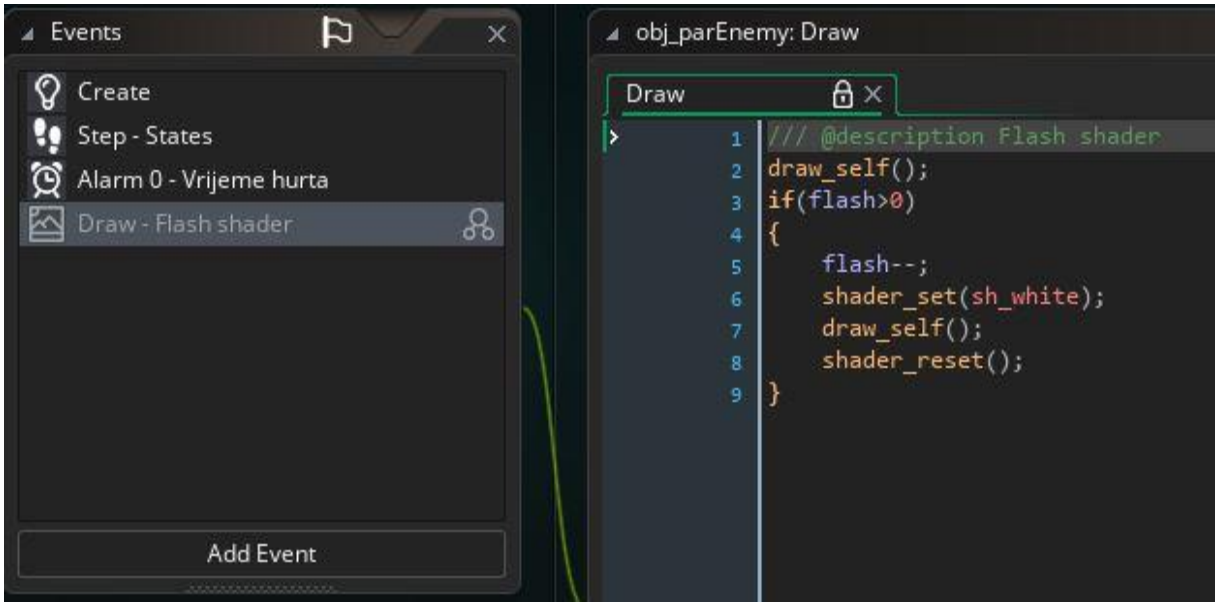
Sl 3.5. *Kartica „Step“ događaja sa kodom*

U gotovo svim igrama instance objekata se dodiruju odnosno sudaraju jedne s drugima. U GMS-u ima događaj za to koji se zove „Collision“. Unutar ovog događaja se definira s kojim drugim objektom treba provjeravati sudare, te kada se ti objekti sudare izvesti će se kod unutar tog događaja. Za primjer se može uzeti objekt glavnog lika. U njemu je definiran „Collision“

događaj koji provjerava sudar s instancom objekta roditelja neprijatelja. Tada će se u slučaju sudara glavnog lika s bilo kojim neprijateljem izvesti kod unutar „Collision“ događaja kao na primjer smanji broj životnih bodova glavnog lika.

„Keyboard“, „Keyboard Press“, „Keyboard Release“ i „Mouse“ događaji služe za upravljanje događajima koji koriste tipkovnicu i miš. Na primjer objektu glavnog lika se može pridružiti događaj pritiska lijevog klika miša u kojem je napisan kod za ispucavanje metaka. Tada će se nakon svakog pritiska lijevog klika izvršiti kod unutar toga događaja, odnosno stvorit će se instanca objekta metak.

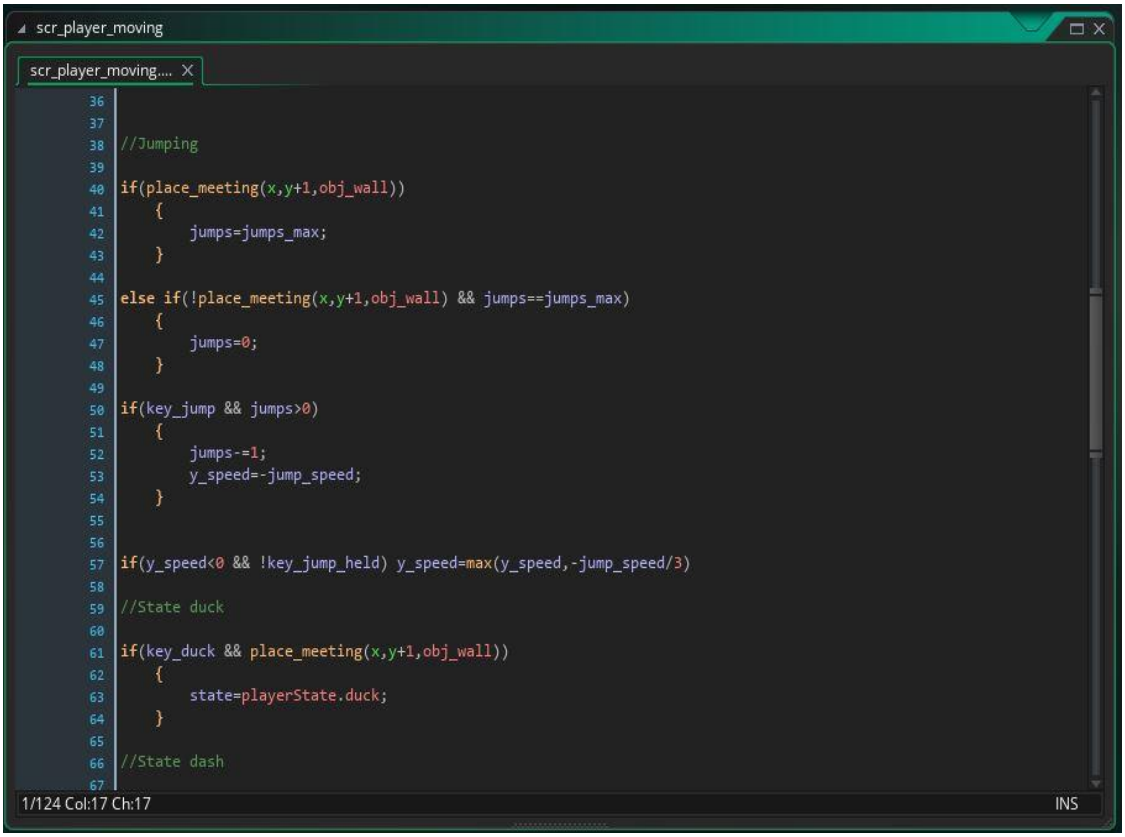
„Draw Event“ je događaj koji upravlja onim što se vidi na zaslonu kada se pokrene igra. Kao i „Step“ događaj „Draw“ također ima poddogađaje. Glavni „Draw“ događaj uvijek će se pozvati za svaku instancu bez obzira ima li ta instanca sprite ili ne. Ako se instanca objekta obilježi kao nevidljiva, tada „Draw“ događaj neće biti aktiviran. Zato se mora pripaziti jer ako postoji kod u „Draw“ događaju on se u tom slučaju neće izvesti. Na slici 3.6. je primjer „Draw“ događaja koji nakon što je varijabla flash instance neprijatelja veća od nula, postavlja shader u sh_white te zatim crta instancu objekta. Drugim riječima kada je neprijatelj ozlijeđen njegov sprite će kratko vrijeme biti bijele boje, a zatim će se vratiti u originalno stanje.



Sl. 3.6. *Primjer „Draw“ događaja i njegovog koda*

3.2. Osnove GML-a

Nakon događaja GMS-a, moraju se znati i kakve sve mogućnosti pruža GML te njegovu strukturu i sintaksu. Svaki blok koda sastoji se od izraza koje GMS prevodi i koristi kako bi se nešto dogodilo unutar igre. Izrazi mogu biti od jednostavnog definiranja varijable pa sve do korištenja kompliciranih funkcija. Izraze treba odvojiti sa znakom „ ; “ kako bi se izbjegle pogreške s deklaracijom varijabli te kako bi kod bio čitak. Na slici 3.7. je primjer bloka koda unutar skripte koji sadrži različite izraze.



Sl. 3.7. *Blok koda unutar skripte*

Pisanje komentara u projektima je jako bitno. Komentari unutar GMS-a se mogu pisati na različite načine ali osnovni je stavljanje dva znaka kose crte „ // “ te nakon toga komentar. Komentiranjem se pišu bilješke sebi i kolegama s kojim se radi, objašnjavaju se dijelovi koda, ostavljaju se podsjetnici što koji dio koda čini i još puno toga. Na slici 3.7. prikazano je korištenje komentara koji opisuje što koji dio koda radi. Tako se ne mora čitati cijeli kod i razmišljati zašto je on napisan i što radi, nego se jednostavno pročita komentar.

Blokovi koda sastoje se od izraza. Izrazi mogu sadržavati brojeve, varijable, stringove, GML funkcije, ali mogu sadržavati i različite operatore kao što su zbrajanje, oduzimanje, dijeljenje itd. Za izraze postoje operatori koji su prikazani u tablici 3.1. po redoslijedu prioriteta. Kada se radi više operacija u jednom izrazu vrlo je bitno koristiti zagrade kako bi se izdvojio redoslijed operacija budući da ih različite platforme mogu izvoditi drugačije.

NAZIV	**OPERATORI**	**OPIS**
Dodjeljivanje	=	Koristi se za dodjelu vrijednosti varijabli.
Kombiniranje	&&, ||, ^^	Kombiniranje bool vrijednosti kako bi dobili „true“ ili „false“.
Uspoređivanje	<, <=, ==, !=, >, >=	Uspoređivanje koje daje „true“ ili „false“.
Operacije bitovima	|, &, ^, <<, >>	Obavljanje operacija bitovima.
Aritmetički operatori	+, -, *, /	Zbrajanje, oduzimanje, množenje i dijeljenje.
Increment / Decrement	++, --	Povećanje i smanjivanje.
Podjela i modulo	div, %, mod	Div daje iznos koji se može podijeliti u proizvodnju kvocijenta cijelog broja, dok mod daje samo ostatak dijeljenja.
„Unary“ operatori	!, -, ~	„ !“ je „not“ npr. „!true “ je isto što i „false“. „-“ negira vrijednost iza znaka. „~“ negira vrijednost bitno.

Tab. 3.1. *Operatori, njihove oznake i opis*

Funkcije u GML-u imaju ulaz i izlaz, a izlaz je na neki način povezan s ulazom. Funkcija se sastoji od imena te argumenata unutar zagrade odvojeni zarezima. Naravno, funkcija ne mora imati argumente. U GMS-u postoje dvije vrste funkcija. GMS sadrži veliku zbirku ugrađenih funkcija koje se mogu koristiti pri razvoju igre i kontroliranju njenih

aspekta. Neke funkcije vračaju vrijednost dok neke izvode naredbe. Postoje i skripte koje korisnik sam piše i koje se mogu koristiti kao funkcije. Skripte se najčešće koriste kako bi nam kod bio čistiji i organiziraniji.

3.3. Varijable i tipovi podataka

GML kao osnovnu jedinicu za programske operacije koristi varijable. Varijablama se pridružuju vrijednosti koje se pohranjuju u memoriju za kasniju ili trenutačnu upotrebu. Svakoj varijabli se treba dati naziv kako bi se moglo odnositi na njih u daljnjem radu. Naziv varijable u GML-u mora početi sa slovom i može sadržavati samo slova, brojeve i znak „ _“, te ne smije biti dulji od 64 simbola.

GMS ima četiri glavne kategorije varijabli od kojih svaka ima svoj vlastiti opseg odnosno „scope“.

Varijabla instance je kreirana unutar instance objekta i ona je jedinstvena toj instanci. Više instanca istog objekta mogu imati iste varijable, ali te varijable mogu sadržavati različite vrijednosti. Na primjer, mogu postojati dva identična neprijatelja koji imaju iste varijable, no kod jednog je brzina veća nego kod drugog.

Lokalne varijable se koriste samo za određeni događaj, a kada taj događaj završi varijabla se odbacuje. One se mogu koristiti samo unutar instance u kojoj su deklarirane. U GML-u lokalnim varijablama se ispred imena dodaje var. Lokalne varijable omogućuju da kod bude čist i organiziran. S obzirom na to da varijable zauzimaju memorijski prostor, korištenjem lokalnih varijabla se održava memorijski prostor optimiziranim za stvari kojima je prostor stvarno potreban.

Globalne varijable su one varijable koje nakon što su deklarirane ne pripadaju ni jednoj instanci posebno, ali svi im mogu pristupiti. Globalna varijabla će ostati u memoriji do kraja igre. Na primjer, ako se u igri mora pratiti broj novaca koje glavni lik ima, jedan način je da se deklarira globalna varijabla koja će spremati vrijednost trenutnog broja novaca kroz cijelu igru.

GML sadrži i ugrađene varijable koje su pridružene objektima i sobama u igri. Primjer nekih ugrađenih varijabli su „sprite_index“ koja sadrži trenutni indeks spritea instance, „speed“ koja definira koliko će se piksela instanca pomjeriti svaki korak igre, „room_speed“

koja sadrži vrijednost brzine rada svih soba u igri itd. Uz njih GML podržava i konstante koje nakon što su deklarirane i dodijeljena im je vrijednost, više ne mogu biti izmijenjene. Na slici 3.8. je primjer deklariranja varijable instance, globalne i lokalne varijable, konstante, te korištenje ugrađene varijable.

```
zivoti=1; // Varijabla instance
var srce=1; // Lokalna varijabla
global.novac=0; // Globalna varijabla
#macro broj_metaka=10; // Konstanta
speed=0; //Korištenje ugrađene varijable
```

Sl. 3.8. *Deklariranje varijabli različitih opsega*

GML podržava različite tipove podataka koje se mogu spremati u varijable. Tipovi podataka, njihov naziv, primjer i kratki opis se nalazi u tablici 3.2..

TIP PODATKA	PRIMJER	OPIS
String	Ime = „matej“;	Tekst koji je stavljen unutar znakova navodnika.
Realan	Godine = 22;	Svi realni brojevi.
Polje	ime_polja = array[50];	Tip podatka koji može sadržavati više vrijednosti
Boolean	Istina = false;	Tip podatka koji može biti 1 ili 0, odnosno „true“ ili „false“.
Pokazivač	ptr(n);	Tip podatka koji pokazuje na memorijsku lokaciju.
Enum	enum boje{plava,crvena};	Koristi se za kreiranje vlastitog tipa podatka.
Nedefiniran	Prezime= null;	Sve izrazi koji nemaju točnu vrijednost vračaju vrijednost „null“.

Tab. 3.2. *Tipovi podatka, njihov primjer i opis.*

GMS omogućuje provjeru vrste podataka dane varijable sa sljedećim ugrađenim funkcijama: is_string, is_bool, is_array, is_real itd. U GML-u su također definirane ključne riječi koje se moraju izbjegavati pri imenovanju varijabli. Neke od njih su: self, other, noone, all itd.

3.4. GML naredbe

GML ima brojne naredbe koje se više puta koriste prilikom pisanja koda. To su korisne naredbe koje stvaraju više mogućnosti i slobode pri pisanju koda i razvoju igara. Zato što su naredbe brojne, opisane su samo one koje su korištene pri razvoju igre.

If naredba omogućava jednostruka i višestruka grananja u programu. Provjerava uvjet unutar zagrada, te ako je uvjet točan izvodi se kod u suprotnom kod se neće izvest. If naredba može se granati s else naredbom.

```
if (<uvjet>)
   {
   <izraz>
   }
else
   {
   <izraz>
   }
```

Sl. 3.9. *If naredba*

While je petlja koja će izvoditi izraz sve dok je uvjet unutar zagrada točan. Potrebno je biti oprezan kada se koristi while naredba jer je moguće napraviti beskonačnu petlju što će dovesti do rušenja igre.

```
while (<uvjet>) <izraz>
```

Sl. 3.10. *While petlja*

For petlja je izuzetno korisna za obavljanje zadataka koji se ponavljaju. Korištenjem for petlje znatno se može smanjiti broj linija koda i uštedjeti na vremenu. For petlja se sastoji od početnog stanja, uvjeta i prirasta. Na slici 3.11. prikazan je primjer for petlje. Ovaj izraz će stvarati novu instancu objekta „obj_coin“ sve dok je početno stanje „i“ manje od lokalne varijable „dropRate“, s time da se početno stanje nakon svakog izvršenja povećava za jedan.

```
for(i=0;i<dropRate;i++)
    {
        instance_create_layer(x,y,"layer_player",obj_coin);
    }
```

Sl. 3.11. *Primjer for petlje*

Ako je potrebno da instanca obavi naredbu s obzirom na vrijednost varijable, može se koristiti switch izraz. Naravno to je moguće napraviti i s if grananjem, ali to bi bilo više koda i može se zapetljati. Na slici 3.12. je prikazan primjer switch izraza. Switch naredba prvo provjerava uvjet, u ovom slučaju je to vrijednost varijable „state“. U slučaju da je vrijednost varijable „state“ jednaka „flyState.wander“ izvest će blok koda unutar tog „casea“ odnosno izvršit će se skripta „scr_fly_wander“. Naredbom break switch se završava ranije i nastavlja se izvršavati kod poslije izraza switch.

```
switch(state)
    {
        case flyState.wander: script_execute(scr_fly_wander); break;

        case flyState.shoot: script_execute(scr_fly_shoot); break;
    }
```

Sl. 3.12. *Primjer switch izraza*

U igri ima više instanca objekta neprijatelja pčela u jednoj sobi. Kada se pčela udari s oružjem potrebno je da se samo toj instanci smanje životni bodovi. To omogućuje with izraz.

```
with(other)
    {

        if(invincible==0)
        {
            hit=1;
            hp=hp-obj_gun.dmg;
            flash=3;
            hitfrom=other.direction;
        }

    }
```

Sl. 3.13. *Primjer with izraza*

Ključna riječ other omogućuje upravo ono navedeno. Oružje će skinut životne bodove samo onoj instanci koja je trenutno udarana. U slučaju da je napisan izraz with(obj_bee), udarcem u jednu instancu pčele svakoj instanci pčele bi se smanjili životni bodovi.

4. RAZVOJ 2D PLATFORMSKE IGRE

Ideja igre je da igrač mora prikupljati novce kako bi prelazio razine i tako došao do kraja igre. Igrač mora koristiti oružja kao što su mač i pištolj kako bi savladao brojne neprijatelje. Kada je neprijatelj savladan on će izbaciti novce. Inspiriran igrama kao što su „Hollow Knight“ i „Ori and the blind forest“, većina igre temelji se na korištenju mehanika glavnog lika i savladavanja njegove kretnje. Glavni lik ima mogućnosti kao što su dupli skok, skok od zida i ubrzanje. Ako igra ima jedinstvene mehanike čije se funkcije malo preklapaju, igraču će to biti zanimljivo i imat će razlog za povratak [6]. Razine su dizajnirane tako da igrač mora kombinirati sve mehanike kako bi savladao tu razinu, a pri tom mora biti oprezan jer ima i neprijatelja. U početku igrač može samo skakati, a kako napreduje kroz razine uči nove mehanike. Sva grafika je preuzeta sa opengameart.org stranice koja je po potrebi uređivana paint.net alatom [7][8].

4.1. Glavni lik

Kako se igra temelji većinom na korištenju mehanika glavnog lika i savladavanja njegove kretnje prvo je napravljen glavni lik. Glavni lik ima animacije za mirovanje, skakanje, kada je ozlijeđen, kada je na zidu i za trčanje. Animacije su prikazane na slici 4.1..

Sl. 4.1. *Animacije glavnog lika*

Kada su sređeni spriteovi i maska sudara za glavnog lika, napravljen je njegov objekt. „Collision mask“ odnosno maska sudara je područje u kojem GMS registrira sudare tog objekta. Maska sudara glavnog lika namještena je na animaciju kada je glavni lik u mirovanju. Zatim su napisane dvije skripte koje su često korištene u razvoju igre. To su skripte za ulaz s miša i tipkovnice i skripta za kretanje i provjeravanje sudara sa zidom. Razlog stavljanja toga u skripte je taj što svi neprijatelji koji su napravljeni također koriste istu skriptu za kretanje i

provjeru sudara sa zidom. Ovim načinom je moguće pozivati napisanu skriptu umjesto da se stalno piše isti kod. Skripta za ulaz sastoji se od ugrađenih funkcija kao što su keyboard_check() i mouse_check() koje provjeravaju je li tipka s miša ili tipkovnice pritisnuta. Te vrijednosti su pridružene varijablama.

scr_get_input

scr_get_input.gml

```
//----------BASIC MOVEMENT
key_left=keyboard_check(ord("A"));
key_right=keyboard_check(ord("D"));
key_jump=keyboard_check_pressed(vk_space);
key_jump_held=keyboard_check(vk_space);
key_duck=keyboard_check(vk_control);
key_down=keyboard_check(ord("S"));

//----------DASH
key_dash=keyboard_check_pressed(vk_shift);

//----------ATTACK
key_attack=mouse_check_button(mb_left);
key_attack_sword=mouse_check_button_pressed(mb_left);


//----------CYCLE WEAPONS
key_sword=keyboard_check_pressed(ord("1"));
key_gun=keyboard_check_pressed(ord("2"));
key_noWeapon=keyboard_check_pressed(ord("0"));
```

Sl. 4.2. *Skripta za ulaz podataka*

Skripta za kretanje i provjeru sudara koristi ugrađenu funkciju place_meeting(). Logika za provjeru sudara je ta da preko funkcije place_meeting() provjeravamo je li ispred, iznad ili ispod lika postoji objekt zida. U slučaju da postoji, brzina glavnog lika postavlja se na nula.

```
//----------VERTICAL COLLISION
for(i=1;i<=100;i++)
{
    if(place_meeting(x,y+y_speed/i,obj_wall))
    {
        while(!place_meeting(x,y+sign(y_speed),obj_wall))
        {
            y=y+sign(y_speed);
        }
        y_speed=0;
    }
}


//Vertical movement
y=y+y_speed;
```

Sl. 4.3. *Provjera vertikalnog sudara korištenjem funkcije place_meeting() i vertikalno kretanje*

Objekt glavnog lika sastoji se od četiri događaja. U „Create“ događaju definirane su sve varijable za lika. Neke od njih su brzina hodanja, vertikalna brzina, horizontalna brzina, broj skokova, trenutno stanje lika i brojne druge. Glavni lik je dizajniran prema „finite state machine“ sistemu. To je sustav koji ima ograničen broj stanja rada [9]. Glavni lik ima sedam stanja: kretanje, čučanje, ubrzanje, na zidu, s pištoljem, s mačem i ozlijeđen. Zadano stanje u „Create“ događaju je postavljeno na stanje kretanja, a zatim iz tog stanja glavni lik može prelaziti u druga stanja. Sva stanja su napisana u različitim skriptama, a skripte se pozivaju u „Step“ događaju ovisno o trenutnom stanju glavnog lika. Stanja se mijenjaju ovisno o događajima unutar igre. Na primjer, ako lik trči njegovo stanje bit će kretanje, a kada se pritisne tipka shift ući će u stanje ubrzanja. No, vrijeme u stanju ubrzanja ograničeno je alarmom. Kada alarm odbroji do nule lik se iz stanja ubrzanja vrača u stanje kretanja. Kada je lik ozlijeđen ulazi u ozlijeđeno stanje i postaje nedodirljiv. To je također regulirano alarmima. Objekt glavnog lika u događaju sudar provjerava sudare s roditeljem objekta neprijatelj te u slučaju sudara smanjuje životne bodove glavnog lika.

```
switch(state)
    {
        case playerState.moving: script_execute(scr_player_moving);
                                 break;

        case playerState.duck: script_execute(scr_player_duck);
                               break;

        case playerState.dash: script_execute(scr_player_dash);
                               break;

        case playerState.onWall: script_execute(scr_player_onWall);
                                 break;

        case playerState.gun: script_execute(scr_player_gun);
                              break;

        case playerState.sword: script_execute(scr_player_sword);
                                break;

        case playerState.hurt: script_execute(scr_player_hurt);
                               break;


    }
```

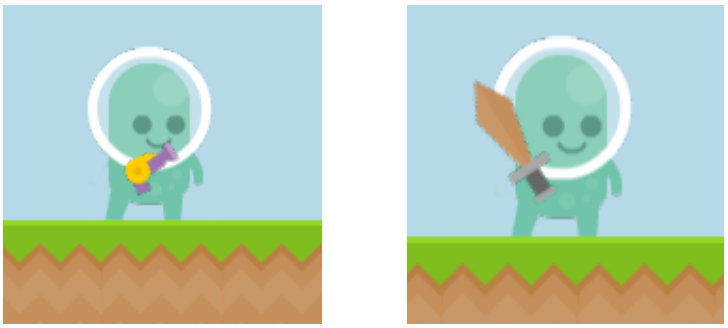
Sl. 4.4. *Reguliranje stanja lika unutar „Step“ događaja*

Na slici 4.5. se nalazi dio skripte stanja kretanja. Možemo vidjeti neke uvjete iz kojih glavni lik prelazi u druga stanja kao što su stanje ubrzanja ili stanje čučanja.

```
scr_player_moving....
59  //State duck
60
61  if(key_duck && place_meeting(x,y+1,obj_wall))
62      {
63          state=playerState.duck;
64      }
65
66  //State dash
67
68  if(key_dash && dash_cooldown<1 && dash_ability==true)
69      {
70          state=playerState.dash;
71          alarm[0]=room_speed/4;
72          dash_cooldown=90;
73      }
74          dash_cooldown--;
75
76  if(dash_cooldown <-100) dash_cooldown=0;
77
78  //State hurt
79  if( hit==1 and invincible==0)
80      {
81          hp--;
82          state=playerState.hurt;
83          alarm[1]=room_speed/4;
84          alarm[2]=room_speed/2;
85      }
86
```

Sl. 4.5. *Dio skripte stanja kretanja*

Glavni lik također može koristiti oružja mač i pištolj nakon što ih pokupi. Mač i pištolj su zasebni objekti. Nakon pritiska tipke „1“ na tipkovnici stvorit će se instanca objekta mača koja će uvijek biti na istom mjestu kao i glavni lik. Istom logikom radi i pištolj. Pritiskom lijevog klika miša mač stvara nevidljivu instancu objekta hitbox koja registrira sudare s maskom sudara neprijatelja, a pištolj stvara instancu objekta metak. Sva oružja prate pokazivač miša.

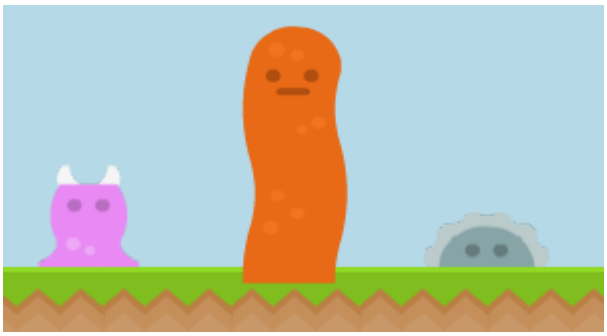


Sl. 4.6. *Glavni lik s pištoljem i mačem*

4.2. Neprijatelji

U igri postoji četrnaest neprijatelja. Svi neprijatelji koriste skriptu za provjeru sudaranja s objektom zida isto kao i glavni lik. Prvo je stvoren objekt roditelja neprijatelja u kojem su definirane varijable koje su svim neprijateljima zajedničke. Neke od tih varijabla su životni bodovi, smjer kretanja, brzina kretanja, jačina udarca i slično. Svi neprijatelji su zapravo djeca objekta roditelja neprijatelja. Neki neprijatelji su slični, odnosno imaju istu logiku, ali se razlikuju po vrijednosti varijabla i maski sudara. Neprijatelji su dizajnirani kao i glavni lik po „finite state machine“ sistemu [9], odnosno imaju različita stanja koja su upravljana unutar „Step“ događaja. Glavni lik od nekih neprijatelja može „odskočiti“, a od nekih ne. Svi neprijatelji imaju svoje animacije ovisno o njihovim stanjima.

Ljepljivac, vruća zmija i polu vrtilica su neprijatelji koji stoje na mjestu. To su jednostavni neprijatelji koji se baziraju samo na maski sudara. Oni su pozicionirani unutar sobe te u slučaju sudara glavnog lika s njima smanjuju se životni bodovi.

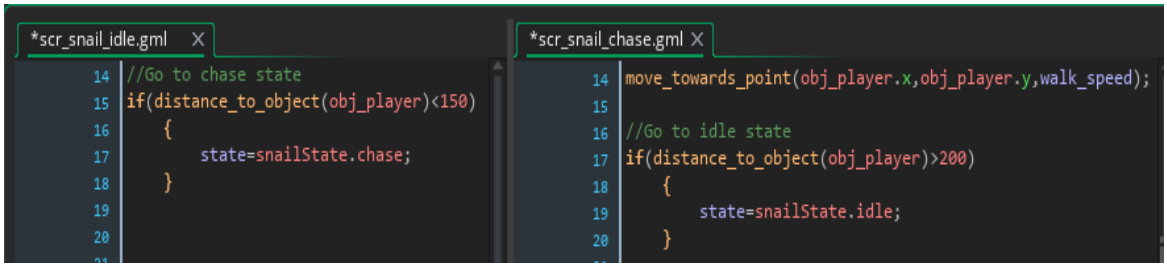


Sl. 4.7. *Ljepljivac, vruća zmija i polu vrtilica*

Slina i vrtilica se kreću lijevo i desno po sobi i kada se sudare sa zidom mijenjaju smjer. Slina se razlikuje od vrtilice po tome što se slina boji visine, odnosno kada dođe do ruba platforme promijenit će smjer dok će vrtilica pasti s platforme. Logika je postignuta ugrađenom funkcijom place_meeting() kojom se provjerava sudar sline ili vrtilice sa zidom te se u slučaju sudara mijenja varijabla za smjer. Kod sline se dodatno provjerava je li ispod nje platforma. U slučaju da nije također se mijenja varijabla smjera.

Sl. 4.8. *Slina i vrtilica*

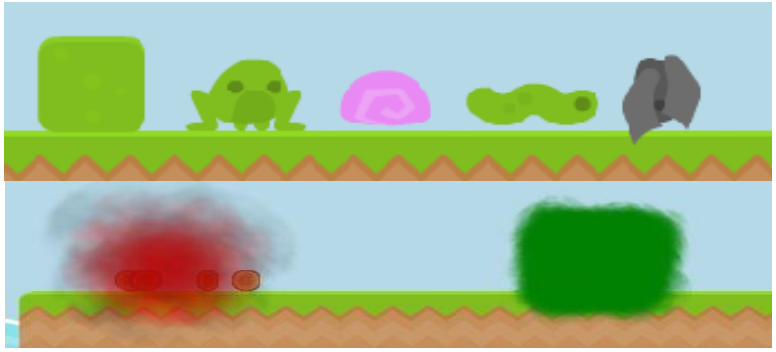
Puž, zmija, kocka, žaba i šišmiš su neprijatelji koji koriste funkcije distance_to_object() i move_towards_point(). Prva funkcija vraća udaljenost između dvije instance, a druga funkcija pomjera instancu do zadane točke. Osnovna ideja je da nakon što se glavni lik približi jednom od ovih neprijatelja, on ih počne pratiti. Na slici 4.9. prikazan je primjer korištenja ove dvije funkcije unutar skripti stanja puža. Puž je prvo u stanju mirovanja. Na lijevoj strani slike funkcijom distance_to_object() se provjerava udaljenost do glavnog lika, te ako je on dovoljno blizu stanje puža se mijenja u praćenje. U skripti za praćenje na slici desno prikazana je funkcija move_towards_point() koja pomjera instancu puža do instance glavnog lika. Ako glavni lik pobjegne dovoljno daleko puž se vraća u stanje mirovanja.



Sl. 4.10. *Skripte stanja mirovanja i praćenja objekta puža*

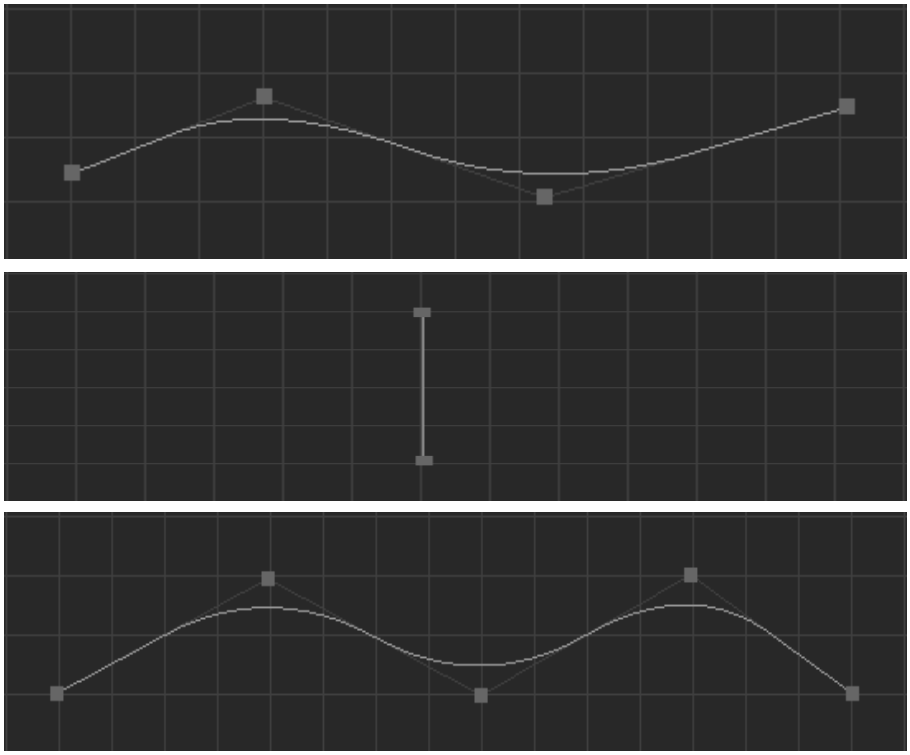
Zmija je napravljena tako da ima zasebni objekt stvarač. Kad se lik približi stvaraču on funkcijom instance_create_layer() stvara instancu objekta zmije koja zatim prati glavnog lika.

Kocka i žaba nakon smrti stvaraju instance objekta hitbox funkcijom instance_create_layer() koje se zadržavaju kratko vrijeme u sobi. Ta instanca može ozlijediti glavnog lika a i druge neprijatelje u slučaju sudara. Ona je nevidljiva ali se kod kocke prikazuje kao eksplozija, a kod žabe kao otrov. Za efekte eksplozije i otrova su korišteni ugrađeni efekti i funkcija effect_create(). Na slici 4.11. gore je prikazana kocka, žaba, puž, zmija i šišmiš, a dolje su prikazani efekti kada kocka i žaba umru, odnosno instance objekta hitbox.



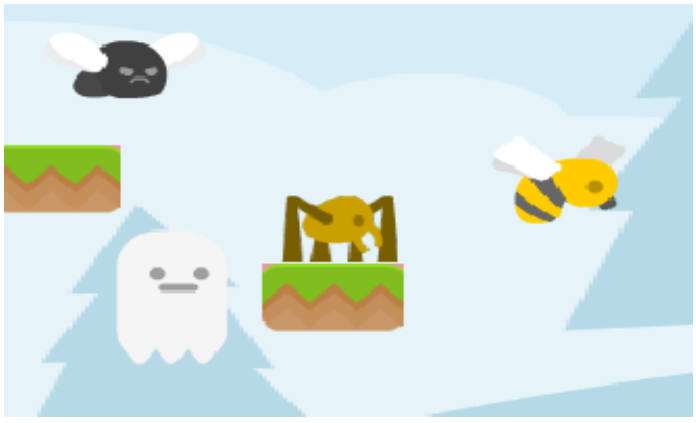
Sl. 4.11. *Kocka, žaba, puž, zmija i šišmiš*

Muha, pčela i duh koriste putanje. Putanje su napravljene u uređivaču putanja unutar GMS-a, a pridružene su im unutar „Create“ događaja funkcijom path_start(). Muha i pčela prate svoje putanje te svakih par sekundi ispucavaju metke na glavnog lika. Razlika je što se instanca metka pčele može zabiti u instancu zida. Nakon što se zabije za par sekundi će stvoriti instancu eksplozije poput one kada se ubije kocka. Duh leti slobodno po sobi dok mu se igrač ne približi nakon čega ga počinje pratiti.



Sl. 4.12. *Putanje muhe, pčele i duha*

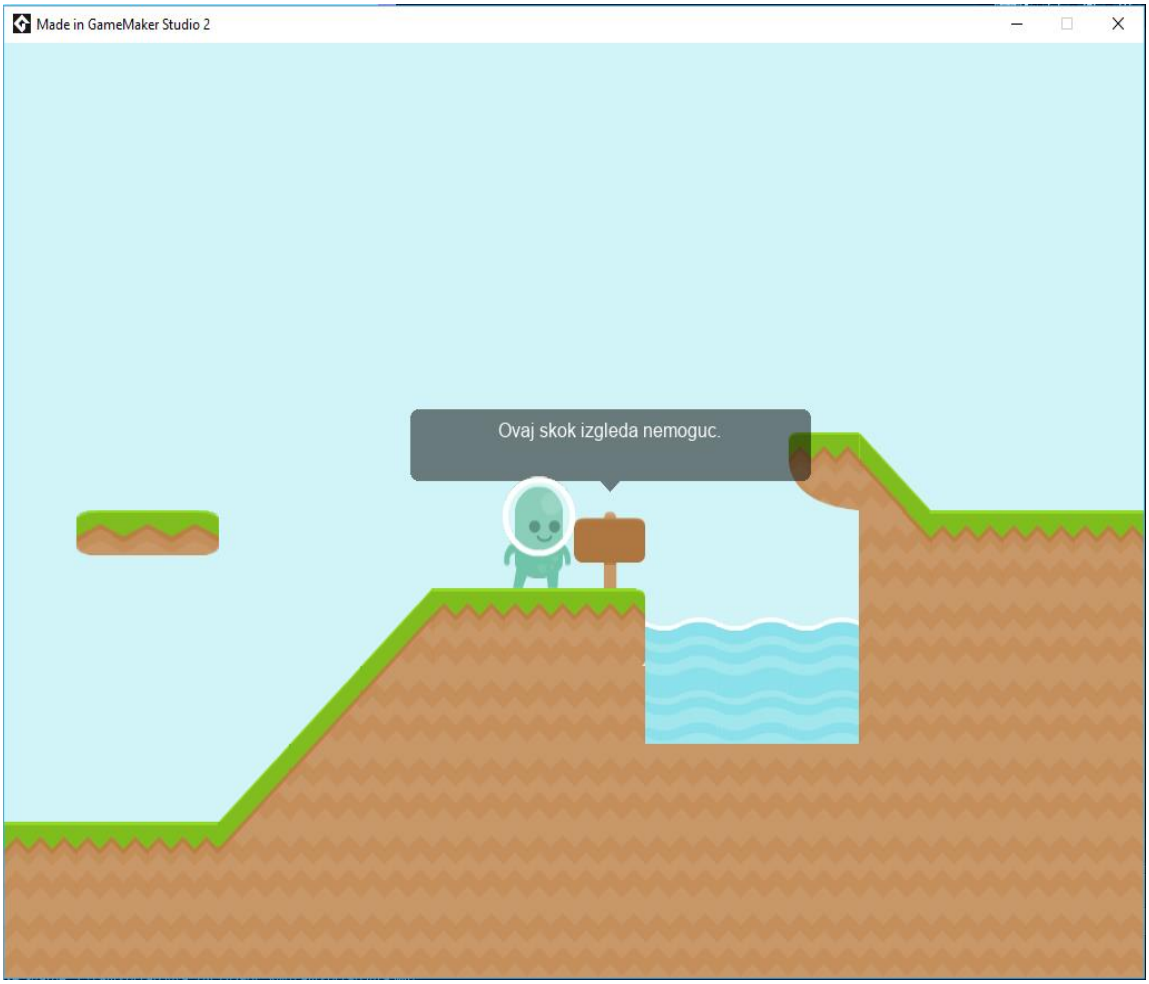
Pauk je neprijatelj koji može hodati u svim smjerovima te poput žabe nakon smrti ostavlja otrov.



Sl. 4.13. *Muha, duh, pauk i pčela*

4.3. Razvoj razina i postavke igre

Nakon što su lik, neprijatelji i svi ostali potrebni objekti razvijeni, počeo je dizajn razina. U postavkama igre podešene su opcije kao što su brzina igre na 60 FPS, skaliranje na zadržavanje omjera te je omogućeno interpoliranje boja između piksela. Sve razine dizajnirane su tako da igrač mora koristiti mehanike lika kako bi prešao na drugu razinu. Razine su bazirane na setovima pločica, odnosno kretanje igrača ograničeno je pločicama [10]. Igrač je uvijek na vrhu određene pločice. Unutar razina postoje znakovi. Desnim klikom na znak stvara se prozor s tekstom. Sadržaj teksta pomaže igraču kako koristiti napade i pokrete, ili daje savjete kako savladati prepreke unutar razine.



Sl. 4.14. *Primjer znaka unutar igre*

Prva soba koju igrač vidi nakon što je igra pokrenuta je naslovna soba. Naslovna soba sadrži animiranu pozadinu koja je sastavljena od više elemenata. Elementi se kreću različitim

brzinama što daje osjećaj dubine prostora. Unutar naslovne sobe imamo izbornik koji se sastoji od tri opcije. Igrač može nastaviti igru od one razine gdje je zadnji put bio, započeti novu igru od prve razine ili izlazak iz igre.

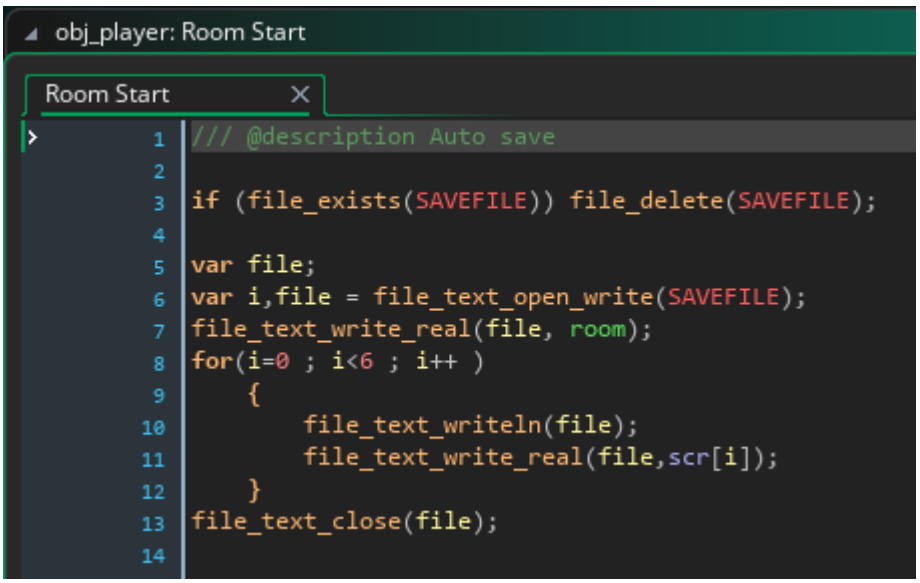
Sl. 4.15. *Naslovna soba igre*

Opcije su spremljene u jednodimenzionalno polje. Takvim pristupom se lagano može dodavati još opcija u izbornik naslovne sobe. Naredbom switch na slici 4.16. se provjerava koja opcija je odabrana i zatim se pokreće odgovarajući kod. Napisana skripta scr_tranzicija se koristi za tranzicije između soba.

```
switch (menu_commited)
    {
        case 2: default: scr_tranzicija(TRANS_MODE.NEXT); break;
        #region CASE 1
        case 0: game_end(); break;
    }
```

Sl. 4.16. *Switch nareba unutar objekta menu*

Kako bi opcija nastavi bila moguća, stvoren je sustav spremanja podataka igre. Sve varijable glavnog lika koje trebaju biti sačuvane, kao što su životni bodovi, naučeni pokreti, broj novaca, broj sobe u kojoj se trenutno nalazimo i slično, su stavljene u zasebni objekt kontrole lika koji postoji od trenutka pokretanja igre pa sve do kraja igre. Sve te varijable su zatim spremljene u jednodimenzionalno polje zbog lakšeg pristupa. Unutar objekta glavnog lika dodan je događaj Room Start koji pokreće kod nakon pokretanja sobe. Događaj je prikazan na slici 4.17..



Sl. 4.17. *Spremanje podataka igre unutar Room Start događaja*

Nakon svakog pokretanja sobe provjerava se je li postoji datoteka „Save.sav“ koja je prethodno definirana. Ako postoji, ta datoteka se briše i otvara se nova datoteka za pisanje. For naredbom prolazi se kroz jednodimenzionalno polje u kojem se nalaze varijable i zapisuju se unutar datoteke svaka u novi red. Kada su svi podaci zapisani datoteka se zatvara.

Kad su potrebne varijable sačuvane u datoteku, prilikom odabira opcije nastavi datoteka se otvara i čitaju se vrijednosti spremljenih varijabli, a zatim se te vrijednosti pridružuju objektu kontrole lika što je prikazano na slici 4.18..

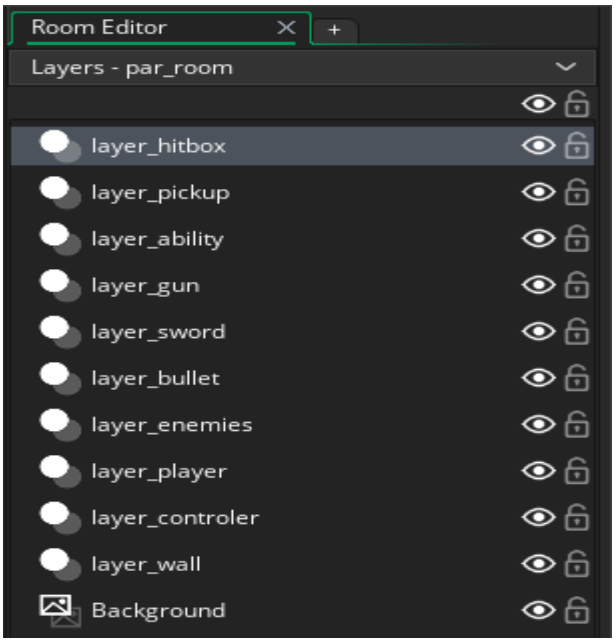
```
case 1:
    {
        if(!file_exists(SAVEFILE))
            {
                scr_tranzicija(TRANS_MODE.NEXT);
            }
        else
            {

                var i, file = file_text_open_read(SAVEFILE);
                for(i=0;i<8;i++)
                    {
                        scr2[i] = file_text_read_real(file);
                        file_text_readln(file);
                    }
                file_text_close(file);
                scr_tranzicija(TRANS_MODE.GOTO,scr2[0]);
                obj_player_control.hp=scr2[1];
                obj_player_control.jumps_max=scr2[2];
                obj_player_control.sword_picked_up=scr2[3];
                obj_player_control.gun_picked_up=scr2[4];
                obj_player_control.dash_ability=scr2[5];
                obj_player_control.wall_jump=scr2[6];

                }
    } break; #endregion
```

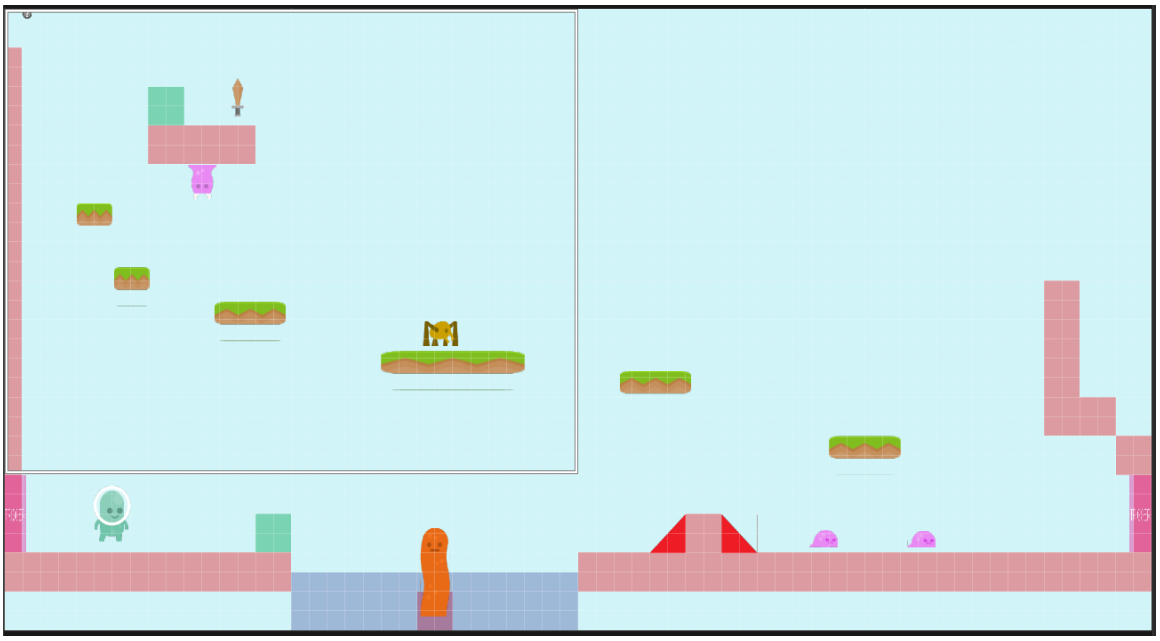
Sl. 4.18. *Slučaj odabira opcije nastavi*

Sve sobe osim naslovne imaju postavljene širinu na 2048 piksela i visinu na 1024 piksela. Prvo je napravljena soba roditelja koja sadrži sve potrebne slojeve koje ostale sobe nasljeđuju. Slojevi su potrebni jer se instance objekata stvaraju na određeni sloj. Na slici 4.19. su prikazani slojevi sobe roditelja.

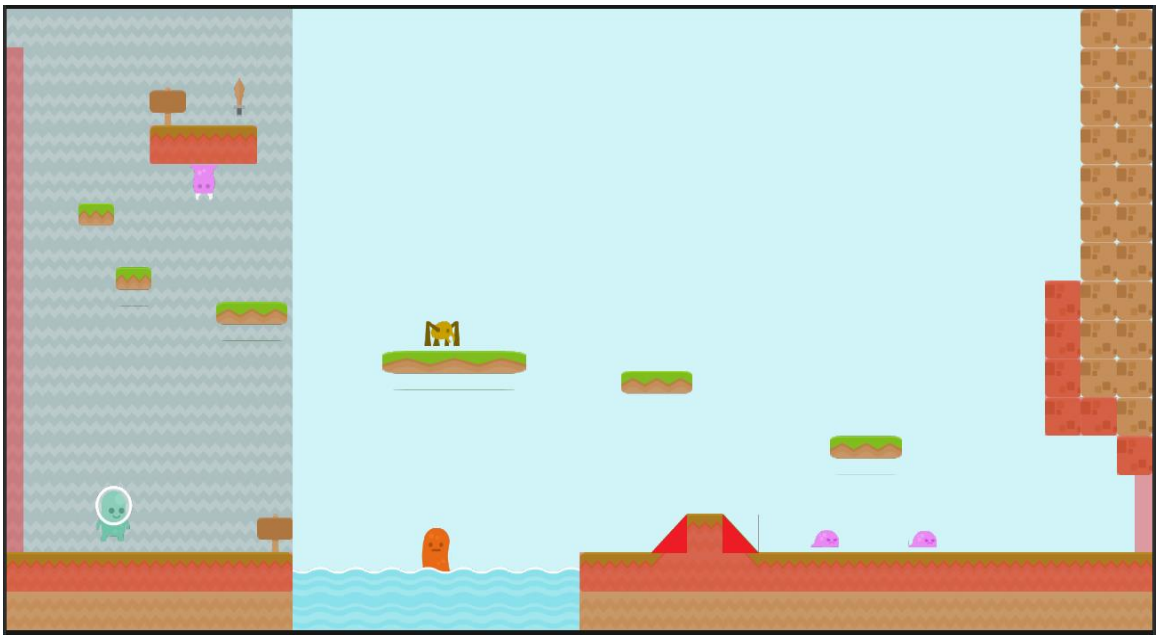


Sl. 4.19. *Slojevi sobe roditelja*

Pri razvoju soba uvijek su prvo stavljeni objekti za sudaranje i kontrolu igre kao što su na primjer zidovi, objekti kamere i objekt praćenja vrijednosti varijabli. Nakon što je razina složena, koristi se definirani set pločica za uređenje sobe. Na slikama 4.20. i 4.21. prikazane su sobe prije i poslije uređenja sa setom pločica.

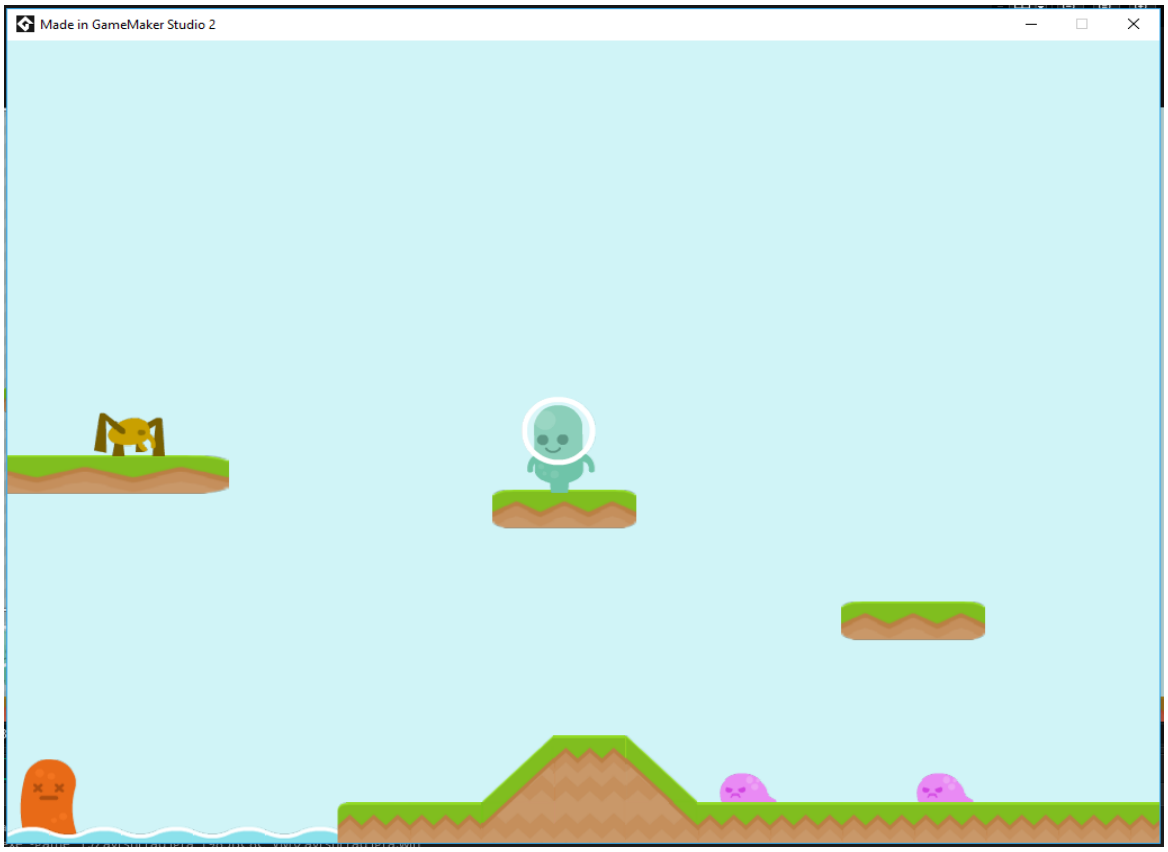


Sl. 4.20. *Postavljanje instanca objekata u sobu*



Sl. 4.21. *Uređenje sobe setom pločica*

Na slici 4.20. bijelim okvirom prikazana je kamera sobe. Kamera je podešena u postavkama sobe. Napravljen je i objekt koji kontrolira kameru. Njegov zadatak je pogled naše kamere uvijek pozicionirati na lika i pratiti ga prilikom njegovog kretanja kroz sobu. Kroz cijelu igru korištena je kamera širine 1024 piksela i visine 768 piksela.



Sl. 4.22. *Izgled prikaza kamere unutar igre*

Tranzicija između soba omogućena je objektom koji nakon sudara s instancom glavnog lika poziva skriptu scr_tranzicja koja nas zatim prebacuje u određenu sobu.

5. ZAKLJUČAK

GMS je alat koji teži što lakšem i bržem načinu stvaranja računalnih igara i ostvarivanja različitih ideja. Prvenstveno je dizajniran za stvaranje 2D računalnih igara koje se prenose na više platformi iz istih inicijalnih baza resursa. GMS sa svojim jednostavnim korisničkim sučeljem omogućava lako stvaranje računalnih igara i organiziranje svih naših resursa pri razvoju igre. GMS pruža dvije metode rada. „Drag and Drop“ metoda je za one koji su novi u razvoju računalnih igara i programiranju. To je sučelje koje omogućuje stvaranje igara vizualnim skriptiranjem. Igru možemo napravit samo sa povlačenjem miša. GMS uz „Drag and Drop“ metodu koristi i programski jezik Game Maker Language odnosno GML. GML pruža više fleksibilnosti u radu, kompliciraniji je od DnD metode ali uz malo vježbe može se savladati. U ovom radu izrađena je 2D platformska igra u GMS-u koristeći ugrađeni GML programski jezik. Pri izradi igre korištene su brojne funkcije i alati koje pruža GMS. Uz te alate korišteni su i neki drugi alati kao Paint.net za uređivanje spriteova i Particle designer za dizajniranje čestica. Svi spriteovi, zvukovi, animacije i pozadine preuzeti su sa stranice opengameart.org. Tema igre je 2D akcijski platformer u kojoj je cilj koristeći različita oružja i mehanike glavnog lika izbjegavati neprijatelje i prelaziti složene razine. Igra je dizajnirana po razinama. Igrač mora skupiti novce kako bi prelazio na druge razine te došao na kraj igre. GMS je odličan alat koji bi preporučio svima koji imaju želju razvijati računalne igre a nemaju iskustva u tom području.

LITERATURA

[1] YoYo Games Ltd., http://www.yoyogames.com/

[2] XenForo Ltd., https://forum.yoyogames.com/index.php

[3] M. Rohde, GameMaker: Studio For Dummies, John Wiley & Sons, Inc.,Hoboken New Jersey, 2014.

[4] Spirte, wikipedia https://en.wikipedia.org/wiki/Sprite_(computer_graphics)

[5] Shader, wikipedia https://en.wikipedia.org/wiki/Shader

[6] D. Silverman, 9 tips for indie game developers i learnd at gdc 2013, EnvatoTuts+, dostupno na: https://gamedevelopment.tutsplus.com/articles/9-tips-for-indie-game-developers-i-learned-at-gdc-2013--gamedev-6591

[7] Grafike u igri, OpenGameArt, dostupno na: https://opengameart.org/

[8] Pain.net alat, dostupno na: https://www.getpaint.net/

[9] G. Howland, A Practical Guide to Building a Complete Game AI: Volume I, GameDev.net, dostupno na: https://www.gamedev.net/articles/programming/artificial-intelligence/a-practical-guide-to-building-a-complete-game-a-r784

[10] R. Monteiro, The guide to implementig 2D platformers, Higher-Order Fun, dostupno na: http://higherorderfun.com/blog/2012/05/20/the-guide-to-implementing-2d-platformers/

SAŽETAK

Cilj ovog rada je izrada 2D platformske igre koristeći GameMaker Studio softversku platformu, te tijekom izrade same igre opisati rad u GMS-u i njegove mogućnosti. Igra je izrađena korištenjem ugrađenog GML programskog jezika. Sva grafika je preuzeta sa opengameart.org stranice koja je po potrebi uređivana paint.net alatom. U igri igrač mora prikupljati novce kako bi prelazio razine i došao do kraja igre. Koristeći mehanike kao što su dupli skok, skok od zida te ubrzanje igrač mora prelaziti različite razine. Uz pomoć oružja mora savladavati neprijatelje kao što su puž, muha koja puca metke, kocka koja eksplodira i druge. Igra je izvedena za Windows desktop.

Ključne riječi: 2D platformska igra, GameMaker Language, GameMaker Studio, GML, GMS, razvoj računalne igre

ABSTRACT

Using GameMaker Studio development enviroment for making computer games

The aim of this paper is to create a 2D platform game using the GameMaker Studio software platform, and to describe the work in GMS and its capabilities during the development of the game. The game is made using the built-in GML programming language. All the graphics are downloaded from the opengameart.org site, which were edited by paint.net tool if needed. In the game the player must collect money to pass the levels and reach the end of the game. Using mechanics such as double jump, wall jump and dash, the player must pass different levels. With the help of weapons, the enemy has to defeat enemies such as a snail, a fly shooting bullets, an exploding cube and many others. The game is exported for Windows desktop.

Keywords: 2D Platform Game, GameMaker Language, GameMaker Studio, GML, GMS, Computer Game Development

ŽIVOTOPIS

Matej Mijić rođen je 26. ožujka 1996. u Osijeku. U Osijeku završava osnovnu školu „Fran Krste Frankopan“ te 2011. upisuje Prirodoslovno-matematičku gimnaziju. 2015. ostvaruje upis na Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, smjer računarstvo.