Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tomaž Treven

Spletno orodje za razvoj iger

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Peter Peer

Ljubljana 2014

Rezultati diplomskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.
LEKTORICA: Ljudmila Treven, prof. slov. in zgod.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00485 / 2013
Datum: 10.4.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **TOMAŽ TREVEN**

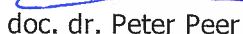
Naslov: **SPLETNO ORODJE ZA RAZVOJ IGER**
WEB TOOL FOR GAME DEVELOPMENT

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Zasnujte in implementirajte spletno razvojno orodje, ki omogoča ustvarjanje iger z vizualnim programiranjem. Uporaba razvojnega orodja naj ne zahteva nobenega nameščanja in nastavljanja potrebnih programov. Ustvarjene igre se naj prevedejo in tečejo v spletnih brskalnikih. Naredite tudi primerjavo s sorodnimi orodji za celovit razvoj igre.

Mentor:



doc. dr. Peter Peer

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Tomaž Treven, z vpisno številko **63090125**, sem avtor diplomskega dela z naslovom:

Spletno orodje za razvoj iger

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Petra Peera,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, 18. marca 2014 Podpis avtorja:

Zahvaljujem se mentorju za usmerjanje, nasvete in dobro voljo, ki jo je izkazal pri mentorstvu moje naloge. Zahvalil bi se tudi vsem tistim ljudem, ki so ustvarili in delajo na vseh tehnologijah, zaradi katerih je bila izdelava te diplomske naloge sploh možna.

Kazalo

Povzetek

Abstract

1 Uvod 1

1.1 Ustvarjanje iger danes 2

2 Vizualno programiranje 5

2.1 Arhitektura našega orodja 6

2.2 Gramatika TiTudi vizualnega jezika 9

2.3 Uporaba animacij 17

2.4 Uporaba fizikalnega pogona 21

3 Interpreter 23

4 Prevajalnik 29

4.1 Optimizacija prevedene kode 30

5 Primerjava 33

5.1 Game Maker 33

5.2 Construct 2 37

5.3 TiTudi 41

6 Vizija razvoja in tržne možnosti 47

7 Sklepne ugotovitve 49

Literatura 49

Slike

1.1 Igra Red Dead Redemption je izšla leta 2010. Je ena izmed najdražjih, saj je stala približno 100 milijonov dolarjev [1]. 2
1.2 Primer ponesrečenega razvoja igre, ki je po petnajstih letih razvoja le izšla. 3

2.1 Primer tekstualnega programiranja v PHP. 7
2.2 Primer vizualnega programiranja v Illumination Software Creator. . 8
2.3 Primer osnovnega gradnika – matrika. 10
2.4 Pomočnik bloka. 10
2.5 Ustvarjanje nove povezave. 11
2.6 Pomočnik povezave. 12
2.7 Pomočnik za ustvarjanje objekta in dokončan objekt. 13
2.8 Pomočnik za objekt in pomočnik pri ustvarjanju nove akcije. 14
2.9 Blok tipa akcija in blok tipa dogodek. 15
2.10 Primer pogojnega bloka s pogojem. 16
2.11 Primer ponavljalnega bloka. 16
2.12 Zgradba animacije. 18
2.13 Pomočnik za animacije. 19
2.14 Spustni meni za izbiro animacij. 19
2.15 Izbira slike za našo animacijo. 20
2.16 Okno za urejanje lastnosti stanja animacije. 20
2.17 Animiranje slikovnih elementov. 21
2.18 Del kode, ki pogleda, če se dva slikovna elementa dotikata. 22

3.1 Struktura razreda TiImage z vsemi lastnostmi. 25
3.2 Logika za izvrševanje ponavljalnih blokov. 26
3.3 Logika za izvrševanje pogojnih blokov. 27

4.1 Razredi, ki sestavljajo prevajalnik. 30
4.2 Primerjava ustvarjene vizualne kode (zgoraj) in ekvivalentna prevedena koda (spodaj). 32

5.1 Integrirano razvijalsko orodje Game Maker, trenutno odprt urejevalnik stopenj. 35
5.2 Primer lastnosti objekta v vizualnem jeziku Game Maker. 36
5.3 Primer končne igre z orodjem Game Maker. 37
5.4 Primer bolj dovršene igre Gunpoint [26], ki jo je moč ustvariti z orodjem Game Maker. 38
5.5 Orodje Construct 2. 39
5.6 Vizualno programiranje v Construct-u. 40
5.7 Primer bolj dovršene igre Super Ubie land [27], narejene z orodjem Construct 2. 41
5.8 Razvojno okolje TiTudi. 42
5.9 Stran za izvoz igre. 43
5.10 Dokončana igra v razvojnem okolju TiTudi. 44
5.11 Primer igre [28], ki jo je možno narediti v našem okolju. 45

7.1 Primer, kako nam dodatek ReSharper pomaga pri programiranju. . 50

Povzetek

Razvoj iger je relativno dolg in zahteven proces, ki zahteva kar nekaj znanja in primerno razvijalsko okolje. V diplomskem delu smo predstavili spletno razvojno okolje, pri katerem se uporabniku oz. razvijalcu ni potrebno ukvarjati z nameščanjem in nastavljanjem vseh potrebnih programov. Ustvarjati je mogoče praktično tudi samo s klikanjem oz. vizualnim programiranjem, tako da se naredi izdelavo iger čim bolj preprosto. Ker je produkt na spletu, lahko izvorno kodo igre vidijo vsi in jo po želji spremenijo, nadgradijo ali pa uporabijo manjše kose za nekaj čisto novega. Na strani so tudi možnosti za interakcijo z ostalimi uporabniki in nalaganje različnih gradiv, kot so slike in zvoki, za uporabo v igrah. Zelo olajšan je tudi razvoj iger s podporo več igralcem, za kar poskrbi Node.js na strežniški strani, komunikacija pa poteka preko WebSocket-a. Nastale igre se tudi prevedejo v bolj optimizirano obliko in tečejo v spletnih brskalnikih s pomočjo HTML-ja in JavaScript-a. Ker je tako izvajalno okolje zelo razširjeno in standardizirano (na vseh računalnikih, neodvisno od operacijskega sistema, na tabličnih računalnikih, pametnih telefonih in tudi na nekaterih igralnih konzolah), je potencialno število ljudi, ki bi igrali, zelo veliko. Na koncu smo dodali tudi primerjavo z nekaterimi najbolj uporabljenimi orodji za celovit razvoj iger.

Ključne besede:

Razvoj iger, spletno razvojno okolje, vizualno programiranje, Apache, PHP, Zend, HTML, JavaScript, jQuery, HTML5 Canvas, WebSocket, Node.js, prevajalnik

Abstract

Development of games is a relatively long and complex process, which requires a lot of knowledge and appropriate development environment. In this work we present web development environment for which user or developer do not need to install or configure any programs. The user is able to create the game with just clicking or by principles of visual programming, so that the development is simplified as much as possible. Because the product is on the web, everyone can see the source code and change it, modify it, enhance it or use smaller pieces of it for something entirely new. On the werb portal are also options for communicating with other users and uploading different resources, such as images and sounds for use in games. Developing game with support for multiple players is also very simplified as for communication WebSocket and Node.js on server side are used. Created games are compiled into more optimized code, which runs in web browsers with help of HTML and JavaScript. Because such execution environment is very widespread and standarized (all personal computers, independent from operating system, tablet computers and smart phones) the potential number of gamers, is huge. In the end there is also a comparision with some of the most used similar tools for complete game development.

Key words:

Game development, web development environment, visual programming, PHP, Apache, Zend, JavaScript, HTML, jQuery, HTML5 Canvas, WebSocket, Node.js, compiler

Poglavje 1

Uvod

Za razvoj lastne igre je potrebno relativno veliko časa, ustrezno delovno okolje in kar nekaj znanja. Danes so za razvoj igre na voljo številna orodja. Razlikujejo se predvsem po tem, komu so namenjena: ali večjim, izkušenim skupinam, ki znajo delati s kompleksnejšimi orodji oz. pogoni, da naredijo zelo napredne in sofisticirane igre, ali pa so namenjena zelo majhnim skupinam oz. kar posameznikom in omogočajo lažji in hitrejši razvoj, pri čemer pa navadno sama igra ne vsebuje veliko vsebine.

Cilj te diplomske naloge je narediti alternativno spletno razvojno okolje, ki bo omogočalo na kar se da preprost način ustvariti igro, brez vzpostavljanja lastnega okolja, le-to bo že na spletni strani. Ker pa bo vse na spletnih straneh, bo delo enega na voljo vsem drugim, da spremenijo že narejeno igro, ali pa vzamejo že ustvarjene elemente in jih uporabijo v svoji igri. V posameznih sklopih te diplomske naloge bomo opisali, kako smo idejo načrtovali in implementirali. Na koncu bomo naredili primerjavo okolja, ki je nastalo v okviru te diplomske naloge, z orodji, ki trenutno obstajajo na trgu in spadajo v podobno kategorijo. Glavni kriteriji primerjave bodo preprostost uporabe orodja, čas, potreben za razvoj orodja, in kvaliteta končnega izdelka, torej orodje in v njem razvite igre.

Slika 1.1: Igra Red Dead Redemption je izšla leta 2010. Je ena izmed najdražjih, saj je stala približno 100 milijonov dolarjev [1].

1.1 Ustvarjanje iger danes

Računalniške igre se lahko po obsežnosti zelo razlikujejo med seboj. Manj kompleksne lahko ustvari že en sam razvijalec oz. manjša skupina ljudi, ki je po navadi neodvisna od zunanjih partnerjev. Od tod tudi njihovo ime "indie" razvijalci oz. "indie" igre iz angleške besede "independent" oz. neodvisen. Danes zaradi rasti spletne distribucije, ki olajša prodajo izdelkov, postaja taka oblika bolj in bolj popularna in relevantna. Na drugi strani veliki studii z več kot 100 razvijalci po navadi sledijo trendu in imajo za sabo založnika, igre pa so zelo kompleksne in pogosto zahtevajo več let za razvoj. Povprečni strošek razvoja tako velike oz. AAA igre je bil v letu 2000 med 1 in 4 milijoni dolarjev, 5 milijonov v letu 2006 in čez 20 milijonov dolarjev v letu 2010 [2], nekatere igre pa imajo še dražji razvoj in obetajo še večje dobičke. Primer takšne igre je Red Dead Redemption (slika 1.1).

Ker so igre umetniške narave, zahtevajo neko mero svežine in novosti, če se želijo uspešno prodajati. Izdajanje pogostih in podobnih nadaljevanj bo kmalu

Slika 1.2: Primer ponesrečenega razvoja igre, ki je po petnajstih letih razvoja le izšla.

znižalo prodajne številke [3]. Ustvarjanje čisto novih igralnih konceptov pa je tvegano početje, zato to večinoma delajo manjše, neodvisne razvijalske ekipe ali pa največji razvijalci, ki niso vezani na tesen finančni in časovni načrt in si lahko privoščijo zavlačevanje, če izdelek še ni dovolj dober za izdajo. Ker so v zadnjem času v vzponu video igre za mobilne telefone, prihaja tudi tam do številnih novosti, saj tam platforma še ni tako zmogljiva in so stroški razvoja nižji ter lahko posledično za njih dela več ljudi.

Kljub veliko načrtovanja za izdelavo igre je še zmeraj veliko primerov, ko gre kaj narobe, ali se preseže zastavljen rok ali se preseže zastavljen proračun ali pa je kvaliteta prenizka in igra ne opraviči svojih stroškov – Primer takšne igre je Duke Nukem: Forever (slika 1.2).

Glavni motiv te diplomske naloge je torej narediti enostavno orodje, ki bo poenostavilo izdelavo in distribucijo iger. Zaradi številnih že narejenih iger in modulov za igre, ki se lahko ponovno uporabijo, naj bi se povečala zanesljivost posameznih komponent in zmanjšalo tveganje za napake. Vse skupaj bi tako povečalo hitrost in znižalo stroške razvoja. Seveda na začetku le za enostavne igre, v prihodnosti pa mogoče tudi za zahtevnejše.

Poglavje 2

Vizualno programiranje

Računalniško programiranje je proces, pri katerem nastane program oz. navodila računalniku ali nekemu stroju, kaj in kako naj nekaj naredi. Večinoma z namenom, da reši nek problem ali opravi določene naloge. Proces se začne z idejo in načrtovanjem, nato pa se začne ustvarjati program oz. izvorno kodo. Pri tem nastanejo različne napake, ki se jih s testiranjem in razhroščevanjem poskuša odpraviti. Ko imamo nato delujoč in relativno stabilen program, je le-ta pripravljen na uporabo, da ga določena platforma, ki nastali program razume, izvrši. Izvorna koda je skupek ukazov, ki računalniku povedo, kaj naj naredi. Napisana je v človeku berljivem formatu, v določenem programskem jeziku, ki je formalni jezik s svojimi pravili. Le-ti pa se lahko med seboj zaradi različnih namenov razlikujejo v [4]:

- **Stopnjah abstrakcije**: Razlika med računalniškim in človeškim jezikom je velika. Skozi čas so nastajali računalniški jeziki, ki so vedno bolj podobni človeškemu. V začetku so bili le jeziki na najnižjem nivoju oz. je en ukaz kode predstavljal en strojni ukaz, ki ga stroj lahko izvede. To je v uporabi še danes za vgrajene sisteme oz. tam, kjer računalnik ni preveč zmogljiv. V teh primerih rabimo ali zelo optimiziran program ali pa direkten dostop do strojne opreme. Zaradi večje kompleksnosti in želje po hitrejšem razvoju pa so jeziki postali bolj abstraktni. To so lahko dosegli zaradi večje računalniške moči in prevajalnikov, ki kompleksen ukaz višjega programskega jezika pre-

vedejo v serijo ukazov, ki jih računalnik razume.

- **Paradigmah**: Za prilagoditev določenemu namenu se jeziki razlikujejo v številnih smernicah, ki vplivajo na strukturo programa. Na najbolj abstraktni ravni se jeziki delijo na imperativne (navedemo postopek, kako naj se nekaj izvede) in dekleratívne (navedemo, kaj naj se zgodi, kako bo to izvedeno, pa naj platforma ugotovi sama). Nato imamo funkcijske in objektivno usmerjene jezike, močno in šibko tipizirane, prevajalne in interpretativne itn.
- **Sintaksi**: Seznam pravil, ki določajo veljavne kombinacije simbolov v dokumentu, imenujemo sintaksa. Pri tekstovnih programskih jezikih sintaksa določa pravilna zaporedja znakov, pri vizualnih programskih jezikih pa vizualno obliko simbolov in povezave med njimi. Po tem tudi najhitreje ločimo programske jezike med seboj.

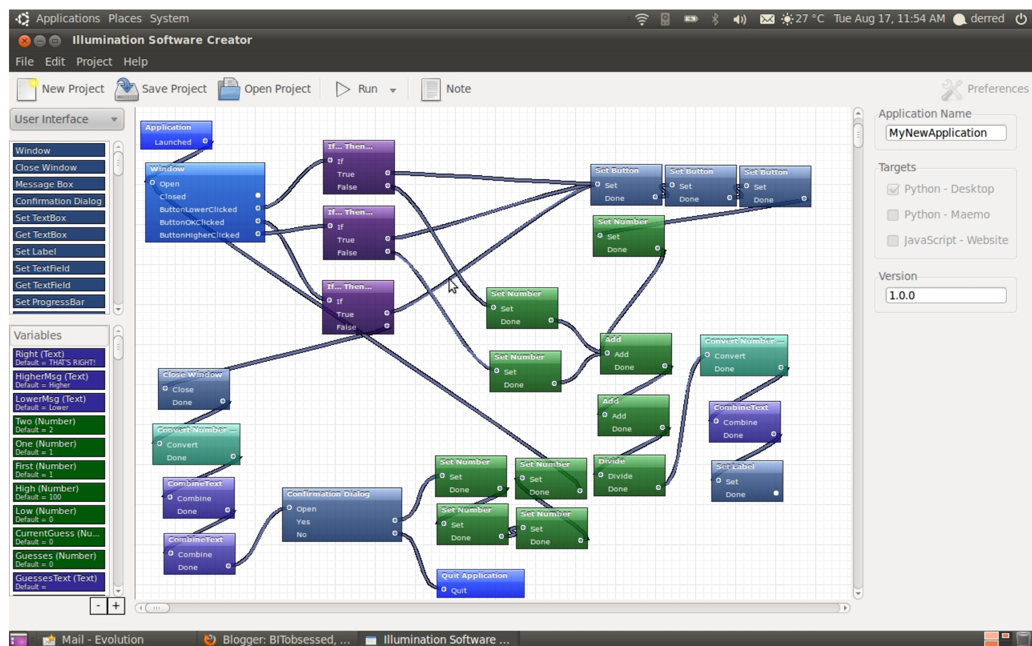
Večina programskih jezikov je bila že od samega začetka tekstovna (slika 2.1). Taka oblika večinoma prevladuje tudi danes. Pri taki obliki programiranja programer piše izvorno kodo v skladu s sintakso. V veliki meri mu lahko to olajša ustrezno pametno razvojno okolje z barvanjem kode glede na namen in z avtomatičnim strukturiranjem kode. V tem delu se bomo osredotočili na vizualno programiranje (slika 2.2), ki nudi potencialno hitrejše ustvarjanje kode, ker kode ni treba pisati, ampak se sestavlja iz vizualnih elementov. Tako se tudi ustvari manj napak, ker je povezovanje elementov samo po sebi bolj striktno in lažje za začetnike.

2.1 Arhitektura našega orodja

Ker je naše orodje za programiranje v celoti spletno, je v osnovi spletna stran. Teče na spletnem strežniku Apache [5]. Za programiranje na strežniški strani smo uporabil programski jezik PHP in knjižnico Zend [6], tako da je strežniška arhitektura pa vzoru MVC [7]. Bistvena spletna stran diplomske naloge je `http://titudi.com/matrix/crea` [8], (slika 33). Ta stran je razvojno okolje, ki omogoča vizualno programiranje v našem jeziku, ki ga bomo poimenovali TiTudi jezik. Program, ki ga ustvarimo,

```
580            $block .= parent::translate($prefix);
581            $block .= $ident ."}\n";
582            return $block;
583        }
584    }
585
586    class Block {
587        public $name;
588        public $commands;
589        public $level;
590        public $parent;
591
592        function __construct($xmlDesc, $level, $parent) {
593            $this->level = $level+1;
594            $this->parent = $parent;
595            $this->commands = array();
596            HyperMorpher::$currentBlock = $this;
597
598            $commands = $xmlDesc->children();
599            for($ci = 1; $ci < count($commands); $ci++) {
600                if($commands[$ci]->getName() == 'Command') {
601                    $command = new Command($commands[$ci]);
602                } else if ($commands[$ci]->getName() == 'Block') {
603                    if($commands[$ci]['type'] == 'Control') {
604                        $command = new ControlBlock($commands[$ci], $this->level, $this);
605                    } else if ($commands[$ci]['type'] == 'Iteration') {
606                        $command = new IterationBlock($commands[$ci], $this->level, $this);
607                    }
608                } else {
609                    print ($commands[$ci]->getName() .'<br />');
610                }
```

Slika 2.1: Primer tekstualnega programiranja v PHP.



Slika 2.2: Primer vizualnega programiranja v Illumination Software Creator.

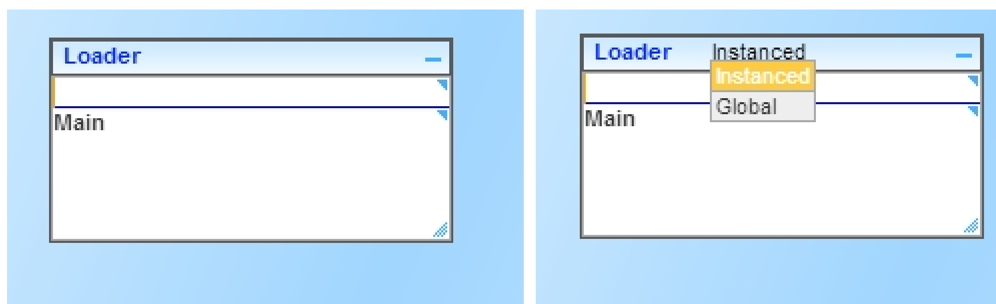
se shrani na strežnik v XML formatu. Ta stran to mogoča z intenzivno uporabo Java Script jezika, uporabljena je bila knjižnica jQuery [9] in njeni dodatki. Vsa ostala koda na uporabniški strani je moja lastna. Na tej strani je tako koda, ki omogoča ustvarjanje vizualne kode, naši lastni razredi, ki so na voljo za uporabo v TiTudi jeziku, med katere spadata tudi razreda za animacije, fiziko in interpreter, ki ustvarjeno kodo poganja, da lahko vidimo, kaj smo ustvarili. Ta koda je sestavljena iz dobrih 5700 vrstic. Ko ustvarimo nek program z našim orodjem in ga izdamo, se naš ustvarjen program v XML formatu avtomatično prebere v php datoteki, ki jo prevede v Java Script program, ki ga lahko uporabniki naše strani potem poženejo. Vse to smo ustvarili v razvojnem okolju PhpED [10], preizkušali in testirali pa v brskalniku Chrome.

2.2 Gramatika TiTudi vizualnega jezika

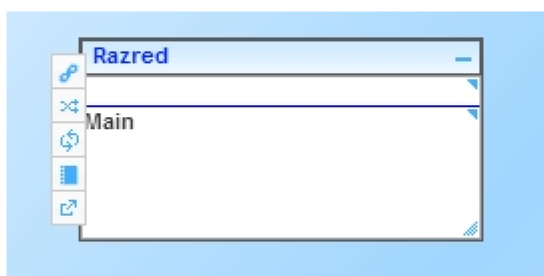
Vizualni programski jezik je objektivno orientiran, kar se vidi takoj, ko odpremo stran za programiranje. Prvi vizualni gradnik, ki nas čaka že pripravljen na strani, je matrika (slika 2.3), ki lahko prestavlja razred ali objekt. V vseh jezikih je objekt posamezna instanca z določenimi lastnostmi, ki jih specificira razred. Oba imata na začetku dva osnovna bloka. V prvi blok vnesemo poljubne spremenljivke oz. vrednosti, ki bodo pripadale temu gradniku (`members`). Naslednji blok je metoda, ki se zažene takrat, ko se zažene objekt. Po želji pa lahko dodajamo poljubno število blokov, ki opravijo različna dela z imenom `action`. Ali se bo ta osnovni gradnik obnašal kot razred ali objekt, izberemo v spustnem seznamu zraven imena gradnika. Izberemo lahko možnosti `Instanced` in `Global`. Prva možnost naredi gradnik za klasični razred, to pomeni, da v ostalih blokih ustvarimo instance tega razreda in tako dobimo posamezne objekte. V tem primeru je drugi blok (akcija, ki se zažene ob zagonu objekta) poimenovan konstruktor. V drugi možnosti pa se gradnik obnaša, kot bi se v drugih jezikih obnašali statični razredi. To pomeni, da so kreirani že ob začetku programa in so unikatni; se pravi, da obstaja samo en primerek. Kot razredi po načrtovalskem vzorcu edinec (angl. Singleton) z želeno inicializacijo (angl. Eager Initialization) [11]. Smiselno je, da ima program vsaj eno matriko tipa `Global`, ki se izvrši na začetku, saj se v nasprotnem primeru ne bi nič zgodilo. V primeru, da imamo eno matriko tipa `Global`, s prvo akcijo `Main`, bi to v programskem jeziku Java ustrezalo metodi public static main v osnovnem razredu. Z dvoklikom na ime matrike se nam odpre vnosno polje, s katerim lahko izbrano matriko preimenujemo.

Ob kliku na enega izmed blokov se nam odpre pomočnik bloka (slika 2.4), ki nam nudi najpogostejše elemente, s katerimi lahko gradimo program. Ti elementi so od zgoraj navzdol:

- **Link** (povezava) oziroma v drugih jezikih spremenljivka ali referenca,
- **Conditional** block (pogojni blok) – blok se glede na pogoj izvrši ali ne,
- **Iteration** block (ponavljalni blok) – glede na pogoj se bo blok različno po-



Slika 2.3: Primer osnovnega gradnika – matrika.



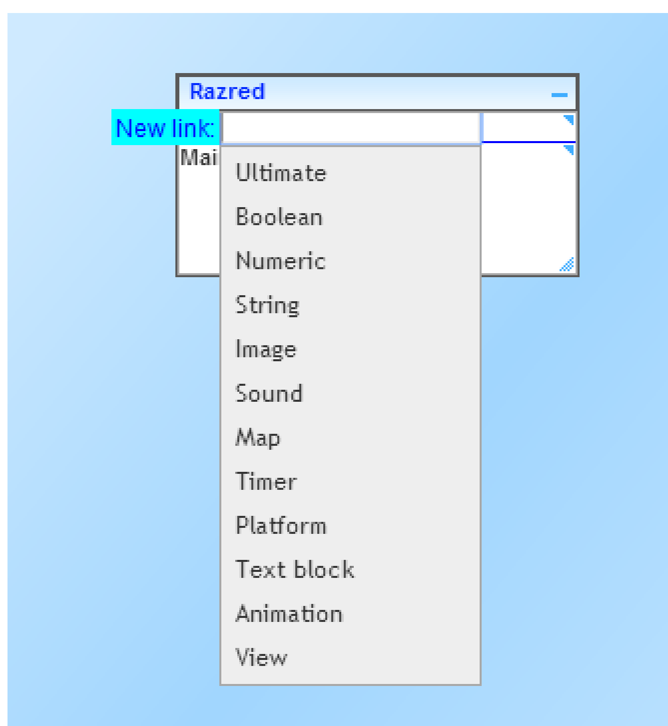
Slika 2.4: Pomočnik bloka.

navljal,

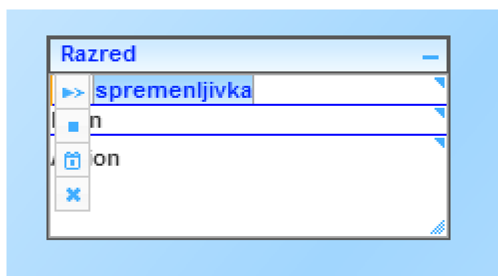
- **Action** (akcija) v matriki kreira nov blok tipa akcija,
- **Return** (vrnitveni element) konča izvajanje trenutnega bloka in opcijsko vrne poljubno vrednost.

Ob kliku na prvi gumb za kreirane nove povezave se nam ustvari nova povezava, ki še nima imena, zato ji ga moramo določiti. Novo povezavo lahko ustvarimo tudi tako, da pritisnemo na tipko R. V spustnem meniju (slika 2.5) so navedeni že nekateri razredi, ki so vgrajeni v okolje in jih lahko naredimo tako, da ta povezava kaže na željen razred. Če naredimo povezavo na razred, lahko potem manipuliramo s tistim razredom prek te povezave.

Povezava pa lahko kaže tudi na manj kompleksne elemente oz. nesestavljene, v tem primeru bi se obnašala podobno kot v ostalih jezikih spremenljivke. To



Slika 2.5: Ustvarjanje nove povezave.



Slika 2.6: Pomočnik povezave.

naredimo tako, da povezavi priredimo nek `expression` oz. izraz. Ob kliku na povezavo se nam odpre pomočnik za povezavo (slika 2.6), ki vsebuje vse ukaze, ki jih lahko izvedemo nad povezavo. Ti ukazi so:

1. Vstavi akcijo, ki izvrši neko akcijo nad elementom, na katerega kaže povezava.

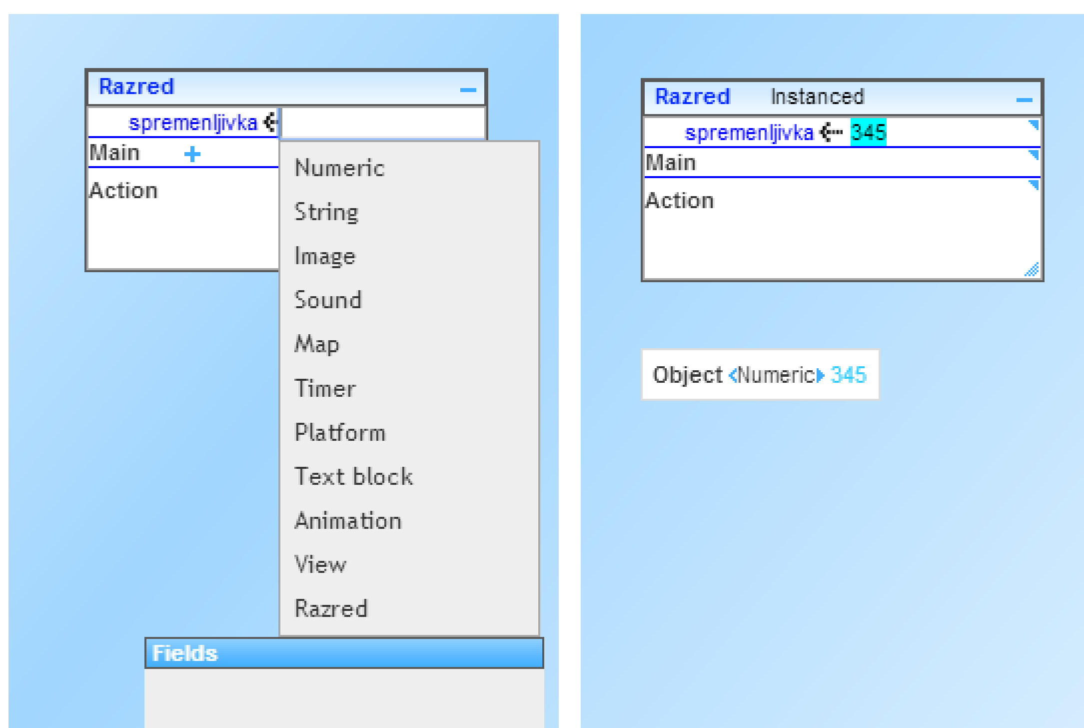
2. Vstavi objekt, ki povezavi priredi neko enostavno vrednost.

3. Vstavi operator za kombiniranje več povezav skupaj.

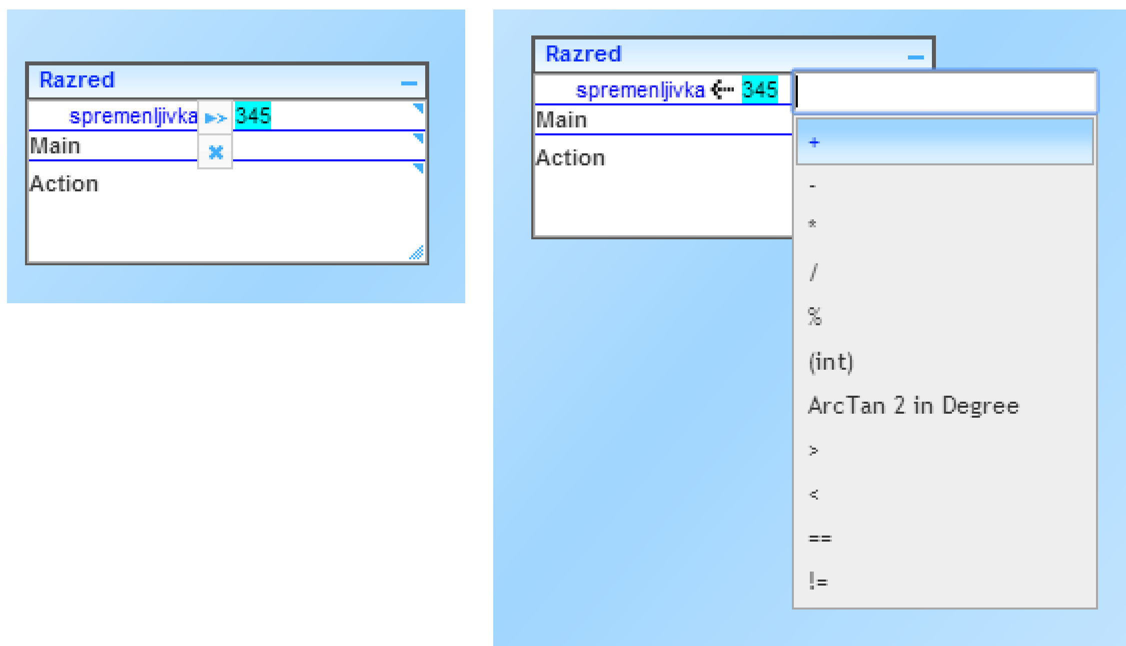
4. Izbriši povezavo.

Če kliknemo na drugi gumb (slika 2.6), da ustvarimo nov objekt, se nam avtomatično ustvari tudi nov izraz. Če bi želeli doseči enak rezultat samo s tipkovnico, bi morali pritisniti tipki E in O. S tem ustvarimo prazen objekt z že odprtim pomočnikom za prirejanje vrednosti temu objektu. Recimo, da želimo, da ima objekt vrednost 345 (slika 2.7). To je že dokončan ukaz, ki pove, da povezava z imenom spremenljivka kaže na objekt z numerično vrednostjo 345, kar nam sporoča tudi splošen pomočnik (slika 2.7) v spodnji levi polovici delovnega okolja.

Poleg povezav in objektov lahko postavimo tudi akcije (slika 2.8). Postopek za ustvarjanje le-te je podoben kot za ustvarjanje ostalih elementov do sedaj. Lahko pritisnemo tipko A ali pa kliknemo na izbrani objekt, da se nam prikaže pomočnik za manipulacijo objekta, ki vsebuje le možnost za dodajanje akcije in izbris objekta, v katerem izberemo prvo možnost. Ustvari se še nenastavljena akcija s spustnim menijem, ki vsebuje nekatere izmed možnih akcij. Ker okolje ve,



Slika 2.7: Pomočnik za ustvarjanje objekta in dokončan objekt.

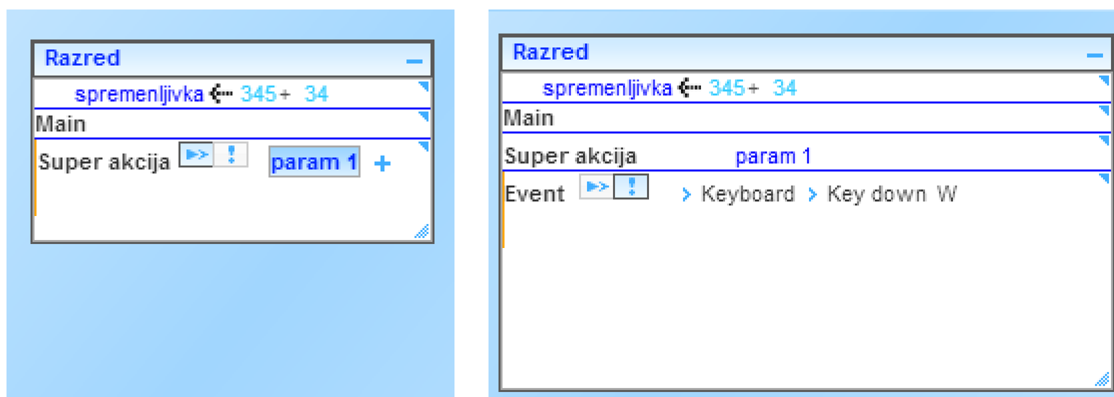


Slika 2.8: Pomočnik za objekt in pomočnik pri ustvarjanju nove akcije.

da ta akcija deluje nad numeričnim objektom, ponudi le tiste akcije, ki se lahko izvedejo nad podatkom numeričnega tipa, torej matematične in logične operacije. Če pa bi ustvarjali akcijo nad povezavo, ki bi kazala na animacijo, pa bi bile možnosti: zaženi animacijo, ustavi animacijo, pojdi na točko v animaciji itn.

V matriki pa lahko sami ustvarimo tudi svoje akcije (slika 2.9), ki jih potem kličemo nad to matriko. V pomočniku bloka kliknemo na gumb za novo akcijo, ki nam jo tudi ustvari. Podobno kot pri matriki tudi tukaj z dvoklikom akciji spremenimo ime. Ko imamo miško nad glavo akcije (zgornji del bloka, ki vsebuje informacije o akciji), se nam prikažeta izbirna gumba, ki spremenita tip bloka iz akcije v dogodek in obratno. Samo delovanje obeh tipov je praktično enako, izvedeta kodo, ki jo vsebujeta. Razlikujeta se le v načinu klica. Dogodek se bo avtomatično sprožil ob nekem dogodku, akcijo pa moramo poklicati sami. V primeru akcije imamo na desni strani še gumb plus, s katerim lahko dodamo parametre, ki jih prenesemo v akcijo ob klicu.

Ko kličemo akcijo `Super akcija`, ji moramo podati en parameter. V nada-

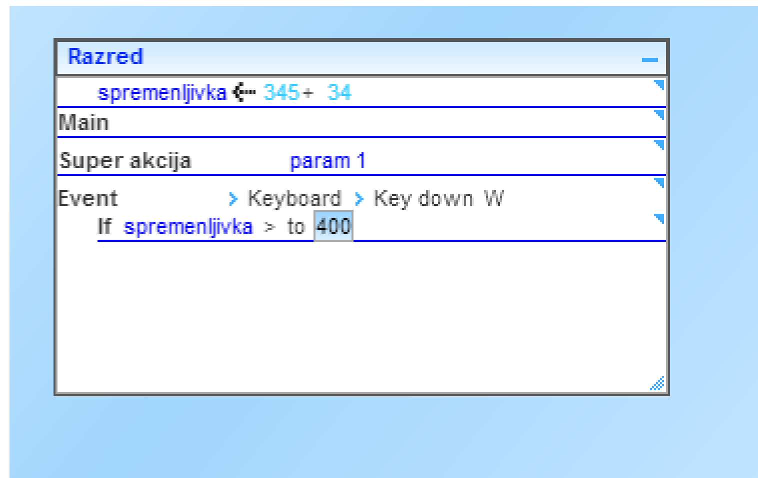


Slika 2.9: Blok tipa akcija in blok tipa dogodek.

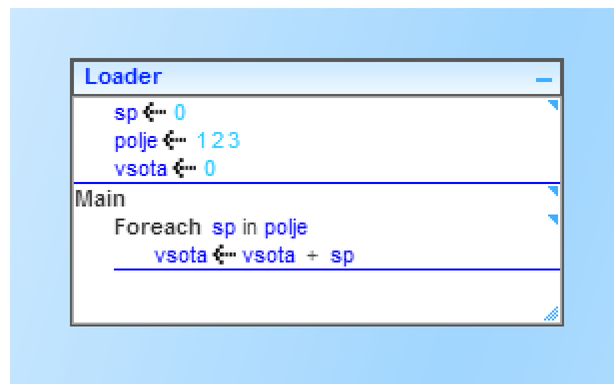
ljevanju pa ima matrika Razred tudi dogodek, ki se bo sprožil vedno, ko bomo pritisnili tipko W. Če želimo, da se neka logika izvede glede na določen pogoj, potem potrebujemo pogojni blok. V pomočniku za blok (slika 2.4) izberemo nov pogojni blok (slika 2.10). Glede na predhodne in naslednje bloke ga lahko preoblikujemo v `If`, `Else If` in `Else` blok. Pri prvih dveh navedemo še nek pogoj, ki mora biti resničen, da se ta blok izvrši.

Zadnji tip bloka pa je ponavljalni blok. Je zelo podoben for blokom v ostalih jezikih, ki se lahko iterirajo po načelu `for each`. Gradnik, ki bi bil v drugih programskih jezikih ekvivalenten zanki `while`, v našem jeziku ne obstaja. Takemu bloku v glavi določimo dva elementa: polje oz. povezavo, ki kaže na več združenih elementov, in povezavo, ki bo ob vsaki ponovitvi bloka kazala na naslednjo vrednost v polju. Najprej kaže na prvi element polja, ob zadnji ponovitvi pa na zadnji element polja. Ko pride do konca bloka, se le-ta ne ponavlja več. Podobno kot pogojni blok lahko ustvarimo tudi ponavljalni blok. Pred tem ustvarimo še povezavo, ki bo kazala na polje elementov.

Primer ponavljalnega bloka (slika 2.11), ki se izvede trikrat, saj povezava polje kaže na polje s tremi elementi. Pri prvem obhodu povezava `sp` kaže na vrednost prvega elementa v polju `polje`, ki je 1, pri zadnjem obhodu pa na vrednost zadnjega elementa, ki je 3. V vsaki iteraciji se spremenljivki `vsota` prišteje trenutna vrednost iz polja `polje`. Ko se `foreach` blok konča, ima spremenljivka vsota



Slika 2.10: Primer pogojnega bloka s pogojem.



Slika 2.11: Primer ponavljalnega bloka.

vrednost 6.

2.3 Uporaba animacij

Za uporabo animacij je na voljo razred `Animation` (slika 2.13). Njegov konstruktor sprejme dva argumenta: ime animacije in ime akcije, ki jo ta animacija ima. Vsaka animacija je namreč sestavljena kot objekt akcij (na sliki 2.12) so to `Stance`, `Damaged`, `Win` ...). Vsebujejo lastnost zamik (`delay`), ki pove, koliko časa v milisekundah preteče med dvema stanjema animacije. Bistvena lastnost je `timeLines`. Le-ta v lastnosti `id` vsebuje ime slike, ki bo uporabljena za animacijo (na sliki 2.12 je sliki za animacijo ime 39). Ta je v večini slikovni atlas (angl. sprite) [12]. Poleg tega pa vsebuje polje frames, ki vsebuje informacije, kaj naj se s to sliko naredi v posameznem stanju. Možnosti so odrez (angl. clip), ki pove, kateri del slikovnega atlasa naj se v posameznem stanju prikaže. Na sliki 2.12 se v prvem stanju akcije `Run` prikazuje kvadratni izrez velikost 44*51 slikovnih elementov, ki je od zgornjega levega kota oddaljen za 386 elementov po horizontali in 466 elementov po vertikali. Poleg tega pa ima lahko vsaka akcija še lastnost `followUp`, ki pove, kaj naj se zgodi, ko se trenutna akcija konča. Če je vrednost prazna, se ne zgodi nič in prikazuje se zadnje stanje. Lahko pa ima ime kakšne druge animacije, ki se bo zgodila po tej ali pa kar svoje lastno ime, če želimo, da se trenutna animacija ponavlja. Na sliki 2.12 se po zaključku animacije `Run` ponovi ista akcija `Run` z začetkom na prvi stopnji.

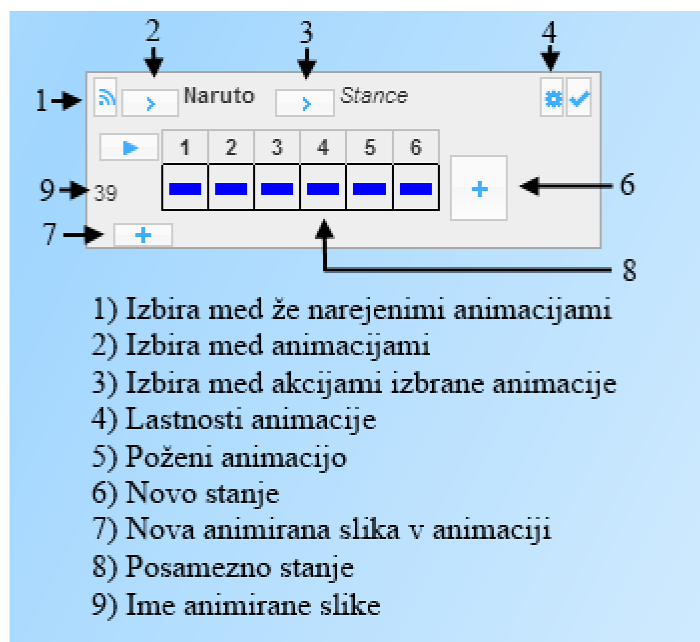
Da bi se animiranje čim bolj olajšalo, je na voljo pomočnik. V njem lahko z gumbom za iskanje animacij izberemo neko že narejeno animacijo, ki jo je nekdo shranil na strežnik. Lahko pa kreiramo svoje animacije s pomočnikom (slika 2.13).

Na tem pomočniku kliknemo na gumb 2 (izbira med animacijami), s čimer se odpre spustni seznam z vsemi že obstoječimi animacijami in z možnostjo za novo animacijo `+New animation` (slika 2.13).

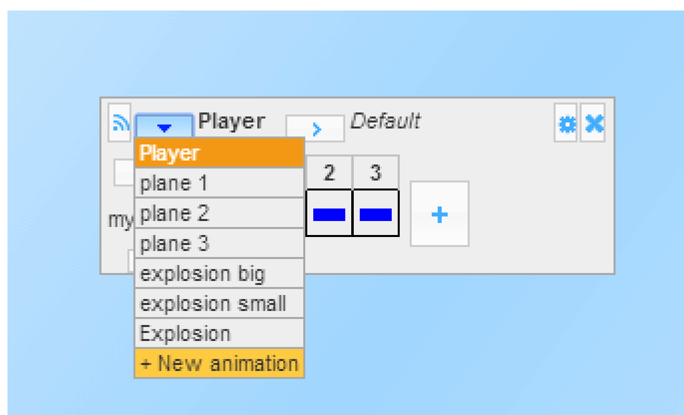
S tem smo ustvarili novo animacijo s privzetim imenom New animation. Z dvoklikom na ime animacije se nam odpre polje za urejanje imena, s čimer lahko animacijo preimenujemo (slika 2.15). S klikom na `New timeline` pa se nam odpre

```
> Animator.animations.Naruto
  ▼ Object {Stance: Object, Damaged: Object, Win: Object, Run: Object, Smash: Object…}
    ▶ Damaged: Object
    ▶ Head butt: Object
    ▶ Jump: Object
    ▼ Run: Object
        delay: 50
      ▼ followUp: Array[3]
          0: "Naruto"
          1: "Run"
          2: 0
          length: 3
        ▶ __proto__: Array[0]
      ▼ timeLines: Array[1]
        ▼ 0: Object
          ▼ frames: Array[6]
            ▼ 0: Object
              ▼ clip: Object
                  h: 51
                  w: 44
                  x: 386
                  y: 466
                ▶ __proto__: Object
              ▶ __proto__: Object
            ▶ 1: Object
            ▶ 2: Object
            ▶ 3: Object
            ▶ 4: Object
            ▶ 5: Object
              length: 6
            ▶ __proto__: Array[0]
            id: 39
            name: null
          ▶ __proto__: Object
          length: 1
        ▶ __proto__: Array[0]
      ▶ __proto__: Object
```

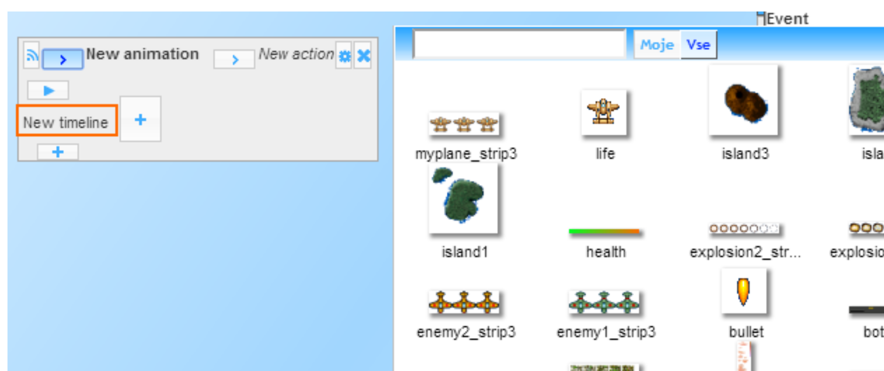
Slika 2.12: Zgradba animacije.



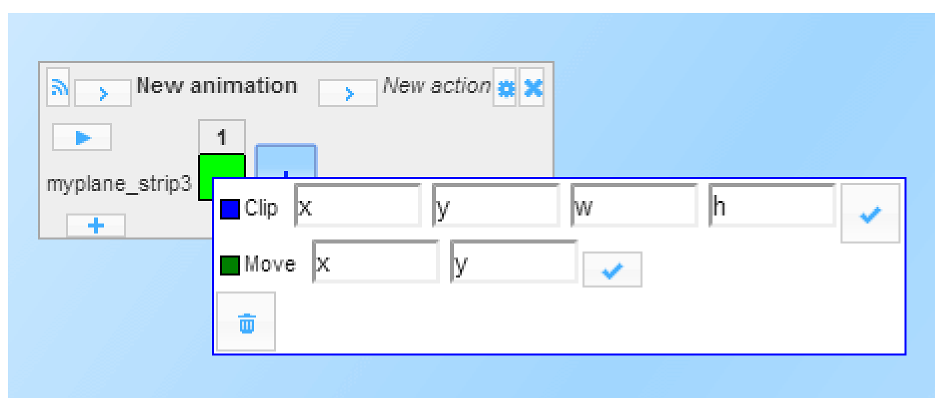
Slika 2.13: Pomočnik za animacije.



Slika 2.14: Spustni meni za izbiro animacij.



Slika 2.15: Izbira slike za našo animacijo.

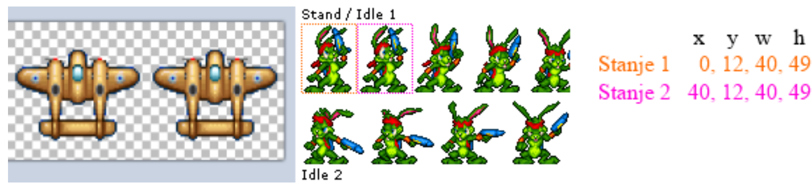


Slika 2.16: Okno za urejanje lastnosti stanja animacije.

okno z vsemi naloženimi slikami, med katerimi izberemo tisto, za katero želimo, da bo prisotna v naši animaciji oz. jo bomo animirali (slika 2.15).

S klikom na gumb 6 (slika 2.13) ustvarimo novo stanje animacije (angl. frame). Ob kliku na to stanje se nam odpre okno z naborom možnosti za urejanje tega stanja oz. kaj naj se z izbrano sliko na tem stanju zgodi (slika 2.16).

Lastnosti, ki jih je trenutno moč spreminjati, sta le dve: kako naj se slika obreže (angl. clip) in premik. Z lastnostjo `Clip` (slika 2.16) povemo, kateri del slike naj se v trenutnem stanju izriše. To povemo s podatki: levo zgornjo točko slike (x, y) in širino w ter višino h. To je vidno na primeru izrezov (slika 2.17). V naši igri imamo le zelo preproste animacije: recimo letala s tremi stanji, pri katerih je



Slika 2.17: Animiranje slikovnih elementov.

v vsakem stanju propeler letala malo drugačen. Ker se to stanje hitro spreminja, dobimo občutek, da se propeler dejansko premika.

2.4 Uporaba fizikalnega pogona

Simulacija fizike je lahko zelo zapletena, če je moramo sami implementirati. Zato je v našem orodju že nekaj našega fizikalnega ogrodja, ki olajša delo s fiziko. Vsak slikovni element ima že lastnost za lasten pospešek in trenutno premikanje v smeri x in y. Do teh lastnosti lahko dostopamo pri programiranju v našem okolju in jih lahko po želji spreminjamo. Poleg teh pa ima vsak slikovni element tudi skrite lastnosti, kot so povezava na podobo, glede na katero naš pogon zaznava odboje (angl. collision detection) [13], lastnost, ki označuje, če je naš objekt že v stiku z drugimi objekti, ter na lastnosti `parent` in `children`. Če je dani slikovni element povezan z nekim drugim v smislu, da je odvisen od njega, potem ima dani slikovni element nadrejen objekt v lastnosti `parent`, nadrejeni objekt pa ima dani element za podrejenega in v svojem seznamu `children`. Če se premakne nadrejeni objekt, se bodo na enak način premaknili tudi vsi podrejeni slikovni elementi, ki jih ima nadrejeni objekt v svojem seznamu `children`. Pred izrisovanjem vsakega slikovnega elementa se pogleda, če ima slikovni element nastavljeno ovojnico (angl. boundary representation). Če jo ima, se konkretni slikovni element posodobi v našem fizikalnem pogonu, glede na želeno obnašanje, ki smo ga določili za posamezno podobo: gravitacija, premikanje, dotik ipd. Če smo v naši kodi ustvarili kakšen dogodek, ki se zgodi, ko se dva slikovna elementa dotikata, se to tudi izračuna v glavni metodi za posodabljanje fizike `Field.Update()`. Dotikanje

```
315    } else if (s[1] == 9) {
316        var rel = Field.relations[img.id][s[2]]
317        for(var f = 0; f < Field.elements.length; f++) {
318            if(Field.elements[f].id == s[2]) {
319                var ti = Field.elements[f]
320                for(var ri = 0; ri < ti.field.regions.length; ri++) {
321                    var r = ti.field.regions[ri]
322                    var t = Field.data.tiles[r[0]]
323                    if(ti.y + t[1] <= img.y + st[1] + st[3] + img.my && img.x + st[0] <=
324                       ti.x + t[0] + t[2] && img.y + st[1] <= ti.y + t[1] + t[3] && ti.x + t[0] <=
325                       img.x + st[0] + st[2]) {
326                        for(var fr = 0; fr < rel.length; fr++) {
327                            if(rel[fr][0] == img.hm) {
328                                hyperMatrixes[rel[fr][0]].stream.push([rel[fr][1],0])
329                                hyperMatrixes[rel[fr][0]].perform([0,null])
330                            }
331                        }
332                    }
333                }
334            }
335        }
336    }
```

Slika 2.18: Del kode, ki pogleda, če se dva slikovna elementa dotikata.

(slika 2.8) se izračuna tako, da se pogleda, če se podobi dveh slikovnih elementov dotikata ali prekrivata. Podobe, ki se pri temu uporabijo, so kvadrati, poravnani s koordinatnim sistemom sveta (angl. AABB oz. axis-aligned bounding box) [14].

Poglavje 3

Interpreter

Način izvajanja kode je v osnovi objektni. Ko uporabnik zažene sestavljeno kodo, se najprej nastavi, da se glavna funkcija render izvrši šestdesetkrat v sekundi. Le-ta predstavlja glavno zanko igre. Najprej pobriše vse, kar je že narisano na HTML canvas elementu. Nato nastopi faza procesiranja delnega vhoda; delnega zato, ker sprocesira le tiste dogodke, ki imajo trajanje, in ne enkratnih (pritisk in sprostitev tipke sta enkraten dogodek, če pa je tipka pritisnjena, pa ima trajanje) in jih je posledično treba vedno upoštevati. To se zgodi tako, da funkcija render pogleda v spremenljivko `hyperBind`, v kateri so povezave na vse dogodkovne bloke. Če je kašen blok z izpolnjenim pogojem, na primer, da ima blok označeno, da se izvede, ko je tipka A pritisnjena, in je ta tipka res pritisnjena, potem se ta blok pošlje v izvrševanje. Preostali del funkcije render pa je zadolžen za izrisovanje vseh slikovnih elementov. Le-ti so namreč povezani v spremenljivko `CanvasElements` glede na njihove lastnosti (velikost, položaj, rotacijo, prosojnost ipd.). Vsi ti slikovni elementi so v resnici instance razreda `TiImage` (slika 2.10), ki je tudi eden od preddefiniranih razredov v našem jeziku. Ko je glavna zanka zagnana, se začne interpretacija ustvarjene kode. Bistveni del pri interpretaciji je razred `hyperMatrix`. V sebi ima povezavo na naš vizualni razred, ki ga interpretira, vse vrednosti spremenljivk, ki jih interpretirani razred potrebuje, in podatek o položaju oz. v kateri vrstici interpretiranega razreda se nahajamo oz. v kateri vrstici trenutno poteka izvajanje. Za vsako matriko, ki je tipa global, se ustvari

živo matriko oziroma nov objekt tipa `hyperMatrix` in nad njim pokliče metodo ctor oz. metodo za prebujanje matrike oziroma objektov. Živa matrika ima v sebi svojo unikatno številko, povezavo na matriko, po kateri je ustvarjena oz. na razred, katerega instanca je, izvrševalno vrsto, kje naj se nadaljuje izvajanje, zbirko svojih spremenljivk in še nekatere druge informacije. Le-ta najprej uvrsti vse dogodkovne bloke v spremenljivko hyperBind, da so hitro na voljo, ko se ustrezen dogodek, ki jih proži, tudi zgodi. Dogodki se delijo na:

- dogodke okolja (ponavljajoči na določen interval, ob koliziji),
- dogodke miške (kateri gumb: levi, srednji, desni in vrsta: pritisk, pritisnjeno, spuščeno),
- dogodke tipkovnice (pritisk, pritisnjeno, spuščeno).

Nato se v izvedbo pošljeta prva dva bloka matrike (če le-ta obstajata). Prvi v spremenljivke matrike naloži začetne vrednosti. Drugi blok pa je metoda, ki deluje kot konstruktor objekta.

Najpomembnejši del pri interpretaciji kode opravlja metoda `perform` nad objektom `hyperMatrix`. Le-ta pogleda, kateri blok je v vrsti za izvrševanje, in ga začne izvrševati. Blok vsebuje zaporedje ukazov, ti so lahko blok (pogojni ali ponavljalni) ali pa enostavni ukazi. Ponavljalni blok iteracijski spremenljivki najprej nastavi ustrezno vrednost iz kolekcije in postavi sam sebe v vrsto za izvrševanje ter se izvrši (slika 3.2). V primeru, da med izvrševanjem ukazov v bloku ne naletimo na kontrolne ukaze, ki lahko prekinejo zanko oz. ponavljalni blok, se bo blok ponovil tolikokrat, kolikor elementov je v kolekciji.

V primeru, da gre za pogojni blok, si v začasno spremenljivko branch shranimo, če je bil pogoj za njega resničen oz. če se je blok izvršil; privzeto ima vrednost `false`. To bomo potrebovali v primeru, da imamo več zaporednih pogojnih blokov, ki so podtipa `if` in `else` oz. `if`, `else if` in `else`, se pravi, če se prvi pogojni blok ne izvrši, naj se drugi oz. če se ne izvrši ne prvi in tudi ne drugi, potem naj se zadnji. V zanki potem gledamo preostale ukaze. Dokler so le-ti pogojni bloki, jih zanka obdela. Če je spremenljivka branch nastavljena na vrednost `true` oz. se je nek pogojni blok že izvršil, vse ostale sosednje pogojne bloke ignoriramo. Če

```
1	CanvasElements = []
2	function TiImage(id,hm) {
3	    this.x = 0
4	    this.y = 0
5	    this.h = null
6	    this.w = null
7	    this.offsetX = 0
8	    this.offsetY = 0
9	    this.scaleX = 1
10	    this.scaleY = 1
11	    this.rotate = 0
12	    this.id = id
13	    this.hm = hm
14	    this.img = new Image()
15	    this.a = 1.0
16	    this.e = []
17	    this.zIndex = CanvasElements.length * 0.001
18	    if(id == 0) return
19	    this.img.onload = _imgLoad
20	    this.img.src = '/res/'+ id
21	    this.img.parent = this
22	    CanvasElements.push(this)
23	    this.field = null
24	    this.mx = 0
25	    this.my = 0
26	    for(var i = 0; i < Field.data.fields.length; i++) {
27	        if(Field.data.fields[i].id == id) {
28	            Field.elements.push(this)
29	            this.field = Field.data.fields[i]
30	            this.c = false
31	            this.parent = null
32	            this.children = []
33	        }
34	    }
35	}
```

Slika 3.1: Struktura razreda TiImage z vsemi lastnostmi.

```
} else if (cs[c].type == Block.type.Iteration) {
    var iteratorRid = cs[c].commands[0][0].rid
    var collectionRid = cs[c].commands[0][1].rid
    for(var i = 0; i < this.essence[collectionRid].length; i++) {
        this.essence[iteratorRid] = this.essence[collectionRid][i];
        stream[bp] = c
        stream.push(0)
        arg = this.perform(arg)
        if(arg[0] == HyperMatrix.Type.Break) {
            stream.pop()
            stream[bp]++
            arg[0] = HyperMatrix.Type.Run
            break
        }
        if(arg[0] == HyperMatrix.Type.Return) return arg
        arg[0] = HyperMatrix.Type.Run
    }
```

Slika 3.2: Logika za izvrševanje ponavljalnih blokov.

pa ima vrednost `false`, pa blok vedno izvršimo, če je podtipa `else`, ki je lahko le zadnji med združenimi pogojnimi bloki, ali pa, če je pogoj bloka resničen (slika 3.3).

V primeru, da gre za enostavni ukaz, pa je sestavljen iz skupka elementov, ki jih bomo obravnavali enega za drugim, od leve proti desni. Elementi, ki so lahko tukaj, so povezava oz. `link`, izraz oz. `expression`, akcija oz. `actionCall`, kontrolni element oz. `return` in operator. V osnovi vsi elementi na nek način manipulirajo z neko spremenljivko. Na začetku zato pričakujemo element `link`, ki bo povedal, nad katero spremenljivko naj operiramo. Element link lahko kaže tudi na nek objekt. V temu primeru lahko takemu elementu sledi nov element `link`, ki specificira spremenljivko v objektu, na katerega kaže prvi element `link`. Če naletimo na element `expression`, bo spremenljivka, nad katero operiramo, kazala na evaluirano vrednost izraza. Če naletimo na element `actionCall` nad operirano spremenljivko, poženemo izbrano akcijo. Če naletimo na element `return`, vrnemo vrednost, ki jo vsebuje, in prekinemo izvajanje trenutnega bloka. Opisan interpreter smo napisali v Java Scriptu in je sestavni del predstavljene arhitekture.

```
if(cs[c].type == Block.type.Control) {
    var branch = false
    for( ; ; c++) {
        if(!branch &&(cs[c].commands[0][0] == Block.controlType.Else ||
            cs[c].commands[0][1].getValue(null, this.rid))) {
            branch = true
            stream[bp] = c
            stream.push(0)
            arg = this.perform(arg)
            if(arg[0] == HyperMatrix.Type.Return) return arg
            if(arg[0] == HyperMatrix.Type.Break || arg[0] == HyperMatrix.Type.Continue) {
                stream.pop()
                return arg
            }
        }
        if(c+1 == cs.length) break
        else if(cs[c+1] instanceof Block && cs[c+1].type == Block.type.Control) {
            if(cs[c+1].commands[0][0] == Block.controlType.If)
                branch = false
        } else break
    }
```

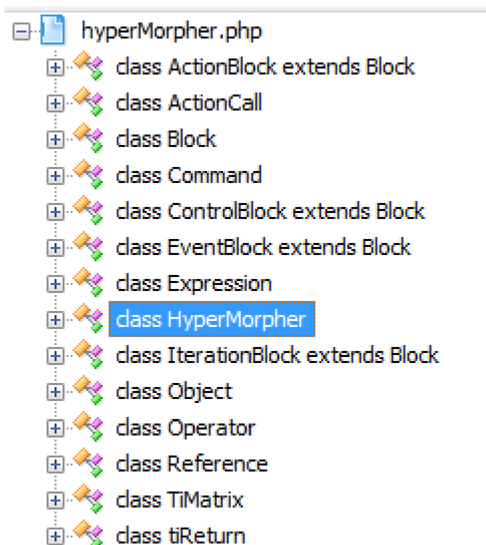
Slika 3.3: Logika za izvrševanje pogojnih blokov.

Poglavje 4

Prevajalnik

Ko je uporabnik zadovoljen z rezultatom, lahko program oz. igro tudi izda. To pomeni, da programu da ime, sliko in še nekatere druge lastnosti, da se bo ta igra pojavila v seznamu iger in bo na voljo za igranje. Takrat se igra tudi prevede iz shranjenega XML-ja, ki predstavlja zapisan vizualni program, v izvršljiv Java Script program. Prevajalnik smo napisali v programskem jeziku PHP. Zgrajen je iz objektov (slika 4.1). Glavni objekt je `HyperMorpher`, v njemu se zgodi celotno branje vhodnega programa in nato shranjevanje izhodnega oz. prevedenega programa. Ostale razrede uporabi za grajenje abstraktnega sintaksnega razreda (angl. abstract syntax tree) [15] na podlagi vhodne datoteke.

Objekt `HyperMorpher` prebere in shrani vse podatke za fiziko in animacijo. Nato gre čez vse matrike in si za vsako shrani ime, njene lastnosti (spremenljivke v prvem bloku) in vse njene akcije. S tem je pridobivanje splošnih podatkov končano in začne se prevajanje matrik, eno za drugo. Prevajanje je rekurzivno. Ko se prevaja nek element, prevede elemente, ki jih vsebuje, doda svoje in konča. To velja pretežno za vse elemente v drevesni strukturi z izjemami, kjer so le-te potrebne, kot so podatki za animacije in fiziko. Ko je drevo končano, se pokliče metodo za prevod v korenskem elementu, to je razred oz. v našem prevajalniku simbol `TiMatrix`. Le-ta prevede matriko v Java Script objekt, ki mu doda še prevedene lastnosti in akcije tako, da kliče metodo za prevod na vseh elementih, ki jih vsebuje, in tako rekurzivno naprej. Za vse dogodkovne bloke pa si, enako



Slika 4.1: Razredi, ki sestavljajo prevajalnik.

kot pri interpretaciji, v spremenljivki `hyperBind` označi, katere metode katerega objekta naj se zgodijo ob določenem dogodku.

4.1 Optimizacija prevedene kode

Koda, ki jo ustvarimo in poženemo, se sproti interpretira oziroma program sproti prebira vizualno kodo in se, glede na prebrano, odloči, kaj naj se zgodi. Zaradi tega je mogoče program spreminjati, medtem ko se le-ta že izvaja. Vendar gre to na račun manjše hitrosti izvajanja. Pri naši igri se to ni poznalo, če pa bi bila igra kompleksnejša, bi bilo dodatnega računanja zaradi interpretiranja toliko, da se program ne bi več tekoče izvajal. Da bi v končni fazi program tekel kar najbolj tekoče, se pri izvozu našega programa vizualna koda prevede v bolj optimizirano in minimizirano Java Script datoteko. Vgrajeni razredi (`Image`, `Animation` ipd.) se še vedno uporabljajo v enaki obliki kot pri interpretaciji. Prevedena in optimizirana pa je koda, ki smo jo ustvarili mi. Primerjava naše vizualne kode in kode, ki jo zgenerira prevajalnik, je vidna na sliki 4.2. Optimizator ni samostojni modul, ampak je že del prevajalnika, tako da je napisan v programskem jeziku PHP.

Minimizira se tako, da se za vse spremenljivke uporabijo kratka imena (črka "r" in identifikacijska številka). Prav tako se v izrazih uporablja čim manj presledkov. Kjer na sam pomen programa ne vplivajo, se izpustijo. V našem jeziku so tudi enostavne matematične operacije (seštevanje, odštevanje, množenje, primerjanje ...) implementirane kot funkcije. To se optimizira tako, da se v izhodnem programu namesto funkcij uporabijo direktni matematični operandi.

```
Event              > Matrix > Sleep 16
    random ⇐ Platform Random min 20  max 600
    plane 1 Change position relatively on X for 0  on Y for 3
    If  420 < to plane 1 Get at 1 GetTop
        plane 1 Change position absolutely on X for random on Y for -20
    plane 2 Change position relatively on X for 0  on Y for 3
    If  420 < to plane 2 Get at 1 GetTop
        plane 2 Change position absolutely on X for random on Y for -20
    plane 3 Change position relatively on X for 0  on Y for 3
    If  420 < to plane 3 Get at 1 GetTop
        plane 3 Change position absolutely on X for random on Y for -20
    If shootPause > to 0
        shootPause ⇐ shootPause - 1
```

```
106 c13e4 = function(t) {
107       var r38=Random(20,600);
108       t.r23.ChangePosition(0,0,3);
109       if (420<t.r23.Get(1).y) {
110         t.r23.ChangePosition(1,r38,-20);
111       }
112       t.r24.ChangePosition(0,0,3);
113       if (420<t.r24.Get(1).y) {
114         t.r24.ChangePosition(1,r38,-20);
115       }
116       t.r25.ChangePosition(0,0,3);
117       if (420<t.r25.Get(1).y) {
118         t.r25.ChangePosition(1,r38,-20);
119       }
120       if (t.r34>0) {
121         t.r34=t.r34-1;
122       }
123 };
```

Slika 4.2: Primerjava ustvarjene vizualne kode (zgoraj) in ekvivalentna prevedena koda (spodaj).

Poglavje 5

Primerjava

Za primerjavo bomo v različnih sorodnih orodjih za izdelavo iger ustvarili podobno igro in primerjali proces izdelave ter s tem kakovost orodij. Izdelali bomo enostavno arkadno igro, pri kateri bo igralec po premikajoči se pokrajini streljal sovražna letala in nabiral točke. V ospredju so bolj tehnični vidiki izdelave in ne toliko vsebinski oz. sama igralnost. V primerjavo sta vključeni orodji Game Maker [16] in Construct 2 [17], ker sta osredotočeni na manj kompleksne 2D igre in imata za sestavljanje skript svoje vizualne programske jezike. Druga orodja za ustvarjanje iger so The 3D GameMaker [18], ki je le orodje za postavljanje že narejenih objektov. Njegov naslednik je FPS Creator Reloaded [19], ki je že naprednejši, vendar njegov skriptni jezik ni vizualen in se tako kot njegov predhodnik osredotoča na 3D. Na trgu je tudi zalo dovršeno orodje Unity [20], ki ima dobro podporo za 2D in 3D, skriptiranje pa je podprto z jezikom C#. Med najboljšimi pa sta še Unreal Developmer's kit [21] in CryEngine [22].

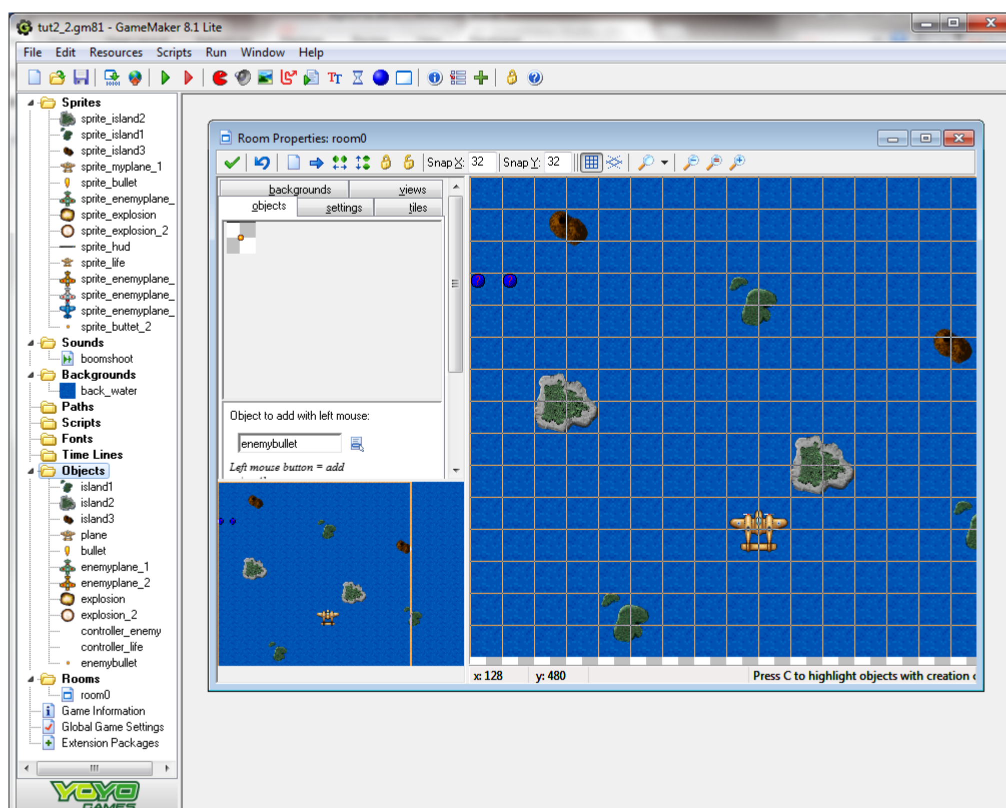
5.1 Game Maker

Program je bil najprej napisan v okolju Delphi, od verzije 8.1 naprej pa v C# in deluje na operacijskih sistemih Windows. Prva izdaja tega programa sega v leto 1999. Takrat je bil namenjen le kreiranju 2D animacij. Leta 2007 so se združili s podjetjem YoYo Games [23], kar je zelo dobro vplivalo na kakovost programa

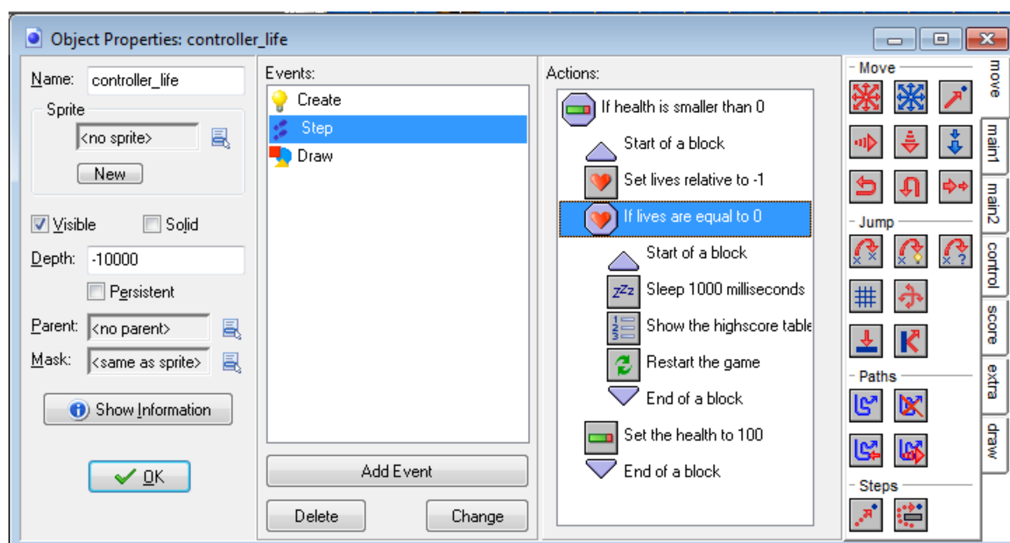
in razširitev skupnosti na spletu. Leta 2012 so vstopili v partnerstvo s podjetjem Valve Software [24] in s tem to orodje ponudili vsem uporabnikom Valvovega distribucijskega ekosistema Steam [25], ki šteje preko 50 milijonov članov. Namen orodja je seveda poenostavitev izdelave iger, pri čemer se nanaša predvsem na uporabo tehnik vleci in pusti (angl. drag and drop), še posebej pri sestavljanju skript. Pri tem iz menija različnih elementov izbrani element kliknemo in povlečemo na želeno mesto. Skripte so lahko čisto grafične, lahko pa jih tudi napišemo v lastnem programskem jeziku z imenom Game Maker Language, ki je objektivno usmerjen programski jezik in je zelo podoben jezikoma C++ in Java. Gre za interpretativni jezik, zaradi česar ni preveč hiter. A imajo v načrtu izdelavo nizkonivojskega prevajalnika, ki naj bi drastično povečal hitrost izdelanih programov. Osredotočen je na 2D grafiko z omejenim naborom možnosti za prikaz 3D modelov. Nudi tudi različne razširitve, ki prinesejo dodatne možnosti, kot so na primer boljša podpora 3D, boljše delo z zvokom in podobno. Orodje lahko dobimo v štirih različnih verzijah:

- **Brezplačna verzija**: nudi osnovne funkcionalnosti in izvoz igre v izvršljiv program za okolje Windows,
- **Standardna verzija**: plačljiva verzija (49,99$), ki omogoča spreminjanje slike (ki je v brezplačni verziji vedno od Game Maker-ja), ki se prikaže, medtem ko se igra zaganja,
- **Profesionalna verzija** (99,99$): omogoča izvoz na različne namizne platforme, ima nekaj več funkcionalnosti in urejeno oglaševanje ter nakupovanje v igri,
- **Master Collection verzija** (799,99$): omogoča izvoz na dodatne platforme, kot so Android, iOS, Windows Phone, Ubuntu in HTML5.

Pri izdelavi igre se je uporabljala brezplačna verzija programa (slika 5.1). Navodila so kratka in jasna ter nove uporabnike hitro uvedejo v način razmišljanja v tem okolju. Postopek izdelave je v osnovi tak, da najprej uvozimo različne medijske elemente (slike in zvoke). Nato naredimo objekte oz. entitete, ki uporabljajo



Slika 5.1: Integrirano razvijalsko orodje Game Maker, trenutno odprt urejevalnik stopenj.



Slika 5.2: Primer lastnosti objekta v vizualnem jeziku Game Maker.

prejšnje elemente, na primer: uvozimo sliko letala, naredimo objekt `plane`, mu določimo, da uporabi to sliko letala ter specificiramo še neko osnovno obnašanje z vizualnim skriptom. Nato ta objekt povlečemo v našo stopnjo oz. sobo. Vizualno programiranje deluje tako, da posameznemu objektu na različne dogodke (dogodki z vhodnih naprav, dogodki na določene časovne mejnike, dogodki, sproženi pri konstrukciji in destrukciji, pri upodabljanju in drugi) povlečemo različne elemente iz menija v polje `Actions`, dodamo svoje parametre in če smo jih napisali pravilno, smo sestavili skript, ki dela to, kar smo želeli (slika 5.2). Dodajanje parametrov je v obliki teksta brez kakšnih pomagal, ki bi omogočala pregled možnosti, ki jih lahko vnesemo, ali avtomatičnega preverjanja pri pisanju, zaradi česar lahko pride do napak, ko program že teče. Na primer: če želimo letalu določiti obnašanje (ko zapusti vidno polje spodaj, naj se prestavi spet na vrh), moramo parametre napisati ročno (x in y koordinato), poznati imena funkcij, metod in sintakso, da lahko napišemo: `random(room_width-40) + 20`. Le majhen tiskarski škrat je potreben in v bo prišlo do napake v izvajanju.

Na koncu končano igro (slika 5.3) izvozimo v format, ki ga podpira naša verzija orodja. V tej naši brezplačni verziji je ta igra le izvršljiv program za Windows,

Slika 5.3: Primer končne igre z orodjem Game Maker.

ki jo lahko gostimo na njihovih straneh. Ima tudi specializirane elemente, ki sami poskrbijo za lestvico najboljših in podobno. V splošnem je cel proces izdelave visoko abstrakten, kar omogoča zelo hitro izdelavo preprostih iger, za večjo kompleksnost ali če želimo kaj bolj svojstvenega, pa hitro naletimo na oviro, pri čemer je treba poseči po naprednejših možnostih, ki jih nudijo plačljive verzije, in polnem skriptnem jeziku. Skupnost je srednje aktivna; če je vprašanje bolj kompleksno, traja dlje časa, da dobimo odgovor na naše vprašanje, ali pa izvemo, da se nečesa ne da narediti ali je plačljivo ali je treba ubrati stranske poti. Orodje pa omogoča izdelavo tudi bolj dodelanih iger, kot je Gunpoint [26] (slika 5.4).

5.2 Construct 2

Construct 2 (slika 5.5) [17] je prav tako integrirano razvojno okolje s poudarkom na enostavnosti razvoja in deluje na operacijskih sistemih Windows. Bistvena razlika je, da ta privzeto izvaža v HTML5 že v brezplačni verziji, s čimer imamo že na začetku veliko večjo potencialno množico uporabnikov (ne glede na platformo, ki jo uporabljajo) in lažjo distribucijo, saj igro le prenesemo na stran in je že na

Slika 5.4: Primer bolj dovršene igre Gunpoint [26], ki jo je moč ustvariti z orodjem Game Maker.

voljo za igranje. Prav tako omogoča vizualno programiranje s podporo polnemu klasičnemu programiranju v programskem jeziku Python. Osredotoča se na 2D in nudi podporo številnim, naprednejšim možnostim, kot so različni grafični efekti, skeletne animacije in napredna fizika. Prihaja v treh verzijah:

- **Brezplačna verzija**: osnovna funkcionalnost,
- **Osebna verzija** (79,73$): naprednejše možnosti, možnost izvoza za različne platforme in omejena komercialna uporaba za nastale produkte,
- **Poslovna verzija** (267,33$): neomejene pravice pri uporabi nastalih produktov v komercialne namene.

Sama filozofija razvoja je zelo podobna kot pri orodju Game Maker. Osredotoča se na posamezne entitete. Navodila in pomoč so zelo obsežni, skupnost je tudi precej aktivna. Ker gre za bolj mlado orodje (verzija 2 je izšla leta 2012, prva verzija pa 2007), se nekatere stvari spreminjajo. Pogosto se zgodi, da nekatera navodila niso posodobljena in je potem potrebno iskati, kako se kaj naredi na drugih

Slika 5.5: Orodje Construct 2.

mestih. Pri uporabi se dobi občutek, da gre za bolj profesionalno orodje od Game Maker-ja. Funkcionalnosti za kreiranje entitet, določanje prikaznih nastavitev in urejevalnik stopenj so zelo dobro podprti. Vizualno programiranje (slika 5.6) pa je na približno istem nivoju kot prejšnje orodje, le da ima to več in bolj napredne funkcije.

Zelo dober vtis sicer pokvari dejstvo, da so nekatere čisto osnovne možnosti plačljive, kot na primer, da določenemu elementu določimo globino oz. z-indeks. Pri Game Maker-ju je to dobro narejeno in za osnovne stvari ne naletimo na funkcije, ki bi bile nedostopne zaradi brezplačne verzije. Izrazov se ne sestavlja vizualno, ampak se jih piše, zaradi česar lahko pride lažje do napak. Igro v osnovni verziji privzeto izvozi v HTML5, zaradi česar je distribucija močno poenostavljena in jo lahko gostimo kar na njihovih straneh. Tudi že ustvarjene igre (slika 5.7) so zaradi boljšega orodja lepše in bolj izpopolnjene v Construct-u.

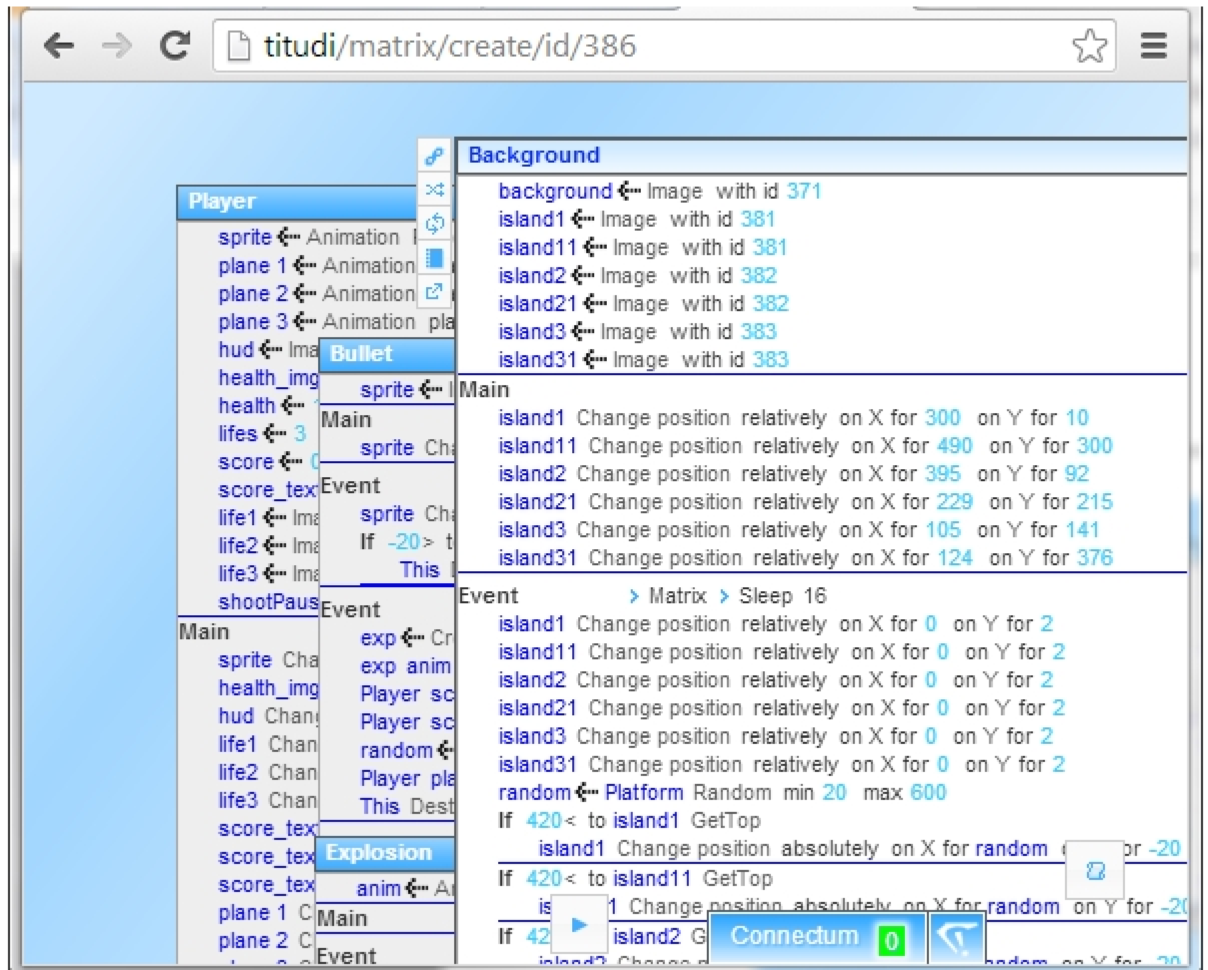
	Object	Condition	Object	Action
37	**Enemy Collisions**			
38	enemy1	On collision with **player**	System	Create object **explosion_1** on layer **0** at (*enemy1.X*, *enemy1.Y*)
			fx	Play **snd_explosion1** not looping at volume 0 dB (tag "")
			enemy1	Set position to (*random(620) + 10* , *-20*)
			player	Set **health** to *player.health - 34*
			health	Set width to *health.Width - 42*
			Add action	
39	enemy2	On collision with **player**	System	Create object **explosion_1** on layer **0** at (*enemy2.X*, *enemy2.Y*)
			fx	Play **snd_explosion1** not looping at volume 0 dB (tag "")
			enemy2	Set position to (*random(620) + 10* , *-20*)
			player	Set **health** to *player.health - 34*
			health	Set width to *health.Width - 42*
			Add action	
40	enemy3	On collision with **player**	System	Create object **explosion_1** on layer **0** at (*enemy3.X*, *enemy3.Y*)
			fx	Play **snd_explosion1** not looping at volume 0 dB (tag "")
			enemy3	Set position to (*random(620) + 10* , *-20*)
			player	Set **health** to *player.health - 34*
			health	Set width to *health.Width - 42*

Slika 5.6: Vizualno programiranje v Construct-u.

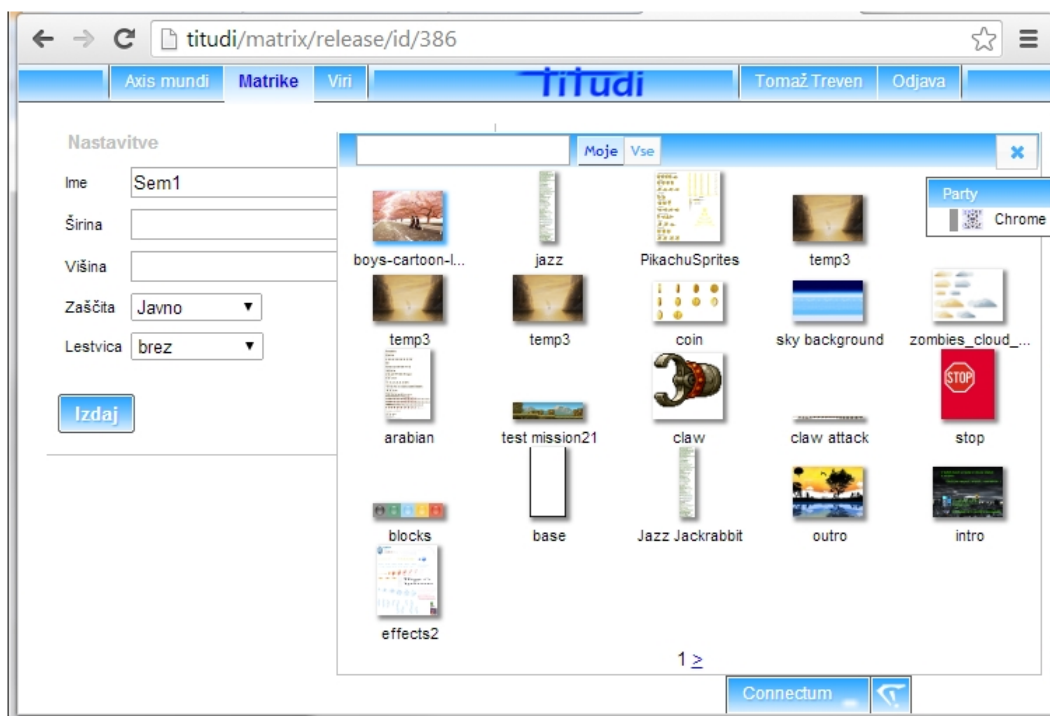
Slika 5.7: Primer bolj dovršene igre Super Ubie land [27], narejene z orodjem Construct 2.

5.3 TiTudi

TiTudi je ime našega orodja. Ime pove, da je igro preprosto narediti, da jo lahko naredi vsak, tudi ti. V primerjavi z ostalimi orodji je to najmlajše in najmanj dodelano. Prva razlika je ta, da je celoten program spleten in ne potrebujemo nobenega nameščanja, prav tako pa lahko ustvarjamo na operacijskem sistemu po izboru in iz kateregakoli računalnika, dokler smo prijavljeni pod istim uporabnikom. Ena izmed večjih razlik v pristopu ustvarjanja iger v primerjavi s prejšnjimi orodji je ta, da se tukaj izhaja iz vizualnega programiranja in ne iz urejanja grafičnih elementov (slika 5.8). Trenutno direktnega manipuliranja grafičnih elementov ni, vse se naredi preko vizualnega programiranja. Tukaj je sintaksa jezika bolj podobna splošnim programskim jezikom, da uporabniku omogoči kar se da veliko stopnjo fleksibilnosti, brez da bi postal jezik preveč kompleksen. Sestavljanje kode je dokaj hitro in enostavno, spreminjanje že obstoječe pa v določenih primerih ni tako lahko; če bi bilo samo urejanje teksta kot pri nevizualnih jezikih, bi bilo spreminjanje hitrejše.



Slika 5.8: Razvojno okolje TiTudi.



Slika 5.9: Stran za izvoz igre.

Potem ko igro prvič shranimo, je le-ta v obliki izvorne kode na voljo tudi vsem ostalim uporabnikom, da jo poljubno spreminjajo, nadgrajujejo ali pa uporabijo samo nekatere dele programa, podobno kot spletni repozitoriji (github in drugi). Le da je tu vse že integrirano in ni potrebno imeti posebnega programa za kontrolo revizij in razvojnega okolja, da preneseno kodo testiramo.

Ko končamo z ustvarjanjem, je tudi distribucija zelo enostavna. Določimo nekaj osnovnih lastnosti (slika 5.9), kot so ime igre, velikost igralnega polja, prikazna slika in še nekaj opcijskih zadev. Naša igra (slika 5.10) se nato prevede v optimizirano obliko in je pripravljena, da jo lahko uporabniki igrajo.

Navodil trenutno ni veliko, prav tako ne vodičev, zato je za nove uporabnike verjetno težje začeti s tem orodjem. Ker je orodje mlado in še bolj kot ne v razvojni stopnji, bi bila kakšna dodatna funkcionalnost zelo dobrodošla, na primer direktno manipuliranje grafičnih elementov. Trenutno pa je mogoče ustvarjati zelo preproste igre (slika 5.11). Prav tako lahko hitro naletimo na kakšne programske hrošče.

Slika 5.10: Dokončana igra v razvojnem okolju TiTudi.

Slika 5.11: Primer igre [28], ki jo je možno narediti v našem okolju.

Prednost pa je v tem, da je vse brezplačno. Trenutno stanje je za nove uporabnike morda preveč zapleteno, tako da verjetno izberejo kakšno izmed prejšnjih orodij.

Poglavje 6

Vizija razvoja in tržne možnosti

Gonilo prodaje nekega sistema, recimo igralne konzole, ko govorimo o igrah, je glavna programska oprema. To se je pokazalo že na začetku prodaje takšnih sistemov, pri igralni konzoli Atari 2600 leta 1977, ko je izšla igra Space invaders, zaradi katere se je prodaja povečala za štirikrat. Taki aplikaciji, ki je zelo popularna in zaradi katere se poveča razširjenost platforme, na kateri se izvaja, so takrat prvič rekli ubijalska aplikacija (angl. killer app). To se je kasneje potrdilo pri izidih vseh večjih konzol: tiste, ki so imele boljše igre, so bile popularnejše. Tudi danes mora vsaka nova konzola imeti vsaj enega paradnega konja, če želi dobre prodajne številke. Za Nintendo je zelo prepoznavna serija Super Mario Bros, Final Fantasy VII in Metal Gear Solid za PlayStation ter seveda serija Halo za Xbox. Izdelki za Apple mobilne naprave pa so integrirani v njegovo trgovino App Store, v kateri lahko enostavno pridobimo neizmerno veliko iger. Če bi naš izdelek dobil zagon, da bi pridobil veliko ustvarjalcev, ki bi delali dobre igre, bi imeli izdelovalci strojne opreme korist, da bi bolj podpirali spletno platformo, saj bi s tem omogočili, da bi njihove stranke dostopale do velikega števila iger. Izdelek pa bi s tem še povečal število igralcev oz. kupcev. Možnost bi tudi bila, da se izvorna koda izdelka prevede za posebej določeno platformo, s čimer bi izdelek prav tako pridobil na trgu. V dogovoru z lastnikom konkretne platforme pa bi se našla še kakšna korist. Predvsem pa bi bil ciljni trg zaradi enostavnega vstopa v ekosistem namenjen občasnim igralcem, od katerih ne pričakujemo velikih igralnih izkušenj, in uporabnikom z

nižjimi dohodki. V zadnjem času je viden trend mikro-transakcij, oz. da je igra zastonj, potem pa so v igri določene koristi ali predmeti, ki jih je potrebno kupiti. Ker naš izdelek pokriva celotno izdelavo in distribucijo iger, bi bila trgovina v igri pod okriljem TiTudi-ja. Pri takih transakcijah bi delež prodaje zadržali za zagotavljanje teh storitev (kot recimo Apple na App Store-u). Če pa bi postalo to močno in vsestransko orodje, pa bi bilo mogoče ustvarjati tudi druge programe, ne le iger, pri katerih bi se tudi našle možnosti za monetizacijo. Možnosti za nadaljnji razvoj so neizmerne. Naštejmo nekatere izmed najbolj zanimivih:

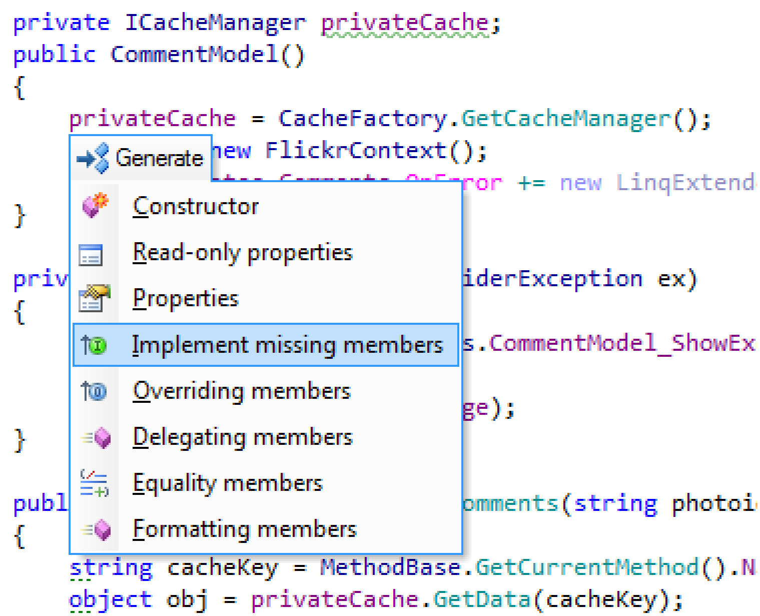
- Bolj optimizirano prevajanje ali prevajanje za kakšno drugo platformo, kot sta Android, iOS.
- Boljše razhroščevanje, ki bi uporabniku razkrilo, kaj točno se v neki točki izvajanja programa dogaja, saj bi to pospešilo ter olajšalo razvoj.
- Podpora 3D, nadgraditev celotnega izrisovalnega pogona, ki bi podpiral 3D, in nadgraditev podporne knjižnice in pripomočkov, ki bi omogočili ustvarjanje v 3D okolju.
- Več knjižnic, ki bi omogočale več različnih stvari oz. kar izdelavo vseh vrst programov.
- Številne izboljšave, nadgraditve in poenostavitve procesov za izdelavo programov.

Poglavje 7

Sklepne ugotovitve

Spreminjanje ali urejanje v vizualnem načinu je relativno dolg proces, ker je treba razbiti zgrajene elemente. Če pa imamo samo tekst, je to hitro. Prav tako je vizualni način primernejši za začetnike, da jim olajša programiranje z vsiljevanjem strukture in predlogi. Za izkušene programerje pa je lahko to ovira, zato mora biti uporaba res dobro premišljena, da pomaga in hkrati ne omejuje programerja. V vseh treh orodjih, ki smo jih primerjali, je za večje projekte tak način vizualnega programiranja omejujoč. Bolje bi bilo, da bi izhajali iz tekstovnega urejanja, ki bi ga nadgradili z vizualnimi nadgradnjami, kot je to pri najboljših orodjih za profesionalno programiranje, na primer Visual Studio, kjer imamo samo vizualne pripomočke, kot je na primer dodatek za Visual Studio ReSharper (slika 7.1).

Tudi če imamo še tako dobra in enostavna orodja, s katerimi se lahko hitro dela, je še vedno potrebno kar nekaj truda, da se naredi privlačna igra. Ta orodja se osredotočajo predvsem na tehnični in programski del. Še vedno pa je potrebno postaviti temo, zgodbo, oblikovati okolje, narisati in zmodelirati različne elemente, ustvariti glasbeno podlago, posneti dialoge, zvočne efekte, poskrbeti za igralnost. Na koncu pa nas čaka še veliko testiranja in distribucija, čeprav naš projekt slednje zelo poenostavi.



Slika 7.1: Primer, kako nam dodatek ReSharper pomaga pri programiranju.

Literatura

[1] (2014) News: Red Dead Redemption costs $100 million. Dostopno na: http://www.computerandvideogames.com/247325/red-dead-redemption-cost-100-million-report/

[2] (2014) Video game development. Dostopno na: http://en.wikipedia.org/wiki/Video_game_development

[3] (2014) Viacom reports decline of sales of Rock Band. Dostopno na: http://www.examiner.com/article/viacom-reports-decline-sales-of-rock-band

[4] (2014) Programming language. Dostopno na: http://en.wikipedia.org/wiki/Programming_language

[5] (2014) Apache HTTP Server. Dostopno na: http://httpd.apache.org/

[6] (2014) Zend PHP Web Application Server. Dostopno na: http://www.zend.com/en/

[7] (2014) Model-view-controller. Dostopno na: http://http://en.wikipedia.org/wiki/Model–view–controller

[8] (2014) TiTudi Create. Dostopno na: http://titudi.com/matrix/create

[9] (2014) jQuery. Dostopno na: http://jquery.com/

[10] (2014) PHP IDE NuSphere PhpED for PHP Developers. Dostopno na: http://www.nusphere.com/products/phped.htm

[11] K. Beck, *Implementation Patterns*, Addison-Wesley, 2007

[12] (2014) Sprite (computer graphics). Dostopno na: http://en.wikipedia.org/wiki/Sprite_(computer_graphics)

[13] (2014) Collision detection. Dostopno na: http://en.wikipedia.org/wiki/Collision_detection

[14] (2014) Minimum bounding box. Dostopno na: http://en.wikipedia.org/wiki/Minimum_bounding_box

[15] K. C. Louden, K. A. Lambert, *Programming Languages Principles and Practice*, Cengage Learning, 2012

[16] (2014) Game maker. Dostopno na: http://en.wikipedia.org/wiki/GameMaker:_Studio

[17] (2014) Construct 2. Dostopno na: http://en.wikipedia.org/wiki/Construct_2

[18] (2014) The 3D Game Maker – The Game Creators. Dostopno na: http://www.thegamecreators.com/?m=view_product&id=2126

[19] (2014) FPS Creator Reloaded by TGC. Dostopno na: http://fpscreator.thegamecreators.com/

[20] (2014) Unity - Game Engine. Dostopno na: http://unity3d.com/

[21] (2014) Free Game Engine for Indie Game Development — UDK Unreal Developer's Kit. Dostopno na: http://www.unrealengine.com/en/udk/

[22] (2014) Crytek — MyCryENGINE. Dostopno na: http://mycryengine.com/index.php?conid=1

[23] (2014) YoYo Games. Dostopno na:
http://www.yoyogames.com/

[24] (2014) Valve. Dostopno na:
http://www.valvesoftware.com/

[25] (2014) Steam. Dostopno na:
http://store.steampowered.com/

[26] (2014) Gunpoint — Showcase — YoYo Games. Dostopno na:
https://www.yoyogames.com/showcase/6

[27] (2014) Null – Scirra Store. Dostopno na:
https://www.scirra.com/store/games/super-ubie-land

[28] (2014) TiTudi. Dostopno na:
http://titudi.com/index/experience/rid/14