

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jernej Plohl

**Primerjava orodij Unity3D in
Construct 2 za razvoj iger za mobilne
platforme.**

DIPLOMSKO DELO

UNIVERZITETNI STROKOVNI ŠTUDIJSKI PROGRAM PRVE
STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: prof. dr. Saša Divjak

Ljubljana 2014

Rezultati diplomskega dela so intelektualna lastnina avtorja. 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 L^AT_EX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Cilj diplomske naloge je primerjava igralnih pogonov Unity 3D in Construct 2 za izdelavo iger za mobilne platforme. Predstavite teoretično ozadje, ki je potrebno za izdelavo igre. Primerjajte izbrana igralna pogona, tako, da s pomočjo obeh igralnih pogonih realizirate enako igro. Raziščite probleme, na katere naletite pri razvoju iger. Preverite delovanje razvitih iger na operacijskih sistemih Android.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Jernej Plohl, z vpisno številko **63100289**, sem avtor diplomskega dela z naslovom:

Primerjava orodij Unity3D in Construct 2 za razvoj iger za mobilne platforme.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Saše Divjaka,
- 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 na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 16. septembra 2014

Podpis avtorja:

*Zahvaljujem se moji družini za vso podporo in pomoč pri mojem študiju.
Zahvaljujem se mentorju dr. Saši Divjaku.*

Kazalo

Povzetek

Abstract

1	Uvod	1
2	Geometrija in transformacije	3
2.1	Vektorji	3
2.2	Afine transformacije	5
2.3	Homogene koordinate	6
3	Fizikalni pogon	9
3.1	PhysX	10
3.2	Box2D	10
4	Igralni pogon	11
4.1	Unity	11
4.2	Construct	12
5	Mobilni operacijski sistemi	15
5.1	Android	16
6	Izdelava igre	19
6.1	Opis igre	19
6.2	Izdelava osnovne scene	20

KAZALO

6.3	Premikanje in fizika	23
6.4	Pobiranje pribolškov in točkovanje	31
6.5	Grafični uporabniški vmesnik	33
6.6	Prehajanje med nivoji in shranjevanje podatkov o igri.	37
6.7	Izvoz	40
6.8	Testiranje	41
7	Sklepne ugotovitve	45

Povzetek

Priljubljenost mobilnih iger se je močno povečala s pojavom pametnih telefonov. Skladno s tem se je povečalo število igralnih pogonov, ki so namenjeni za izdelavo iger za mobilne platforme. Cilj diplomske naloge je primerjava igralnih pogonov Unity 3D in Construct 2 za izdelavo iger za mobilne platforme. V prvem delu je predstavljeno teoretično ozadje, ki je potrebno za izdelavo igre. Drugi del je namenjen primerjavi igralnih pogonov. Primerjava igralnih pogonov je potekala tako, da smo v obeh igralnih pogonih realizirali enako igro. Problem izdelave igre je razdeljen na več podproblemov. V nalogi je opisan vsak podproblem in kako je podproblem rešen v obeh igralnih pogonih. Na koncu naloge je opisano delovanje obeh iger na mobilnem telefonu z operacijskim sistemom Android.

Ključne besede: Unity, Construct, mobilne igre, igralni pogon, Android.

Abstract

With smart phones the popularity of mobile games has increased significantly. Accordingly, the number of game drives, which are designed to produce games for the mobile platform has also increased. The aim of this thesis is comparison of gaming drives Unity 3D and Construct 2 regarding production of games for mobile platforms. The first part presents theoretical background needed for production of a game. The second part compares the gaming drives. Comparison was made by implementing the same game on both drives (platforms). The task of manufacturing a game is divided into several subtasks. Each subtask is presented and how it is implemented on both drives. At the end of the thesis we describe how this two versions of the same game (made in two different environments) are working on mobile phone with Android operating system.

Keywords: Unity, Constuct, mobile games, game engine, Android.

Poglavje 1

Uvod

Tehnologije mobilnih iger vse bolj razvijajo vse od prihoda prvih iger. Prva mobilna igra je bila Tetris razvita leta 1994. Nato je sledila ena izmed najbolj poznanih iger Kača, ki jo je predstavilo podjetje Nokija. Na začetku so bile igre enostavne brez posebnih učinkov in napredne grafike. Igre so se odvijale v dvo dimenzionalnem prostoru. Upravljanje je potekalo preko smernih tipk na telefonu ali kar preko številčnice. Ob pojavu tako imenovanih pametnih telefonov pa so se mobilne igre začele hitro razvijati.

Novejši mobilni telefoni nam ponujajo novo uporabniško izkušnjo pri igranju iger. Upravljanje igre zdaj poteka preko zaslona na dotik. Ker so mobilni telefoni vse bolj zmogljivi, lahko na njih poganjamo tudi zahtevnejše igre z napredno grafiko in naprednimi učinki. Sodobni telefoni omogočajo igre v 2D in 3D prostoru. V igri pa lahko uporabimo tudi razne senzorje, ki jih imajo danes mobilni telefoni. Tako lahko uporabimo senzor, ki meri gravitacijske pospeške in GPS senzor in tako naprej.

Za pojavom kompleksnih mobilnih iger, pa se je razvilo tudi kar nekaj igralnih pogonov, ki omogočajo enostavnejši in hitrejši razvoj igre. In razvijalec je pogosto v dilemi katerega, naj izbere. Prav tako je vedno bolj zaželeno, da je omogočen enostaven izvoz igre za različne platforme.

V diplomskem delu smo predstavili dva igralna pogona Unity 3D in Construct 2. Predstavili smo ju tako, da smo enako igro razvili v obeh pogonih

in ju primerjali in opisali prednosti in slabosti.

Prvi del diplomskega dela vsebuje teoretično podlago, ki je potrebna za samo izdelavo igre in opis nekaterih osnovnih pojmov. V drugem delu pa smo se posvetili sami izdelavi mobilne igre v obeh igralnih pogonih in opisu osnovnih problemov, na katere smo naleteli ob izdelavi igre.

Poglavje 2

Geometrija in transformacije

2.1 Vektorji

Za izdelavo igre je potrebno dobro znanje iz področja vektorjev in linearne algebre. Linearno algebro potrebujemo, za opis položaja in gibanje objektov, za opis položaja objektov glede na navidezno kamero in projekcijo scene iz koordinatnega sistema sveta v koordinatni svet kamere. Pomembnejše operacije nad vektorji so:

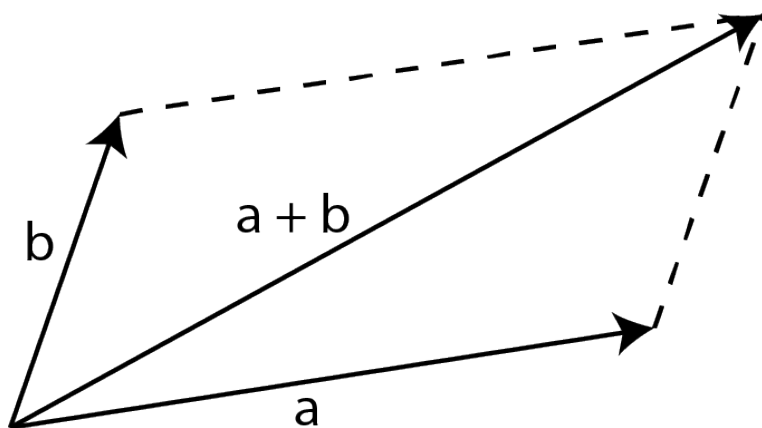
Seštevanje vektorjev (2.1). Rezultat vsote vektorjev je vektor, ki poteka od začetne točke prvega vektorja (A), do končne točke drugega vektorja (C). Slika 2.1.

$$\vec{AB} + \vec{BC} = \vec{AC} \quad (2.1)$$

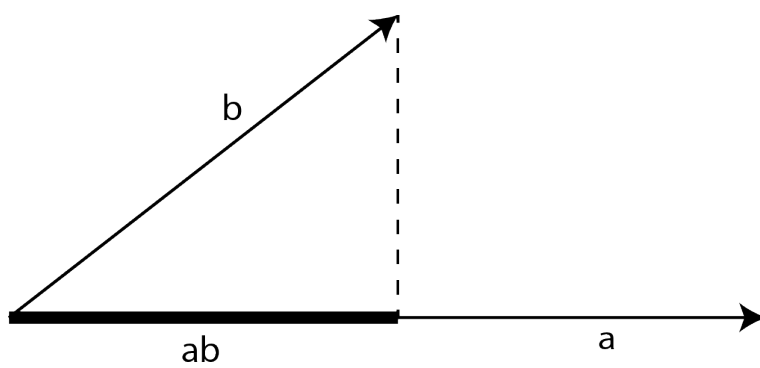
Skalarni produkt vektorjev (2.2). Skalarni produkt je operacija, ki dvema vektorjema priredi število (skalar). Skalarni produkt je dolžina projekcije enega vektorja na drugi vektor. Slika 2.2.

$$\vec{a}\vec{b} = |\vec{a}||\vec{b}| \cos \varphi \quad (2.2)$$

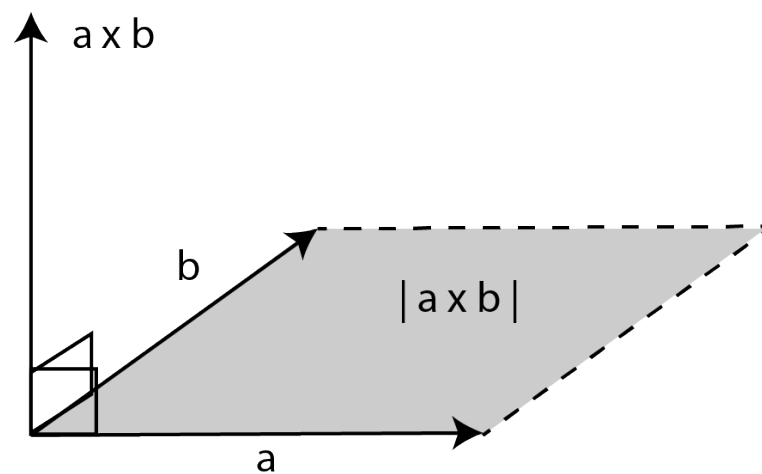
Vektorski produkt $\vec{a} \times \vec{b}$. Vektorski produkt dvema vektorjema priredi rezultat, z naslednjimi lastnostmi (slika 2.3):



Slika 2.1: Seštevanje vektorjev.



Slika 2.2: Skalarni produkt vektorjev.



Slika 2.3: Vektorski produkt vektorjev.

- rezultat je pravokoten na oba vektorja,
- dolžina rezultata je enaka ploščini paralelograma, ki ga oklepata oba vektorja,
- smer rezultata je v desnosučnem koordinatnem sistemu.

2.2 Afine transformacije

Transformacija je funkcija, ki poljubno točko ali vektor preslika v drugo točko ali vektor. Za afine transformacije je značilno, da ohranjajo vzporednost daljic, ne ohranjajo pa dolžin in kotov. V dvodimenzionalnem prostoru poznamo naslednje transformacije.[9]

2.2.1 Translacija

Translacija je premik objekta, za določen vektor premika. Translacijo izvedemo s prištevanjem premestitvenih razdalj t_x in t_y h koordinatam točke.

Translacijo v vektorski obliki zapišemo kot:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (2.3)$$

2.2.2 Skaliranje

Skaliranje je povečevanje oziroma pomanjševanje objekta. Skaliranje v matrični obliki zapišemo kot:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.4)$$

Enačba (2.4) opisuje skaliranje glede na koordinatno središče. Če velja $s_x = s_y$ govorimo o enakomernem skaliranju, drugače pa o neenakomernem skaliranju.

2.2.3 Rotacija

Rotacija je zasuk objekta okoli točke. Enačba (2.5) opisuje zasuk okoli koordinatnega središča. Pri čemer α predstavlja kot, za katerega želimo rotirati objekt. Rotacija poteka v matematično pozitivni smeri oziroma v nasprotni smeri gibanja urinega kazalca.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.5)$$

2.3 Homogene koordinate

Cilj je predstaviti vse afine transformacije kot množenje matrik, da lahko na enostaven način sestavljamo poljubne transformacije. Točko $P(x, y)$ predstavimo v homogenih koordinatah kot $P_h(hx, hy, h)$, pri čemer je h različen od nič. Afine transformacije v homogenem prostoru:[9]

- Translacija

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.6)$$

- Skaliranje

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.7)$$

- Rotacija

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.8)$$

Poglavje 3

Fizikalni pogon

Fizikalni pogon je programska oprema, ki skrbi za približno simulacijo fizikalnih sistemov. Fizikalni pogon skrbi za simulacijo togih teles, dinamiko mehkih teles, ter simulacijo tekočin. Za nadziranje gibanja objektov fizikalni pogon uporablja drugi Newtonov zakon $F = ma$. Fizikalne pogone v splošnem delimo na takšne, ki izvajajo izračun fizike v realnem času in na visoko natančnostne. Za igre uporabljamo relanočasovne fizikalne pogone. Za njih je značilno, da so izračuni fizike poenostavljeni, na račun tega izgubimo natančnost izračuna fizike. Vendar se ta izračun izvede v primernem času, da ta ne vpliva na odzivnost igre. V splošnem realnočasovne fizikalne pogone uporabljamo za vse interaktivne aplikacije, pri katerih želimo takojšen odziv. Visoko natančnostni fizikalni pogoni zahtevajo veliko procesorske moči, za izračun zelo natančne grafike. Le ti se uporabljajo za znanstvene simulacije in animacijo. Fizikalni pogon naravno razdeli fizikalno simulacijo na dve fazi, zaznavanje trkov in odziv na le te. Zaznavanje trkov se nanaša na zaznavanje ali se dva objekta trenutno sekata, ali se bosta morda sekala v prihodnosti. Fizikalni pogon se mora na detekcijo pravilno odzvati, tako na primer, če žoga pade na tla pričakujemo, da se bo ta od tal odbila.[6]

Nadaljnje bomo opisali dva fizikalna pogona, ki jih uporabljata tudi Unity in Construct.

3.1 PhysX

PhysX je fizikalni pogon, ki simulira fiziko v realnem času. Razvila ga je Ageia, od leta 2008 pa je v lasti podjetja Nvidia.

PhysX je na voljo za operacijske sisteme Microsoft Windows, Mac OS X, Linux, PlayStation 3, PlayStation 4, Xbox360 in Android. PhysX razvojno okolje je za Windows na voljo tako za ne komercialno, kot tudi komercialno rabo. Za platforme Linux, Mac OS in Android je PhysX brezplačno na voljo za namene izobraževanja in v ne komercialne namene.

Fizikalni pogon PhysX omogoča simulacijo togih teles, simulacijo mehkih teles, kontroliranje karakterjev, dinamiko vozil, simulacijo fluidov in simulacijo oblčil.[14][13]

3.2 Box2D

Box2D je odprtokodni fizikalni pogon, ki simulira dvodimenzialno grafiko. Napisan je v programskem jeziku C++. Razvil ga je Erin Catto. Box2D je prilagojen tudi drugim programskim okoljem, kot so Java, Adobe Flash, C#, Lua, JavaScript.

Box2D omogoča simulacijo togih teles. Simulira lahko konkavne poligone in kroge. Simulira tudi gravitacijo, trenje in odziv na trke. Omogoča tudi pritrditev objekta na drug objekt in združevanje objektov. [4]

Poglavje 4

Igralni pogon

Igralni pogon (game engine) je programsko orodje, ki je namenjeno izdelavi računalniških iger. Namen igralnega pogona je poenostaviti izdelavo računalniških iger. Uporabnikom ni potrebno poznati vseh nizko-nivojskih konceptov izdelava igre, ampak se lahko osredotočijo na višje nivojske funkcionalnosti igre. Običajno igralni pogon vsebuje naslednje funkcionalnosti: pogon za upodabljanje 2D in 3D grafike, fizikalni pogon, sistem za dodajanje zvoka, podporo animaciji. Veliko igralnih pogonov skrbi tudi za enostaven izvoz igre za različne platforme. V nadaljnje bomo opisali igralna pogona Unity in Construct, ki jih bomo kasneje uporabili za primerjavo in razvoj igre.[8]

4.1 Unity

Unity je igralni pogon, ki ga je razvilo podjetje Unity Technologies. Namenjen je izdelavi igre za različne platforme. Omogoča razvoj 3D iger od leta 2013 pa vsebuje tudi neposredno podporo za razvoj 2D iger.

Unity omogoča izvoz igre za različne platforme. Podprte so naslednje platforme:

- Namizne platforme: Linux, Mac, in Windows
- Mobilne platforme: Android, BlackBerry 10, iOS in Windows phone 8

- Igralen konzole: PS3, Wii U, Xbox 360
- Spletne platforme, preko spletnega vmesnika Unity.

Programiranje v Unity se vrši preko odprtokodne .NET platforme Mono. Omogočeno je programiranje v jezikih C#, JavaScript in Boo. Od verzije 3.0 Unity uporablja orodje MonoDevelop za razhroščevanje skript.

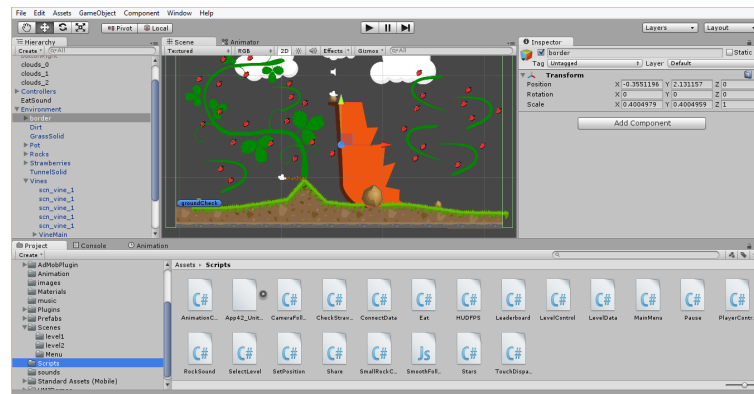
Unity je na voljo v brezplačni in plačljivi verziji Unity Pro. Za plačljivo verzijo je treba odšteti 1500 dolarjev, ali pa 75 dolarjev na mesec. Unity Pro ima dodatne funkcionalnosti, ki pripomorejo k še večji realističnosti samih iger. Predvsem vsebuje več orodij za bolj realistično grafiko, boljši zvok in boljši nadzor nad animacijo. Vsebuje tudi profiler, ki nam pomaga pri optimizaciji igre. Sporoči nam, koliko se je porabilo časa za prikaz, animacijo, fiziko, logiko igre in tako naprej. Če uporabljamo brezplačno verzijo Unity se nam vedno od zagonu igre, prikaže Unity začetni zaslon, pri spletnih igrah pa je vseskozi vidna digitalna vodna oznaka z napisom Unity.[17]

Za simulacijo tridimenzionalne fizike Unity uporablja fizikalni pogon PhysX, ki smo ga opisali v 3.1. Za simulacije dvodimenzionalne fizike se uporablja fizikalni pogon Box2D opisan v 3.2.

Uporabniški vmesnik programa Unity (Slika 4.1) je v osnovi razdeljen na več modulov. V sredini je modul, v katerem urejamo sceno. Levo je modul, ki prikazuje hierarhijo objektov, ki se nahajajo na sceni. Spodaj so prikazane vse datoteke in mape, ki se nahajajo znotraj objekta. Desno se prikažejo vse možne nastavitve izbranega objekta. Zgoraj se nahaja orodna vrstica [10].

4.2 Construct

Construct je igralni pogon, ki omogoča izdelavo dvodimenzionalnih HTML5 iger. Razvilo ga je podjetje Scirra. Modeliranje igre temelji na sistemu povleci in spusti. Za nadzor poteka igre se uporablja seznam dogodkov, ki ga sestavljajo dogodki, ki vsebujejo pogojne stavke ali sprožilce. Dogodke med seboj povezujemo s pomočjo logičnih opretorejv *in* in *ali*. Ta način omogoča



Slika 4.1: Uporabniški vmesnik programa Unity.

izdelavo igre brez, da bi se pred tem morali učiti katerega od kompleksnejših programskih jezikov.

Construct temelji na tehnologiji HTML5 vsi dogodki se pretvorijo v programski jezik JavaScript, saj je to standardni jezik za spletne aplikacije. Za izračun fizike uporablja Construct fizikalni pogon Box2D, ki smo ga opisali v 3.2.

Construct je na voljo z različnimi licencami. Construct2 Free Editino je popolnoma brezplačen. Namenjen je za ne komercialno rabo. Uporabljajo ga lahko tudi ne komercialne ustanove, kot je na primer šola. Brezplačna različica ima določene omejitve, in sicer omejeno število dogodkov na projekt, omejeno število plasti in efektov, ne omogoča iskanja dogodkov in podobno. Da se izognemo tem omejitvam si lahko priskrbimo program z osebno licenco, ki je namenjena osebni rabi in za ne profitne organizacije. Za profitne organizacije je na voljo Construct s poslovno licenco.[18]

Poglavje 5

Mobilni operacijski sistemi

Mobilni informacijski sistemi so informacijski sistemi, ki upravljajo pametne telefone, tablice, dlančnike in ostale mobilne naprave. Za sodobne mobilne informacijske sisteme je značilno, da kombinirajo nekatere lastnosti namiznih operacijskih sistemov z ostalimi funkcionalnostmi, ki so značilne za mobilne naprave. Tako nudijo podporo za zaslone na dotik, kamero, Bluetooth, dostop do mobilnega omrežja, gravitacijske senzorje, GPS, NTF (near field communication), prepoznavanje zvoka in glasovno upravljanje, predvajanje glasbe in podobno.[11]

Po podatkih in februarja 2014 je razširjenost mobilnih informacijskih sistemov naslednja:[16]

- Android: 52,1%
- iOS: 41,3%
- Windows Phone: 3,4%
- BlackBerry: 2,9%
- Symbian: 0,2%

Ker je Android najbolj razširjen mobilni informacijski sistem, ga bomo tudi podrobneje opisali.

5.1 Android

Android je odprtokodni informacijski sistem, ki temelji na jedru Linux. Uporabniški vmesnik temelji na neposredni manipulaciji in je bi zasnovan predvsem za mobilne naprave z zaslonom občutljivim na dotik, kot so pametni telefoni in tablice. Danes operacijski sistem srečamo tudi na drugih napravah, na televizijah, igralnih konzolah, fotoaparatih in tako naprej. Odprtokodno izvorno kodo operacijskega sistema Android je izdalo podjetje Google. Za večino naprav Android je značilno, da uporabljajo kombinacijo odprtokodnega in lastniškega sistema, torej večina proizvajalcev k osnovnemu Androidu doda nekaj svojih funkcionalnosti.[2]

Android je izšel v različnih verzijah. Vsaka verzija je označena s številko in imenom. Vsaka verzija se imenuje po sladici, imena si sledijo po abecednem redu. Tabela 5.1 prikazuje vse do zdaj izdane verzije operacijskega sistema Android.[3] Za aplikacije je značilno, da je napisana za določeno verzijo sistema Android, bo ta delovala tudi na vseh naslednjih verzijah, obratno seveda ne velja. Torej je treba pred razvojem aplikacije premisliti, na katerih sistemih Android bo le ta poganjala.

Največ aplikacij za sistem Android se prenesi iz spletne trgovine Google store, dosegljive pa so tudi na drugih spletnih trgovinah kot je na primer Amazon Appstore. Google store je uradna spetna trgovina podjetja Google. Uporabnikom je omogočeno iskanje, prenašanje in posodabljanje aplikacij, ki jih je razvilo podjetje Google ali kateri drugi razvijalci. Trenutno (18. 6. 2014) ja za sistem Android na voljo 1.261.642 aplikacij.[12]

Aplikacije za Android so primarno napisane v programskem jeziku Java z uporabo Android razvojne programske opreme (SDK - software development kit). SDK vsebuje orodja za razvoj, razhroščevanje, programske knjižnice, emulator, dokumentacijo in primere kode. Uradno podprto razvojno okolje za aplikacije Android je Eclipse, ki ga razširimo z razvojnim orodjem Android (ADT - Android Development Tools). Na voljo pa so tudi monoga druga razvojna okolja. Android aplikacije imajo končnico .apk, ko na Android napravi zaženemo datoteko s končnico .apk se aplikacija samodejno namesti

Verzija	Ime
1.0	/
1.5	Cupcake
1.6	Donut
2.0, 2.0.1, 2.1	Eclair
2.2 – 2.2.3	Froyo
2.3 – 2.3.7	Gingerbread
3.0 - 3.2	Honeycomb
4.0 – 4.0.4	Ice Cream Sandwich
4.1 - 4.3	Jelly Bean
4.4	KitKat

Tabela 5.1: Verzije mobilnega informacijskega sistema Android.

na napravo.

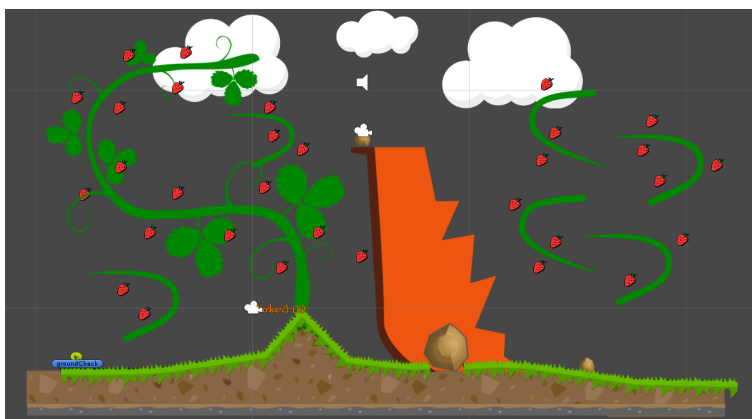
Poglavje 6

Izdelava igre

V tem poglavju je opisana izdelava igre s pomočjo igralnih pogonov Unity3D in Construct 2. Poglavje je razdeljeno na posamezne ključne probleme, ki jih je bilo tekom izdelave treba rešiti. Ob vsakem problemu je opisano, kako smo problem rešili v obeh igralnih pogonih.

6.1 Opis igre

Igra se dogaja v dvodimenzionalnem svetu. Igralec vodi glavni lik s pomočjo smernih tipk. Omogočeno mu je premikanje levo in desno in skakanje. Glavni lik posnema premikanje mravlje, lahko hodi tudi obrnjen z glavo navzdol. Igralec tekom igre pobira razno sadje tako imenovane priboljške. Cilj igre je, v čim krajšem času pobrati vse sadje. Igralcu se ob končanju posameznega nivoja dodeli določeno število zvezdic, med 1 in 3 glede na to, kako hitro je rešil določeni nivo. Ob zagonu igre se nam prikaže začetni meni, na katerem imamo možnost začetka igre s prvim nivojem, lahko pa sami izberemo nivo, ki ga želimo odigrati seveda, če je le ta odklenjen. Nivo se odklene, takrat ko rešimo nivo, ki je pred njim (torej, če želimo odkleniti drugi nivo moramo rešiti prvi nivo.). Ob končanju vsakega nivoja se nam pokaže dialog med nivoji, ki prikazuje v kolikšnem času smo opravili nivo, koliko zvezdic smo si prisvojili in koliko točk smo zbrali. Ta dialog vsebuje dva gumba in sicer



Slika 6.1: Scena prvega nivoja.

enega, ki zažene naslednji nivo in enega, s katerim lahko ponovno igramo nivo, ki smo ga nazadnje zaključili, če z rezultatom nismo zadovoljni.

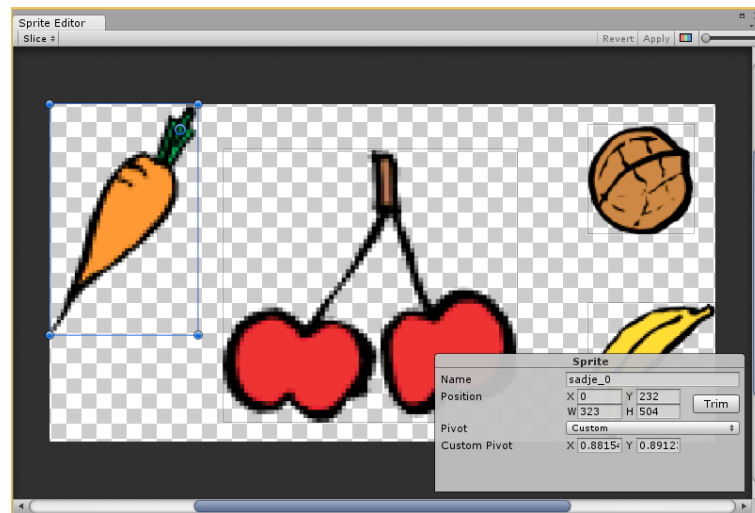
6.2 Izdelava osnovne scene

V tem delu se bomo posvetili izgradnji osnovne scene, načinu dodajanja elementov na sceno. Vse elemente smo narisali s pomočjo programa Inkscape¹. Osnovno sceno prvega nivoja prikazuje slika 6.1.

6.2.1 Unity

Unity omogoča enostavno sestavljanje scene po principu povleči in spusti. Vse objekte dodamo v mapo Asset znotraj projekta (lahko tudi podmapo) in jih nato znotraj programa Unity iz te mape enostavno povlečemo na sceno. Pri tem se kreira nov objekt tipa Sprite (grafični lik). Unity vsebuje orodje za urejanje grafičnih likov (Sprite editor). S pomočjo tega orodja lahko eno sliko razdelimo na več grafičnih likov. Omogoča dva načina delovanja in sicer

¹Inkscape je brezplačni in odprtokodni za izrisovanje vektorske grafike. Podpira informacijske sisteme Windows, Mac OS X in Linux. Kot osnovni grafični format uporablja SVG (Scalable Vector Graphics) [1].



Slika 6.2: Orodje za urejanje grafičnih likov.

avtomatski način, ki sam zazna, kje se nahaja grafični lik in tako razreže sliko. Pri tem lahko območje izrezovanja tudi ročno popravimo, če z avtomatskim izrezovanjem nismo bili zadovoljni, prestavimo lahko tudi center objekta. Drugi način delovanja je s pomočjo mreže. Pri tem načinu določimo velikost enega kvadrata mreže nato se čez sliko nariše mreža s kvadrati takšne velikosti in slika se razreže na grafične like, tako da vsak kvadrat mreže predstavlja svoj grafični lik. En primer uporabe prikazuje slika 6.2, kjer smo vse približske narisali na eno sliko in jih kasneje izrezali s pomočjo orodja za urejanje grafičnih likov. Vse slike, ki smo jih uporabili za izdelavo scene so formata PNG, saj omogoča transparentno ozadje.

Pri izgradnji scene je pomembno, kakšen je vrstni red izrisa objektov. Kateri grafični liki so v ospredju in kateri v ozadju, ter kako se med sabo prekrivajo. Za to, da določimo kakšen je vrstni red izrisa objektov jih razporedimo na sloje (layers). Potem določimo kakšen je vrstni red slojev, kateri so v ospredju in kateri v ozadju. Mi smo tako dali vse objekte, ki predstavljajo tla na svoj sloj, igralca na svojega in tako naprej. Ko imamo tako objekte razporejene po nivojih, pa jih lahko razvrstimo tudi znotraj samega nivoja. Vsak objekt ima številski atribut, ki določa razvrstitev znotraj sloja, večja

kot je ta številka bolj v ospredju je objekt znotraj sloja. Unity pa omogoča še en način za določanje vrstnega reda izrisa objektov. In sicer lahko znotraj urejevalnika scene na enostaven način preklopimo med dvodimenzionalnim in trodimenzionalnim pogledom, tako lahko vsak objekt prestavimo tudi po koordinati z . Objekti z nižjo koordinato z so bolj v ospredju kot objekti z višjo. Vendar je pri tem potrebno paziti, da nima objekt nižje koordinate z kot kamera, saj potem objekt ne bo več viden na sliki. Ker se v dvodimenzionalnem pogledu uporablja izometrična projekcija je objekt videti enako ne glede na to, kako spremenimo njegovo koordinato z , ta vplila samo na vrstni red izrisa. Prav tako objekti z različno koordinato z trkajo med sabo, kot če bi bili v enaki ravnini.

6.2.2 Construct

V programu Construct lahko elemente na sceno dodajamo na dva načina. Nove elemente lahko povlečemo na postavitev iz mape, kjer se nahajajo na disku ali pa jih vstavimo tako, da desno kliknemo na postavitev in izberemo, kateri objekt želimo vstaviti. Construct vsebuje tudi orodje za urejanje slik. To orodje vsebuje nekaj osnovnih pripomočkov za urejanje slik, kot so čopič, radirka, pisalo, orodje za izris črt in kvadratov. S pomočjo tega orodja lahko narišemo preprosti grafični lik (Sprite), ali pa spremenimo že obstoječega. Znotraj tega ogrodja je mogoče tudi urejati poligone, ki skrbijo za detekcijo trkov.

Prav tako kot smo že opisali v poglavju 6.2.1 je tudi tukaj pomembno, da se liki izrisujejo v pravilnem vrstnem redu. Tudi v programu Construct poznamo sloje in lahko vrstni red izrisa objektov določimo glede na to, na katerem sloju se nahajajo. Prav tako lahko določimo, kateri objekti so v ospredju sloja in kateri v ozadju. To storimo z desnim klikom na objekt in nato izberemo pošlji v ozadje sloja ali pa pošlji v ospredje. Za bolj natančno razvrščanje po osi Z pa potrebujemo plačljivo verzijo programa.

6.3 Premikanje in fizika

V tem delu bomo opisali najpomembnejši korak pri izdelavi igre in sicer se bomo posvetili kontroli samega lika, premikanju levo in desno in skakanju. Dodajanju fizikalnih lastnosti na sceno ter obračanje lika glede na podlago. Pri tem ji bil glavni problem, ki ga je bilo treba rešiti ta, da se lik drži podlage ne glede na to, kako je obrnjen.

6.3.1 Unity

Preden smo se posvetili premikanju glavnega lika, je bilo potrebno objektom določiti fizikalne lastnosti. Glavnemu igralnemu liku smo dodali komponento tega telo (rigid body 2D), tako zdaj nanj vpliva gravitacija. Za detekcijo trkov smo na igralca namestili dva okrogla detektorja trkov, enega za noge in enega za telo. Na sliki 6.3 je prikazan glavni igralec s pripetimi detektorji trkov, ki so obarvani svetlo zeleno. Za detekcijo trkov na ostalih elementih kot so tla, stebila rastlin, torej vseh po katerih lahko igralec hodi, smo uporabili poligonski detektor trkov (Polygon collider). Pri tem Unity sam obriše objekt s poligonom, ki služi za detekcijo trkov. Ta poligon je mogoče na enostaven način spreminjati. S klikom na levi miškin gumb in istočasnem držanju tipke *shift*, lahko premikamo obstoječe točke na poligonu in dodajamo nove. S klikom levega miškinega gumba in istočasnem držanju tipke *control*, lahko odstranjujemo točke. V našem primeru je najpogosteje Unity zgeneriral preveč točk in jih je bilo potrebno nekaj odstraniti, saj niso bile potrebne in bolj, kot zmanjšamo število točk na poligonu, manj obremenjujemo procesor z računanjem trkov.

Igralec se lahko premika levo, desno in skače. Za premikanje igralca skrbi skripta. Igralca je mogoče premikati s pomočjo smernih tipk na tipkovnici, na napravah za dotik pa se premikanje upravlja z dotiki zaslona. Na zaslonih za dotik je za skok potrebno pritisniti kjer koli na desni polovici zaslona. Za premikanje levo in desno pa sta na zaslon dodani dve puščici levo in desno, če pritisnemo na eno od puščic, se igralec premakne v ustrezno smer. Da smo



Slika 6.3: Glavni lik s detektorjem trkov.

zagotovili, da lahko igralca upravljamo tako z dotiki, kot tudi s tipkovnico, smo znotraj metode `FixedUpdate`² dodali navodila prevajalniku, ki glede na to, na kateri napravi se nahajamo, pokliče ustrezno metodo za naprave za dotik ali namizne naprave, uporabili smo naslednjo kodo:

```
#if UNITY_EDITOR || UNITY_STANDALONE || UNITY_WEBPLAYER
    DesktopDeviceUpdate();
#elif UNITY_ANDROID || UNITY_IPHONE || UNITY_WP8
    TouchDevicesUpdate();
#endif
```

Pri napravah s tipkovnico smo preverili, ali je pritisnjena, katera od horizontalnih smernih tipk in smo glede na to premaknili igralca. Pri napravah na dotik pa smo najprej preverili, ali smo se dotaknili katerega od gumbov, s pomočjo metode `HitTest()`, ki preveri ali se koordinate pritiska nahajajo znotraj GUI texture, primerjava se dogaja v koordinatah zaslona. Koda za zaznavanje pritiska izgleda tako:

```
if (Input.touchCount > 0)
{
    foreach(Touch t in Input.touches)
    {
        if(buttonLeft.HitTest(t.position))
            moveX = -maxSpeed;
        else if(buttonRight.HitTest(t.position))
            moveX = maxSpeed;
    }
}
```

Igralca nato prestavimo tako, da mu dodamo hitrost v ustrezni smeri glede na vektor, ki kaže desno. Metoda, ki skrbi za premikanje igralca izgleda tako:

```
private void PlayerMove(float moveX, float moveY)
{
```

²Metoda `FixedUpdate` se kliče enkrat na vsak okvir. Uporablja se namesto funkcije `Update`, kadar imamo opravka z manipulacijo togih teles. [7]

```

Vector2 newVelocity;
//preverimo ali se igralec dotika tal
if (grounded)
    newVelocity = new Vector2 (transform.right.x * moveX
        * maxSpeed, transform.right.y * moveX * maxSpeed
    );
else
    newVelocity = new Vector2 (moveX, moveY);
rigidbody2D.velocity = newVelocity;
//preverimo ali je igralec obrnjen v smeri premikanja
if (moveX > 0 && !facingRight)
    Flip ();
else if (moveX < 0 && facingRight)
    Flip ();
}

```

Premikanje igralca je različno glede na to, ali se dotika tal, ali je v zraku, saj je na tleh premika glede na vektor, ki kaže desno, v zraku pa ne glede na rotacijo samo levo in desno. Na koncu metode preverimo, ali je igralec pravilno obrnjen glede na smer premikanja, če njegovo komponento skaliranja po x pomnožimo z -1 , kar je ekvivalentno kot, če bi ga prezrcalili čez y os. Za preverjanje ali se igralec dotika tal uporabljamo naslednjo kodo:

```

grounded = Physics2D.OverlapCircle (groundCheck.position
    , groundRadius , whatIsGround);

```

groundCheck je prazen objekt, ki smo ga pripeli na noge igralca in služi temu, da s pomočjo njega določimo položaj nog, *groundRadius* je polmer, v katerem odkrivamo, ali se igralec dotika tal, *whatIsGround* je maska sloja (LayerMask), to so vsi objekti, ki smo jim določili sloj tla. Za skakanje igralcu dodamo silo v vektorja, ki kaže navzgor (*transform.up*).

Ko smo rešili problem s premikanjem, je bilo potrebno rešiti problem, da se bo igralec premikal glede na to, kako so obrnjena tla pod njim. Spodaj je prikazana koda, ki skrbi, da je igralec pravilno obrnjen glede na teren.

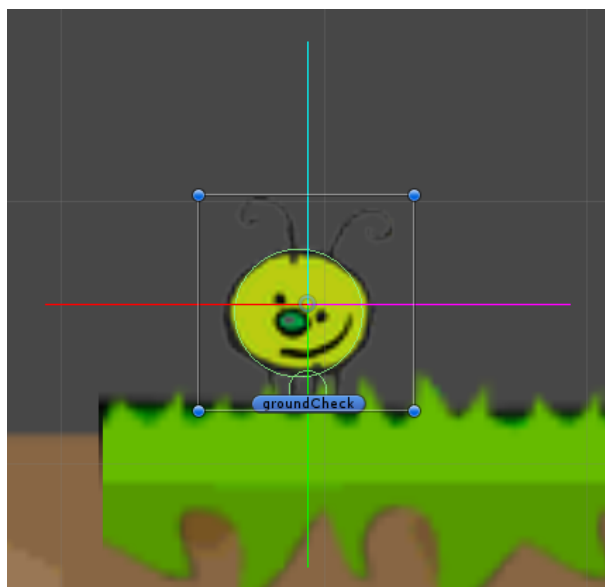
```
void PlayerRotate()
{
    if (grounded)
    {
        RaycastHit2D hit = Physics2D.Raycast (transform.
            position, -transform.up, 5.0f, whatIsGround);
        transform.rotation = Quaternion.FromToRotation (
            Vector2.up, hit.normal);
    } else {
        transform.rotation = Quaternion.Euler (transform.up)
            ;
    }
}
```

S pomočjo metode *Raycast* pošljemo žarek iz položaja igralca v smer navzdol, glede na rotacij igralca in preverimo pod kakšnim kotom žarek seče normale tal, glede na to ustrezno obrnemo igralca. Če je, igralec v zraku se obrne tako, da pade naravnost navzdol. Ko igralec skoči in če se pri tem z glavo dotakne objekta, po katerem lahko hodi se mora igralec prijeti tega objekta, se pravilno obrniti in nadaljevati hojo po njem. Za zagotovitev tega se pošlje žarek tudi navzgor, ko je igralec v zraku in preveri, ali se dotika tal, prav tako pošljemo žarek levo in desno, da ugotovimo, ali je ob strani zadel tla. V načinu za razhroščevanje lahko izrišemo vse poslane žarke, prikazuje jih slika 6.4.

Ko je bilo poskrbljeno za pravilno obračanje igralca je bilo potrebno poskrbeti za pravilno delovanje gravitacije. Torej, da ko je igralec, obrnjen na glavo, ne pade navzdol, ampak lahko nadaljuje hojo po objektu. To smo zagotovili tako, da smo popravili smer delovanja gravitacije glede na to, kako je igralec obrnjen, in sicer z naslednjo kodo.

```
Physics2D.gravity = -transform.up * gravitySpeed;
```

Seveda se gravitacija takoj ko igralec skoči v zrak, nastavila na privzeto

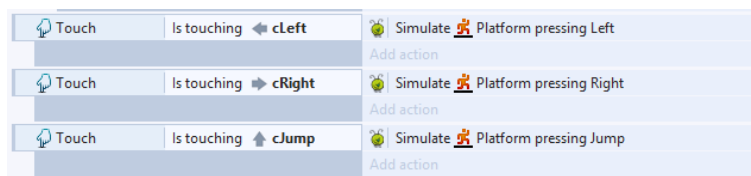


Slika 6.4: Igralec z izrisanimi žarki.

vrednost, torej ravno navzdol.

6.3.2 Construct

V igralnem pogonu Construct igralnim likom dodelimo določene lastnosti (na primer fizikalne) tako, da jim v nastavitvah objekta dodelimo obnašanje, enemu objektu lahko dodelimo tudi več obnašanj. Glavnemu igralnemu liku smo dodelili obnašanje, ki se imenuje Platform. Ta že vsebuje premike igralca levo, desno in skakanje, ki jih lahko nadzorujemo s smernimi tipkami na tipkovnici. Prav tako je poskrbljeno, da na igralca deluje gravitacija. Torej lahko za osnovne gibe igralca poskrbimo na zelo enostaven način, samo s tem, da mu dodamo ustrezno obnašanje. Ker pa je ta igra namenjena tudi napravam z zaslonom na dotik, pa je bilo potrebno zagotoviti tudi upravljanje s pomočjo dotikov. Za zagotovitev tega je bilo potrebno na sceno dodati objekt Touch, ki skrbi za detekcijo dotikov. Opis dogodkov, ki skrbijo za premikanje s pomočjo dotikov prikazuje slika 6.5. Ta ob pritisku na katero od puščic, ki smo jih dodali na sceno, kot Sprite, ustrezno simulira premikanje

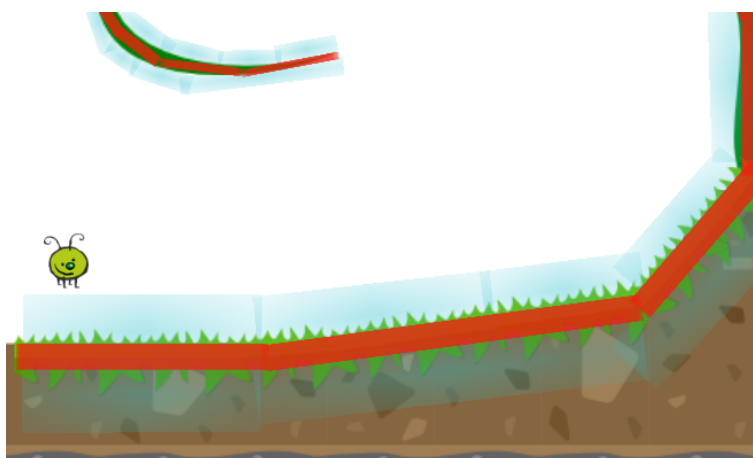


Slika 6.5: Dogodki, ki skrbijo premikanje igralca s pomočjo dotikov.

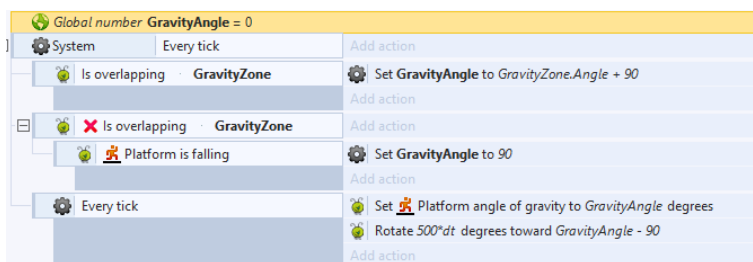
igralca v izbrano smer.

Ko smo uredili gibanje glavnega lika smo se posvetili detekciji trkov na tleh, steblih rastlin in tako naprej. Tudi orodje Construct omogoča uporabo poligonskega detektorja trkov, vendar je ta v primerjavi z Unity dosti manj zmogljivo. Omogoča samo osem točkovni poligon in zato ni primeren za uporabo na bolj zapletenih objektih, prav tako je avtomatsko obrisovanje objekta zaradi malega števila točk zelo nenatančno. Poligonu sicer lahko dodamo več kot osem točk, vendar nas začne opozarjati, da ima poligon preveč točk in bo zaradi tega igra delovala počasneje. To težavo smo rešili tako, da smo namesto poligonskega detektorja trkov uporabili več kvadratov, ki smo jih razporedili po objektih. Ti kvadrati so tipa Sprite in imajo dodano obnašanje Solid. To obnašanje predstavlja trden objekt, s katerim je mogoče trkati. Tem kvadratom smo določili, da so nevidni. Kvadrato smo dodali tako, da smo najprej narisali en kvadrat in ga postavili na sceno. Na to smo ob držanju tipke *CTRL* pritisnili na grafični lik in ga povlekli. To v programu Construct, naredi kopijo grafičnega lika z enakimi lastnostmi, vendar pa ne kreira novega objekta, ampak oba grafična lika spadata pod isti objekt. Nato lahko vsakemu grafičnemu liku posebej spreminjamo položaj, velikost in rotacijo, če pa spremenimo katero od drugi lastnosti se bo ta spremenila na vseh grafičnih likih znotraj tega objekta. Na sliki 6.6 so ti kvadrati prikazani z rdečo barvo.

Za pravilo obračanje igralca glede na teren in pravilo delovanje gravitacije. Smo na teren na enak način dodali kvadrato, ki se nahajajo tik nad tlemi in imajo enako rotacijo kot tla pod njimi. Te kvadrato smo poimeno-



Slika 6.6: Prikaz kvadratov za detekcijo trkov, ter pravilo obračanje igralca.



Slika 6.7: Dogodki, ki skrbijo za pravilo obračanje igralca in pravilo delovanje gravitacije.

vali GravityZone in so na sliki 6.6 prikazani z modro barvo. Nato nastavimo rotacijo igralca na enako, kot jo ima kvadrat s katerim se prekriva in ustrezno glede nato popravimo kot gravitacije. Dogodki, ki skrbijo za to so prikazani na sliki 6.7. Vedno preverimo, ali se igralec prekriva, s katerim od kvadratom (ker so vsi kvadrati isti objekt, Construct sam zazna, s katerim se prekriva), nato shranimo kot rotacije tega kvadrata v spremenljivko. Če se igralec ne prekriva z nobenim kvadratom in hkrati pada, pa spremenljivko nastavimo tako, da bo gravitacija delovala navzdol. Spremenimo kot delovanja gravitacije na igralca in rotacijo igralca glede na to spremenljivko.

6.4 Pobiranje priboljškov in točkovanje

V tem delu bomo opisali, kako je realizirano pobiranje priboljškov, merjenje časa in podeljevanje točk glede na to, v kakšnem času smo opravili nivo. Priboljški so jagode, češnje in ostalo sadje, ki ga lahko igralec poje. Nivo je zaključen ko igralec pobere ves pribolške. Uspešnost igralca se prikaže tudi s številom zvezdic. Igralec lahko prejme tri, dve ali eno zvezdico, pri čemer več zvezdic pomeni boljši čas.

6.4.1 Unity

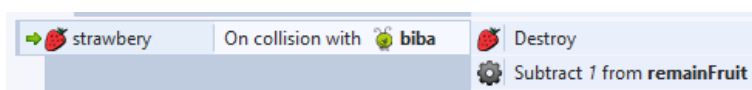
V Unity smo najprej vsem priboljškom dodali krožni detektor trkov in določili, da so sprožilci (trigger). In sicer zato, da igralec ne trka z njimi, ampak se samo sproži funkcija *OnTriggerEnter*. Vsakemu priboljšku smo dodali skripto, ki je ob sprožitvi uničila objekt. Metoda izgleda tako:

```
void OnTriggerEnter2D(Collider2D col)
{
    //preverimo ali trka z igralcem
    if (col.collider2D.gameObject.name == "Player")
    {
        Destroy(gameObject);
    }
}
```

Nato smo vse priboljške v hierarhiji pripeli kot otroke objekta priboljški. Na starševski objekt smo pripeli skripto, ki vsak okvir preveri kakšno je število otrok tega objekta, če je število otrok enako nič se ta nivo zaključi. Da ugotovimo kakšno je število otrok, uporabimo tole kodo.

```
n = transform.childCount;
```

Za merjenje časa smo uporabili metodo *Time.time*, ki vrne čas v sekundah od začetka igre. Na začetku igre znotraj metode *Start()* pokličemo metodo *Time.time* in si shranimo začetni čas. Nato vsak okvir ponovno pokličemo metodo *Time.time*, nato od trenutnega časa odštejemo začetni čas



Slika 6.8: Dogodek, ki skrbi za manipulacijo s priboljški.

in tako dobimo dejanski čas igre. Čas je nato potrebno pretvoriti v ustrezen format, saj metoda *Time.time* vrne vrednost tipa *float*. Iz te vrednosti dobimo minute tako, da čas celo številsko delimo s 60, za sekunde pa izračunamo ostanek pri deljenju s 60.

Točke in zvezdice se podeljujejo ob končanju posameznega nivoja. Točke določimo, tako da čas v sekundah, ki smo ga porabili za posamezne nivo odštejemo od 300. Točke se nato seštejejo za vse nivoje, ki smo jih že končali skupaj. Za podeljevanje zvezdic pa smo realizirali razred, ki hrani tabelo, v kateri vrstice predstavljajo posamezen nivo, prvi stolpec prikazuje naj več, koliko moramo imeti čas, da osvojimo 3 zvezdice v drugem stolpcu pa čas, ki ga je potrebno doseči za 2 zvezdici. Če je naš čas večji od obeh časov v vrstici za ta nivo, dobimo eno zvezdico.

6.4.2 Construct

V programu Construct smo priboljške dodali tako, da smo vstavili nov grafični lik in jih nato po že opisanem postopku ob držanju tipke *CTRL* kopirali po igralni površini. Dodali smo še objekt tipa *Array* v katerem smo za vsak nivo določili, koliko je začetno število priboljškov za ta nivo. Ob začetku nivoja smo začetno število priboljškov za trenutni nivo iz tabele shranili v novo spremenljivko, ki hrani, koliko priboljškov nam je še ostalo. Potem smo dodali dogodek, ki ob trku igralca s priboljškom le tega uniči in zmanjša število ostalih priboljškov v tem nivoju za ena. Dogodek, ki skrbi za pravilo manipulacijo s priboljški je prikazan na sliki 6.8. Ko je število ostalih priboljškov za ta nivo enako nič, se trenutni nivo zaključí.

Za določanje časa uporabimo funkcijo *time*, ki vrne čas od začetka igre. Ob začetku nivoja (On start of layout), nastavimo spremenljivko, ki hrani

vrednost začetni čas, na vrednost, ki jo vrne funkcija *time*. Da pretvorimo čas v ustrezn format za prikaz, uporabimo tole kodo: `floor((time - StartTime)/60)&" : "&floor((time - StartTime)%60)`.

Izračun točk poteka enako, kot smo opisali za program Unity. Za izračun zvezdic pa smo dodali tabelo, ki ima enako sestavo kot tabela v Unity. Torej je ta rešitev zelo podobna rešitvi v programu Unity, le da tukaj za to ni treba realizirati novega razreda, ampak je tabela, ki smo jo dodali na eni sceni, nato vidna na vseh scenah.

6.5 Grafični uporabniški vmesnik

Na tem mestu se bomo posvetili načinu izdelave grafičnih uporabniških vmesnikov (GUI) v obeh igralnih pogonih. Glavna naloga je bila realizacija grafičnega vmesnika za glavni meni, izbiranje že osvojenih levelov in dialog med nivoji. Izgled glavnega menija prikazuje slika 6.9. Opisali bomo tudi, kako smo realizirali prikaz štoparice, ki nam prikazuje, koliko časa že igramo trenutni nivo. Za prikaz časa smo s programom Inkscape izdelali lastno pisavo in jo shranili kot sliko PNG.

6.5.1 Unity

V programu Unity lahko grafični uporabniški vmesnik realiziramo na dva načina. Prvi način je, da videz grafičnega uporabniškega vmesnika opišemo znotraj funkcije *OnGUI()*. Ta način je tudi najpogosteje uporabljen in smo ga uporabili tudi mi. Vendar pa omenimo še drugi način. Pri drugem načinu dodamo gumbe na sceno kot grafične elemente, jim dodamo detektor trkov, ter nato s pošiljanjem žarkov ugotovimo, kateri gumb je pritisnjen. Prednos drugega načina je, da vidimo sam videz menija že med sestavljanjem scene, saj pri prvem načinu celoten videz menija opišemo v kodi in vidimo, kako izgleda šele, ko zaženemo igro. Prednost drugega načina je tudi, da se gumbi nahajajo v koordinatah sveta ter se, zato ohrani razmerje velikost gumbov glede na celotno sliko, ko igro zaženemo na napravah, ki imajo



Slika 6.9: Glavni meni.

različno ločljivost zaslona. Pri prvem načinu pa so elementi v koordinatah zaslona in je potrebno za ohranitev razmerja vse velikosti podajati relativno glede na višino in širino zaslona in imamo zaradi tega nekoliko več dela. Vendar pa ima drugi način tudi kar nekaj slabosti, saj moramo v kodi posebej ločiti ali gre za pritisk miške ali dotik zaslona, prvi način ima to že realizirano in nam na to ni potrebno paziti. Prav tako je težava, če želimo, da se videz gumba ob pri tisku nanj ali ob prehodu miške spremeni, saj moramo vse spremembe sprogramirati sami. Pri prvem načinu je to dosti enostavneje, saj samo dodamo gumbu stil in znotraj njega določimo različne izgleda, gleda na različne dogodke kot so pritisk gumba, prehod z miško in tako dalje.

Primer izgradnje grafičnega uporabniškega vmesnika si bomo pogledali na primeru glavnega menija. Kot smo že omenili, smo za to uporabili funkcijo *OnGUI()*. Funkcija, ki skrbi za izgled glavnega menija je prikazana spodaj.

```
void OnGUI()
{
    if (GUI.Button (new Rect (width/4, height/8, width/2,
        height/4), "Play", startStyle))
```

```
{
    Application.LoadLevel (3);
}
if (GUI.Button (new Rect (width/4, 5 * height/8, width
    /2, height/4), "Select level", selectLevelStyle))
{
    Application.LoadLevel (2);
}
}
```

Gumb realiziramo z ukazom *GUI.Button*, kot argument pa podamo pravokotnik, ki predstavlja položaj in velikost gumba, drugi argument je napis, ki ga bo imel gumb in zadnji stil gumba. Spremenljivki *width* in *height* predstavljata širino in višino zaslona. Njuni vrednosti smo določilo z ukazom *Screen.width* oziroma *Screen.height* znotraj funkcije *Start()*, ki se kliče samo enkrat ob zagonu scene. Vse vrednosti so podane relativno glede na širino in višino zaslona, da bodo gumbi prikazani v enakih razmerjih glede na celotno sliko na napravah z različnimi ločljivostmi zaslona. Stilu smo dodane različne teksture glede na to ali je gumb pritisnjen ali ne.

Brezplačna različica Unity ne vsebuje orodij, ki bi avtomatsko iz slike z znaki naredila pisavo. Zato smo k problemu pristopili nekoliko drugače. Iz slike s številkami od 0 do 9, smo s pomočjo orodja za urejanje grafičnih likov izrezali posamezne številke. Nato smo jih razporedili v tabelo tako, da je imela številka isti indeks v tabeli, kot je njena vrednost. Na sceno smo dodali nove grafične objekte. Enega za prikaz minut, enega za prikaza desetice pretečenih sekund in enega za prikaz enic, vmes pa smo dodali še dvopičje. Torej je izpis časa imel obliko M:SS. Nato smo z naslednjo kodo poskrbeli za ustrezen izpis.

```
public Sprite[] timeFont;
void showTime(int time)
{
    int second = time % 60;
    int min = time / 60;
```

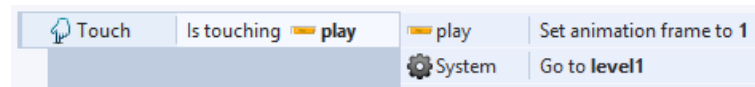
```
sec0.GetComponent<SpriteRenderer> ().sprite = timeFont
    [second % 10];
sec1.GetComponent<SpriteRenderer> ().sprite = timeFont
    [second / 10];
minute.GetComponent<SpriteRenderer> ().sprite =
    timeFont [min];
}
```

Najprej iz časa izločimo sekunde in minute. Nato vsakemu objektu zamenjamo teksturo, ki se nahaja na ustreznem mestu v tabeli s teksturami števil.

6.5.2 Construct

V programu Construct lahko gumb dodamo tako, da vstavimo nov element tipa gumb (button). Vendar mu težje spremenimo njegov izgled. To nam sicer omogočajo stili CSS, vendar je, zato potrebno kar nekaj dodatnega dela, saj je potrebno opisati več dogodkov. Za to smo se mi odločili za drugačno rešitev. Vnaprej smo pripravili texture gumbov ko so le ti pritisnjeni in ko niso, tako smo imeli tudi več svobode pri oblikovanju gumbov. Nato smo glavni meni realizirali tako, da smo na sceno dodali nov grafični lik, s teksturo našega gumba. Da smo zagotovili spremembo gumba med pritiskom nanj, smo nastavili animacijo grafičnega lika. In sicer na prvo sliko animacije smo dodali teksturo gumba, ko le ta ni pri pritisnjen, na drugo pa, ko je pritisnjen. Potem smo nastavili hitrost animacije na 0, da se animacija ne začne avtomatsko izvajati, ko zaženemo igro. Delovanje gumba za začetek igre upravlja dogodek, ki je prikazan na sliki 6.10. Ob pritisku na gumb najprej nastavimo animacijo gumba na sliko z indeksom ena, nato pa zaženemo sceno s prvim nivojem igre.

Prikaz časa v programu Construct poteka nekoliko enostavneje kot v Unity. Construct vsebuje orodje, ki omogoča izdelavo pisave iz slike. Na sceno dodamo objekt tipa *Sprite font* in izberemo zeleno sliko z našo lastno pisavo. Nato določimo višino in širino posameznega znaka na sliki, torej mo-



Slika 6.10: Dogodek, ki upravlja gumb za začetek igre.

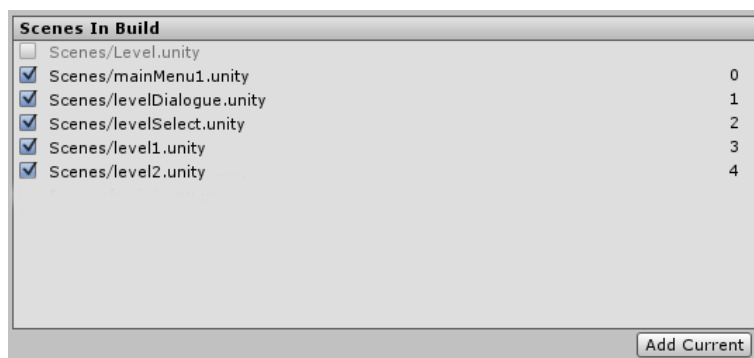
ramo med izdelavo pisave paziti, da smo vsi znaki enakih dimenzij. Potem pa vnesemo zaporedje znakov tako kot si sledijo po vrsti na sliki. Ko dokončamo vse te korake program Construct avtomatski izreže znake iz slike in ko objektu določimo željen tekst, se bo ta prikazal v naši pisavi.

6.6 Prehajanje med nivoji in shranjevanje podatkov o igri.

V tem delu se bomo posvetili, kako prehajamo iz enega v drug nivo, kako pošljemo podatke med nivoji in kako shranimo podatke o igri, da se ohranijo tudi, ko se igra zapre. Po koncu vsakega nivoja je potrebno dialogu, ki skrbi za prehajanje med nivoji in podatke o tem, v kakšnem času smo končali nivo. Ob končanem nivoju je potrebno tudi preveriti ali smo nivo končali v najboljšem časi in če smo ta čas tudi shraniti. Preveri se tudi, ali je to najvišji do zdaj rešeni nivo, da na ta način poskrbimo za odklepanje naslednjega nivoja.

6.6.1 Unity

V Unity so nivoji in meniji vsak na svoji sceni. Vsako sceno je treba dodati v nastavitvah. Nato jih lahko znotraj nastavitv razvrstimo v poljubnem vrstnem redu. Za naš primer smo scene razvrstili tako, da so vsi meniji na začetku nato sledijo nivoji razvrščeni po vrsti, razvrstitev scen v našem primeru prikazuje slika 6.11. Vsaka scena dobi svojo številko od 0 naprej, te številke so na sliki 6.11 prikazane desno. Če ni določeno drugače se ob zagonu igre zažene scena s številko 0. Za odprtje scene se uporablja ukaz



Slika 6.11: Razvrstitev scen.

Application.LoadLevel(), ki kot argument prejme številko scene, ki jo želimo zagnati.

Pošiljanje podatkov v Unity lahko realiziramo na dva načina in sicer s pomočjo statičnega razreda ali s `PlayerPrefs`³. Za pošiljanje podatkov med končanim nivojem in dialogom, ki se prikaže med nivoji, uporabljamo statični razred. Ta razred hrani spremenljivke, končani novo, čas končanega nivoja in število prejetih zvezdic. Dialog med nivoji prikaže dobljene podatke torej čas in število zvezdi. Če na dialogu med nivoji pritisnemo gumb za igranje naslednjega novo, bo le ta po zgoraj opisanem postopku odpru nivo, ki ima za ena večjo številko scene od končanega nivoja.

`PlayerPrefs` uporabljamo za shranjevanje podatkov o tem, kakšen je najboljši čas posameznega nivoja, največ, koliko zvezdic smo prejeli za posamezni nivo in kateri nivo smo že odklenili. Primer uporabe `PlayerPrefs` prikazuje spodnja koda, ki skrbi za najboljšega časa za določen nivo.

```
void saveBestTime(int time, int endLevel)
{
```

³`PlayerPrefs` hrani igralčeve podatke med posameznimi sejami igre. Podatki se hranijo na pomnilnik naprave, lokacija hranjenih podatkov pa je odvisna, na kateri napravi zagajamo igro. Ti podatki se ohranijo tudi, ko igro zapremo. Za shranjevanje celo številske vrednosti uporabimo metodo `PlayerPrefs.SetInt`, ki kot argument prejme ključ in vrednost. Podobno z `GetInt` dobimo vrednost, ki je pripisana določenemu ključu. Z metodo `HasKey` pa preverimo ali ključ obstaja [15].

```
if(!PlayerPrefs.HasKey("bestTime"+endLevel))
    PlayerPrefs.SetInt("bestTime"+endLevel, time);
else
    if(PlayerPrefs.GetInt("bestTime"+endLevel) > time)
        PlayerPrefs.SetInt("bestTime"+endLevel, time);
}
```

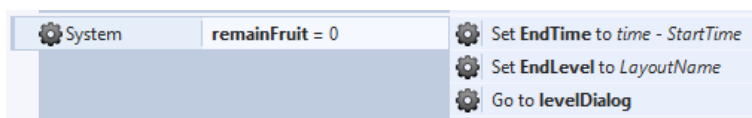
V tem primeru najprej preverimo ali za ta nivo še ne obstaja ključ z najboljšim časom tega nivoja in če ne potem shranimo ta čas. Drugače pa preberemo najboljši čas in ga primerjamo s časom, ki smo ga podali metodi, če je ta čas manjši od časa, ki se nahaja v PlayerPrefs potem zapišemo novi čas.

6.6.2 Construct

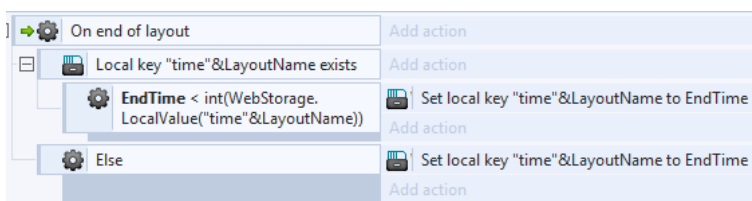
Tudi pri programu Construct se vsak nivo nahaja na svoji sceni. Scen ni potrebno posebej razporejati, tako kot je to potrebno pri Unity. V nastavitvah projekta pa je potrebno nastaviti, katera scena naj se zažene najprej. Nove sceno odpremo z ukazom *Go to layout* in preko spustnega menija izberemo, katero sceno želimo odpreti. Lahko pa izberemo opcijo *Go to layout (by name)* in v polje vnesemo niz, ki predstavlja ime zelene scene. Za enostavnejše delo smo scene nivojem poimenovali s številkami (prvi nivo s številko 1 itd.).

Za pošiljanje podatkov med nivoji se uporabljajo globalne spremenljivke, ki so vidne v vseh listih dogodkov tako imenovanih Event sheet. Tako kot v Unity smo tudi tukaj realizirali dialog med nivoji, ki prikaže čas končanega in število zvezdic, ki smo jih prejeli. Tako ob končanju nivoja shranimo v globalne spremenljivke, čas končanega nivoja, in ime nivoja, ki smo ga končali. Dogodek ob koncu nivoja je prikazan na sliki 6.12. Ko v dialogi med nivoji pritisnemo gumb za naslednji nivo, ugotovimo kateri novo je potrebno zagnati tako, da iz globalne spremenljivke preberemo ime končanega nivoja, prištejemo ena in zaženemo nivo s tem imenom. Ravno s tem razlogom smo nivoje poimenovali s številkami, kot smo opisali nekoliko prej.

V programu Construct smo za shranjevanje podatkov, ki se ne smejo iz-



Slika 6.12: Dogodek ob koncu nivoja.



Slika 6.13: Shranjevanje najboljšega časa posameznega nivoja.

brisati, ko zapustimo igro, uporabili WebStorage. Tudi tukaj bomo primer uporabe prikazali na shranjevanju najboljšega časa posameznega nivoja. Primer prikazuje slika 6.13. Izvede se, ko se scena konča. Najprej preverimo, ali ključ že obstaja. Če ne obstaja, ga naredimo. V nasprotnem primeru pa preverimo, ali je dobljeni čas manjši od shranjenega najmanjšega časa, če je, shranimo novi čas.

6.7 Izvoz

V tem poglavju bomo opisali, kako v obeh programih izvozimo igro tako, da jo je mogoče zagnati na mobilni platformi.

6.7.1 Unity

Izvoz v programu Unity, poteka na enostaven način. Odpremo dialog *Build Settings*, izberemo platformo, za katero želimo izvoziti igro in pritisnemo tipko *Build*. Program zgenerira ustrezne datoteke, ki so potreben za zagon aplikacije na želeni platformi. Če izberemo platformo Android, na Unity zgenerira datoteko s končnico *.apk*. To datoteko nato prenesemo na napravo Android in ob

zagonu te datoteke se igra samodejno namesti na našo napravo.

6.7.2 Construct

Brezplačna verzija programa Construct omogoča izvoz samo za aplikacije HTML 5. Izvoz aplikacije v HTML 5 poteka enostavno. Kliknemo izvozi projekt, izberemo HTML 5 in izberemo lokacijo na disku, kamor želimo izvoženi projekt shraniti. Ker pa želimo aplikacijo, ki jo lahko zaženemo na mobilni platformi smo uporabili spletno orodje CocoonJS[5]. Za uporabo orodja se je treba najprej brezplačno registrirati. Nato ustvarimo nov projekt in naložimo .zip datoteko našega projekta. Ko uredimo vse potrebne nastavitve in kliknemo gumb prevedi, se naš projekt postavi v vrsto za prevajanje. Ko je projekt preveden prejmemo elektronsko pošto, s povezavo, iz katere lahko prenesemo datoteko .apk za zagon na android napravah. Orodje omogoča tudi izvoz za iOS naprave. Prevajanje je v našem primeru trajalo približno 2 minuti.

6.8 Testiranje

Na koncu smo igre, izdelane v obeh orodjih stestirali. V tem delu bomo opisali delovanje igre ko smo jo zagnali na mobilnem telefonu. Za testiranje smo uporabili mobilni telefon HTC Desire X, z dvojedrnim procesorjem frekvence 1 GHz in naloženim operacijskim sistemov Android različice 4.1.1. Za namene testiranja smo na zaslonu izpisali FPS⁴. Opisali bomo tudi, kako poteka sprotno testiranje v obeh orodjih. Torej testiranje med samo izdelavo igre.

Najprej smo na mobilnem telefonu zagnali igro, ki smo jo izdelali s pomočjo programa Unity. Ob zagonu igre se najprej pojavi pojavno okno z napisom Unity. Ker smo uporabili brezplačno verzijo programa Unity se tega okna, ne da odstraniti. Igra je delovala tekoče in brez težav. Zagon igre je trajal

⁴FPS - frames per second (sličice na sekundo), nam pove kolik sličic generira naprava v eni sekundi.

približno 3 sekunde (od pritiska gumba za zagon, do prikaza menija). Tudi za zagon prvega nivoja je naprava potrebovala 3 sekunde. Torej zagon same igre ne traja predolgo. FPS se je v poprečju gibal okoli 53 sličic na sekundo, nikoli pa ni padel pod 50. To je dovolj hitro, da igranje poteka nemoteno in omogoča dobro uporabniško izkušnjo.

Nato smo na isti napravi zagnali še igro, ki smo je naredili s programom Construct. Zagon igre je trajal okoli 15 sekund. Torej dosti dlje kot pri igri, ki je bila narejena s pomočjo programa Unity. Za zagon prvega nivoja po je trajal približno enako dolgo torej okoli 3 sekunde. Igra je delovala majn tekoče in se je včasih na kratek čas tudi popolnoma ustavila. FPS se je v povprečju gibal dokaj nizko, in sicer okoli 25, včasih tudi nižje. To je dokaj nizko in nam ne omogoča tekočega igranja. Med testiranjem igre se nam je tudi naredile, da je igra brez razloga zmrznila in se popolnoma ustavila.

Torej se je med testiranjem dosti bolje izkazala igra narejena z Unity. Delovala je bolj tekoče. Igranje je bilo nemoteno. Ni se nepričakovano ustavila. Ponujala je boljše uporabniško izkušnjo, kot igra narejena s programom Construct.

Za sprotno testiranje v programu Unity pritisnemo tipko predvajaj. V oknu *Game* začne teči igra. Med tem ko igra teče lahko tudi spreminjamo parametre vseh objektov na sceni in vse javne spremenljivke. Vse spremembe so takoj vidne. Vendar pa se, ko končamo testiranje spremembe, ki smo jih naredili med testiranjem ne shranijo, ampak se ponastavijo na vrednost pred testiranjem. Na ta način lahko testiramo vnose s tipkovnico in miško, ne moremo pa testirati vnosov z dotiki. V ta namem nam služi orodje Unity Remote. Orodje namestimo na mobilni telefon, zaženemo in preko USB kabla povežemo računalnik z mobilnim telefonom. Ko zaženemo testiranje se nam na zaslonu mobilnega telefona pojavi igra in jo lahko upravljamo z dotiki. Vse spremembe so prav tako vidne v programu Unity na računalniku.

V programu Construct izgleda sprotno testiranje tako, da se nam ob zagonu testiranja igra zažene v spletnem brskalniku kot, HTML 5 igra. Delovanje igre lahko nadziramo s tipkovnico in miško. Dotik lahko simuliramo

z miško, seveda pa na takšen način ne moramo simulirati dotika z več prsti na enkrat. Lahko pa izberemo tudi način za razhroščevanje, v tem primeru se nam poleg igre v brskalniku prikaže dodatna vrstica. Ta vrstica, prikazuje vse parametre objektov in njihove spremenljivke. Prikazuje pa tudi koliko sličic na sekundo se zgenerira, koliko procesorja zasede igre in podobno. Ni pa mogoče tekom testiranja spreminjati parametrov objektov ali pa vrednosti globalnih spremenljivk tako kot to lahko počnemo v programu Unity.

Poglavje 7

Sklepne ugotovitve

Cilj dela je bila primerjava orodij Unity in Construct za izdelavo iger za mobilne platforme. Na začetku smo predstavili nekaj teoretične podlage, ki je potreba za razvoj iger. Nato smo orodij testirali tako, da smo enako igro izdelali v obeh orodjih. Izdelavo igre smo razdelili na manjše podprobleme in jih podrobno opisali. Na koncu smo igri testirali na mobilni napravi in primerjali delovanje.

Gradnja scene v obeh orodjih poteka podobno. Oba omogočata hitro in enostavno izgradnjo scene, po principu povleci in spusti. Torej se glede sam gradnje scene programa ne razlikujeta dosti. Za gradnjo osnovne scene porabimo v obeh programih približno enako časa. Imamo pa v programu Construct več dela, če moramo poskrbeti za trkanje z bolj kompleksnim predmetom, zaradi majn zmogljivega poligonskega detektorja trkov.

Večjo razliko med orodji opazimo pri realizaciji nadzora nad delovanjem igre. Pri tem Unity uporablja programerski pristop. V orodju Construct je igra vodena preko dogodkov. Sestavljanje dogodkov poteka dosti hitreje kot programiranje v Unity. K temu pripomore tudi to, da je v orodju Construct za dosti stvari že vnaprej poskrbljeno. Tako na primer za premikanje karakterja, mu samo določimo obnašanje in že deluje. Medtem ko moramo vse v Unity vse sprogramirati sami. Vendar pa je slabost tega to, da smo omejeni samo na dogodke, ki so vgrajeni v samo orodje. V primerjavi s

tem na programiranje v orodju Unity ponuja več svobode in lahko naredimo naprednejše in bolj kompleksne igre. Vendar pa orodje Construct omogoča izdelovanje igre tudi tistim, ki ne poznajo nobenega programskega jezika.

Med testiranje pa se je izkazala glavna slabost programa Construct. Igra narejena s programom Construct je delovala dosti počasneje kot tista narejena v Unity. Za dobro uporabniško izkušnjo pa je pomembno ravno to, da igra deluje tekoče brez prekinjanja.

Torej orodje Construct bi izbral nekdo, ki ni toliko večč programiranje in želi na hiter in enostaven način narediti ne preveč zapleteno igro. Orodje Construct se zaradi hitre realizacije igre, lahko uporablja tudi za izdelavo prototipa igre. Tisti pa, ki je večč programiranje in želi, da njegova igra deluje tekoče in želi izdelati bolj kompleksno igro bo izbral orodje Unity.

Literatura

- [1] About Inkscape dostopna na:
<http://www.inkscape.org/en/about/>
- [2] Android dostopno na:
[http://en.wikipedia.org/wiki/Android_\(operating_system\)](http://en.wikipedia.org/wiki/Android_(operating_system))
- [3] Android version history dostopno na:
http://en.wikipedia.org/wiki/Android_version_history
- [4] Box2D dostopno na:
<http://box2d.org/manual.pdf>
- [5] CocoonJS dostopno na:
<https://www.ludei.com/>
- [6] D. H. Eberly, Game physic, 2010
- [7] FixedUpdate dostopno na:
<http://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html>
- [8] Game engine dostopno na:
http://en.wikipedia.org/wiki/Game_engine
- [9] N. Guid, Računalniška grafika, 2001
- [10] Learninh the interface dostopno na:
<http://docs.unity3d.com/Manual/LearningtheInterface.html>

- [11] Mobile operating system dostopno na:
http://en.wikipedia.org/wiki/Mobile_operating_system
- [12] Number of android apps dostopno na:
<http://www.appbrain.com/stats/number-of-android-apps>
- [13] PhysX dostopno na:
<http://en.wikipedia.org/wiki/PhysX>
- [14] PhysX dostopno na:
<http://www.geforce.com/hardware/technology/physx>
- [15] PlayerPrefs dostopno na:
<http://docs.unity3d.com/ScriptReference/PlayerPrefs.html>
- [16] Top Smartphone Platforms dostopno na:
https://www.comscore.com/Insights/Press_Releases/2014/4/comScore_Reports_February_2014.L
- [17] Unity3D dostopno na:
<http://unity3d.com/>
- [18] Using a license dostopno na:
<https://www.scirra.com/manual/14/using-a-license>