Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

FERI

Skripta

Bločni jeziki: programiranje z delčki

Tomaž Kosar, Matej Črepinšek

Maribor, 2021

Naslov: Bločni jeziki: programiranje z delčki

Avtorji: doc. dr. Tomaž Kosar
mailto://tomaz.kosar@um.si
izr. prof. dr. Matej Črepinšek
mailto://matej.crepinsek@um.si

Strokovna recenzija: red. prof. dr. Marjan Mernik
viš. pred. mag. Matija Lokar

Izdajateljica: Fakulteta za elektrotehniko, računalništvo in informatiko (UM FERI)
Koroška cesta 46, 2000, Maribor, Slovenija
https://feri.um.si
mailto://feri@um.si

Grafična priloga in oblikovanje: doc. dr. Tomaž Kosar
izr. prof. dr. Matej Črepinšek

Izdaja: Prva izdaja

Vrsta publikacije: skripta
e-publikacija

Različica: R1.0

Dostopna na: https://dk.um.si/IzpisGradiva.php?id=78774

Izid: Maribor, 2021

Predgovor

Spoznavati in spodbuditi nove načine ustvarjalnosti je eden izmed ključnih elementov dobre prakse vseživljenjskega učenja. Vsi uporabljamo mobilne naprave, vsi smo potrošniki različnih aplikacij od sporočil, elektronske pošte, družabnih omrežij do iger. Za napredno uporabo teh aplikacije je potrebno imeti vsaj osnovno razumevanje delovanja aplikacij. S pomočjo te knjige, ki je primerna za starejše od 12 let, boste stopili v svet izdelave mobilnih aplikacij. Za uspešno sledenje nalogam pa je potrebno še poznavanje osnov digitalnih kompetenc in osnovno znanje angleškega jezika. Predstavljeni način izdelave mobilne aplikacije sloni na uporabi pripravljenih delčkov oz. na programiranju na osnovi delčkov.

Programiranje na osnovi delčkov (angl. block-based coding), znano tudi kot bločno programiranje, je oblika programskega jezika, kjer so ukazi jezika predstavljeni kot delčki. Ker imajo delčki vnaprej zapisana pravila o povezovanju (kateri delčki sodijo skupaj), programiranje z delčki omejuje opis programov, tako da onemogoči povezovanje nezdružljivih delčkov ali pa zapis ukaza, ki ne obstaja. Tipičen primer takšnega jezika je programski jezik Scratch. Vendar Scratch še zdaleč ni edini primer bločnega jezika. V zadnjih letih opažamo porast programskih jezikov s takšno predstavitvijo. Primerni so še posebej za tiste, ki se želijo uriti v računalniškem mišljenju ali pa narediti prve korake v svet programiranja.

Poleg primarnega cilja, tj. razvijanja računalniškega mišljenja, ki ni starostno omejeno, pa zasleduje knjižica tudi dodaten cilj, ki je povezan z razvojem bločnih jezikov. Tako knjižica lahko oz. služi študentom kot primer dobre prakse na področju razvoja bločnih jezikov in njihovih orodij. Sistematičen opis in prikaz posameznih elementov bločnih programskih jezikov je nujen za vstop v svet razvoja bločnih jezikov. Za dosego ciljev je izbrano spletno okolje MIT App Inventor [1, 3], razvito na znani tehnični univerzi MIT (Massachusetts Institute of Technology). MIT App Inventor je popularen primer uporabe bločnega jezika [2], ki nudi vse osnovne koncepte, ki jih najpogosteje vključujemo v lastno razvite bločne jezike. Omogoča pa razvoj aplikacije za mobilne naprave, ki temeljijo na operacijskem sistemu Android.

Na eni strani je delo nastajalo kot podpora izvedbi poletnih šol in številnih gostovanj na osnovnih šolah. Po drugi strani pa se uporablja kot vzorčni primer pri predmetu Domensko specifičnih modelirnih jezikov, ki je del študija računalništva na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Na fakulteti se učimo, kako razviti takšna orodja, kot je MIT App Inventor.

Razvoj na področju računalništva in informatike poteka hitreje, kot si lahko človek sploh predstavlja. Skripta, kot je ta, je lahko posledično vedno zastarela, vendar temeljna znanja in pristopi ostajajo. Skripto sva izdala v elektronski obliki, ki omogoča njeno redno posodabljanje. Za vse dobronamerne popravke in predloge se priporočava.

Avtorja

KAZALO

Je potrebno?

KAZALO SLIK

1 MIT APP INVENTOR

S pomočjo okolja MIT App Inventor bomo stopili v svet bločnih programskih jezikov. Izbrano okolje omogoča mnogo več kot samo programiranje [3, 5]. Je lep primer za učenje osnovnih konceptov programskih jezikov, po drugi strani pa spoznamo tudi razvoj grafičnih uporabniških vmesnikov, razvoj mobilnih aplikacij, ki omogočajo uporabo številnih pripomočkov (npr. senzorjev, različnih povezav ...). Kljub temu, da sta namen in izgled klasičnih aplikacij in računalniških iger različna, so osnovni koncepti programiranja enaki. Okolje MIT App Inventor je še posebej zanimivo za učenje mladih, ker omogoča učenje programiranja hkrati z razvojem preprostih mobilnih računalniških iger, ki jih s ponosom kažejo bližnjim.

Sledi kratek opis orodja in osnovnih konceptov, ki so nujni, da lahko pristopimo k ustvarjanju svoje aplikacije.

1.1 Kje ga najdemo?

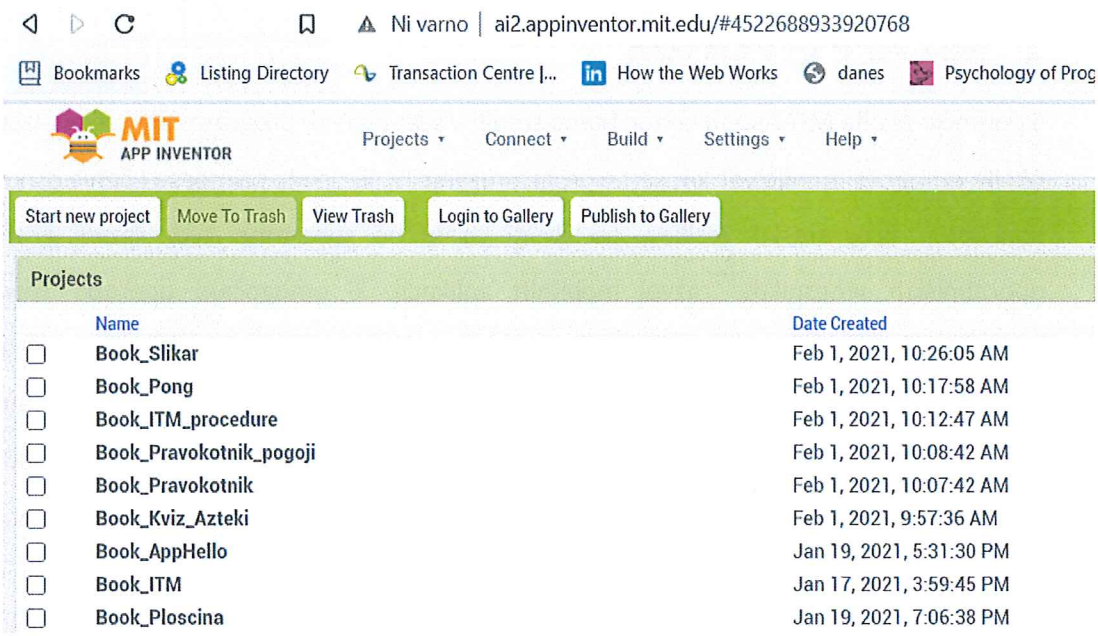
Razvojno okolje deluje po principu oblačne storitve, podobno kot deluje Gmail ali Facebook. Za uspešen začetek potrebujemo samo spletni brskalnik. Spletno aplikacijo najdemo na naslovu:

- http://ai2.appinventor.mit.edu/

Za prijavo je potrebno opraviti prijavo z računom Google, npr.:

- ime.priimek@gmail.com

Po prijavi se nam prikaže seznam naših projektov v okolju MIT App Inventor (na spodnji sliki vidimo, da v našem primeru že imamo devet projektov). Cilj vsakega projekta je mobilna aplikacija. Na začetku je ta seznam prazen. Nov projekt naredimo s pritiskom na gumb *Start new project*.



Slika 1.1: Seznam projektov v okolju MIT App Inventor

1.2 Razvojno okolje

V okolju MIT App Inventor je razvoj aplikacij razdeljen na dva dela. En del sestavlja grafična podoba programa (oblika), drugi del pa opis funkcionalnosti (akcij), ki se prožijo ob posameznem dogodku, npr. ob kliku na gumb.

V ta namen je uporabniški vmesnik razdeljen na dva pogleda (označeno na sliki):

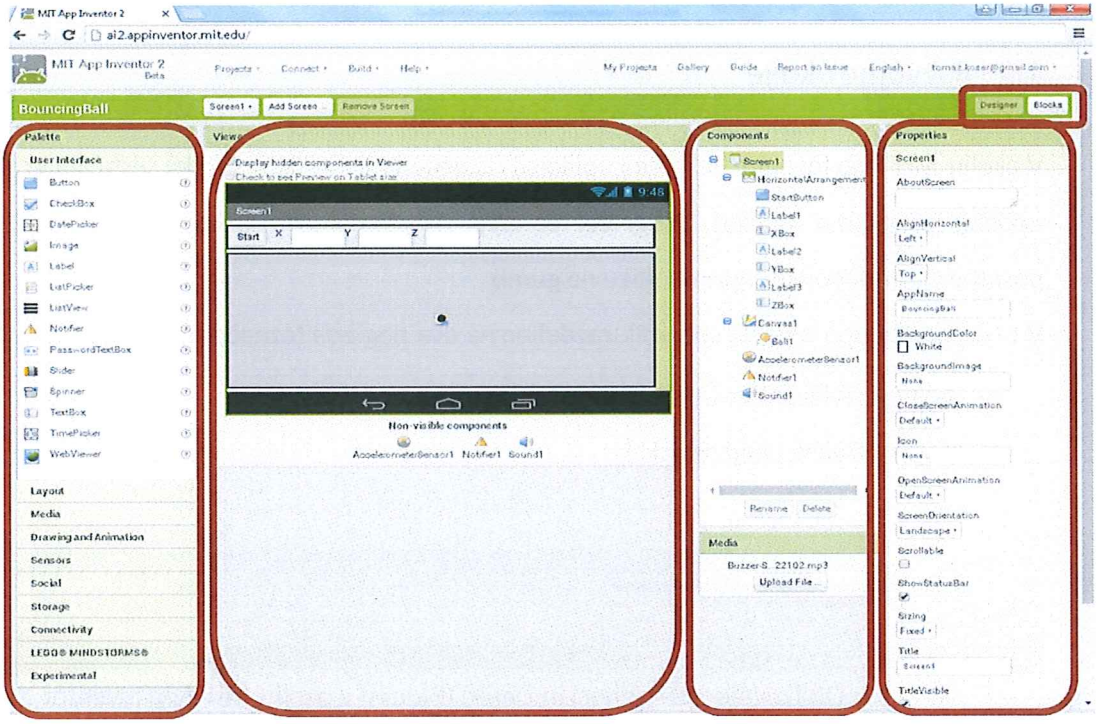
- pogled Oblikovalec (angl. Designer) in
- pogled Delčki (angl. Blocks).

Slika 1.2: Oblikovalec (Designer) in Delčki (Blocks) v okolju MIT App Inventor

Takšna delitev razvoja je zelo naravna in omogoča, da ločimo dve različni nalogi razvijalca računalniške aplikacije. Da bomo razumeli pomen vsake naloge, ju bomo spoznali s pomočjo opisa pogledov.

1.2.1 Pogled Oblikovalec

V načinu Oblikovalec (Designer) oz. v Oblikovalcu poskrbimo za videz mobilne aplikacije – kaj se nam prikaže na zaslonu, ko odpremo aplikacijo na mobilni napravi. Delo v Oblikovalcu je podobno urejanju dokumentov, predstavitev itd. Uporablja se kar nekaj znanih konceptov, kot so poravnava, vrsta pisav, tabela itd. Osnovne digitalne kompetence se tukaj razširijo in definirajo bolj podrobno.



Slika 1.3: Oblikovalèc (Designer) v okolju MIT App Inventor

Pogled

Kot je razvidno iz slike, je Oblikovalec sestavljen iz štirih delov:

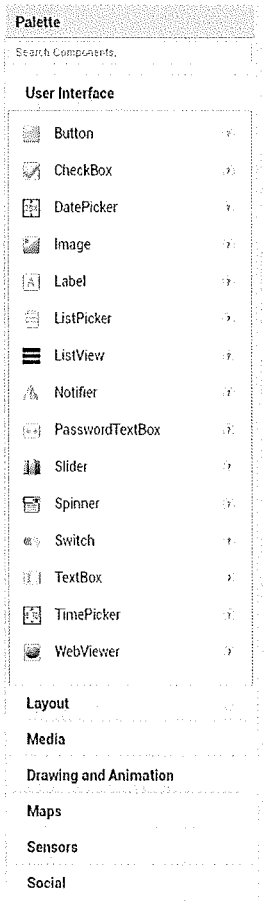
- **Paleta** (angl. Palette)
  Vsebuje seznam gradnikov, ki jih lahko uporabimo v aplikaciji.

- **Pogled aplikacije** (angl. Viewer)
  Prikaže, kako bo aplikacija videti, ko jo poženemo na telefonu ali tablici.

- **Grafični gradniki** (angl. Components)
  Seznam vseh grafičnih gradnikov, ki smo jih uporabili v trenutni aplikaciji.

- **Lastnosti** (angl. Properties)
  Skrajno desno na zaslonu se prikažejo vse nastavitve (lastnosti) za trenutno izbrani gradnik aplikacije.

Najprej poglejmo **Paleto**. Ta vsebuje seznam gradnikov, ki so razdeljeni v različne skupine: Uporabniški vmesnik (User Interface), Postavitev (Layout), Mediji (Media), Risanje in animacije (Drawing and Animation) itd.

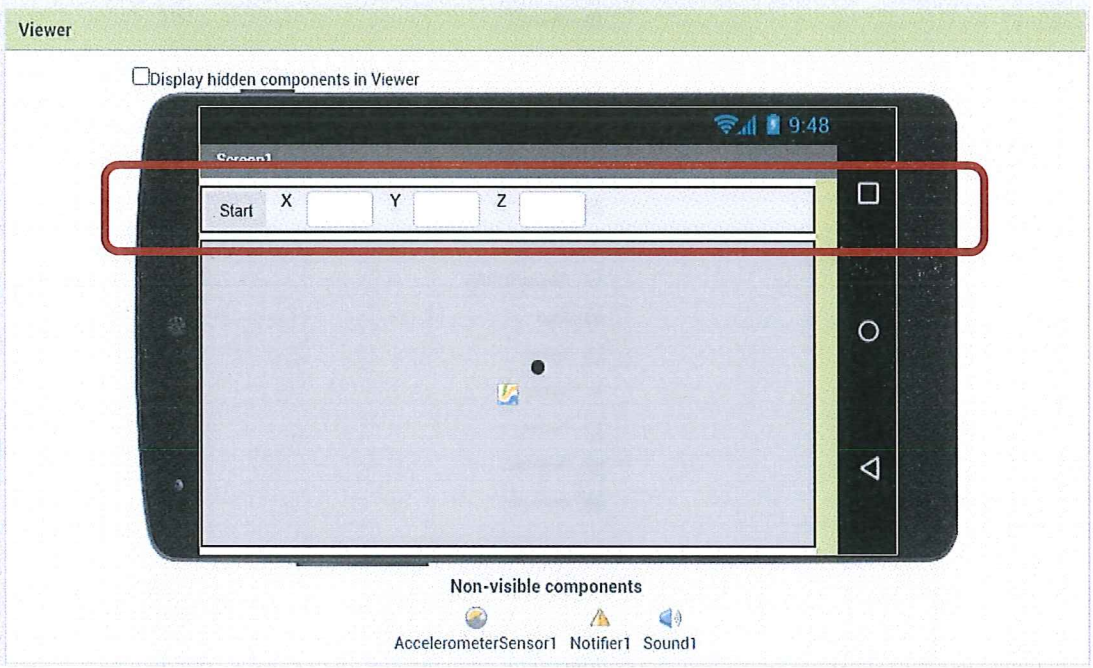


Slika 1.4: Skupine gradnikov v okolju MIT App Inventor

Vsaka skupina pa vsebuje več gradnikov. Npr. skupina **Uporabniški vmesnik** (User Interface) vsebuje naslednje gradnike (v angleščini): *Button*, *CheckBox*, *DatePicker*, *Image*, *Label*, *TextBox* itd. Pri izdelavi prvih aplikacij bomo uporabljali samo to skupino gradnikov. Najpogosteje uporabljeni gradniki iz skupine Uporabniški vmesnik so:

- *Button* je gumb, ki ga pritisnemo.
- *Label* je besedilo, ki se prikaže.
- *TextBox* pa je gradnik, ki omogoči uporabniku vnos nekega podatka (vnosno polje).

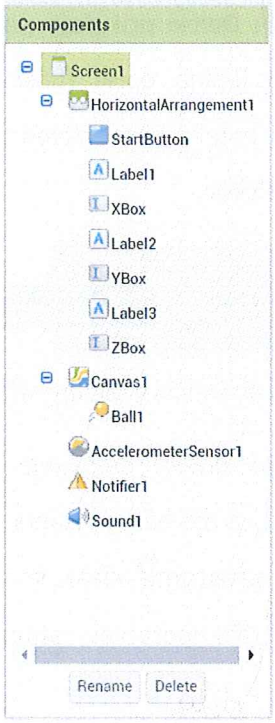
Pogled aplikacije (angl. Viewer) je na sredini pogleda Oblikovalec in obsega največji del zaslona. To je dokaj natančna predstavitev videza aplikacije. Končni videz na mobilni napravi se lahko spremeni. V Pogledu aplikacije so razvidni vsi gradniki, ki smo jih dodali v aplikacijo. Na primeru si poglejmo samo označen vrh aplikacije, ki je prikazan s pomočjo opcije Pogled aplikacije (Viewer). Razviden je gumb *Start*. Nato sledi labela *X* in vnosno polje (bel prazen okvir), ki ga predstavlja gradnik *TextBox*. Sledita še dve ponovitvi – labela *Y* in gradnik *Textbox* ter labela *Z* in gradnik *Textbox*.



Slika 1.5: Pogled na mobilno aplikacijo med razvojem

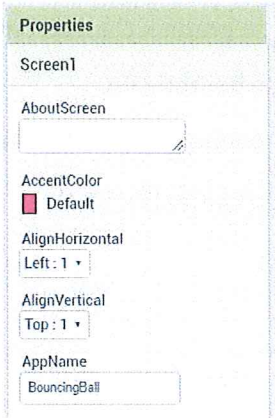
Gradniki (angl. Components) se nahajajo desno od Pogleda aplikacije. Gradnike (kot so gumbi, labele in vnosna polja) dodajamo na zaslon (angl. screen). Če pogledamo primer na

sliki, lahko kot prvi gradnik, zapisan zamaknjeno, vidimo *Screen1*. To pomeni, da so vsi ostali gradniki dodani na zaslon *Screen1*. Kot drugi gradnik nastopa *HorizontalArrangement,* ki skrbi, da so gradniki v vrstici urejeni. V tem gradniku so vključeni: *StartButton, Label1, Xbox, Label2, YBox, Label3* in *ZBox*. Vsak gradnik, ki ga uporabimo, ima svoje lastnosti.



Slika 1.6: Gradniki v aplikaciji

Lastnosti (angl. properties) najdemo skrajno desno. V tem delu nastavljamo lastnosti trenutno označenega gradnika, v primeru slike je to gradnik Zaslon, *Screen1*. Nekaj lastnosti zaslona: *AboutScreen, AlignHorizontal, AlignVertical, AppName* itd. Lastnost *AppName* tako nastavi ime naše aplikacije (npr. BouncingBall).

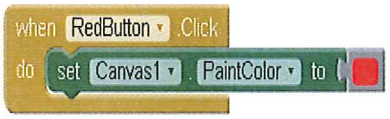


Slika 1.7: Lastnosti gradnika zaslon (Screen1)

1.2.2 Pogled Delčki

V pogledu Oblikovalec smo zastavili videz aplikacije. Njeno delovanje pa predstavimo v pogledu Delčki (Blocks). V tem pogledu zapisujemo obnašanje posameznih gradnikov z delčki. Namen tega pogleda je sestavljanje delčkov v smiselne celote. Vsaka celota predstavlja neko nalogo ali funkcionalnost aplikacije. Za vsak uporabljen gradnik iz Oblikovalca se nam v pogledu Delčki prikažejo podprti delčki dogodkov. Izbrani delček dogodka predstavlja zunanjo lupino opisa funkcionalnosti, ki se izvede ob proženju dogodka. Opis funkcionalnosti poteka interaktivno s sestavljanjem različnih delčkov.

Poglejmo si nekaj primerov delčkov.



Slika 1.8: Delčki v okolju MIT App Inventor

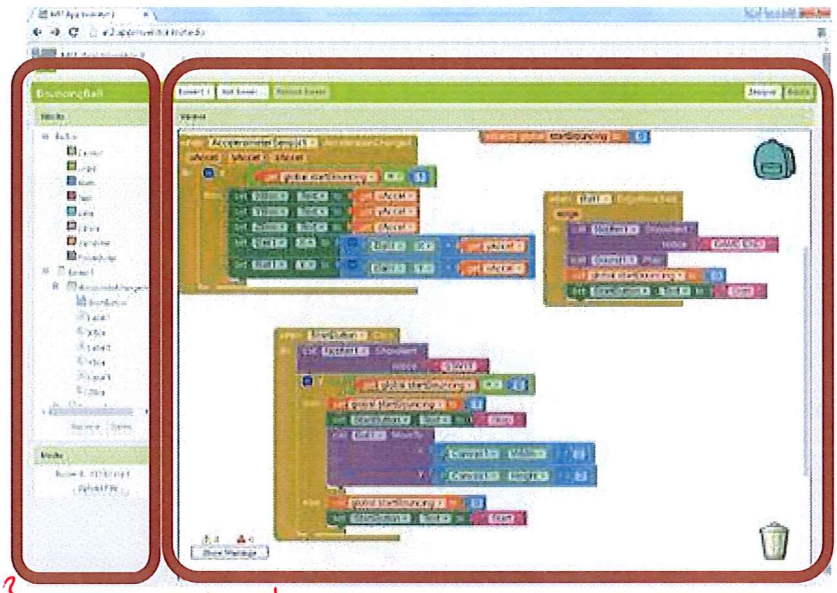
Zgornji delček *RedButton.Click* določa delovanje gumba *RedButton*. Ko bo uporabnik pritisnil gumb (dogodek *Click*), se bo barva risanja (*PaintColor*) platna *Canvas1* nastavila na rdečo. Platno je podobno programu Slikar, kamor lahko rišemo pike, črte ali like. V našem primeru na platno ne riše uporabnik, ampak program simulira uporabnika, npr. izbira barve.

Poglejmo si še en tak primer.

Slika 1.9: Dogodek *Click* za gradnik *CleanButton*

Zgornji delček pomeni, da ko gumb (z imenom *CleanButton*) pritisnemo (dogodek *Click*), bomo platno pobrisali (delček za gradnik *Canvas1* in njegov dogodek *Clear*). Torej vse, kar je na platno narisal program, se bo izbrisalo.

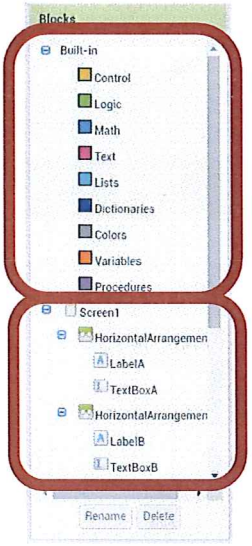
Zaslon, kjer urejamo delčke, vidimo na spodnji sliki. Na levi strani izbiramo med delčki (Blocks), na desni strani pa jih urejamo (Viewer).



Slika 1.10: Okolje Delčki (Blocks) v okolju MIT App Inventor

V Pogledu (Viewer) lahko namestimo več delčkov, tako kot jih vidimo na prejšnji sliki. Delčki so lahko zelo preprosti, lahko pa so sestavljeni iz mnogih drugih delčkov.

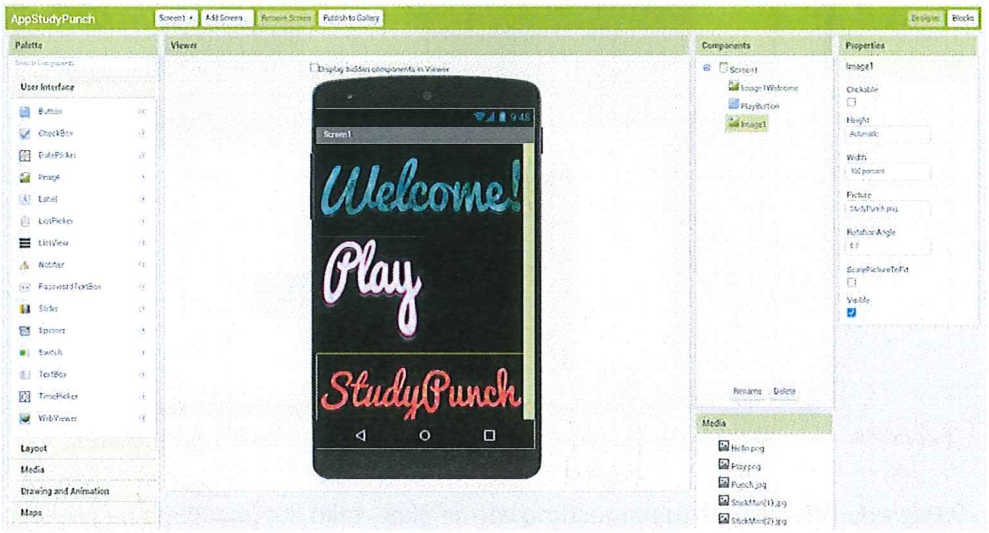
Med delčki najdemo vgrajene delčke (Built-in) in delčke za uporabljene gradnike. Med vgrajenimi delčki najdemo take, ki jih pogosto potrebujemo v aplikacijah. Druga skupina delčkov pa je odvisna od gradnikov, uporabljenih v aplikaciji. Na sliki tako vidimo gradnike: *Screen1*, *HorizontalArrangement*, *LabelA*, *TextBoxA* itd. Ob pritisku na posamezen gradnik se odprejo njegovi delčki.



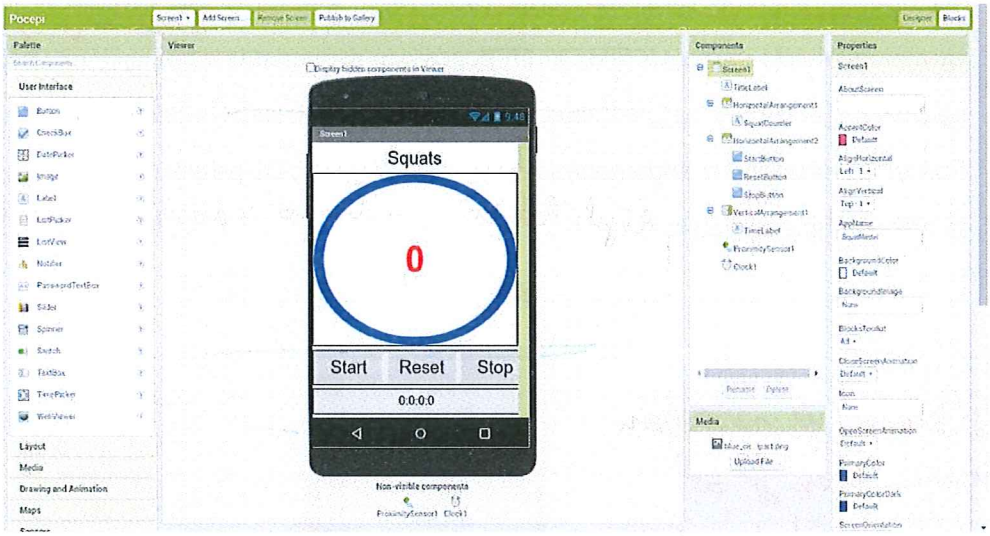
Slika 1.11: Vgrajeni delčki (Built-in) in delčki aplikacije

1.3 Naloge

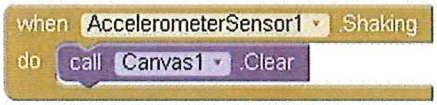
1. Katere gradnike najdemo v naslednji aplikaciji?



2. Katere gradnike najdemo v naslednji aplikaciji?



3. Kaj počne naslednji delček?



4. Kaj počne naslednji delček?

2 MOJA PRVA MOBILNA APLIKACIJA

Spoznali bomo osnovno delovanje okolja MIT App Inventor ter naredili prvo mobilno aplikacijo.

2.1 Navodilo

Za prvi primer želimo narediti aplikacijo, v katero vstavimo svoje ime, in aplikacija izpiše pozdravno sporočilo. Izdelana aplikacija je videti takole:

Slika 2.1: Prva aplikacija v okolju MIT App Inventor

2.2 Ustvarjanje projekta

Ustvarimo nov projekt in ga primerno poimenujmo. V našem primeru je to lahko »Vaja1 Pozdrav«. Dobro poimenovanje je ena izmed pomembnih vrlin uspešnega projekta. Pri razvoju moramo poimenovati skoraj vse gradnike in funkcionalnosti. Čeprav poimenovanje v končnem izdelku praviloma ni vidno, je ključno za uspešno delo razvijalca aplikacije.

Poimenovanje je pomembno tudi za sodelovanje med razvijalci in na koncu za sodelovanje med aplikacijami. Natančni opazovalci bodo opazili, da je dobro poimenovanje ali oblikovanje gradnikov značilno za vse gradnike uspešnih aplikacij ali okolij. Izkušeni uporabniki takšnih sistemov lahko že iz samega imena, oblike ali barve sklepajo o namenu gradnika.

2.3 Videz aplikacije

Najprej se postavimo v pogled Oblikovalec. Na kratko bomo opisali enega izmed načinov, kako lahko izdelamo želeni videz. Iz slike prepoznamo, da potrebujemo tri besedila (»Vnesi ime:«, »Pozdravljeni«, »Tomaž«), eno vnosno polje in dva gumba. Na delovno površino prenesimo naslednje gradnike:

- vstavimo prvo labelo (Label)
- vstavimo vnosno polje (TextBox)
- vstavimo prvi gumb (Button)
- vstavimo drugi gumb (Button)
- vstavimo še drugo labelo (Label)
- vstavimo še tretjo labelo (Label)

Sam premik gradnikov na površino še ne prinese želenega videza. Želimo uporabljati vrstice, poravnave in podobno. V ta namen dodajmo še gradnik *HorizontalArrangement* (v skupini Layout), ki poskrbi, da lahko imamo v isti vrstici več gradnikov. V primeru gradnika *HorizontalArrangement* vidimo, da nimajo vsi gradniki svojega videza. Omenjeni gradnik pomaga urediti oziroma določiti položaj skupini izbranih gradnikov. Skupino izbranega gradnika določimo tako, da v ta gradnik prenesemo:

- prvi gumb
- drugi gumb

Med komponente dodajmo še en gradnik *HorizontalArrangement*. V ta gradnik prenesimo:

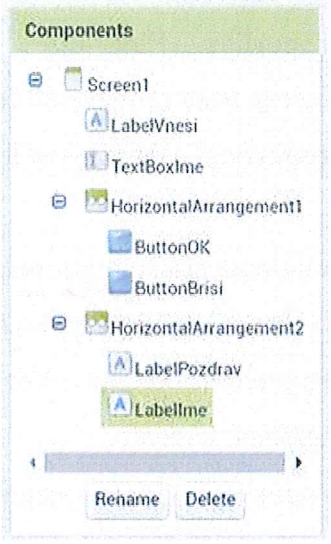
- drugo labelo
- tretjo labelo (pri tem pazimo, da se tretja labela postavi desno od druge labele)

Gradnike preimenujemo tako, da označimo gradnik in pritisnemo gumb *Rename* (nahaja se na dnu okvirja, kjer so našteti uporabljeni gradniki):

- prvo labelo poimenujemo: LabelVnesi
- vnosno polje: TextBoxIme

- prvi gumb: ButtonOK
- drugi gumb: ButtonBrisi
- druga labela: LabelPozdrav
- tretja labela: LabelIme

Kako poimenujemo gradnike, je stvar uporabnika. Dobra praksa je, da vrsto gradnika ohranimo v imenu, zato v našem primeru nismo poimenovali gumba *Brisi*, ampak *ButtonBrisi*. Zgornji razpored gradnikov in poimenovanja vidimo na naslednji sliki.



Slika 2.2: Vsi uporabljeni gradniki in njihova razporeditev v aplikaciji

Videz gradnikov želimo nekoliko spremeniti, tako kot vidimo na naslednji sliki. Zato je potrebno spremeniti lastnosti posameznih gradnikov:

- Screen1
  - ~~Gradnik predstavlja posamezno okno aplikacije.~~
  - V lastnost Naslov zapišemo »Izpiši ime«. Naslov se pojavi na vrhu aplikacije.
    - Properties → Title → »Izpiši ime«
  - Zaslonu spremenimo barvo ozadja.
    - Properties → BackgroundColor → »Cyan«
- HorizontalArangement1
  - Gradnik razširimo čez celotno širino zaslona.
    - Properties → Width → »Fill parent«
- HorizontalArangement2

- Nastavimo lastnost širina (Width) enako kot pri gradniku HorizontalArangement1.

- LabelVnesi
  - V lastnost naslov zapišemo »Izpiši me«.
    - **Properties → Text → »Vnesi ime«**
- TextBoxIme
  - V lastnosti namig izbrišimo namig »Hint for TextBox1«. Namig v naši aplikaciji ni potreben.
    - **Properties → Hint → »«**
  - Širino vnosnega polja raztegnemo čez celo širino zaslona.
    - **Properties → Width → »Fill parent«**
- ButtonOK
  - Izberemo lastnost širina in v procent zapišemo 50. To pomeni, da bomo raztegnili velikost gumba na polovico širine zaslona.
    - **Properties → Width → Percent → 50**
  - V gumb zapišemo besedilo.
    - **Properties → Text → »Izpiši«**
- ButtonBrisi
  - Izberemo lastnost širina in v procent zapišemo 50.
    - **Properties → Width → Percent → 50**
  - V gumb zapišemo besedilo.
    - **Properties → Text → »Briši«**
- LabelPozdrav
  - Med lastnostmi poiščemo velikost pisave in nastavimo na 40.
    - **Properties → FontSize → 40**
  - Med lastnostmi poiščemo barvo pisave in nastavimo na rdečo barvo.
    - **Properties → TextColor → Red**
  - Med lastnostmi poiščemo besedilo in ga izbrišemo.
    - **Properties → Text → »«**
- LabelIme
  - Labeli nastavimo lastnosti enako kot pri labeli labelPozdrav.

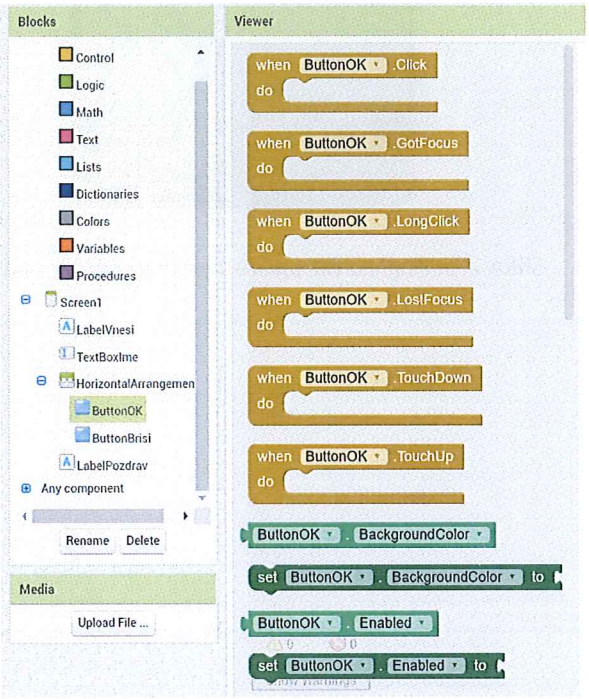
Slika 2.3: Pogled na uporabljene gradnike v prvi aplikaciji

2.4 Delovanje aplikacije

Za določanje funkcionalnosti aplikacije se je potrebno premakniti v pogled Delčki (Blocks). Na levi strani najdemo vse gradnike, ki smo jih vključili v aplikacijo.

Med njimi je tudi gumb *ButtonOK*. Ob pritisku na ta gumb želimo izpisati besedilo »Zdravo« ter ime, ki je bilo vpisano v vnosno polje.

Pritisk na gumb je dogodek. Za vsak gradnik je kar nekaj vnaprej definiranih dogodkov. Vidimo jih, če pritisnemo na posamezen gradnik. Če pritisnemo na gumb *ButtonOK*, se prikažejo naslednji dogodki, zapisani v rjavi barvi, kot vidimo na sliki 2.4.



Slika 2.4: Delčki za gumb *ButtonOK*

Ob dogodkih vidimo še lastnosti tega gradnika, ki so zapisane v zeleni barvi. Lahko bi pa bili delčki tudi vijolične barve, kar bi pomenilo, da gre za metode.

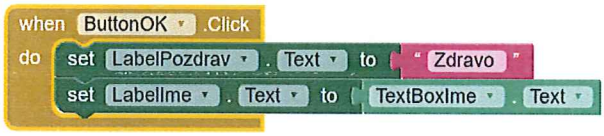
Če želimo implementirati pritisk na gumb, potem moramo uporabiti dogodek *Click*. Delček *ButtonOK.Click* prenesemo na delovno površino:

Slika 2.5: Delček z dogodkom *Click*

Sedaj želimo dogodku *Click* dodati funkcionalnost:

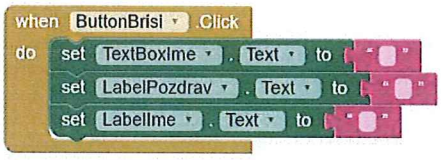
- Poiščemo gradnik LabelPozdrav in izberemo gradnik za nastavljanje nove vrednosti lastnosti Text. Torej pazimo, da izberemo *set*.
  - Blocks → LabelPozdrav → setText
- Med vgrajenimi gradniki poiščemo skupino delčkov Besedilo (Text), v njej izberemo prvi delček  in vanj vpišemo »Zdravo«.
  - Blocks → Built-in → Text → »Zdravo«
- Poiščemo gradnik LabelIme in izberemo lastnost Text. Pazimo, da izberemo set.
  - Blocks → labelIme → Text → set Text
- Poiščemo gradnik TextBoxIme in izberemo lastnost Text. Tokrat izberemo delček za vračanje (z leve strani ima delček uho) in ga dodamo delčku iz prejšnje točke.
  - Blocks → TextBoxIme → Text

Opisani postopek nam da naslednji sestavljeni delček:



Slika 2.6: Dogodek *Click* za gumb *ButtonOK* in njegova funkcionalnost

V prvi aplikaciji imamo še gumb *ButtonBrisi*. Ideja tega gumba je, da izbrišemo vsebino vnosnega polja *TextBoxIme* ter obeh label *labelPozdrav* in *labelIme*. To naredimo tako, da za vse tri delčke izberemo lastnost *Text* in jo nastavimo na prazen niz besedila. Delovanje delčka *ButtonBrisi.Click* prikazuje naslednja slika.



Slika 2.7: Dogodek *Click* za gumb *ButtonBrisi* in njegova funkcionalnost

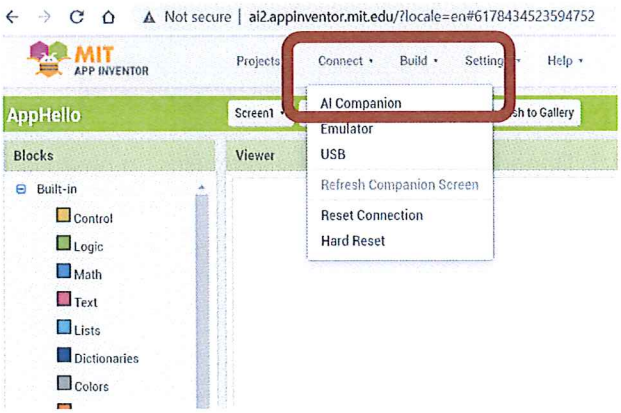
2.5 Moj prvi zagon aplikacije

MIT App Inventor ima uporabno orodje za spremljanje aplikacije na napravi Android med razvojem. Vsaka sprememba, ki jo naredimo v okolju MIT App Inventor, bo vidna takoj na mobilni napravi. Aplikacijo **MIT AI2 Companion** (glejte sliko spodaj) najdete v Trgovini Google Play z iskanjem "MIT AI2 Companion".

Na mobilni napravi je torej potrebno namestiti aplikacijo MIT AI2 Companion.

Slika 2.8: Namestitev aplikacije MIT AI2 Companion

Ko je aplikacija MIT AI2 Companion na mobilni napravi nameščena, na računalniku v meniju Connect izberemo *AI Companion*:

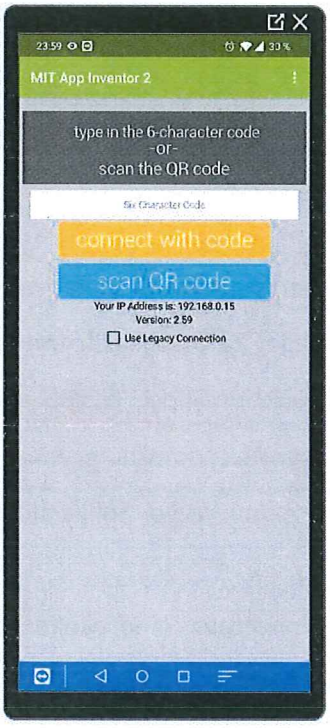


Slika 2.9: Povezovanje okolja MIT App Inventor z aplikacijo MIT AI2 Companion

Prikaže se QR-koda.

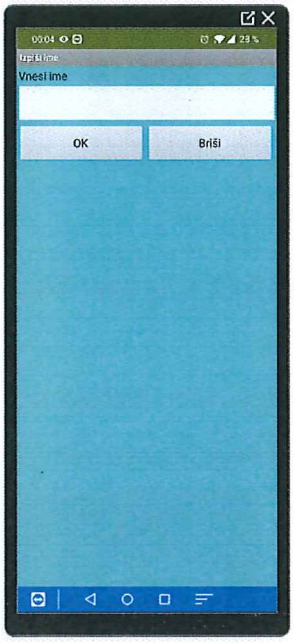
Slika 2.10: QR-koda v okolju MIT App Inventor

Premaknemo se na mobilno napravo in poiščemo pravkar nameščeno aplikacijo MIT App Inventor 2. Izberemo *Scan QR code* in preberemo kodo na zaslonu računalnika.



Slika 2.11: Optično prebiranje kode QR na mobilni napravi

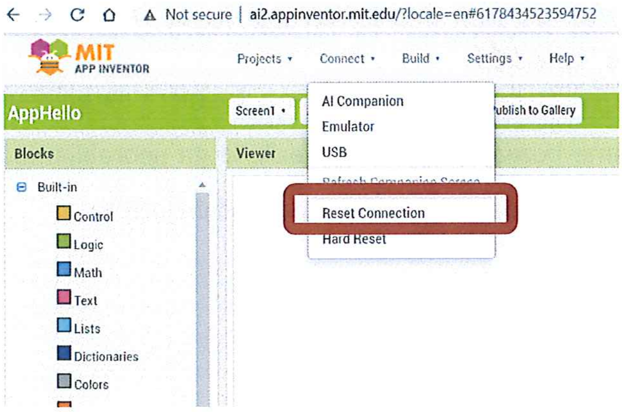
Začne se prenos aplikacije iz oblačne storitve MIT App Inventor na mobilno napravo. Čez nekaj časa se na telefonu odpre aplikacija in sedaj jo lahko testiramo.



Slika 2.12: Prvi zagon aplikacije na mobilni napravi

Priporočeno je, da sproti preverimo pravilnost delovanja posamezne funkcionalnosti, temu pravimo, da testiramo delovanje. Zato aplikacijo med razvojem večkrat zaganjamo in celo po potrebi dodamo kakšno funkcionalnost, ki služi samo kot potrditev delovanja. Primer takšne funkcionalnosti je pogosto pomožni gradnik labela, kjer si izpisujemo vmesne izračune ali rezultate. Ko končamo razvoj aplikacije, takšne funkcionalnosti skrijemo ali odstranimo.

Včasih se tudi zgodi, da se povezava med okoljem MIT App Inventor in aplikacijo AI Companion na mobilni napravi preprosto izgubi. Povezavo ponovno vzpostavimo tako, da v meniju *Connect* poiščemo *Reset Connection* in potem ponovno vzpostavimo povezavo s QR-kodo.



Slika 2.13: Prekinitev povezave med okoljem MIT App Inventor in aplikacijo MIT AI2 Companion

2.6 Ideje za dopolnitev aplikacije

Sledi nekaj idej, kako spremeniti ali izboljšati program. Najprej predlagamo, da namesto spreminjanja obstoječega projekta ustvarite kopijo (poiščite možnost *Save as*... v meniju *Projects*), na kateri nato vadite dopolnitve.

1. Spremenite besedilo »Zdravo« v »Ustvarjalec«.
2. Spremenite velikost in barvo besedila in gumbov.
3. Ob kliku se izpiše še ena vrstica z labelo. Izpišite »Pozdravljeni ustvarjalec«. Preverite delovanje gumba »Briši«.
4. Dodajte nov gumb in si izmislite, kaj naredi.

3 OSNOVNI KONCEPTI BLOČNIH JEZIKOV

Za nami je prva aplikacija. Da lahko opišemo, kaj mora narediti aplikacija ob posameznem dogodku, moramo poznati programski jezik, ki ga uporabljamo. V naslednjem poglavju bomo spoznavali nekaj osnovnih konceptov bločnih jezikov.

3.1 Nizi

V aplikacijah pogosto uporabljamo podatke, ki jih zapišemo kot zaporedje znakov. Znaki so lahko črke, številke in posebni znaki, kot je npr. presledek. Skupaj sestavljajo posamezne besede ali pa celo daljša besedila. Takšno zaporedje znakov imenujemo **nizi**. Nekaj primerov:

»T«

»Tomaž«

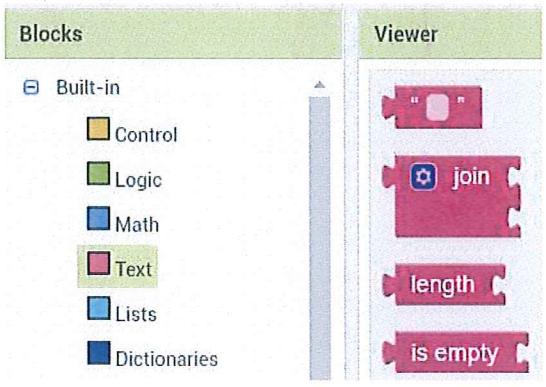
»Zdravo Tomaž«

»Zdravo Tomaž!«

Nad nizi lahko izvajamo operacije, kot so sestavljanje več nizov, spreminjanje ali brisanje posameznih znakov ali delov niza ter iskanje. Delčke za delo z nizi najdemo med vgrajenimi delčki v skupini Besedilo (Text):

Blocks → Built-in → Text

Na vrhu najdemo delček *prazen niz*  , v katerega lahko vpišemo poljubno besedilo. Sledi delček *join*, ki sestavi dva ali več nizov v nov niz, itd.



Slika 3.1: Delčki, povezani z nizi

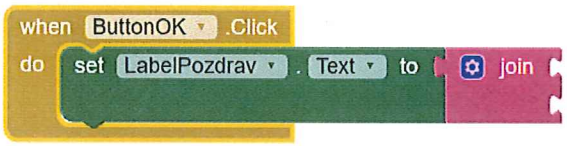
Delčki z nizi so takšni, da jih vključujemo v druge delčke (uho na levi strani delčka). Tipičen primer gradnikov, kjer uporabljamo nize, so labele in vnosna polja.

3.1.1 Aplikacija Izpiši ime – drugič

Vzemimo aplikacijo iz poglavja 2 in jo nekoliko spremenimo. V aplikaciji imamo dve labeli (labelPozdrav in labelIme), ki sta postavljeni ena poleg druge in izpišeta dva niza: »Zdravo« in naše ime. To lahko spremenimo tako, da zapišemo oba niza v eno samo labelo, in sicer labelo *labelPozdrav*. Potrebujemo delček *join*, ki nam pomaga združiti nize.

Blocks → Built-in → Text → join

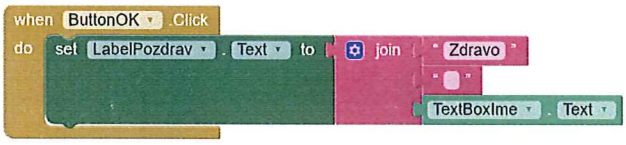
Dodamo ga z desne strani v delček *LabelPozdrav*, kot prikazuje slika.



Slika 3.2: Sestavljanje nizov z delčkom *join*

Ker želimo med »Zdravo« in našim imenom še presledek, sledimo naslednjemu postopku:

- Delček *join* ima modro ikono, pritisnemo jo in k dvema nizoma dodamo še tretjega.
- Med vgrajenimi gradniki poiščemo skupino Besedilo in v njej izberemo prvi gradnik (prazen niz) ter vanj zapišemo »Zdravo«.
  - **Blocks → Built-in → Text → prazen niz → »Zdravo«**
- Ponovimo postopek ter v nov gradnik zapišemo samo presledek.
- Vsebino vnosnega polja dodamo v tretje uho delčka *join*:
  - **Blocks → TextBoxIme → Text**

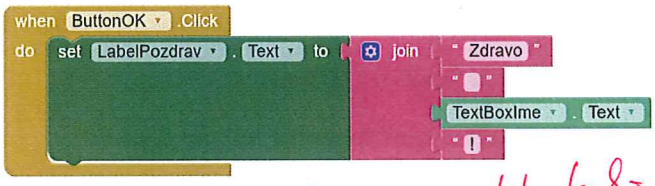


Slika 3.3: Dogodek *Click* in njegova funkcionalnost

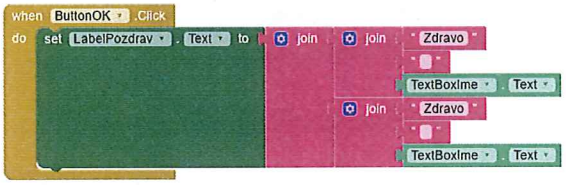
3.1.2 Ideje za dopolnitev aplikacije

Preskusimo uporabo nizov in nadgradimo aplikacijo. Spodaj podajamo nekaj idej, kaj lahko naredimo.

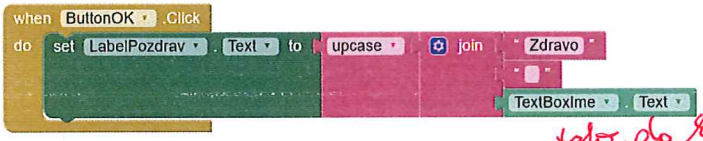
1. Na koncu pozdrava izpišite »!«.



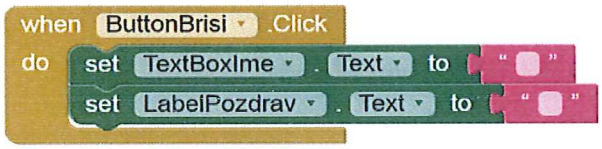
2. Pozdrav zapišite dvakrat.



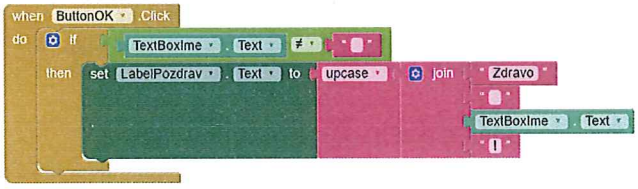
3. Pozdrav izpišite s samimi velikimi črkami.



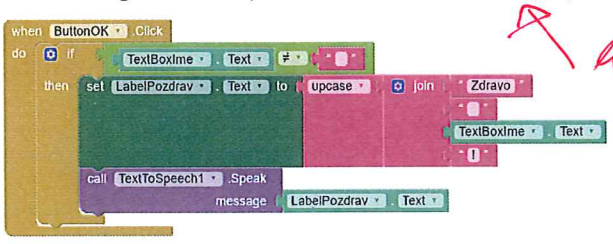
4. Popravite funkcionalnost v gumbu *ButtonBrisi* (izbrišemo pozdrav in vnosno polje).



5. Če je vnosno polje prazno, ne izpišemo pozdrava.



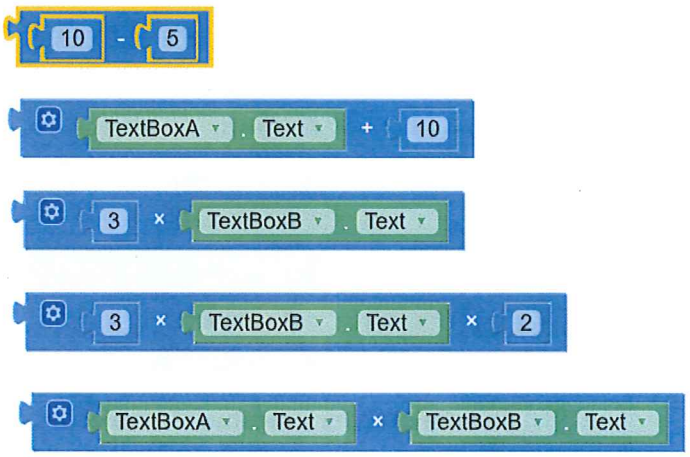
6. Pozdrav izgovorimo. (Pozor! Med lastnostmi je potrebno nastaviti jezik.)



3.2 Izrazi

Poleg nizov v aplikacijah pogosto uporabljamo tudi števila. Podobno kot pri matematiki števila v aplikacijah seštevamo, odštevamo in izvajamo druge matematične operacije. Temu zapisu v matematiki pravimo **izraz** in enako je tudi v bločnem programiranju.

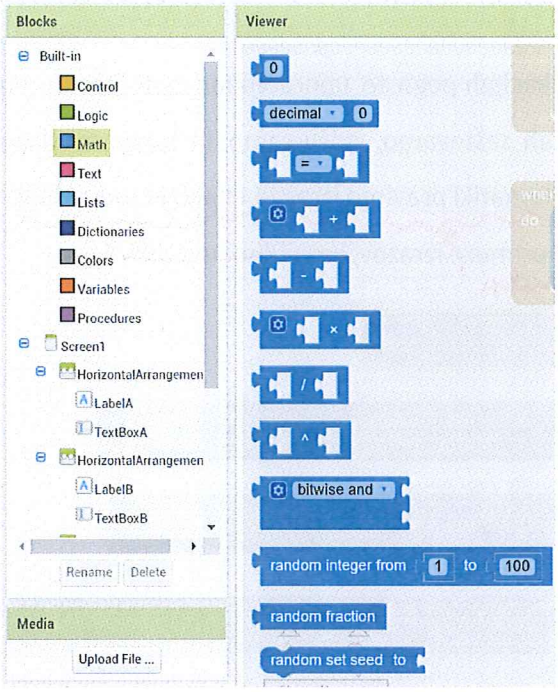
Imejmo naslednje primere izrazov, kot vidimo na sliki.:



Slika 3.4: Primeri aritmetičnih izrazov

Vsi primeri na sliki zgoraj so primeri aritmetičnih izrazov. Prvi aritmetični izraz odšteje številu 10 število 5. Drugi primer iz zgornje slike je podoben. Iz vnosnega polja *TextBoxA* vzamemo vrednost (lastnost *Text*) in k njej prištejemo 10.

Kot vidimo na zgornji sliki, so aritmetične operacije, uporabljene v aritmetičnih izrazih, lahko različne: podatke lahko seštevamo, odštevamo, množimo itd. Delčke za delo z aritmetičnimi izrazi najdemo med vgrajenimi delčki v skupini Matematika (Math), kot vidimo na naslednji sliki.



Slika 3.5: Primeri matematičnih blokcev

Število operandov je lahko tudi več kot samo dve. Če pritisnemo temnomodri zobnik na začetku delčka  , lahko povečamo število podatkov, ki jih vključujemo v izraz. Lep primer je četrti izraz na sliki, kjer množimo tri vrednosti (3, TextBoxB in 2).

Prav tako so lahko podatki v izrazu iz različnih virov: lahko so števila, lahko pa do podatka pridemo preko gradnikov (label, vnosnih polj itd.). V zadnjem primeru aritmetičnega izraza na sliki sta podatka, ki ju množimo, shranjena v lastnost *Text* vnosnega polja (TextBoxA in TextBoxB).

Opazimo, da imajo vsi izrazi na levi strani delčka uho. To pomeni, da je izraz potrebno pripeti delčku, ki to omogoča. Spodaj je primer, kjer vrednost izraza shranimo v lastnost *Text* labele *labelIzracun*.

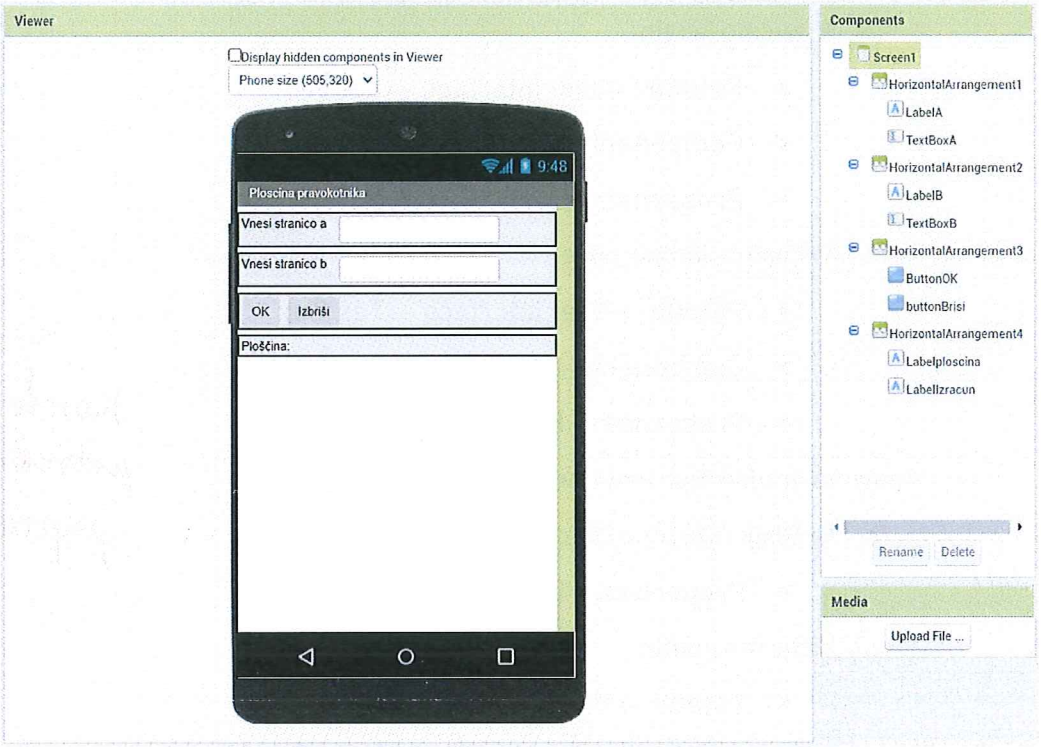
Slika 3.6: Vrednost izraza zapišemo v labelo

3.2.1 Aplikacija Pravokotnik

Izrazé bomo spoznali s pomočjo izdelave aplikacije za izračun ploščine pravokotnika. Ustvarimo nov projekt.

Navodilo

Videz aplikacije na mobilni napravi vidimo na spodnji sliki.



Slika 3.7: Aplikacija Pravokotnik v okolju MIT App Inventor

Videz aplikacije

Sedaj smo že spretnejši in vemo, da potrebujemo gradnike, ki združujejo gradnike v vrstice.

Tokrat začnemo tako, da:

- Vstavimo gradnik HorizontalArrangement.
  - Gradnik razširimo čez celotno širino zaslona.
    - Properties → Width → »Fill parent«
  - Vstavimo labelo po naslednjem navodilu:
    - Palette → User interface → Label
    - Components → rename → LabelA
    - Properties → Text → »Vnesi stranico a«
  - Vstavimo vnosno polje.

        - Palette → User interface → TextBox
        - Components → rename → TextBoxA
        - Properties → Hint → »«
- Vstavimo gradnik HorizontalArrangement.
    - Gradnik razširimo čez celotno širino zaslona.
        - Properties → Width → »Fill parent«
    - Vstavimo labelo.
        - Palette → User interface → Label
        - Components → rename → LabelB
        - Properties → Text → »Vnesi stranico b«
    - Vstavimo vnosno polje.
        - Palette → User interface → TextBox
        - Components → rename → TextBoxB
        - Properties → Hint → »«

navedovaje je potrebno povsod popraviti!

- Vstavimo gradnik HorizontalArrangement.
    - Gradnik razširimo čez celotno širino zaslona.
        - Properties → Width → »Fill parent«
    - Vstavimo gumb.
        - Palette → User interface → Button
        - Components → rename → ButtonOK
        - Properties → Text → »OK«
    - Vstavimo gumb.
        - Palette → User interface → Button
        - Components → rename → ButtonBrisi
        - Properties → Text → »Izbriši«
- Vstavimo gradnik HorizontalArrangement.
    - Gradnik razširimo čez celotno širino zaslona.
        - Properties → Width → »Fill parent«
    - Vstavimo labelo.
        - Palette → User interface → Label
        - Components → rename → LabelPloscina
        - Properties → Text → »Ploščina:«

- Vstavimo labelo.
  - Palette → User interface → Label
  - Components → rename → LabelIzracun
  - Properties → Text → »«

Delovanje aplikacije

Za urejanje delovanja aplikacije se je ponovno potrebno premakniti v pogled Delčki (Blocks). Na levi strani najdemo vse gradnike, ki smo jih vključili v aplikacijo. Izberemo gradnik *ButtonOK* in dogodek Pritisni (Click):

Blocks → ButtonOK → Click

Ko se ta dogodek izvrši, želimo izračunati ploščino in vrednost zapisati v labelo *LabelIzracun*. Poiščemo delček, ki nastavi besedilo tega gradnika:

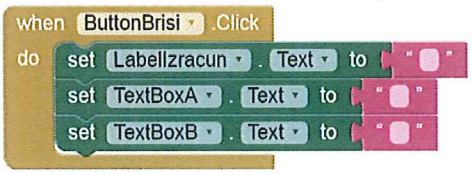
Blocks → LabelIzracun → setText

Zapišemo matematični izraz, tako da zmnožimo dva podatka iz vnosnih polj (TextboxA in TextBoxB), kot je prikazano na naslednji sliki.

Slika 3.8: Množenje podatkov iz dveh vnosnih polj

Še enkrat razložimo delčke na zgornji sliki. Zmnožimo podatka (stranici A in B), ki ju je uporabnik vnesel v vnosni polji *TextBoxA* in *TextBoxB*, vrednost pa zapišemo v labelo *labelIzracun* takrat, ko uporabnik pritisne gumb *ButtonOK*.

Na aplikaciji imamo še gumb *ButtonBrisi*. V njem ne uporabljamo matematičnih izrazov. Naloga tega gumba je, da izbriše vse vnose v vnosna polja in labele. Uporabimo prazen niz, kot vidimo na sliki spodaj.

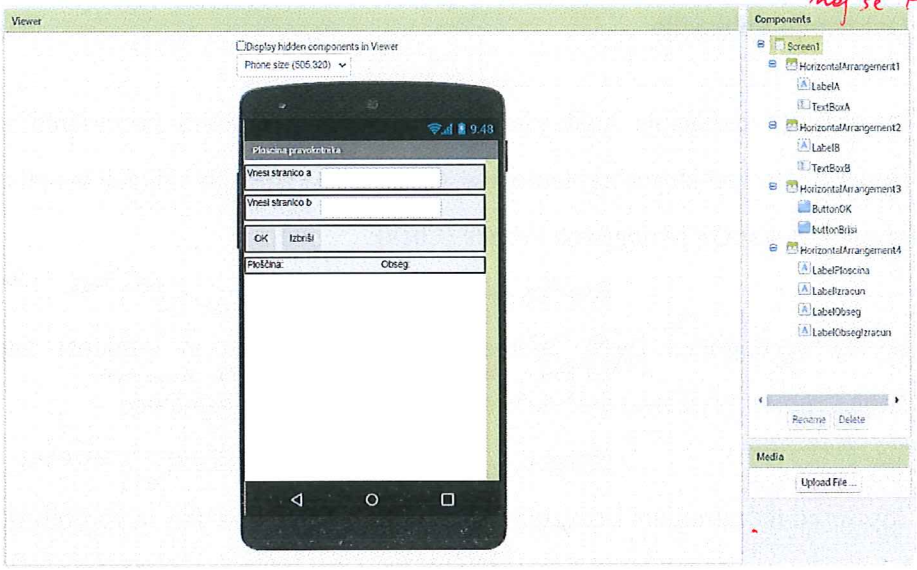


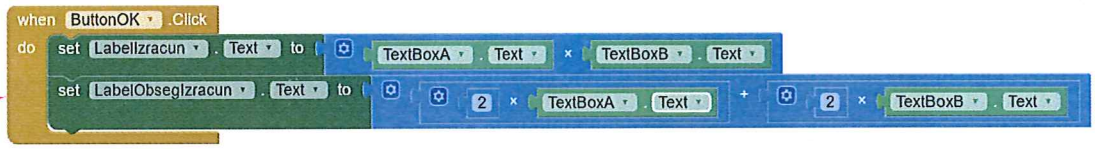
Slika 3.9: Gumb *Brisi*

3.2.2 Ideje za dopolnitev aplikacije

Aplikacijo Pravokotnik lahko nadgradimo. Spodaj je nekaj idej.

1. Obema stranicama in izračunu manjkajo podatki o enotah. Dodajte nove labele, kjer nastavite vrednosti na »cm« in »cm2«.
2. Dodajte izračun obsega pravokotnika. Zgledujte se po sliki spodaj.





Slika 3.10: Dodan izračun obsega pravokotnika

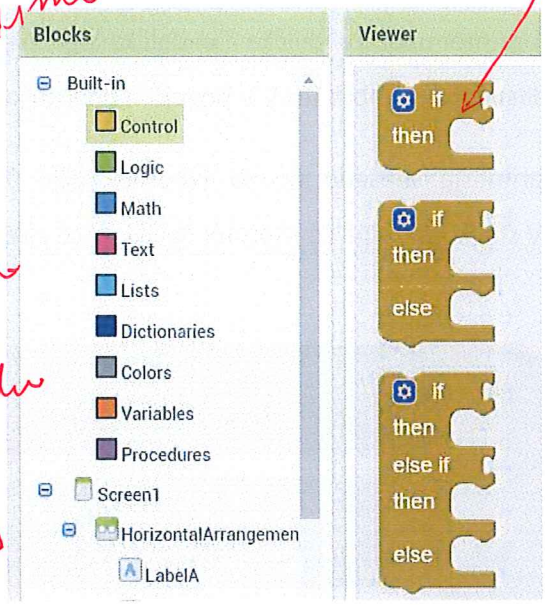
3. Aplikacijo dopolnite z izpisom ploščine v različnih enotah (cm^2, dm^2 itd.).
4. Ob kliku na gumb *OK* onemogočite popravljanje vrednosti (lastnost Enabled nastavite na logično vrednost »false«), ob kliku na gumb Briši pa nazaj na logično vrednost »true«.
5. Naredite še aplikacijo za izračun obsega in ploščine kroga.

3.3 Pogojni stavki in logični izrazi

V življenju pogosto dajemo navodila v obliki pogojev. Nekaj primerov:

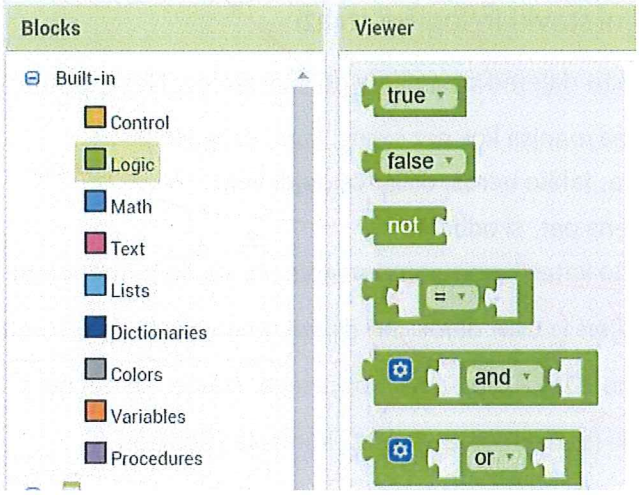
- Če je cena manjša kot pet evrov, kupi, drugače pa ne.
- Če dežuje, lahko bereš, drugače pojdi ven.
- Če je ocena pet, si odličen.

Stavke, s pomočjo katerih opisujemo pogoje (če stavke), imenujemo pogojni stavki. Pogojni stavek je sestavljen iz treh delov. Pri delček je pogoj, ki določa, ali se izvede prvi ali drugi delček programa. Delček za opis pogojnega stavka se začne z **if** in ga najdemo med vgrajenimi delčki (Built-in) pod skupino Kontrola (Control).



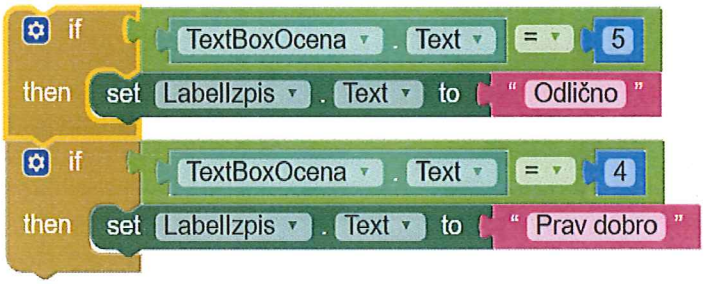
Slika 3.11: Skupina Kontrola z delčki if

Delček if nastopa v paru z **then**. Ta združuje bloke, ki jih izvedemo, če je pogoj izpolnjen. Pogoj dodamo na vrhu pogojnega stavka v obliki **logičnega izraza**. Logični izrazi imajo vrednost *pravilno* (angl. true) ali *nepravilno* (angl. false). Logične izraze najdemo med vgrajenimi (Built-in) delčki in so združeni v skupini Logika (Logic).



Slika 3.12: Skupina Logika z delčki za logične izraze

Pokažimo še skupni primer uporabe delčka *if* in delčka logični izraz. Za nalogo imamo izpis ocene z besedo (odlično, prav dobro itd.). V pogoju *if* uporabljamo logični izraz primerjave  , ki primerja vneseno oceno (vnosno polje *TextBoxOcena*) s številsko vrednostjo ocene. V stavku *then* pa so zapisani delčki, ki se izvedejo samo v primeru, ko je ta pogoj izpolnjen.



Slika 3.13: Primer pogojnega stavka in logičnega izraza

3.3.1 Aplikacija Pravokotnik – drugič

Uporabo pogojnega stavka bomo spoznali z aplikacijo Pravokotnik, ki jo že poznamo.

Navodilo

Cilj dobrega programa je, da prepreči nepravilno uporabo. V primeru aplikacije Pravokotnik si predstavljajte naslednji scenarij uporabe. Če uporabnik vnese podatke o stranicah a in b, se v aplikaciji izračuna ploščina. Kaj pa, če uporabnik ne vnese stranice b (vnosno polje ostane prazno) in pritisne gumb *ButtonOK*? Imamo samo en podatek (stranico a), in ta ni dovolj za izračun ploščine. Pravilno je, da v aplikaciji, preden izračunamo ploščino,

nista podani (TextBoxA in TextBoxB), uporabnik pa pritisne gumb *ButtonOK*, v labelo izpišemo »VNESI STRANICI«!

OBE STRANI POTREBUJEMO
OBE STRANICI !!!

Slika 3.16: Primer stavka *else*

Verjetno bi bilo smiselno zapisati celo zadevo nekako tako:

- Kaj pa, če pogoj ni izpolnjen. Takrat bi radi uporabnika opozorili, da morata biti vnešeni obe stranici.
- To bi lahko naredili tako, da bi dodali še en stavek if, kjer bi uporabili zanikani pogoj:

SLIKA

- A to je nerodno. Na med vgrajenimi delčki je tudi tale, ki vsebuje še prostor za ukaze, ki se izvedejo, ko pogoj ni izpolnjen (del else)

SLIKA

~~Takle uporabimo~~ Naš delček if nadomestimo s tem in prelozimo vse delčke, ter dopišemo delčke v del else.

- Gre pa tudi lažje. ~~Ko ma~~ Če ~~na~~ kliknemo na modro ikono pri delčku if dobimo možnost, da dodamo del else...

preverimo, ali je uporabnik res vnesel stranico b. V nasprotnem primeru izračuna ploščine ne bo.

Delovanje aplikacije

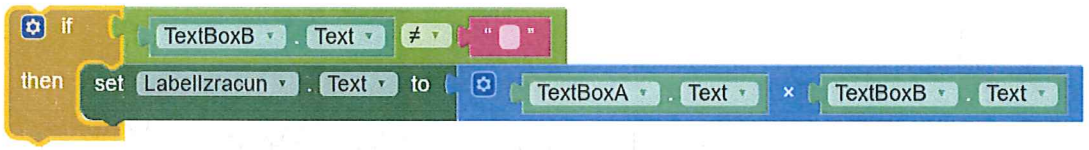
Na spodnji sliki z logičnim izrazom tako preverimo, da vnosno polje (lastnost *text* v gradniku *TextBoxB*) ni prazno:

Built-in → TextBoxB → Text

Logični izraz neenako najdemo:

Built-in → Logic → enako (ter ga nato spremenimo na neenako)

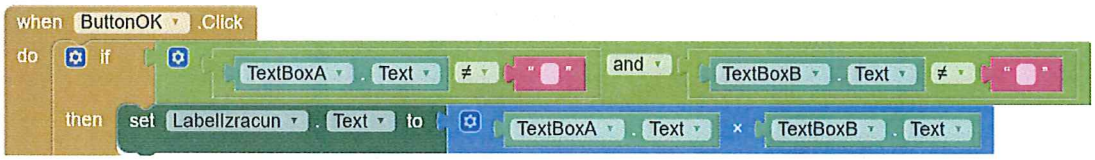
Če to drži, izračunamo ploščino in jo vpišemo v labelo *LabelIzracun*.



Slika 3.14: Preverimo vnosno polje *TextBoxB*

Prečrtan simbol ≠ pove, da primerjani vrednosti ne smeta biti enaki. Samo v tem primeru se izvršijo delčki, zapisani pod *then*.

Velikokrat pa se zgodi, da moramo logične izraze sestaviti. Tudi v primeru aplikacije Pravokotnik je tako. Ne samo da preverimo, ali je uporabnik vnesel stranico B, tudi stranico A je moral vnesti. Uporabiti je potrebno logični operator *in* (angl. and). V okolju MIT App Inventor je to delček and. Sedaj logični izraz nadgradimo tako, da preverimo v obeh vnosnih poljih, ali so bili podatki vneseni.



Slika 3.15: Primer sestavljenega logičnega izraza v delčku *if*

Pogoju *if* lahko dodamo tudi stavek *else*, ki se izvede, kadar pogoj ni izpolnjen. Stavek *else* dodamo s pomočjo temnomodrega koleščka pri gradniku *if*. V našem primeru, če stranici

3.4 Spremenljivke

Naslednji koncept bločnih jezikov, ki ga bomo spoznali, je **spremenljivka**. Delček spremenljivka je prostor, kamor se lahko shrani **vrednost**. Za razliko od spremenljivke v matematiki se lahko njena vrednost med izvajanjem aplikacije spreminja.

V bločnih jezikih je spremenljivka določena z imenom, vrednostjo in tipom. Tipi so pa lahko cela števila, realna števila, lahko tudi niz znakov ali še kaj drugega. Tip spremenljivke v bločnih jezikih definiramo s prvo uporabo.

Delček za vnos podatkov *TextBox* je poseben primer spremenljivke, v katero uporabnik s pomočjo tipkovnice vnese vrednost, drugače je vrednost spremenljivke uporabniku aplikacije skrita. Spremenljivke, ki so pomembne za uporabnika, izpišemo s pomočjo delčka *Label*.

Delčke **Variables** za delo s spremenljivkami, ki so skrite uporabniku aplikacije, najdemo med seznamom Vgrajenih delčkov. Novo spremenljivko vstavimo z delčkom in imenom, ki sprejme ime spremenljivke in začetno vrednost:

Blocks → Built-in → Variables → initialize global/local X to

Vrednost spremenljivke preberemo s pomočjo delčka *get*, nastavimo pa s pomočjo delčka *set*. Poglejmo si primer, ko ustvarimo spremenljivki z imeni *a* in *b*, izpišemo pa njun produkt.

Slika 3.17: Primer ustvarjanja in branja spremenljivk

3.4.1 Aplikacija Indeks telesne mase

Cilj aplikacije je povezati do sedaj obravnavane koncepte nizov, aritmetičnih in logičnih izrazov in spremenljivk.

Navodilo

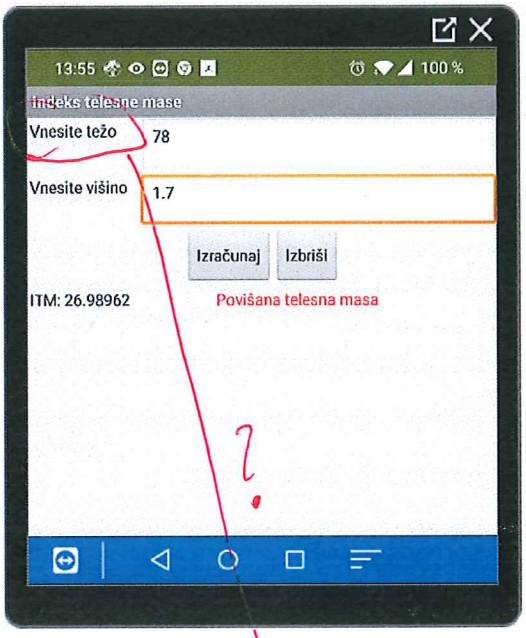
V tej aplikaciji želimo izračunati indeks telesne mase (ITM). Uporabnik vnese svojo težo in svojo višino (v metrih). ITM izračunamo po sledeči formuli:

$$itm = teža / (višina * višina)$$

V aplikaciji želimo izpisati vrednost ITM ter enega od treh tekstov:

- »Nizka telesna masa«, če je ITM manjši od 18,
- »Normalna telesna masa«, če je ITM med 18 in 25,
- »Povišana telesna masa«, če je ITM večji od 25.

Delovanje aplikacije na mobilni napravi vidimo na spodnji sliki.



Slika 3.17: Aplikacija »Indeks telesne mase«

Videz aplikacije

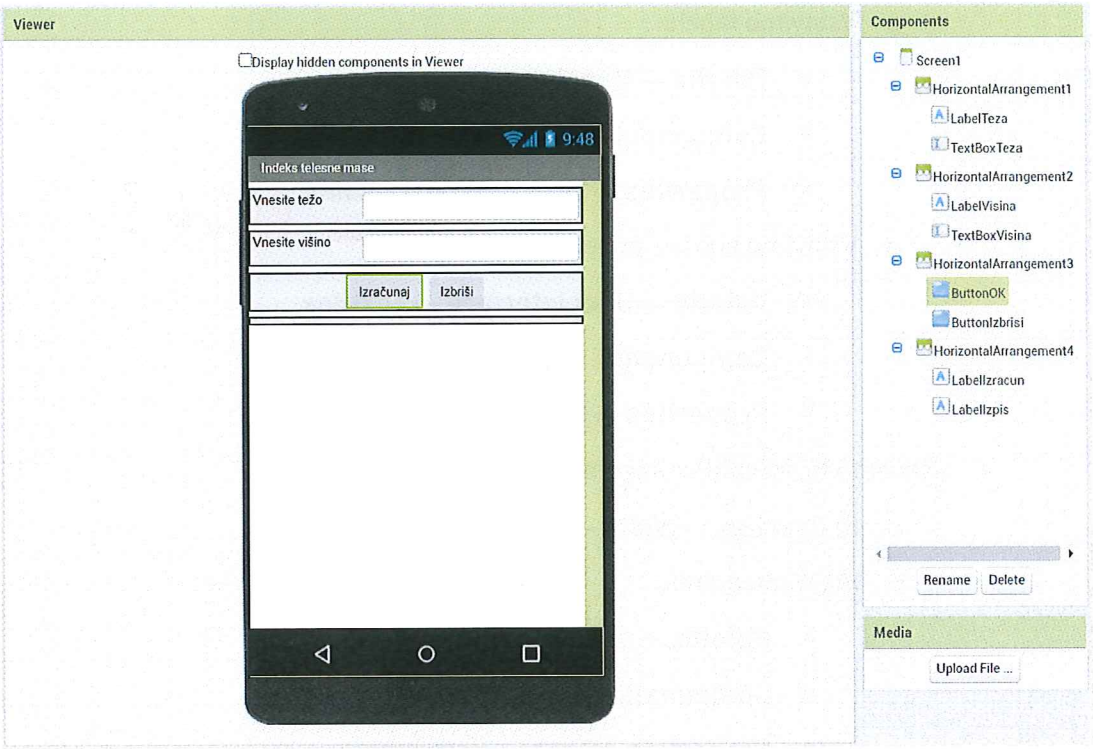
Na delovno površino prenesimo naslednje gradnike:

- Vstavimo HorizontalArrangement.
  - Properties → Width → »Fill parent«
  - Vstavimo labelo.
    - Palette → User interface → Label
    - Components → rename → LabelTeza
    - Properties → Text → »Vnesi težo«
  - Vstavimo vnosno polje.
    - Palette → User interface → TextBox
    - Components → rename → TextBoxTeza
    - Properties → Hint → »v kilogramih«
- Vstavimo HorizontalArrangement.
  - Properties → Width → »Fill parent«

- Vstavimo labelo.
  - Palette → User interface → Label
  - Components → Rename → LabelVisina
  - Properties → Text → »Vnesi višino«
- Vstavimo vnosno polje.
  - Palette → User interface → TextBox
  - Components → Rename → TextBoxVisina
  - Properties → Hint → »v metrih«

- Vstavimo HorizontalArrangement.
  - Properties → Width → »Fill parent«
  - Vstavimo gumb.
    - Palette → User interface → Button
    - Components → Rename → ButtonOK
    - Properties → Text → »Izračunaj«
  - Vstavimo še en gumb.
    - Palette → User interface → Button
    - Components → Rename → ButtonIzbrisi
    - Properties → Text → »Izbriši«
- Vstavimo HorizontalArrangement.
  - Properties → Width → »Fill parent«
  - Vstavimo labelo.
    - Palette → User interface → Label
    - Components → Rename → LabelIzracun
    - Properties → Text → »«
  - Vstavimo še eno labelo.
    - Palette → User interface → Label
    - Components → Rename → LabelIzpis
    - Properties → Text → »«

Velikost posameznih gradnikov poskusite ~~zapisati~~ določiti sami s pomočjo spodnje slike.



Slika 3.18: Gradniki aplikacije »Indeks telesne mase«

Delovanje aplikacije

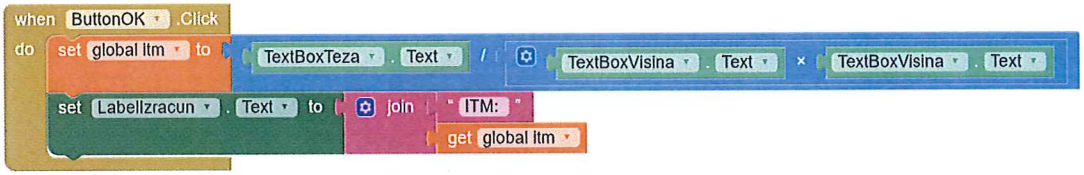
Postavimo se v pogled Delčki (Blocks). Ustvarimo si spremenljivko itm in jo nastavimo na začetno vrednost 0.

Slika 3.19: Spremenljivka *itm*

Na delovno površino zanesimo delček, ki predstavlja pritisk gumba *ButtonOK*:

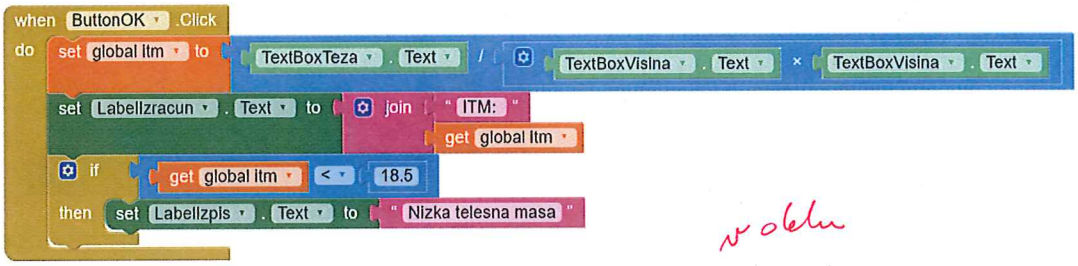
Blocks → ButtonOK → Click

Ko kliknemo gumb *ButtonOK* (delček *ButtonOK.Click*), se nastavi vrednost spremenljivke itm, ki je definirana z matematičnim izrazom deljenja in množenja, kot vidimo na spodnji sliki. Rezultat shranimo v spremenljivko itm. Spremenljivko itm izpišemo v labelo *LabelIzracun*.



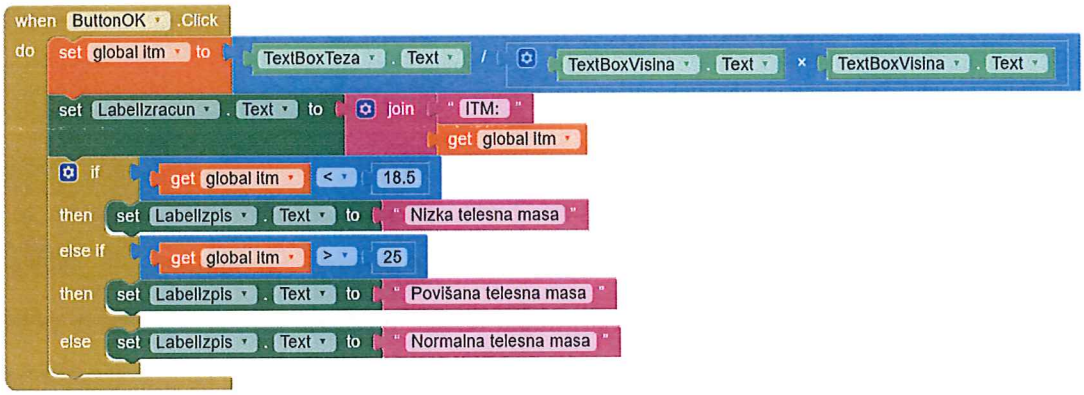
Slika 3.20: Primer uporabe spremenljivke

Sedaj se lotimo izpisa, ali je izračunana vrednost ITM nizka, normalna ali povišana. Najprej dodamo pogoj, če je *itm* manjši od 18.5. V tem primeru v gradnik *LabelIzpis* izpišemo »Nizka telesna masa«.



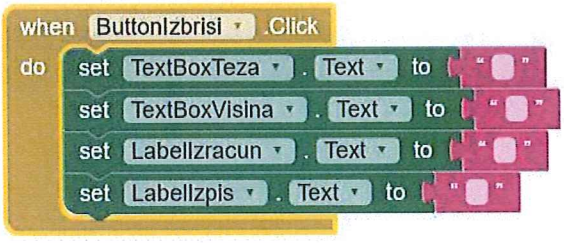
Slika 3.21: Pogoj preveri, ali je vrednost spremenljivke manjša od 18.5

Če to ne drži, lahko preverimo še kak drug pogoj z blokom **else if**. Preverjanje pogojev z *else if* je lahko tudi več. Če nobeden od logičnih izrazov v pogojih *if* in *else if* ne drži, potem se izvede blok *else*. Spodnji primer prikazuje uporabo sestavljenega pogojnega stavka.



Slika 3.22: Primer uporabe pogojev: delčki *if*, *else if* in *else*

Dodamo še delovanje gumba *ButtonIzbrisi*. Z uporabo gumba izbrišemo vse vnose v vnosna polja in labele.



Slika 3.23: Gumb *Brisi*

Ideje za dopolnitev aplikacije

Izpis indeksa telesne mase naj se izpiše v različnih barvah glede na težo:

- če je vrednost *itm* pod 18, se besedilo izpiše v zeleni barvi,
- če je vrednost *itm* med 18 in 25, se besedilo izpiše v črni barvi,
- če je vrednost *itm* nad 25, se besedilo izpiše v rdeči barvi.

Slika 3.24: Nastavitev barve besedila za labelo *LabelIzpis*

3.5 Procedure

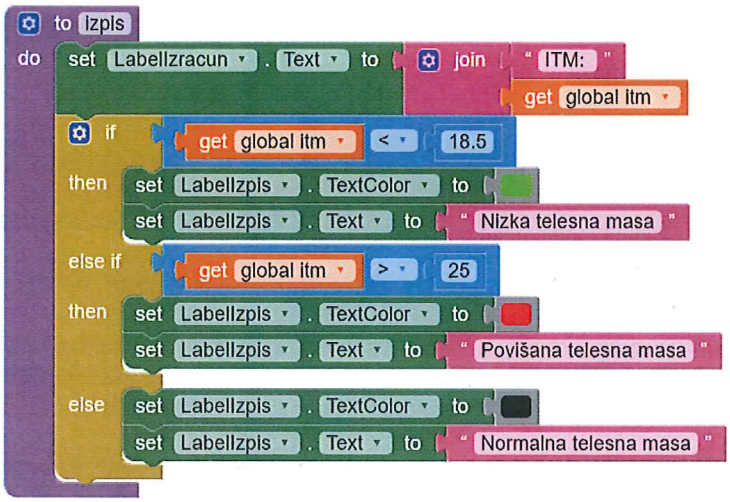
Naslednji koncept bločnih jezikov, ki ga bomo spoznali, je procedura. Pogosto se nam dogaja, da z razširitvami in bogatenjem aplikacij postane implementacija delčka zelo kompleksna, delčki pa sestavljeni iz mnogih drugih delčkov. V tem primeru je potrebno razmisliti, ali bi se dalo sestavljene delčke razdeliti v manjše smiselne celote, ki bi omogočile razumljivejši zapis algoritma.

Lep primer je zadnja slika na prejšnji strani, kjer je dogodek *Click* gradnika *ButtonOK* že precej velik. Primerjajmo delček iz prejšnje slike z naslednjim.

Slika 3.25: Klic procedure *izpis*

Opazimo lahko, da je vsa koda, povezana z izpisom indeksa telesne mase, umaknjena, namesto nje se v delčku *ButtonOK.Click* pojavi vijoličast delček *izpis*. Pravimo, da je *izpis* **klic procedure**.

Sedaj predstavimo še vsebino procedure *izpis*. Kot vidimo, je njena vsebina enaka tisti, ki je bila na sliki na prejšnji strani v delčku *ButtonOK.Click*.

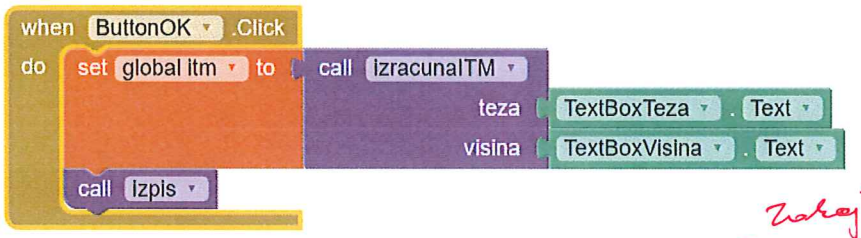


Slika 3.26: Vsebina procedure *izpis*

Smisel procedur je v tem, da lahko na ta način problem znotraj delčka razdelimo na manjše podprobleme in vsakega rešujemo posebej s proceduro, ki jo potem pokličemo, ko jo potrebujemo pri reševanju glavnega problema. Kot vidimo zgoraj, delček *ButtonOK.Click* postane krajši, preglednejši in s tem bolj razumljiv.

Še večjo prednost pa s proceduro dosežemo, kadar potrebujemo enako skupino delčkov na različnih mestih v aplikaciji. Takrat je smiselno oblikovati proceduro in jo klicati na tistih mestih, kjer to skupino delčkov potrebujemo. **Klicev procedure je torej lahko poljubno veliko**.

Poglejmo si še enkrat spremenjeni delček *ButtonOK.Click*. V naslednji sliki opazimo še klic procedure *izracunajITM*.

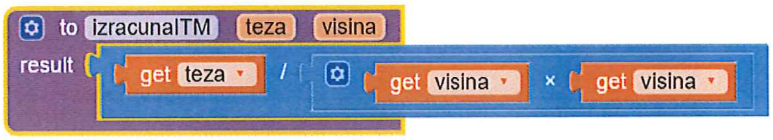


Slika 3.27: Klic procedure *izracunajITM*

Klic procedure *izracunajITM* je drugačen od klica metode *izpis*. Poglejmo desno stran tega delčka. Opazimo dva **parametra**: *teza* in *visina*. V parametru sta pripeti vrednosti dveh vnosnih polj: *TextBoxTeza* in *TextBoxVisina*. Procedure pogosto za svoje delovanje potrebujejo dodatne podatke. Parametri jim te podatke dostavijo.

Procedura *izracunajITM* se razlikuje od *izpisa* še po eni lastnosti. Če pogledamo levo stran tega delčka, opazimo uho, ki smo ga pripeli spremenljivki *itm*. Ugotovimo lahko, da ta

procedura nekaj **vrača,** in to zapišemo v omenjeno spremenljivko. Takšnim proceduram pravimo tudi **funkcije**.



Slika 3.28: Funkcija izracunajITM

Slika prikazuje funkcijo *izracunajITM*. Bodimo pozorni na označbo *result*, kjer zapišemo v funkciji tisto, kar želimo, da funkcija vrača. Poleg imena funkcije (npr. izracunajITM) pa desno najdemo vse parametre, ki jih lahko uporabimo v funkciji (npr. teza in visina).

Funkcije in procedure so zelo uporaben programski koncept. Zato jih bomo v prihodnjih nalogah s pridom izkoriščali.

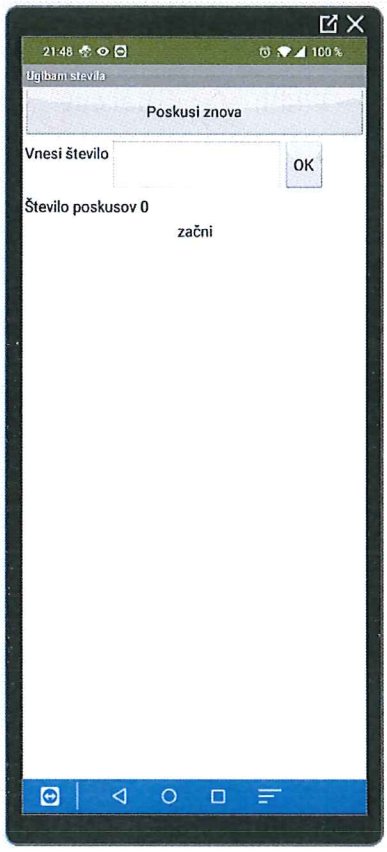
4 APLIKACIJE

Do sedaj smo spoznali že različne koncepte: nize, aritmetične izraze, pogojne stavke, logične izraze, spremenljivke in procedure. Povežimo jih v naslednji nalogi.

4.1 Aplikacija Ugibam števila

Navodilo

V tej igri želimo ugibati število med 1 in 20. Pri tem štejemo število poskusov. Videz aplikacije na mobilni napravi je prikazan na spodnji sliki.



Slika 4.1: Aplikacija »Ugibam stevila«

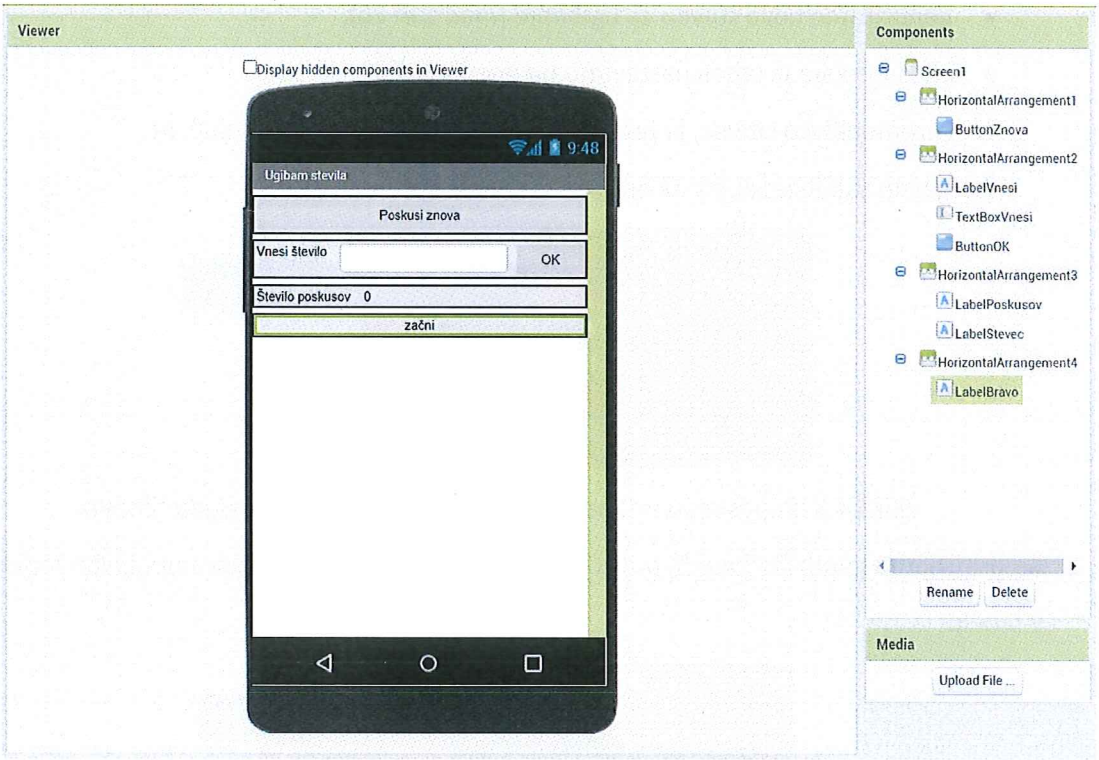
Videz aplikacije

Na delovno površino zanesimo naslednje gradnike:

- Vstavimo HorizontalArrangement1.
  - Properties → Width → »Fill parent«
  - Vstavimo gumb ButtonZnova.
    - Palette → User interface → Button
    - Components → rename → ButtonZnova
    - Properties → Text → »Poskusi znova«
    - Properties → Width → FillParent
- Vstavimo HorizontalArrangement2.
  - Properties → Width → »Fill parent«
  - Vstavimo labelo.
    - Palette → User interface → Label
    - Components → rename → LabelVnesi
    - Properties → Text → »Vnesi število«
  - Vstavimo vnosno polje.
    - Palette → User interface → TextBox
    - Components → rename → TextBoxVnesi
    - Properties → Hint → »«
  - Vstavimo gumb.
    - Palette → User interface → Button
    - Components → rename → ButtonOK
    - Properties → Text → »OK«
    - Properties → Width → FillParent
- Vstavimo HorizontalArrangement3.
  - Properties → Width → »Fill parent«
  - Vstavimo labelo.
    - Palette → User interface → Label
    - Components → rename → LabelPoskusov
    - Properties → Text → »Število poskusov«
  - Vstavimo še eno labelo.
    - Palette → User interface → Label

- Components → rename → LabelStevec
- Properties → Text → »0«

- Vstavimo HorizontalArrangement4.
  - Properties → Width → »Fill parent«
  - Vstavimo labelo.
    - Palette → User interface → Label
    - Components → rename → LabelBravo
    - Properties → Text → »začni«
    - Properties → Width → »Fill parent«
    - Properties → TextAlignment → »center«



Slika 4.2: Gradniki aplikacije »Ugibam števila«

Delovanje aplikacije

Ustvarimo spremenljivko *stevec* in jo nastavimo na začetno vrednost 0. Spremenljivka šteje število naših poskusov pri ugibanju števila.

Slika 4.3: Spremenljivka *stevec*

Potrebujemo še spremenljivko *stevilo*, ki bo hranila število, ki ga ugibamo. Med delčki poiščemo delček za naključno generiranje števila:

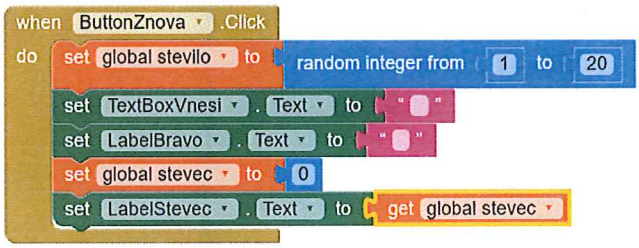
Blocks → Built-in → Math → random integer

V njega vpišemo spodnjo in zgornjo mejo, med katerima se naključno izžreba celo število.

Slika 4.4: Spremenljivka *stevilo*

S pritiskom gumba *ButtonZnova* želimo ponovno pričeti z igro. Naredimo naslednje:

- generiramo novo število za ugibanje (med 1 in 20),
- vnosno polje in labeli nastavimo na prazen tekst,
- spremenljivko stevec, ki hrani število poskusov, nastavimo na 0, in
- labelo LabelStevec nastavimo na vrednost spremenljivke.

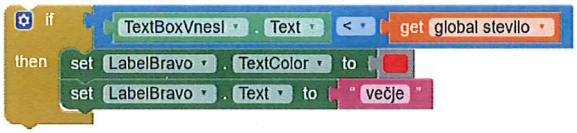


Slika 4.5: Ponastavitve label in vnosnega polja v gumbu *ButtonZnova*

Ko pritisnemo gumb *ButtonOK*, povečamo števec poskusov za 1 in spremenljivko zapišemo v labelo.

Slika 4.6: Gumb *ButtonOK* in povečamo število poskusov

Sedaj je potrebno v gumbu *ButtonOK* preveriti še, ali smo ugotovili število. Če je uporabnik zapisal manjše število, kot je ~~število, ki ga ugibamo~~ izžrebano število, izpišemo v labelo »večje« in jo obarvamo z rdečo barvo.



Slika 4.7: Ugibano število je večje od vpisanega

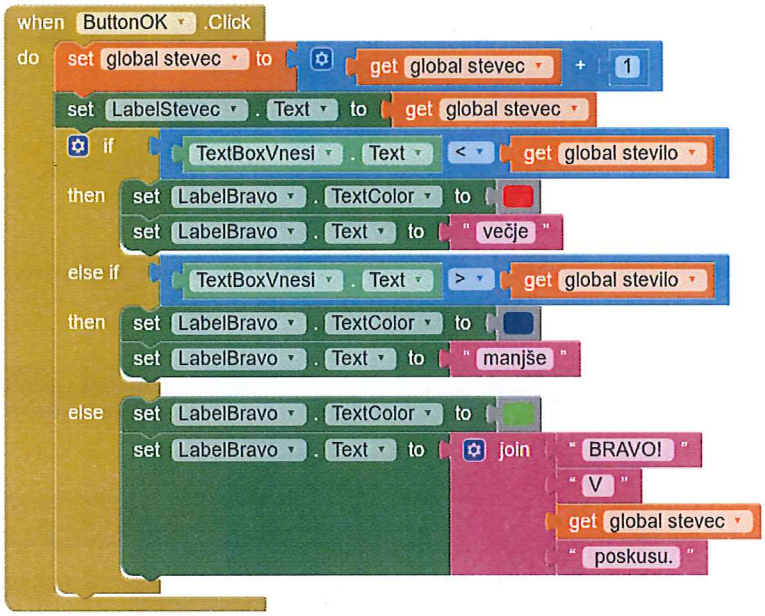
Če je uporabnik zapisal večje število, kot je število, ki ga ugibamo, izpišemo v labelo »manjše« in jo obarvamo z modro barvo.

Slika 4.8: Dodan pogoj, če je ugibano število manjše od vpisanega

V nasprotnem primeru je uporabnik vpisal pravilno število. V tem primeru v labelo izpišemo »BRAVO!« in besedilo obarvamo zeleno.

Slika 4.9: Blok else, če je ugibano število enako vpisanemu

Dodamo še podatek, v katerem poskusu je uporabnik pravilno zadel ugibano število.



Slika 4.10: Izpis števila poskusov ob uspešnem ugibanju števila

4.2 Ideje za dopolnitev aplikacije

1. Zapišite navodilo reševanja naloge (»Ugibaj števila med 1 in 20!«).
2. Za gradnik *LabelBravo* uporabite večjo pisavo.
3. Po vsakem poskusu izbrišite vsebino *TextBoxVnesi*.
4. Dopolnite aplikacijo z novo labelo, ki bo sestavljena iz predhodnega poskusa in zapisa manjše/večje, npr. »manjše od 10«.
5. Uporabite procedure za izboljšanje preglednosti aplikacije.
6. Ustvarite nov števec, ki šteje število odigranih iger.
7. Če je izbrano število manjše, naj ga aplikacija izpiše v modri barvi, drugače v rdeči barvi.

Ne ustrezno mesto v aplikaciji. Lahko na dodate gumb Navodila, ki ob kliku izpiše navodilo za reševanje naloge.

4. Dopolnite aplikacijo tako, da bo ~~namesto~~ nalogo "manjše/večje" napisalo še vrednost poskusu (npr. "manjše od 7".

8. Izpišite še najmanjše, največje in povprečno število poskusov med vsemi odigranimi igrami.

5 RAČUNALNIŠKE IGRE

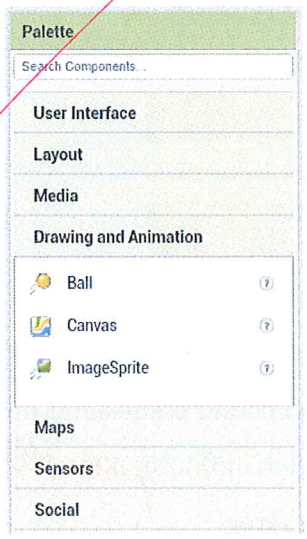
V prejšnjih poglavjih smo naredili nekaj aplikacij z uporabniškim vmesnikom, ki so vsebovale gradnike, kot so npr. labele, gumbi, vnosna polja ipd.

Posebno skupino aplikacij, ki so primarno namenjene zabavi, imenujemo računalniške igre. Za izdelavo računalniške igre pa potrebujemo dodatne gradnike. V tem poglavju bomo tako spoznali platno, po katerem lahko rišemo in premikamo objekte.

Sodobne mobilne naprave vsebujejo tudi številne pripomočke. Tak primer so senzorji. Poglejmo si gradnike, ki jih bomo potrebovali za sestavo naše igre.

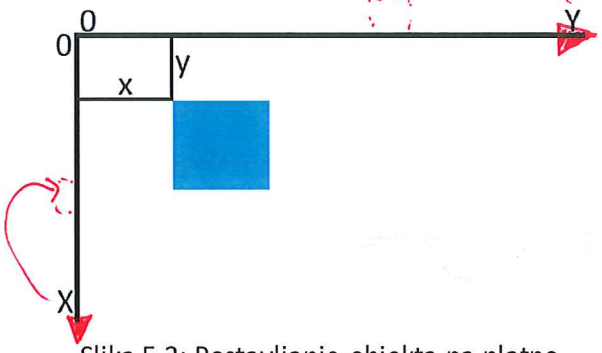
5.1 Platno in grafični objekti

Platno (Canvas) je pravokotna plošča, po kateri lahko s pomočjo gradnikov rišemo in premikamo objekte. Najdemo ga v paleti pod skupino Risanje in animacija (Drawing and Animation).



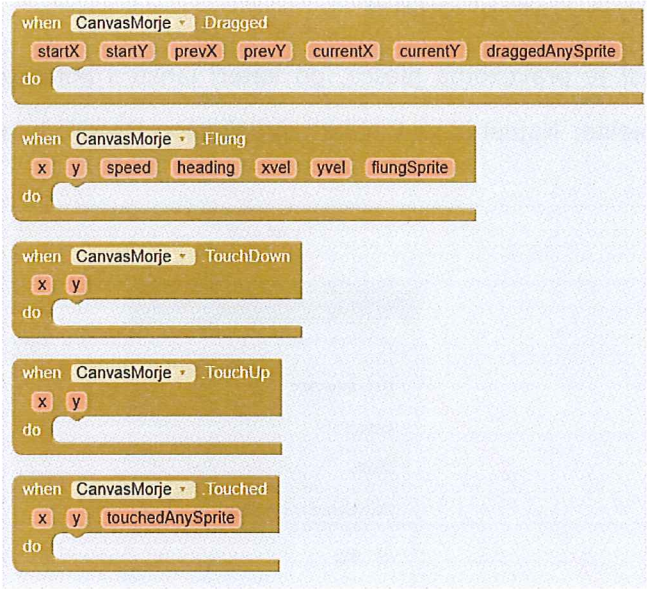
Slika 5.1: Skupina gradnikov Risanje in animacija (Drawing and Animation)

Platno je v bistvu dvodimenzionalno polje in vsaka točka na platnu ustreza paru (X,Y), kjer je X oddaljenost od levega roba, Y pa oddaljenost od zgornjega roba platna. Par (0,0) je torej zgornji levi kot platna. Če pa želimo postaviti objekt na platno, se držimo pravila, kot je predstavljeno na naslednji sliki.



Slika 5.2: Postavljanje objekta na platno

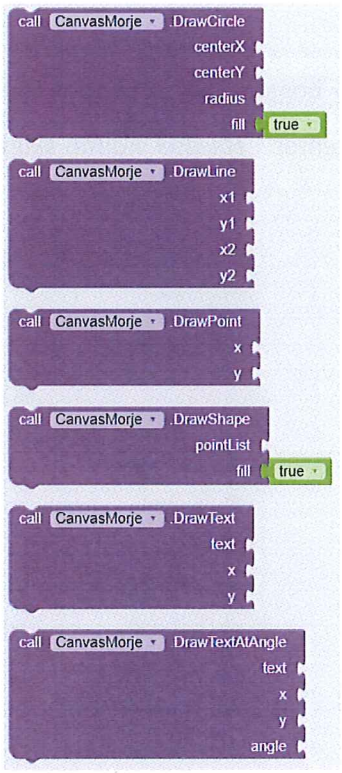
Ker uporabljamo mobilne naprave, je platno občutljivo na dotik. Dogodki platna in objektov nam tako povedo, kje smo se dotaknili platna.



Slika 5.3: Dogodki gradnika Platno

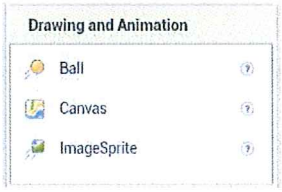
Poleg objektov, ki jih lahko po platnu premikamo, ima platno veliko možnosti za risanje in pisanje. Slika prikazuje nekaj teh možnosti:

- DrawCircle – narišemo krog
- DrawLine – narišemo črto
- DrawPoint – narišemo točko
- DrawText – napišemo besedilo
- DrawTextAtAngle – narišemo besedilo pod kotom
- Clear – izpraznimo platno
- DrawArc – narišemo lok
- itd.



Slika 5.4: Metode gradnika Platno

V isti skupini Risanje in animacija (Drawing and Animation) najdemo še **Žogico** (Ball).



Slika 5.5: Gradniki Ball, Canvas in ImageSprite

Žogica je grafični objekt, ki ga lahko premikamo po platnu ter reagira na trke z drugimi elementi ali s platnom samim (npr. rob platna).

Sličica (*ImageSprite*) je podoben element žogi in se lahko premika po platnu. Medtem ko je žogica vedno enakega videza, lahko za sličico izberemo poljuben grafični prikaz. Sličico je mogoče spreminjati (povečevati/pomanjševati itd.).

Žogica (*Ball*) in Sličica (*ImageSprite*) sta edini komponenti, ki se lahko premikata po platnu.

5.2 Senzorji

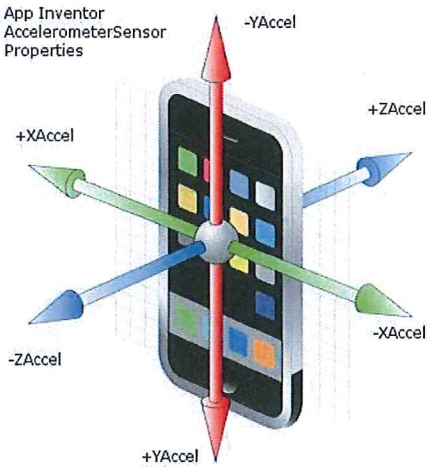
Mobilne naprave vsebujejo bogat nabor senzorjev in njihova uporaba je omogočena tudi znotraj okolja MIT App Inventor. Na sliki vidimo skupino gradnikov **Senzorji** (Sensors) in različne tipe senzorjev.

Slika 5.6: Skupina gradnikov Senzorji

Poglejmo si primer senzorja. **Pospeškometer** (AccelerometerSensor) je senzor, s katerim zaznavamo nagib mobilne naprave. Deluje s pomočjo merjenja sil oz. tresljajev (v m/s^2) v posameznih oseh X, Y in Z. Če pogledamo sliko, so to naslednji trije parametri z vrednostmi:

- xAccel:
  - 0: mobilna naprava miruje na ravni površini
  - večja od 0: mobilni napravi dvignemo levi rob
  - manjša od 0: mobilni napravi dvignemo desni rob
- yAccel:
  - 0: mobilna naprava miruje na ravni površini
  - večja od 0: mobilni napravi dvignemo spodnji rob
  - manjša od 0: mobilni napravi dvignemo zgornji rob
- zAccel:
  - 0: mobilna naprava miruje pravokotno na tla

- 9.8: mobilna naprava miruje vzporedno s tlemi z zaslonom navzdol
- -9.8: mobilna naprava miruje vzporedno s tlemi z zaslonom navzgor
- če mobilno napravo premikamo navzgor ali navzdol, se vrednost zAccel spreminja



Slika 5.7: Pridobivanje podatkov o nagibu s pomočjo pospeškometra [6]

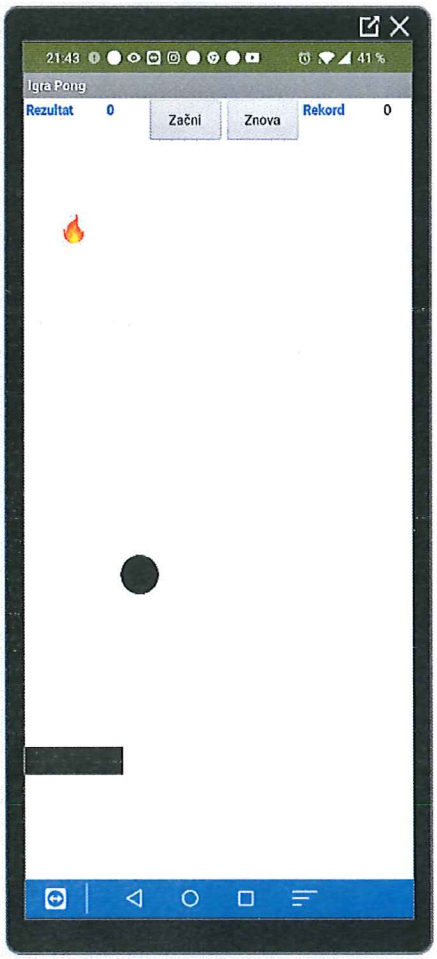
Dogodke in lastnosti pospeškometra vidimo na spodnji sliki.

Slika 5.8: Dogodki in lastnosti pospeškometra

5.3 Računalniška igra Pong

Navodilo

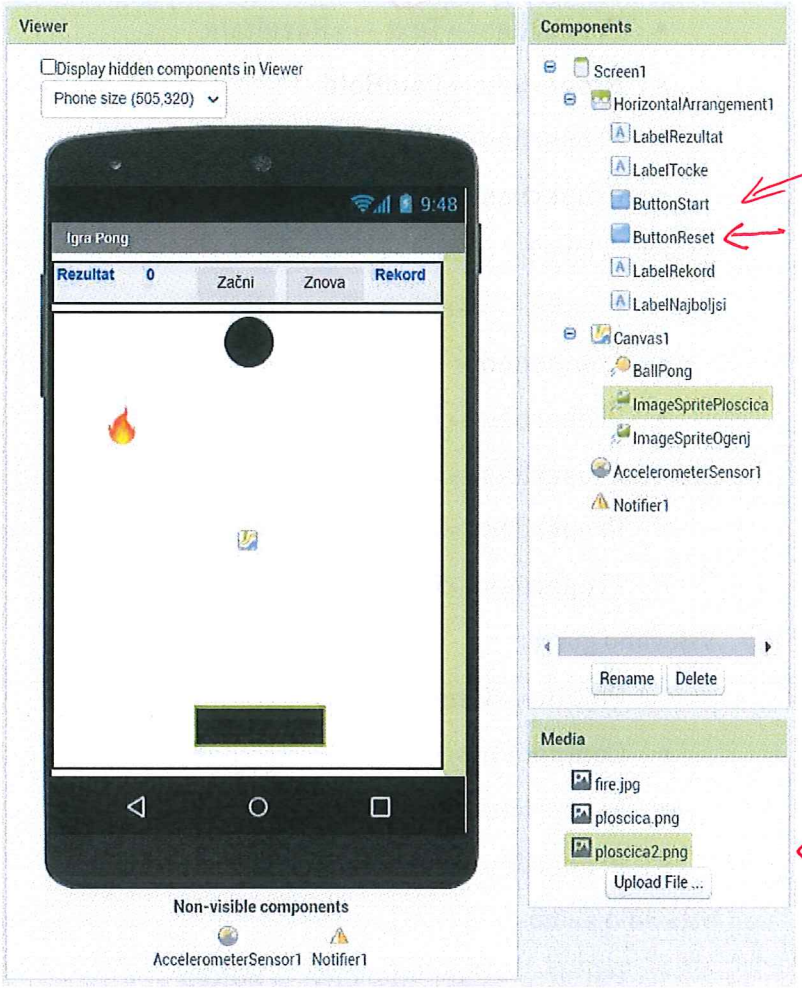
Cilj igre Pong je, da s pomočjo premikanja ploščka preprečimo, da nam žogica uide v spodnji del ekrana, medtem ko je levo, desno in zgoraj omejena z zidom. Ko se žogica dotakne ploščka, se odbije. Za vsak odbitek pridobimo po 1 točko. Na vrhu igralne površine je tudi ogenj. Če ga zadenemo, si prislužimo dodatnih 5 točk. Igra se prične s pritiskom na gumb *Začni*. Če igro končamo, jo lahko ponovno pričnemo igrati s pritiskom na gumb *Znova*. Igra se konča, ko se žogica dotakne spodnjega roba zaslona oz. nam je žogica ušla. V igri podprimo tudi beleženje najboljšega rezultata (rekord).



Slika 5.9: Igra Pong

Videz aplikacije

Na delovno površino prenesemo gradnike, kot vidimo na sliki.



Slika 5.10: Gradniki igre Pong

Podajmo razlago gradnikov.

- Screen
  - Zaslonu nastavimo, da orientacije zaslona ni mogoče spremeniti – če obrnemo telefon, se aplikacija ne bo odzvala in obrnila. Postavimo se na gradnik Screen1 ter nastavimo pokončno postavitev mobilne naprave.
    - **Properties → Screen Orientation → Portrait**
- Vstavimo HorizontalArrangement1.
  - **Properties → Width → »Fill parent«**
  - Nastavimo širino poravnave na celotno širino zaslona.
    - **Properties → width → Fill parent**
  - Vstavimo labelo.

- Palette → User interface → Label
- Components → rename → LabelRezultat
- Properties → Text → »Rezultat«
- Properties → FontBold
- Properties → Width → »20 percent«
- Properties → TextColor → »Blue«

- Vstavimo labelo.
  - Palette → User interface → Label
  - Components → rename → LabelTocke
  - Properties → Text → »0«
  - Properties → FontBold
  - Properties → Width → »10 percent«
  - Properties → TextColor → »Blue«
- Vstavimo gumb.
  - Palette → User interface → Button
  - Components → rename → ButtonStart
  - Properties → Text → »Start«
  - Properties → Width → »20 percent«
- Vstavimo gumb.
  - Palette → User interface → Button
  - Components → rename → ButtonReset
  - Properties → Text → »Znova«
  - Properties → Width → »20 percent«
- Vstavimo labelo.
  - Palette → User interface → Label
  - Components → rename → LabelRekord
  - Properties → Text → »Rekord«
  - Properties → FontBold
  - Properties → Width → »20 percent«
  - Properties → TextColor → »Blue«
- Vstavimo labelo.
  - Palette → User interface → Label

- Components → rename → LabelNajboljsi
- Properties → Text → »0«
- Properties → FontBold
- Properties → Width → »10 percent«
- Properties → TextColor → »Blue«

- Vstavimo platno.
  - Vstavimo platno (Canvas). Platno razširimo čez cel preostanek zaslona.
    - Palette → Drawing and Animation → Canvas
    - Properties → width → Fill parent
    - Properties → height → Fill parent
- Vstavimo žogico (*Ball*).
    - Palette → Drawing and Animation → Ball
    - Components → rename → BallPong
  - Nastavimo začetno smer gibanja (*Heading*). Podatek predstavlja kot, ki je podan v stopinjah.
    - Properties → Heading → -40
  - *Interval* je podatek v milisekundah, ki pove, kako hitro se osveži premik žogice.
    - Properties → Interval → 50
  - Velikost žogice nastavimo z lastnostjo Polmer (*Radius*).
    - Properties → Radius → 20
  - Hitrost premika (lastnost *Speed*) pove, za koliko točk na zaslonu se ob vsakem premiku prestavi žogica.
    - Properties → Speed → 5
  - Lastnosti *X* in *Y* predstavljata začetni položaj žogice. Če nastavimo X in Y na 0, pomeni, da se bo žogica pojavila v zgornjem levem kotu platna.
    - Properties → X → 140
    - Properties → Y → 0
- Vstavimo plošček za odbijanje (*ImageSprite*).
    - Palette → Drawing and Animation → ImageSprite
    - Components → rename → ImageSpritePloscica
  - Nastavimo smer gibanja.

- Properties → Heading → 0

- Lastnost Interval v milisekundah nastavimo na 100.
  - Properties → Interval → 100
- Izberemo sliko ploščka (Picture).
  - Podolgovat črn plošček narišemo v aplikaciji Slikar (PaintBrush). Sliko shranimo z imenom, npr. »ploscek.png«.
  - Properties → Picture → upload file → »ploscek.png«
  - Properties → width → 100
  - Properties → height → 25
- Lastnost Hitrost (Speed) nastavimo na 0.
  - Properties → Speed → 0
- Položaj ploščka (lastnosti X in Y) pa nastavimo na vrednost, ki je odvisna od velikosti zaslona.
  - Properties → X → 114
  - Properties → Y → 300

- Vstavimo ogenj (ImageSprite).
  - Za dodatne točke lahko na platno namestimo majhen element (ogenj). Kot je že bilo omenjeno, ko zadenemo ta element z žogico, pridobimo dodatne točke v igri.
    - Palette → Drawing and Animation → ImageSprite
    - Components → rename → ImageSpriteOgenj
  - Nastavimo smer gibanja.
    - Properties → Heading → 0
  - Velikost elementa (ognja) prilagodimo tako, da ni prevelik.
    - Properties → width → 40
    - Properties → height → 40
  - Nastavimo osveževanje sličice.
    - Properties → Interval → 100
  - Izberemo sliko.
    - Sliko ognja poiščemo na internetu in jo shranimo na računalnik (npr. ogenj.png).
    - Properties → Picture → upload file → »ogenj.png«

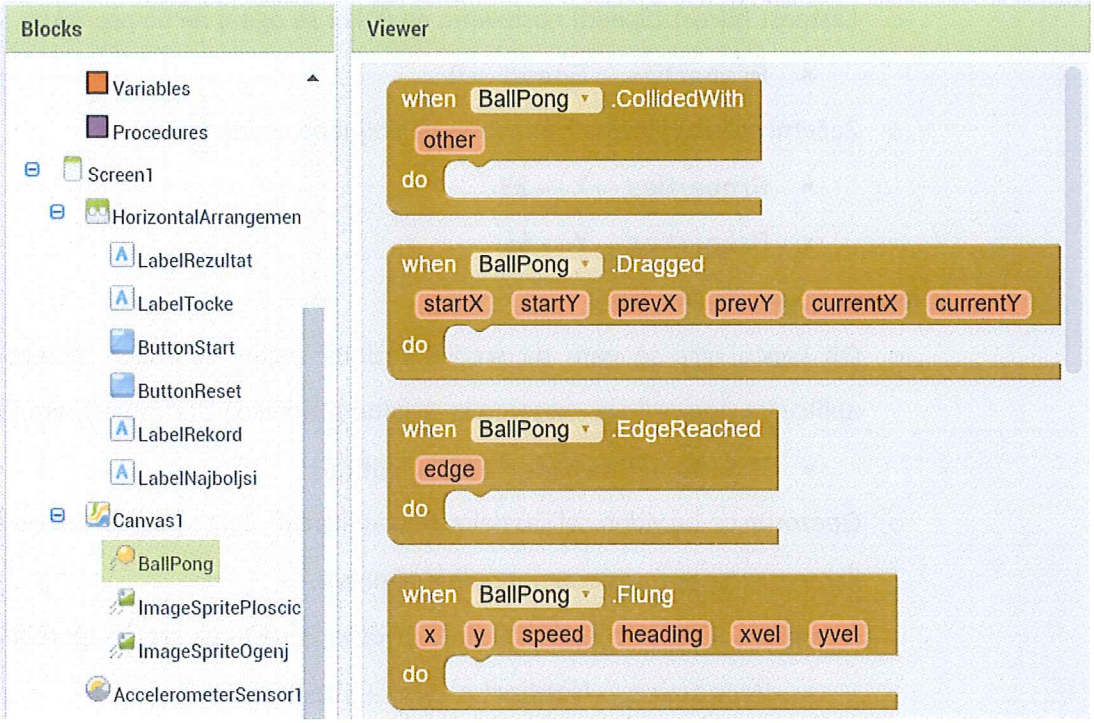
- Lastnost hitrost (Speed) nastavimo na 0, ker se ogenj ne bo premikal.
  - **Properties → Speed → 0**
- Začetni položaj ploščka nastavimo na poljubno vrednost.
  - **Properties → X → 33**
  - **Properties → Y → 66**

- Vstavimo Opozorilo (*Notifier*).
  - Ob koncu igre se nam na sredini zaslona pojavi sporočilo »Konec«. V aplikacijo Opozorilo prenesemo iz skupine gradnikov Uporabniški vmesnik.
    - **Palette → User interface → Notifier**
  - Opozorilo ne bo vidno cel čas delovanja aplikacije. Zato se pojavi kot nevidna komponenta pod zaslonom mobilne naprave.
  - Nastavimo dolžino prikaza opozorila (*NotifierLength*) na kratko sporočilo.
    - **Properties → NotifierLength → short**
- Vstavimo Pospeškometer (*AccelerometerSensor*).
  - Plošček bomo po zaslonu premikali z nagibanjem mobilne naprave, torej z uporabo pospeškometra.
  - Pospeškometer prenesemo v aplikacijo. Pojavil se bo kot **nevidna komponenta** pod zaslonom mobilne naprave (podobno kot Notifier).
    - **Palette → Sensors → AccelerometerSensor**
  - Nastavimo natančnost zaznavanja pospeškometra.
    - **Properties → MinimumInterval → 400**

Delovanje aplikacije

Najprej bomo sprogramirali delovanje žogice. Vse akcije, ki jih ima gradnik Žogica, vidimo na seznamu, ki ga odpremo iz seznama gradnikov aplikacije. Na sliki tako najdemo naslednje dogodke:

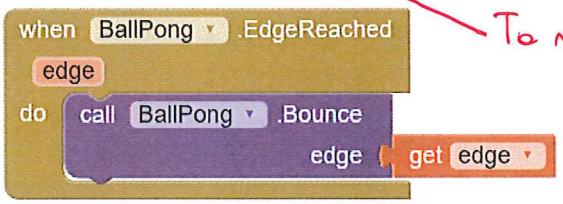
- CollidedWith – objekt žogice trči v drug objekt
- Dragged – objekt žogice počasi vlečemo oz. premikamo po zaslonu
- EdgeReached – objekt žogice zadene rob platna
- Flung – objekt žogice na hitro premaknemo oz. podrsamo



Slika 5.11: Delčki gradnika *Ball*

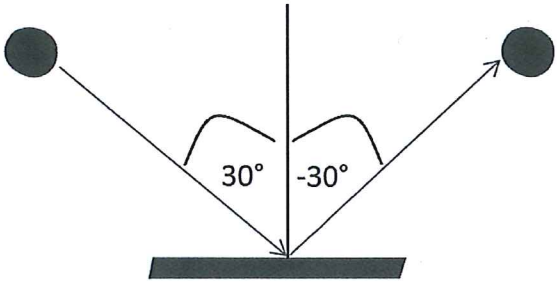
Gradnik *BallPong* ima še več dogodkov (rjave barve), metod (vijolične barve) in lastnosti (zelene barve). Dogodki lahko stojijo sami zase, medtem ko je uporaba metod in lastnosti vezana na uporabo dogodkov in jih lahko uporabimo samo znotraj dogodka.

Sprogramirajmo, kaj se zgodi, ko žogica zadene rob aplikacije. Da se žogica odbije od zidu (rob ali ovira na platnu), uporabimo dogodek *EdgeReached*. Znotraj delčka *BallPong.EdgeReached* pa bomo uporabili metodo *Bounce* (najdemo jo prav tako v gradniku *Ball*) za odboj od zidu. V metodo je potrebno na desni strani pripeti vrednost, kako naj se žogica odbije. Če želimo odboj pod enakim kotom, kot je žogica priletela v zid, potem uporabimo vrednost *edge* iz dogodka *EdgeReached*. Poglejmo si to na sliki.



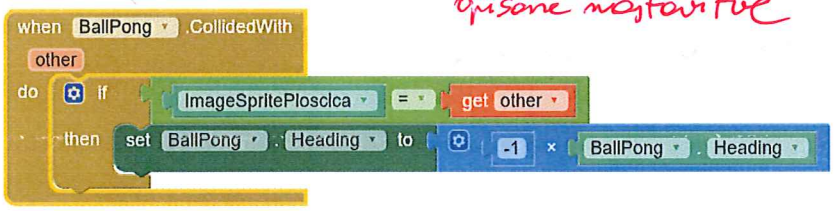
Slika 5.12: Odboj žogice od roba platna

Žogica pa se ne odbija samo od roba platna, ampak tudi od drugih elementov. V naši aplikaciji bomo žogico odbijali s ploščkom *ImageSpritePloscica*. Odboj mora biti naraven, zato je potrebno poznavanje fizike odboja. Poglejmo si spodnji primer.



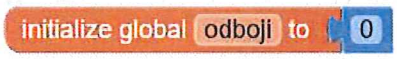
Slika 5.13: Pravilni odboj žogice

Žogici nastavimo kot pri odboju tako, da damo lastnosti *Heading* novo vrednost, ki je nasprotna kotu, ki je bil pred odbojem. Staro vrednost kota torej pomnožimo z –1. Dogodek, ki spremlja odboj žogice, je *CollidedWith* in v njem lahko preverimo, ob kaj je žogica trčila. To naredimo tako, da primerjamo gradnik *ImageSpritePloscica* in objekt *other*, ki nosi informacijo o tem, s čim je žogica trčila. Zapisano prikazuje naslednja slika.



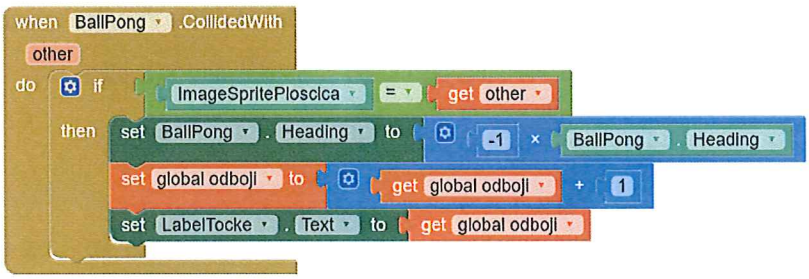
Slika 5.14: Odboj žogice v nasprotno smer

V igri želimo šteti število odbojev od ploščice. Ustvarimo si spremenljivko:



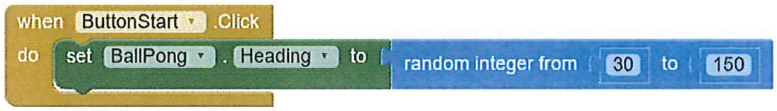
Slika 5.15: Spremenljivka za štetje odbojev

Vsakič, ko se žogica dotakne ploščice, štejemo število odbitkov. Uporabimo torej spremenljivko odboji in ji prištejemo 1. Hkrati zapišemo trenutno število odbojev v labelo (delček *LabelTocke.Text*), da jih vidi tudi uporabnik.



Slika 5.16: Štetje odbojev žogice od ploščice

V lastnostih smo žogici začetni kot in s tem smer gibanja nastavili na –40 (lastnost Heading). To pomeni, da se bo žogica vedno pričela gibati v isto smer. Ker želimo v našo igro dodati naključnost, zapišemo, da naj se ob pritisku gumba *ButtonStart* nastavi smer gibanja naključno (npr. med 30 ° in 150 °).



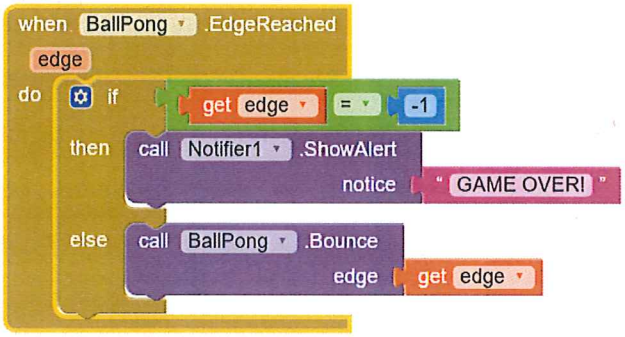
Slika 5.17: Naključna smer gibanja žogica

Konec igre je takrat, ko žogica doseže spodnji del ekrana. Dogodek *EdgeReached*, ki preveri, ali se je žogica dotaknila roba platna, je sprogramiran tako, da ob stiku s poljubnim robom odbije žogico. Sedaj bomo to spremenili in preverili, ali se je žogica dotaknila spodnjega roba platna. V tem primeru igro zaustavimo in izpišemo »Konec igre!« na sredino zaslona (uporabimo gradnik *Notifier*). V primeru, če se dotakne ostalih robov, pa žogico odbijemo. Sliši se, kot da bo potrebno uporabiti pogoj.

Še pred tem si poglejmo, kaj nosi parameter *edge*, ki ga dobimo ob uporabi dogodka *EdgeReached*. Ta spremenljivka vrne številsko vrednost, ki predstavlja rob, ki ga je zadela žogica. Vrednosti parametra *edge* so lahko številke od –4 do 4 in predstavljajo smeri neba:

- 1: sever (vrh mobilne aplikacije)
- 2: severovzhod (zgornji desni kot)
- 3: vzhod (desna stran mobilne aplikacije)
- 4: jugovzhod (spodnji desni kot)
- -1: jug (dno mobilne aplikacije)
- -2: jugozahod (spodnji levi kot)
- -3: zahod (leva stran mobilne aplikacije)
- -4: severozahod (zgornji levi kot)

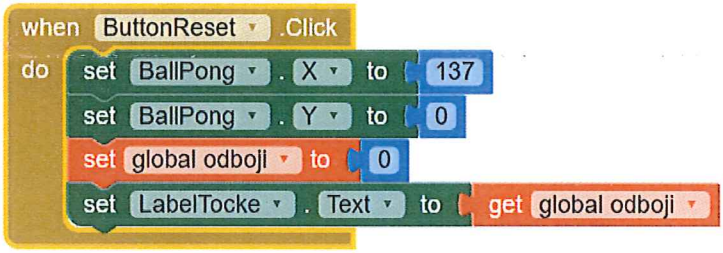
Spodnji rob zaslona tako predstavlja –1. Sedaj lahko zapišemo znotraj dogodka *EdgeReached* pogoj, ki preveri, katerega roba se je dotaknila žogica.



Slika 5.18: Preverjanje roba, ki se ga žogica dotakne

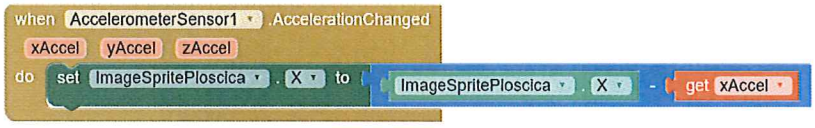
Recimo, da smo sedaj končali igro, žogica se je dotaknila spodnjega roba zaslona. Želimo pričeti z novo igro. Razmisliti moramo, kaj je potrebno narediti, da lahko pričnemo z novo igro. Tukaj je nekaj navodil za gumb *ButtonReset*:

- Žogico želimo vrniti na vrh zaslona.
- Spremenljivko *odboji*, ki vodi število odbojev, vrnemo na 0.
- Labelo, ki uporabniku prikaže število točk, postavimo na vrednost spremenljivke *odboji*.



Slika 5.19: Funkcionalnost gumba *Znova* v igri Pong

Plošček bi lahko premikali z drsanjem po zaslonu. Imamo pa tudi možnost, da z nagibom mobilne naprave kontroliramo premik ploščka. V vseh mobilnih napravah je namreč nameščen pospeškometer, ki zaznava nagibe v vse smeri: naprej/nazaj, levo/desno in gor/dol. Količine, ki jih zaznava, so pospeški. Poglejmo si dogodek *AccelerationChanged* za senzor *AccelerometerSensor1* na sliki spodaj.

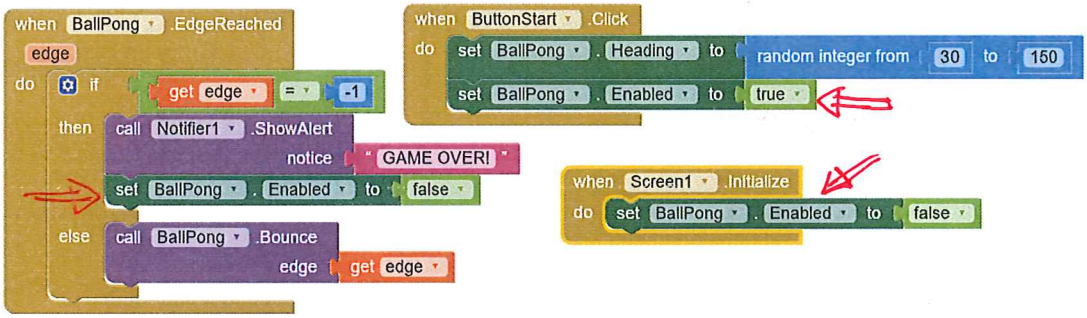


Slika 5.20: Dogodek *AccelertationChanged*

Dogodek se sproži ob spremembi nagiba mobilne naprave. Kot parametre pa ima xAccel, yAccel in zAccel. Uporabimo pospešek xAccel (nagib levo/desno) za nov položaj ploščka na zaslonu tako, da staremu položaju X odštejemo premik, ki ga zaznamo s pospeškometrom. Ko se igra konča, lahko opazimo, da se žogica še vedno premika. Potrebno jo je onemogočiti, da se ne bo več premikala. Poiščemo lastnost *Enabled* za žogico. Lastnost *Enabled* nastavimo na treh mestih:

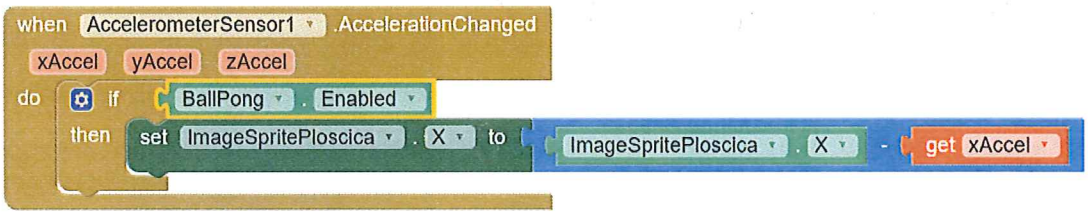
- Ko se igra prikaže na zaslonu mobilne naprave, žogico onemogočimo (dogodek Initialize).
- Ko pritisnemo gumb (dogodek *Click* za gumb *ButtonStart*), bomo omogočili žogico.
- Ko izgubimo (dogodek *EdgeReached*), pa žogico onemogočimo.

Zgoraj zapisano vidimo na spodnji sliki.



Slika 5.21: Nastavitev lastnosti *Enabled* za žogico

Ko je igra končana, premikanje ploščka ni potrebno. V dogodek *AccelerationChanged* dodajmo pogoj, ki bo preverjal, ali je žogica ustavljena.



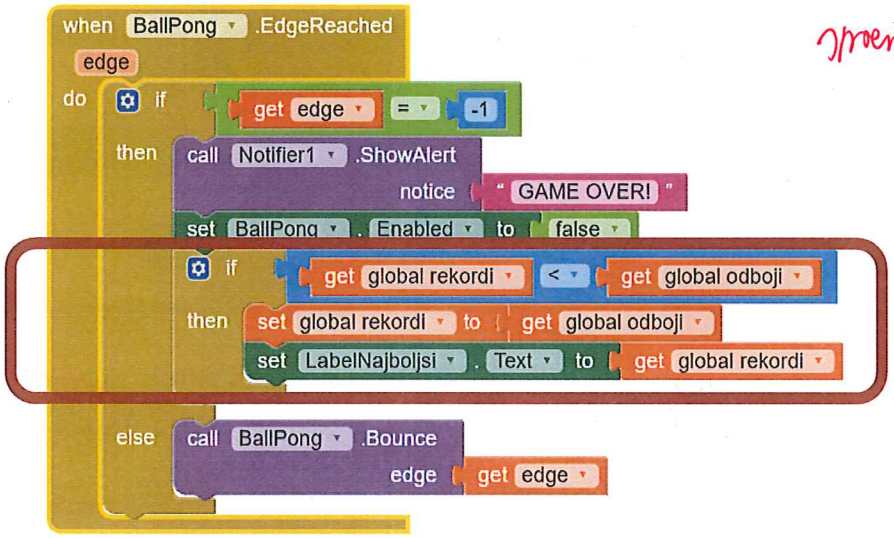
Slika 5.22: Ustavimo premikanje ploščka, če je ustavljena tudi žogica

Naredimo še spremenljivko, ki bo hranila najboljši rezultat.

Slika 5.23: Spremenljivka *rekord*

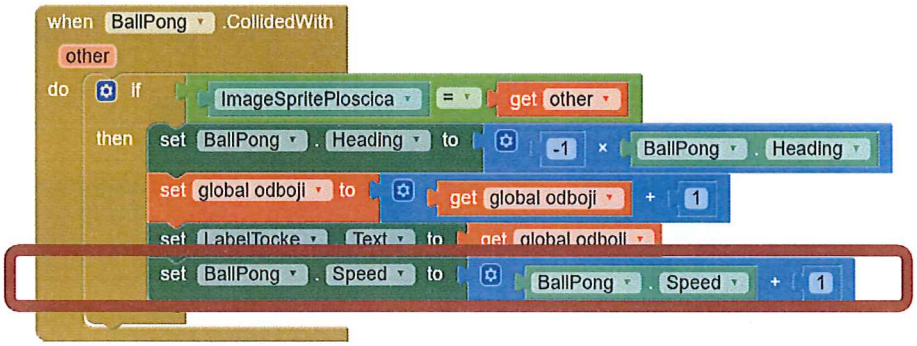
Če je spremenljivka, ki hrani število odbojev za zadnjo igro, višja od rekorda, potem imamo nov rekord. V tem primeru nastavimo spremenljivko rekord na vrednost spremenljivke

odboji. Nov najboljši rezultat prepišemo v labelo *LabelNajboljsi*. Vrednost odbojev preverimo, ko je konec igre (v dogodku *EdgeReached*).



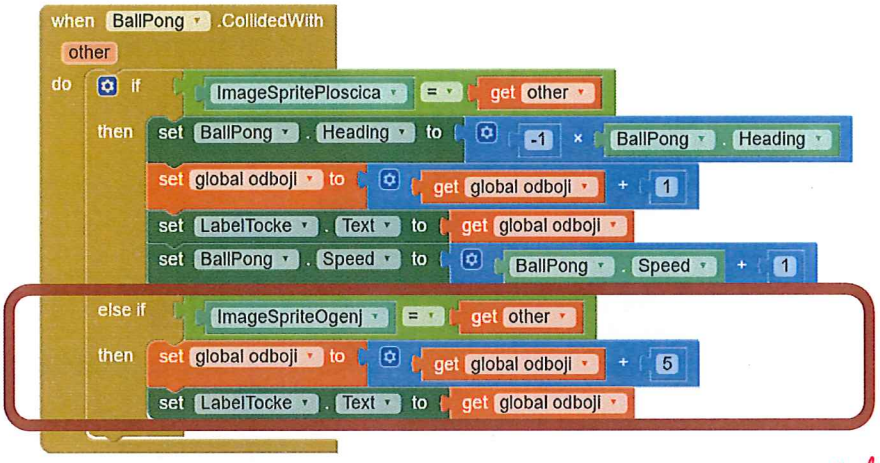
Slika 5.24: Dopolnjena koda za konec igre s preverjanjem najboljšega rezultata

Zahtevnost igre lahko povečamo na različne načine. Najprej bomo povečevali hitrost potovanja žogice. Z vsakim odbojem (dogodek *CollidedWith*) povečamo hitrost za 1. Uporabimo lastnost *Speed*.



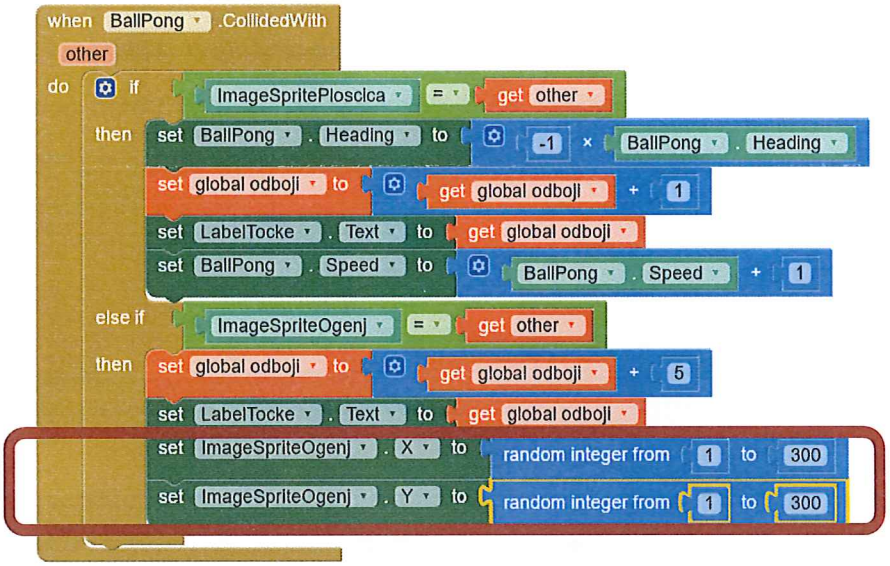
Slika 5.25: Ob vsakem odboju povečamo hitrost premikanja žogice

Pri oblikovanju videza aplikacije smo med komponente dodali ogenj kot sličico *ImageSpriteOgenj*. Uporabimo ga. Ko se bo žogica zaletela vanj (dogodek *CollidedWith*), dodajmo k trenutnemu številu točk dodatnih 5 ter izpišimo trenutno stanje točk v labelo *LabelTocke*.



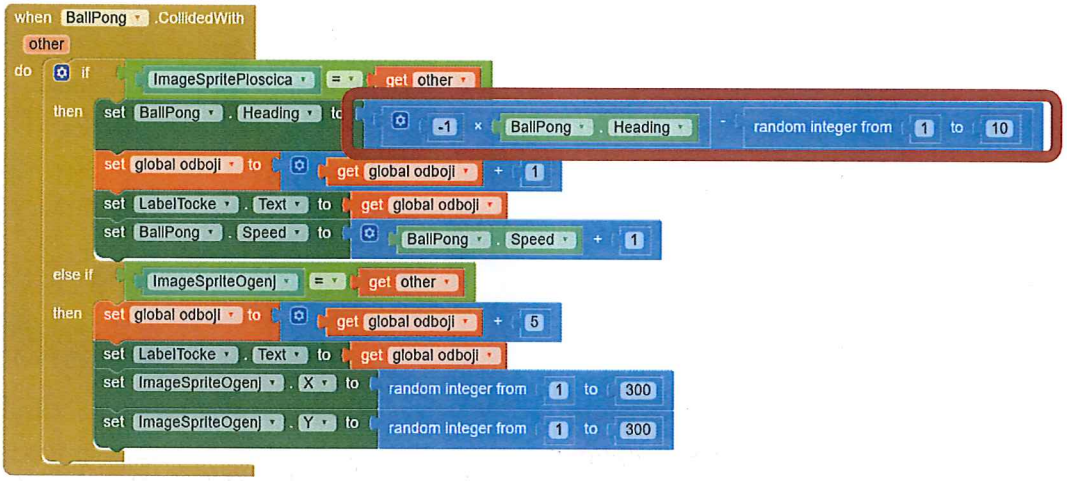
Slika 5.26: Ogenj prinese dodatnih 5 točk

Ko zadenemo ogenj, se ogenj prestavi na naključen položaj. Lastnosti X in Y gradnika *ImageSpriteOgenj* nastavimo na naključno vrednosti med 1 in 300.



Slika 5.27: Naključni premik ognja

Žogica potuje vedno po isti poti. Dodamo še nekaj naključnosti v njeno pot.

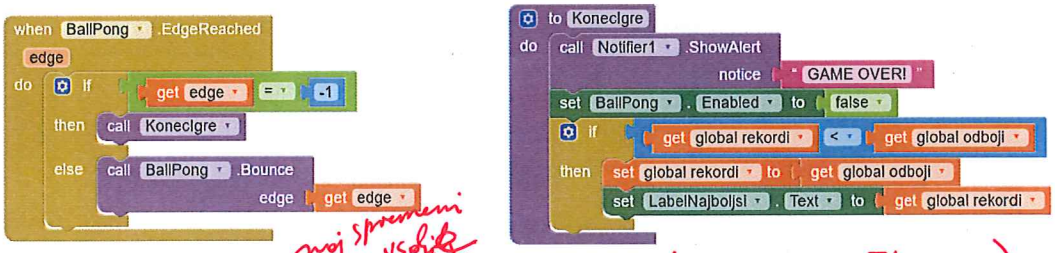


Slika 5.28: Naključni premik žogice ob dotiku ploščka

zdaj na novi strani!

5.3.1 Ideje za dopolnitev aplikacije

1. Dodajte še eno sličico, podobno ognju, ki pa bo prinesla k rezultatu 10 točk.
2. Preučite spodnjo kodo. Razložite, kaj smo naredili v pogoju znotraj delčka *BallPong.EdgeReached*. Ali se vam zdi koda bolj berljiva?



naj spremeni vsakič!

3. Položaj sličice ognja se ~~prestavi~~ po dveh sekundah. (Nomig: ...Timer...)
4. Velikost žogice se med ~~delovanjem~~ spreminja.
5. Velikost sličice ognja se med ~~delovanjem~~ spreminja.

igranjem

5.4 Računalniška igra Morski pes

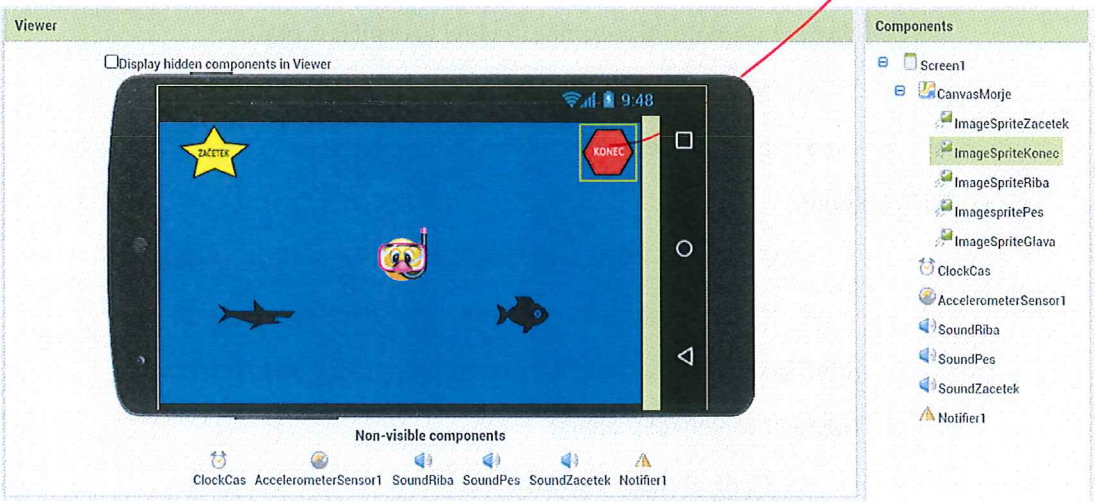
Navodilo

Naredili bomo igro, kjer morski pes lovi smeška z dihalko. Smeško se mu izmika in pri tem pobira ribice. Beležijo se točke, za vsako pobrano ribico pridobimo 10 točk. Ko ribico ulovimo, se nova naključno pojavi na drugem mestu. Igra se prične s pritiskom na zvezdo »Začetek«, konča pa se, ko nas morski pes ulovi. Če želimo, jo lahko predčasno končamo s pritiskom na rdeči šestkotnik »Konec«.

Slika 5.29: Igra »Morski pes«

Videz aplikacije

Na delovno površino prenesemo gradnike, kot vidimo na sliki.



Slika 5.30: Gradniki igre »Morski pes«

Podajmo nastavitev gradnikov mobilne igre:

TU MANJKA sporočilo/razlaga, da ne bomo (ne moremo) uporabljali ozadij, ampak da bomo vse lesedilo pisali na platno!

- Screen
  - Postavitev zaslona ~~obrnimo~~ nastavimo tako, da bo vedno ležeče.
    - **Properties → Screen Orientation → Landscape**
- Canvas
    - **Palette → Drawing and Animation → Canvas**
  - Platno preimenujmo v »CanvasMorje«.
    - **Components → rename → CanvasMorje**
  - Barvo ozadja nastavimo na modro.
    - **Properties → BackgroundColor → Blue**
  - Razširimo platno čez cel zaslon.
    - **Properties → width → Fill parent**
    - **Properties → height → Fill parent**
- Sličica Začetek
    - **Palette → Drawing and Animation → ImageSprite**
    - **Components → rename → ImageSpriteZacetek**
  - Rumeno zvezdico narišemo v orodju Slikar (npr. »start.png«).
  - Nastavimo velikost sličice (lastnosti width in height na 50).
    - **Properties → width → 50**
    - **Properties → height → 50**
  - Sličico povežemo s sliko, ki smo jo narisali v Slikarju.
    - **Properties → Picture → upload file → »start.png«**
  - Nastavimo položaj zvezdice na levi zgornji kot platna.
    - **Properties → X → 0**
    - **Properties → Y → 0**
- Sličica Konec
    - **Palette → Drawing and Animation → ImageSprite**
    - **Components → rename → ImageSpriteKonec**
  - Rdeč šestkotnik narišemo v orodju Slikar (npr. »konec.png«).
  - Nastavimo velikost sličice.
    - **Properties → width → 50**
    - **Properties → height → 50**
  - Sličico povežemo s sliko, ki smo jo narisali v Slikarju.

        - Properties → Picture → upload file → »start.png«
    - Nastavimo položaj sličice na desni zgornji kot.
        - Properties → X → 400
        - Properties → Y → 0
- Sličica Riba
        - Palette → Drawing and Animation → ImageSprite
        - Components → rename → ImageSpriteRiba
    - Ribico poiščemo na internetu (npr. »riba.png«).
    - Nastavimo velikost sličice.
        - Properties → width → 50
        - Properties → height → 50
    - Nastavimo, kako pogosto se osveži premik sličice.
        - Properties → Interval → 20
    - Sličico povežemo s sliko z interneta.
        - Properties → Picture → upload file → »riba.png«
    - Nastavimo položaj ribe.
        - Properties → X → 318
        - Properties → Y → 150
- Sličica Morski pes
        - Palette → Drawing and Animation → ImageSprite
        - Components → rename → ImageSpriteMorskiPes
    - Ribico poiščemo na internetu (npr. »morski_pes.png«).
    - Nastavimo velikost sličice.
        - Properties → width → 75
        - Properties → height → 50
    - Lastnost Interval nastavimo na 20 milisekund.
        - Properties → Interval → 20
    - Sličico povežemo s sliko z interneta.
        - Properties → Picture → upload file → »morski_pes.png«
    - Nastavimo položaj morskega pesa.
        - Properties → X → 50
        - Properties → Y → 150

- Sličica smeška
    - Palette → Drawing and Animation → ImageSprite
    - Components → rename → ImageSpriteGlava
    - Smeška poiščemo na internetu (npr. »emoji.png«).
    - Nastavimo velikost sličice.
        - Properties → width → 50
        - Properties → height → 50
    - Sličico povežemo s sliko z interneta.
        - Properties → Picture → upload file → »emoji.png«
    - Nastavimo položaj smeška.
        - Properties → X → 200
        - Properties → Y → 90

V igri pa imamo tudi nekaj gradnikov, ki jih ne vidimo na delovni površini, a jih vseeno uporabljamo, zato jih dodamo:

- Ura
    - Components → rename → ClockCas
    - Iz skupine gradnikov Senzorji izberemo gradnik Ura (Clock), ki jo bomo uporabljali, da igro časovno omejimo. V igri bomo tako odštevali čas.
        - Palette → Sensors → Clock
- Pospeškometer
    - Iz skupine gradnikov Senzorji izberemo gradnik Pospeškometer.
        - Palette → Sensors → AccelerometerSensor
- Zvok Ribe
    - Zvok uporabimo, ko bo smeško »pobral« ribico.
    - Primeren zvok poiščemo na internetu (npr. »riba.mp3«).
    - Iz skupine gradnikov Media izberemo gradnik zvoka.
        - Palette → Media → Clock
        - Components → rename → SoundRiba
        - Properties → Source → upload file → »riba.mp3«

- Zvok MorskegaPsa
  - Zvok bomo uporabili, ko morski pes ujame smeška.
  - Primeren zvok poiščemo na internetu (npr. morski_pes.mp3).
  - Dodamo gradnik zvoka.
    - **Palette → Media → Clock**
    - **Components → rename → SoundMorskiPes**
    - **Properties → Source → upload file → »morski_pes.mp3«**
- Zvok Začetek
  - Zvok uporabimo, ko igralec pritisne na sličico Začetek.
  - Primeren zvok poiščemo na internetu (npr. »zacetek.mp3«).
  - Dodamo gradnik zvoka.
    - **Palette → Media → Clock**
    - **Components → rename → SoundZacetek**
    - **Properties → Source → upload file → »zacetek.mp3«**
- Obvestilo
  - Uporabimo ga, da izpišemo na sredini zaslona »Konec igre!«.
    - **Palette → User interface → Notifier**
  - Nastavimo ~~dolžino pojavitve obvestila~~, kolikо jaz se vidno dvedrt
    - **Properties → NotifierLength → short**

Delovanje aplikacije

Najprej si zapišimo spremenljivke, ki jih potrebujemo za delovanje igre:

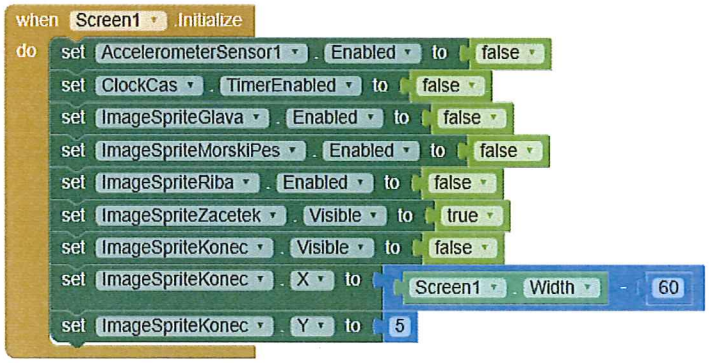
- *cas*: Spremenljivka bo odštevala čas igranja igre (v sekundah). Ko pridemo do 0, je igre konec. V primeru, ko nas morski pes ulovi, se igra konča predčasno.
- *rezultat*: Beleži število pobranih ribic.

Slika 5.31: Spremenljivki v aplikaciji »Morski pes«

Prvi dogodek zaslona, ki se izvrši, je *Initialize*. Vanj zapišemo vse, kar želimo, da se uporabniku najprej prikaže. V naši aplikaciji bomo naredili naslednje:

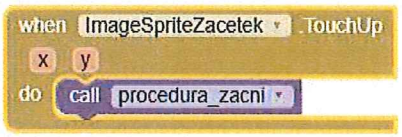
- Onemogočili gradnike (pospeškometer, ura, glava, morski pes in riba). To pomeni, da se ti objekti ne bodo premikali po zaslonu, dokler ne pritisnemo »Začetek«.
- Ko odpremo aplikacijo, je vidna sličica *ImageSpriteZacetek*, skrijemo pa sličico *ImageSpriteKonec*.
- Sličico *ImageSpriteKonec* postavimo na desni konec zaslona.

Zgornji postopek je zapisan na spodnji sliki.



Slika 5.32: Dogodek *Initialize* za zaslon *Screen1*

Ko pritisnemo sličico *ImageSpriteZacetek*, imamo opravka z dogodkom *TouchUp*. V njem pokličemo proceduro *procedura_zacni*, kot vidimo na spodnji sliki, ki pa jo moramo najprej ustvariti.

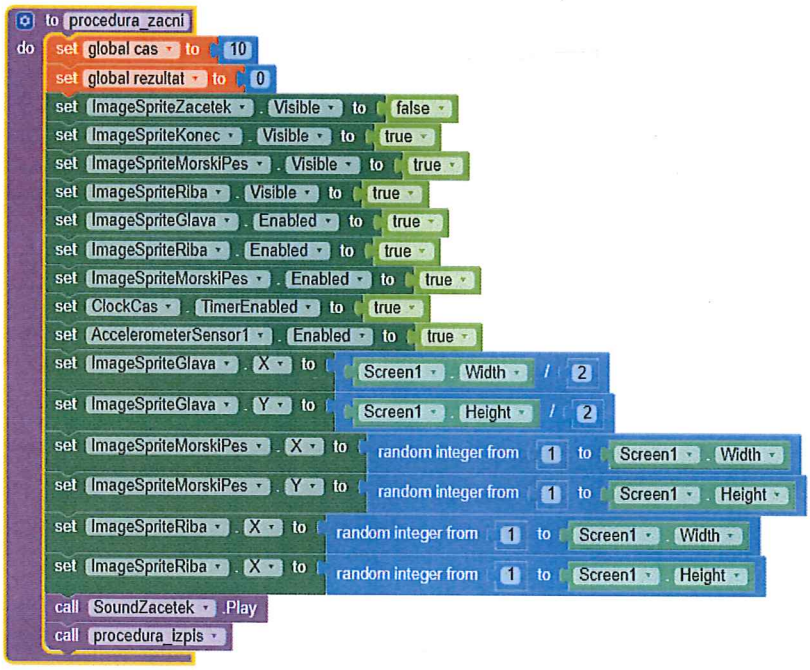


Slika 5.33: Dogodek *TouchUp* za sličico *ImageSpriteZacetek*

Procedura *procedura_zacni* ima naslednje delovanje:

- Čas igranja (spremenljivka *cas*) nastavimo na 10 sekund.
- Rezultat (spremenljivka *rezultat*) nastavimo na začetku igre na 0.
- Ko smo pritisnili sličico *ImageSpriteZacetek*, jo skrijemo (lastnost *Visible* na »false«), prikažemo pa sličico *ImageSpriteKonec* (lastnost *Visible* nastavimo na »true«).
- Prikažemo tudi sličici *ImageSpriteRiba* in *ImageSpriteMorskiPes* (lastnost *Visible* nastavimo na »true«).
- Omogočimo, da se sličice (glava, riba, morski pes) začnejo premikati po zaslonu (lastnost *Enable* nastavimo na »true«).
- Omogočimo delovanje ure in pospeškometra (lastnost *Enable* nastavimo na »true«).
- Postavimo sličico *ImageSpriteMorskiGlava* na sredino zaslona (lastnosti X in Y).
- Postavimo sličici *ImageSpriteMorskiPes* in *ImageSpriteRiba* na poljubno mesto na zaslonu (lastnosti X in Y).

Zgoraj opisana funkcionalnost je prikazana na naslednji sliki.



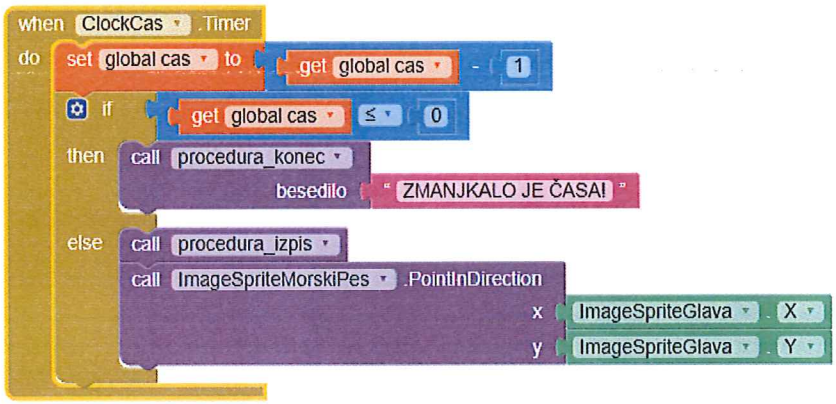
Slika 5.34: Procedura *procedura_zacni*

Ura (Clock) je gradnik, ki ga najdemo med skupino Senzorji (Sensors). S tem gradnikom tvorimo časovnik, ki v intervalih proži dogodke. Ura omogoča delčke za pretvorbe in rokovanje z različnimi časovnimi enotami.

Postopek za vodenje časa do konca igre:

- Spremenljivko časa zmanjšamo za 1.
- S pogojnim stavkom preverimo, ali je čas igre potekel.
  - V primeru, da je igralni čas potekel, pokličemo proceduro *procedura_konec* s parametrom »ZMANJKALO JE ČASA!«. Proceduro si bomo pogledali v nadaljevanju.
  - Če je ostalo še kaj tekmovalnega časa, izpišemo trenutno število točk. Tudi tukaj si naredili proceduro, ki to opravi (procedura *izpis*). Druga naloga je, da morskega psa usmerimo proti glavi.

Opisani postopek je predstavljen na spodnji sliki.



Slika 5.35: Odštevanje časa v igri

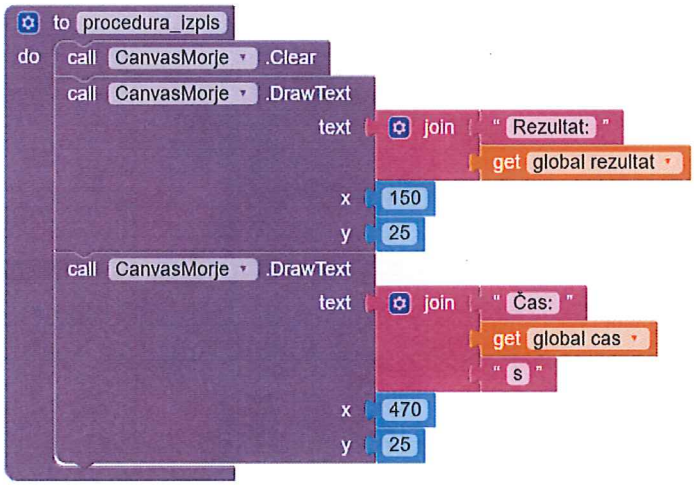
Na vrhu aplikacije želimo izpisovati rezultat in čas, ki je preostal, tako kot vidimo na spodnji sliki.

Slika 5.36: Izpis rezultata in časa

Pripravimo proceduro *izpis*, ki bo izpisala obe besedili (število trenutno doseženih točk in preostali čas igranja igre). V tej proceduri:

- Najprej izbrišemo vso vsebino platna.
- Izpišemo besedili z metodo *DrawText*, ki jo najdemo v platnu.

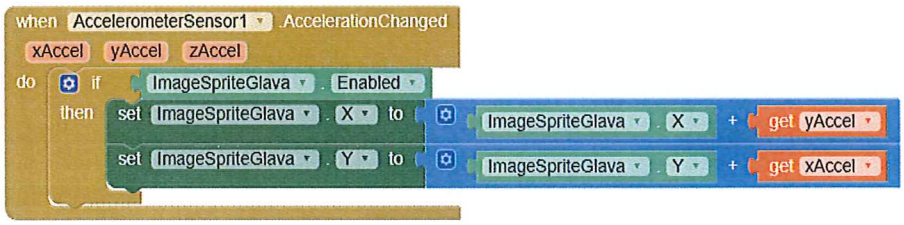
- Besedili sestavimo z delčkom *join* iz besedila (npr. »Rezultat:«) in spremenljivko *rezultat*.
- Postavimo besedili v isto vrstico (*y* pri obeh je 25), začetek besedila pa je različen (*x* pri prvem besedilu 150, pri drugem besedilu pa 470).



Slika 5.37: Izpis rezultata in časa na vrhu zaslona

Proceduro *izpis* kličemo v dogodku *Timer*, in sicer pod pogojem, da čas še ni potekel.

Naslednji korak je uporaba pospeškometra, ki bo premikal našega junaka (gradnik *ImageSpriteGlava*) po zaslonu. Z nagibanjem telefona se bo sličica junaka premikala. Naloga igralca je, da junaka s pomočjo nagiba mobilne naprave usmeri proti ribi. Pri izračunu novega položaja junaka bomo upoštevali pospeška xAccel in xyAccel. Novi položaj preračunamo le, če je gradnik junaka omogočen (smo med igranjem igre).



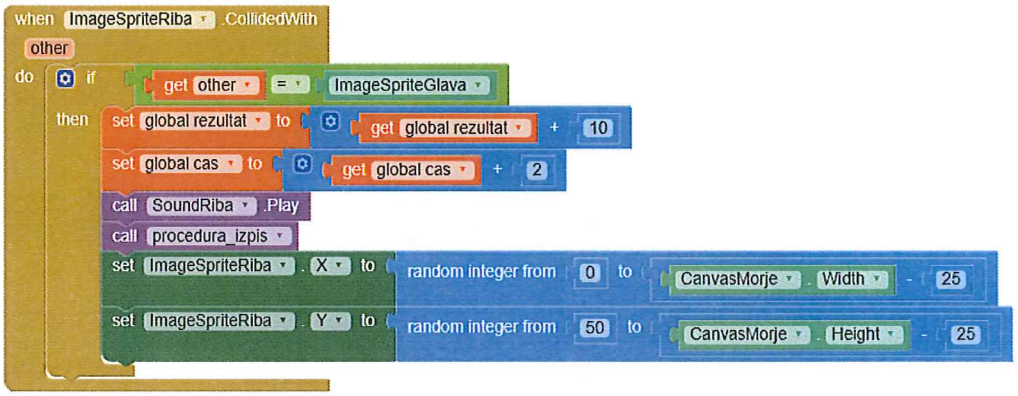
Slika 5.38: Uporaba pospeškometra za premik glavnega junaka

Na zgornji sliki bodimo pozorni, da je pospešek yAccel uporabljen pri določanju komponente *X* glave. Razlog je v ležečem zaslonu.

Sedaj se lotimo trkov. Najprej poskrbimo za trk glave in ribe. V tem primeru je postopek naslednji:

- Preverimo, ali je junak res ujel ribo (se je dotaknil ribe).
- Prištejemo 10 točk k rezultatu (spremenljivka rezultat).

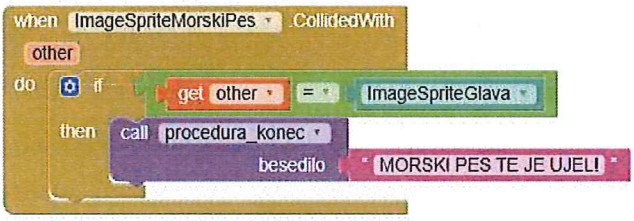
- Kot nagrado za to, da smo ulovili ribo, k času dodamo 2 dodatni sekundi.
- Predvajamo zvok, ki igralca opozori, da smo ujeli ribo.
- Izpišemo novo stanje v igri (klic procedure izpisi).
- Ribo premaknemo na nov položaj.



Slika 5.39: Dotik junaka (gradnik *ImageSpriteGlava*) in ribice

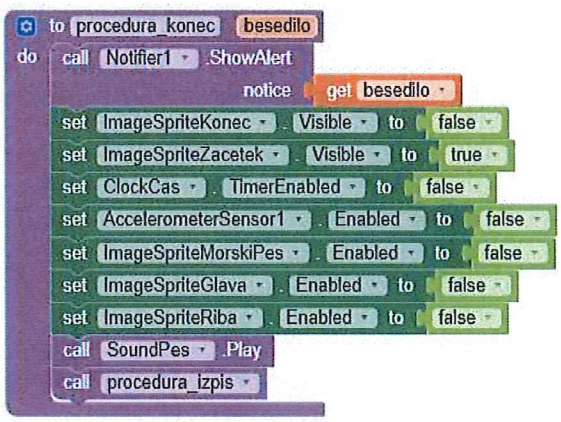
Naslednji dogodek je dotik junaka in morskega psa. Če nas je morski pes ujel, je igre konec. Na spodnji sliki tako vidimo dogodek *ColidedWith* z vsebino:

- Preverimo, ali je morski pes trčil v junaka.
- Če drži, pokličemo proceduro *procedura_konec* z besedilom »Morski pes te je ujel«.



Slika 5.40: Trk glave in morskega psa

Delovanje ob koncu igre je tako združeno v proceduri *procedure_konec*. Poglejmo si jo.



Slika 5.41: Procedura, ki se izvede ob koncu igre »Morski pes«

Na zgornji sliki vidimo naslednje akcije:

- Prikažemo obvestilo, ki pove, kako je prišlo do zaključka igre.
- Onemogočimo gumb *Konec* (v našem primeru je za gumb uporabljena sličica *ImageSpriteKonec*), ker je igra že zaključena (lastnost *Visible* nastavimo na »false«).
- Omogočimo sličico *ImageSpriteZacetek*, da lahko uporabnik ponovno začne igrati igro (lastnost *Visible* nastavimo na »true«).
- Onemogočimo še naslednje objekte na platnu (lastnost *Enable* nastavimo na »false«):
  - uro
  - pospeškometer
  - morskega psa
  - glavo
  - ribo
- Predvajamo zvok morskega psa.
- Osvežimo izpis ob koncu igre.

Kot zadnji korak nam je ostalo, da nastavimo delovanja sličice »Konec«, ko se uporabnik odloči, da predčasno konča igro. Definirati je potrebno dogodek *TouchUp* za gradnik *ImageSpriteKonec*. V njej pokličemo proceduro *procedura_konec* s parametrom »IGRA JE PREKINJENA«.

Slika 5.42: Prekinitev igre

5.4.1 Ideje za dopolnitev aplikacije

V takšni igri je veliko možnosti za razširitve. Poskusi najti svojo. Tukaj pa je nekaj predlogov:

1. Dodaj zvočne učinke, kjer jih še ni.
2. Ob vsaki pobrani ribici naj postane morski pes hitrejši.
3. Morski pes naj vsako sekundo naključno spremeni hitrost.
4. Dodaj še drugo ribico.
5. Morski pes vsako sekundo naključno izbira, ali bo plaval proti glavi ali ribi.
6. Dodaj še kakšno morsko pošast.
7. Dodaj kakšen dogodek moči. Npr. ko pobereš morsko mino, morski pes zbeži za nekaj sekund.

6 PRENOS APLIKACIJ IN NALOŽITEV V OKOLJE MIT APP INVENTOR

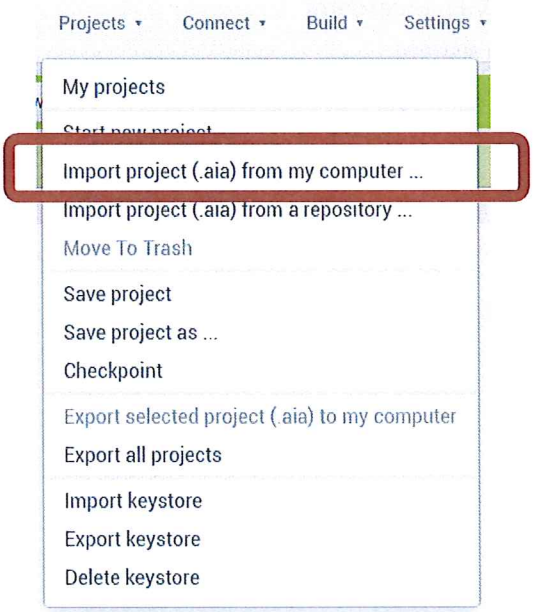
Vse aplikacije, ki smo jih razvili v tej knjigi, lahko prenesete z URL-naslova:

https://github.com/tomazkosar/AppInventorBook/archive/main.zip

Spomnimo, naredili smo naslednje aplikacije in igre:

- Vnesi ime
- Vnesi ime (z nizi)
- Pravokotnik
- Pravokotnik (s pogojnimi stavki)
- Indeks telesna mase
- Indeks telesna mase (s procedurami)
- Ugibanje števil
- Pong
- Morski pes

Ko si prenesete aplikacije z zgornjega naslova, jih v okolje MIT App Inventor naložite tako, kot prikazuje spodnja slika.



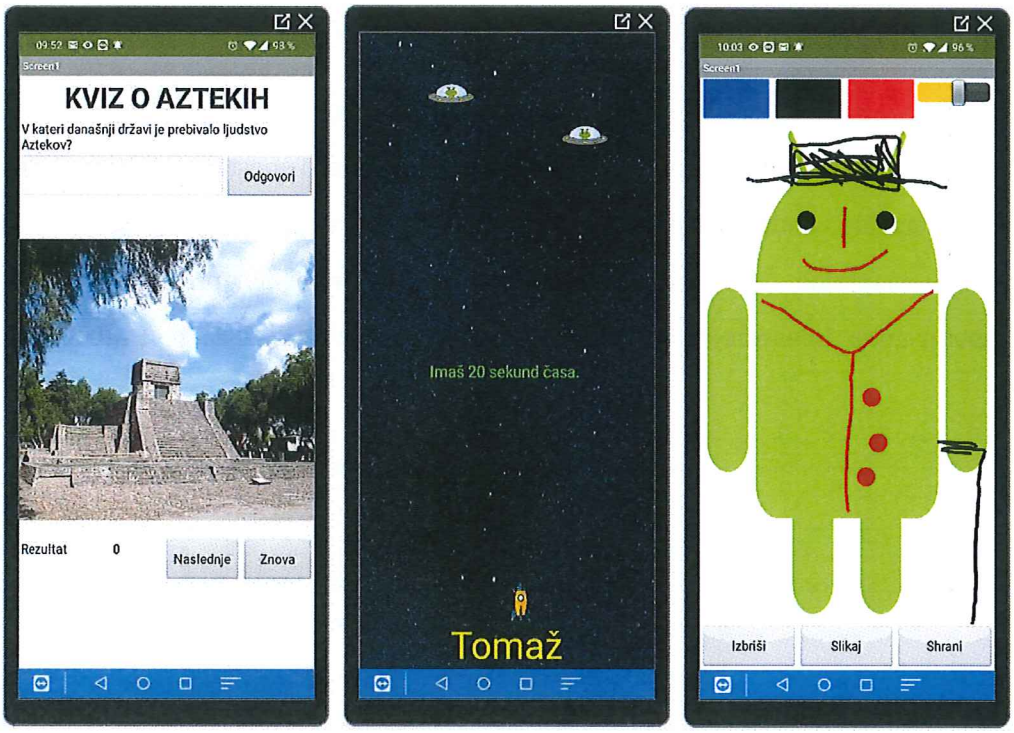
Slika 6.1: Nalaganje aplikacije

7 IDEJE ZA NOVE APLIKACIJE

Da preizkusite osvojeno znanje, je tukaj seznam idej za nove aplikacije in igre, ki jih lahko ustvarite sami:

- Aplikacije
  - Izračunaj prostornine posode
  - Kviz: kaj sem se naučil o Aztekih
  - Predstavitev malo drugače: Zdrava prehrana
- Medijske aplikacije
  - Zaigraj Zdravljico
  - Seznam najljubših skladb
- Senzorji
  - Počepi
  - Sklece
- Igre
  - Slikar (okrasimo sliko)
  - Žogica se odbija
  - Ulovi pico
  - Vesoljci
  - Spomin
- Podatkovne baze
  - Učim se tujega jezika
- Povezovanje (WiFi, Bluetooth)
  - Domači sistemi
  - Prižiganje in ugašanje LED-luči

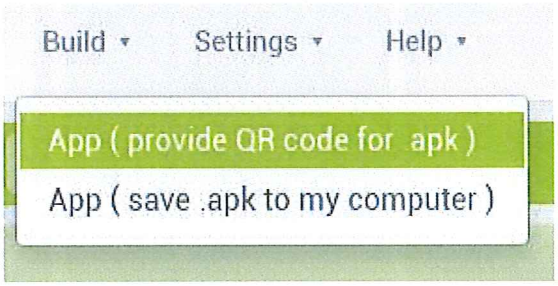
Če se udeležite naših delavnic, boste zagotovo kaj od zgoraj omenjenega tudi preizkusili. Spodaj so slike kviza o Aztekih ter iger Vesoljci in Slikar, ki smo jih dodali tudi med aplikacije, ki jih lahko prenesete k sebi.



Slika 7.1: Kviz o Aztekih, igri Vesoljci in Slikar

8 TRAJNI PRENOS APLIKACIJ NA MOBILNE NAPRAVE

Med razvojem aplikacij in iger z okoljem MIT App Inventor na mobilnih napravah uporabljamo MIT AI2 Companion, ki nam omogoča hitro in enostavno testiranje sprememb v aplikacijah. Vendar orodje MIT AI2 Companion ne omogoča trajnega nalaganja aplikacij in iger na telefon. Če želimo trajen zapis aplikacije na mobilni napravi, potem v okolju MIT App Inventor izberemo meni Zgradi (*Build*) in eno izmed možnosti, ki ju vidimo na sliki.



Slika 8.1: Trajno nalaganje aplikacij na mobilno napravo

V obeh primerih okolje MIT App Inventor izdela datoteko s končnico APK (npr. Pong.apk), ki jo potem prenesemo in zaženemo na mobilni napravi. Če smo izbrali »Priskrbi kodo QR za .apk« (Provide QR code), potem s programom MIT AI2 Companion prenesemo aplikacijo k sebi in jo namestimo. Včasih je potrebno na mobilni napravi omogočiti še nalaganje aplikacij iz neznanih virov (kar pa je odvisno od mobilne naprave, ki jo uporabljate). Pridobljeno aplikacijo v obliki datoteke apk lahko delite s prijatelji.

9 ZAKLJUČEK

V tej knjigi smo spoznali izdelavo preprostih aplikacij in računalniških iger. Za utrjevanje priporočamo, da najprej ponovite katero izmed aplikacij, ki smo jih naredili. Poskusite brez uporabe te knjige. Šele ko boste znali aplikacije narediti sami, brez pomoči knjige, priporočamo, da stopite korak naprej.

Nadaljujete pa lahko v različnih smereh. Okolje MIT App Inventor nam nudi še veliko funkcionalnosti, ki v knjigi niso opisane. V paleti (Pallette) lahko opazite še skupine gradnikov, kot so:

- Navigacija (Maps)
- Socialna omrežja (Social)
- Podatkovne baze (Storage)
- Povezljivost z WiFi ali Bluetooth (Connectivity)
- LEGO Mindstorms

Pridobljeno znanje tako lahko nadgradite s še bogatejšimi aplikacijami, ki bodo uporabljale še naprednejše gradnike. Kot vidimo, ni nujno, da se omejimo samo na uporabo mobilne naprave. Zraven lahko vključimo še robote (npr. LEGO Mindstorm) ali pa povežemo svojo napravo z mikrokrmilnikom in ga upravljamo z mobilno napravo. In še bi lahko naštevali.

Ne smemo pa pozabiti, da je okolje MIT App Inventor učno okolje. Za resnejše programiranje za mobilne naprave priporočamo druga razvojna okolja (npr. AndroidStudio itd.). Ampak korak po korak, ne prehitevajmo, najprej osvojimo osnovne koncepte programiranja v okolju MIT App Inventor, preden se pomaknemo naprej.

10 VIRI

1. Orodje MIT App Inventor. Dostopno na: https://appinventor.mit.edu/ [17. 1. 2021]
2. MIT App Inventor Youtube kanal. Dostopno na: https://www.youtube.com/user/appinventor [17. 1. 2021].
3. David Wolber. What you can do with App Inventor, 2010. Dostopno na: https://appinventorblog.com/about/ [17. 1. 2021].
4. MIT App Inventor for Android. Dostopno na: https://en.wikipedia.org/wiki/App_Inventor_for_Android [17. 1. 2021].
5. Andrew Clark. App Inventor launches second iteration, 2013. Dostopno na: https://news.mit.edu/2013/app-inventor-launches-second-iteration [17. 1. 2021].
6. Delovanje pospeškometra v okolju MIT App Inventor [17. 1. 2021], https://ai2inventor.blogspot.com/2017/03/accelerometer-tester.html