Soccer Stars

Projekt

Python Kivy

Autor: Dominik Babić
Mentor: Goran Boneta

Sadržaj:

Autor: Dominik Babić
Mentor: Goran Boneta

1. UVOD

1.1 Ideja i cilj projekta

Ideja za igricu Soccer Stars je izašla iz već postojeće igrice istoga imena koja se može naći na Trgovini Play. Igrica je namijenjena za dva igrača koja međusobno trebaju postići gol odbijajući loptu pomoću svojih igraćih "kuglica". Moja namjera je bila isprobati tu sličnu aplikaciju/igru iskodirati samostalno.

1.2 Python Kivy. Zašto baš taj modul?

Kivy jedan od manje poznatih modula za izradu GUI-a u Pythonu, ali ono što njega čini posebnim je to da se njime mogu napraviti grafička sučelja u Pythonu koja se mogu koristiti i na Androidu, IOS-u i Raspberry Pi-u. Dakle, da bi mogao što približnije iskopirati mobilnu aplikaciju, iskoristio sam modul kojim bi se moja aplikacija mogla koristiti i na mobitelima.

1.3 Tijek rada

Prije samog početka kodiranja, trebao sam naučiti dosta novosti koje su došle korištenjem Kivy-a. Osim što modul ima svoj vlastiti jezik koji služi za uređivanje sučelja, njegova interakcija unutar Python koda je drugačija nego kod ikojeg drugog modula s kojim sam do sad radio. Nakon što sam prošao kroz par lekcija koje se mogu naći na stranici https://kivy.org/#home, ponovio sam još par lekcija o objektno-orijentiranom programiranju i nakon par isprobanih zadataka, bacio sam se na posao, a to je bilo programiranje glavnog koda.

1.4 Autor

Moje ime je Dominik Babić. Učenik sam 4. razreda Gimnazije Andrije Mohorovičića. Do srednje škole nisam znao programirati, ali sam oduvijek volio igrice, a još više bi ih htio izrađivati. Uz mentora Gorana Bonetu, naučio sam esencijalne vještine programiranja u Pythonu koje mogu proširiti i na druge jezike. Da ne duljim, slijedi opis rada i samog koda.

2. OPIS RADA I KODA

2.1 Početak

Da bi započeo s pisanjem glavnog koda, bilo mi je potrebno ubaciti potreban modul Kivy, a potom iz njega izvući potrebne elementa. Inače se pojedini elemente iz Kivy-a nužno moraju ubaciti na početku koda. Osim Kivy-a, ubacio sam i modul Math jer je potreban u daljnjim procesima aplikacije. Nakon što je sve potrebno ubačeno, odredio sam veličinu prozora aplikacije tako da bude što bolje prilagođen samom izgledu ekrana mobitela.

```
import kivy
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.properties import NumericProperty, ReferenceListProperty, ObjectProperty, BooleanProperty
from kivy.core.window import Window
from kivy.vector import Vector
from kivy.clock import Clock
from kivy.core.audio import SoundLoader
from math import *

Window.size = (600, 950)
```

2.2 Izrada pokretača igre

Da bi se Aplikacija pokrenula potrebno je imati App klasu s funkcijom "*build*" koja poziva za svaki App. Unutar funkcije *build* se pozivaju ostali dijelovi aplikacije, a to je u ovom slučaju klasa Igra koju ćemo u nastavku spomenuti. Osim što se poziva Igra, uz pomoć funkcije *Clock* iz modula *kivy.clock* se poziva funkcija "*update*" (isto definirana unutar klase Igra) za Igru 60 puta u sekundi. Da bi se aplikacija pravilno pokrenula na kraju postavljamo *if __name__ = "__main__"* petlju i tada pokrećemo aplikaciju.

```
class SoccerStars(App):
    def build(self):
        igra = Igra()
        Clock.schedule_interval(igra.update, 1.0 / 60.0) #funkcija update se poziva 60x u sekundi
        return igra

if __name__ == "__main__":
    SoccerStars().run()
```

2.3 Definiranje klase Igra

Igra će nam biti glavno sučelje na kojem će se odvijati borba između dva igrača. U ovom slučaju za Igru sam koristio klasu Widget. Unutar same Igre sam postavio tri objekta:

1. Igrač 1
2. Igrač 2
3. Lopta

Sva tri objekta definirana su s *ObjectProperty*(None) zato jer ćemo ih tek u Kivy jeziku povezati s likovima koje trebaju prezentirati. Kako bi se znalo čiji je red na odigravanje postavljeni su jedan brojač vrijednosti 1 i jedan *BooleanProperty* bez vrijednosti. Također se pokreče zvuk publike uz pomoć *SoundLoadera*, a za kad se postigne pogodak je sačuvan zvuk komentatora.

```
class Igra(Widget):
    lopta = ObjectProperty(None)

    igrac1 = ObjectProperty(None)
    igrac2 = ObjectProperty(None)

    slijed = BooleanProperty(None) #provjerava ako si kliknuo igraca koji je na redu
    red = NumericProperty(1) #broji red odigranih koraka kako bi se znalo tko je na redu

    publika = SoundLoader.load("source/sound1.mp3").play()
    goal = SoundLoader.load("source/sound2.mp3")
```

2.3.1 Kv jezik - Igra

Uz glavni kod, za bolje funkcioniranje Kivy aplikacije, može se napisati i .kv kod kojim oblikujemo Widgete i druge osnovne elemente koji će se prikazati pri pokretanju glavnog programa.

Kv jezik je jednostavan i može se pisati ili unutar Python programa ili u odvojenom dokumentu, ali onda se mora povezati s glavnim programom. Komplikacije nastaju pri povezivanju objekata iz .py kod s objektima iz .kv koda.

Da bi oblikovali našu Igru potrebno ju je spomenuti unutar .kv koda, a potom odrediti sve što želite da se prikaže i što želite da se poveže. U ovo slučaju ja sam prvo da ID igračima i lopti tako da ih mogu povezati s njihovim pripadnim widgetima. Zatim sam definirao *Canvas* koji mi predstavlja teren i Labele koji će prikazivati rezultate te im zadao koordinate za pozicije(koordinatni sustav Kivy-a nije isti kao i kod Tkintera, doljnji lijevi kut je točka (0,0)).

```
<Igra>:
    lopta: lopta
    igrac1: igrac1
    igrac2: igrac2

    canvas:
        Rectangle:
            source: 'source/teren.png'
            size: 600, 908
            pos: 0, 0

    Label:
        font_size: 30
        center_y: root.height - 21
        right: root.right - 20
        text: "Igrac 2: " + str(root.igrac2.score)

    Label:
        font_size: 30
        center_y: root.height - 21
        x: 20
        text: "Igrac 1: " + str(root.igrac1.score)
```

2.4 Definiranje klase Igrac

Naravno, da bi se moglo igrati, potrebni su nam igrači. Njihovu klasu sam napravio tako da sadrži svojstva Widgeta isto kao i teren. Klasi sam pridodao vrijednost *score* koja predstavlja rezultat za svakog igrača. Da bi se igrač kretao mora imati vrijednost brzine, koja će ovdje biti izražena kao par dviju brzina za svaku os. Tako će Igrač imati *velocity_x* i *velocity_y*, a zajedno stvaraju *velocity* (brzinu).

```
class Igrac(Widget):
    score = NumericProperty(0)
    velocity_x = NumericProperty(0)
    velocity_y = NumericProperty(0)

    velocity = ReferenceListProperty(velocity_x, velocity_y)
```

Još sam ih uredio u .kv datoteci i postavio i na glavni zaslon Igra, a zatim ih spojio s njihovim ID-em.

```
<Igrac>:
    size: 60,60
    canvas:
        Ellipse:
            size: 60,60
            pos: self.pos
            segments: 360
```

```
Igrac:
    id: igrac2
    center: (300, 708)
    canvas:
        Ellipse:
            source: "source/zastava2.png"
            size: 60,60
            pos: self.pos
Igrac:
    id: igrac1
    center: (300, 200)
    canvas:
        Ellipse:
            source: "source/zastava1.png"
            size: 60,60
            pos: self.pos
```

Na slici lijevo sam s <, > definirao samu klasu *Igrac*, isto kao što sam izrazio <Igra>. Na slici desno sam upisao samu klasu *Igrac* dva puta za svakog igrača i time sam ih "postavio" na teren (Desno se prikazan kod koji se nalazi unutar <Igra>). Dodatno objašnjenje: u .kv kodu ako se klasa napiše unutar < > definiramo kakva će svojstva imati svaki objekt te klase, ali ti objekti se opet mogu definirati ako se napišu unutar definiranja neke druge klase i bez < >. Tako mogu imati beskonačno mnogo objekata iste klase ali različitih svojstva.

2.4.1 Funkcija Move

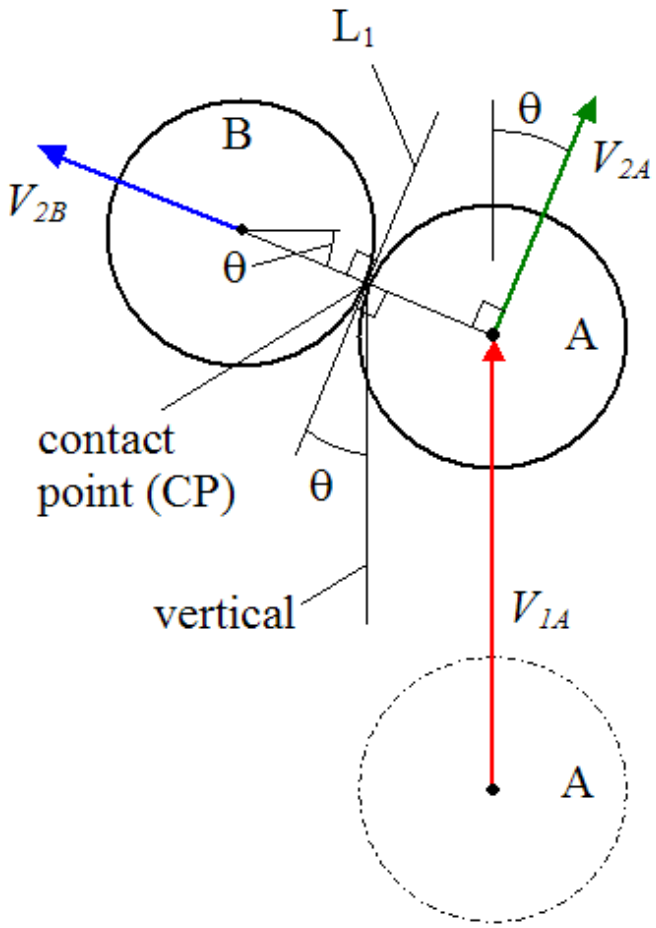
Unutar klase *Igrac* potrebno je definirati funkciju kretanja koja će se kasnije pozivati u funkciji *Update*. S njom mijenjamo poziciju igrača tako da brzinu izraženu kao Vector (Vektor) nadodamo već poznatoj poziciji igrača. Vektor je određen smjerom i iznosom. Njegove komponente *velocity_x* i *velocity_y* se stalno smanjuju za 10% kako bi igrač nakon nekog vremena stao. Kada brzina svake komponente padne ispod određene vrijednosti, brzina mu se zaokružuje na nulu, jer su znali dolaziti problemi zbog izrazito malih vrijednosti, pa brzina nikad ne bi došla do 0 nego npr do 10^{-21} i tako dalje nastavilo.

```
def move(self):
    self.pos = Vector(*self.velocity) + self.pos

    #stalno smanjivanje brzine igrača
    self.velocity_x *= 0.9
    self.velocity_y *= 0.9
    if abs(self.velocity_x) <= 0.5:
        self.velocity_x = 0
    if abs(self.velocity_y) <= 0.5:
        self.velocity_y = 0
```

2.4.2 Funkcija player_collide

Glavna i najteža interakcija između dva igrača ili igrača s loptom je odbijanje. Sama fizika i geometrija odbijanja dvaju kugli je kompliciranije nego što sam očekivao i na ovome sam potrošio bar ¾ vremena na usavršavanju, ali i dalje ima djelomičnih poteškoća.



Za provođenje funkcije se prvo treba postići uvjet. A to je da se rubovi samih likova dodiruju. Kivy ima uređenu funkciju *collide_widget* ,ali problem je u tome da ta funkcija promatra pravokutne widgete, tako da iako su moji widgeti okrugli, njihovo dodirivanje se gledalo kao da su kvadrati. Dakle morao sam upotrijebiti malo matematike i odrediti da dokle je god udaljenost središta krugova veća od zbroja polovičnih širina krugova, nema sudara. Da bi postigao slične rezultate odbijanja krugova, trebao sam otkriti točku o kojoj se kugle sudaraju. Ta točka je točno na pola udaljenosti od središta kugli tako da sam tu koordinatu bez problema mogao i zapisati u kodu. Potom sam uz malo novonaučenog znanja o derivacijama izračunao čemu je jednaka derivacija kruga s obzirom na njegovu poziciju u koordinatnom sustavu. S obzirom da se krug se sudara odbija po liniji tangente potrebno je izračunati kut kojim treba zakrenuti vektor brzine, a to je $\tan^{-1}$ k, pri čemu je k derivacija u točki sudara. A za kut odbijanja drugog kruga je normala pa $\tan^{-1}$ k. Zbog funkcioniranja funkcije *rotate* to je obrnuto.

```
#sudar dva igraca
def player_collide(self, player):
    if sqrt((self.center_x - player.center_x)**2 + (self.center_y - player.center_y)**2) <= (self.width /2) + (player.width/2):
        pocx = (player.center_x + self.center_x)/2  #koordinate dodira
        pocy = (player.center_y + self.center_y)/2  #dve kugle
        if pocy == self.y:
            k = 0
        else:
            k = (self.x - pocx)/(pocy - self.y) #derivacija u točki dodira kugla, kako bi mogao odrediti kut pod kojima će se one odbiti


        if abs(k) > 0.4:
            kut_1 = degrees(atan(k))
        else:
            kut_1 = 0
        vel1 = Vector(self.velocity).rotate(kut_1)
        player.velocity = vel1

        kut_2 = degrees(atan(-1/k))
        vel2 = Vector(self.velocity).rotate(kut_2)
        self.velocity = vel2 /2

        player.move()
        self.move()
```

2.5 Definiranje klase Lopta

Kako bi se mogao postići gol, ako ne loptom. Nju sam definirao s istim svojstvima kao i klasu *Igrac* , ali sam unutar .kv nadodao par svojstva. Dakle, lopta isto ima funkciju odbijanja *ball_collide* i kretanja *move*. Iako sam mogao napraviti loptu da bude ista klasa kao igrač jer ima sve iste funkcije i svojstvakao igrači, ali da promijenim izgled i vizualne karakteristike, prekasno sam se toga sjetio, pa u ovom trenutnom stanju je to tako, ali još ću raditi na projektu i nakon ove prezentacije, tako da ću sigurno i to promjeniti.

```
class Lopta(Widget):
    velocity_x = NumericProperty(0)
    velocity_y = NumericProperty(0)

    velocity = ReferenceListProperty(velocity_x,velocity_y)

    def move(self):
        self.pos = Vector(*self.velocity) + self.pos
        self.velocity_x *= 0.9
        self.velocity_y *= 0.9
        if abs(self.velocity_x) <= 0.5:
            self.velocity_x = 0
        if abs(self.velocity_y) <= 0.5:
            self.velocity_y = 0

    def ball_collide(self, player):
        if sqrt((self.center_x - player.center_x)**2 + (self.center_y - player.center_y)**2) <= (self.width /2) + (player.width/2):
            pocx = (player.center_x + self.center_x)/2  #koordinata u kojima se lopta
            pocy = (player.center_y + self.center_y)/2  # i igrač dodiruju, poc = point of collision
            if pocy == self.y:
                k = 0
            else:
                k = (self.x - pocx)/(pocy - self.y) #derivacija u točki dodira kugla, kako bi mogao odrediti kut pod kojima će se one odbiti

            if abs(k) > 0.4:
                kut_1 = degrees(atan(k))
            else:
                kut_1 = 0
            vel1 = Vector(self.velocity).rotate(kut_1)
            player.velocity = vel1 /2

            kut_2 = degrees(atan(-1/k))
            vel2 = Vector(self.velocity).rotate(kut_2)
            self.velocity = vel2 /2

            player.move()
            self.move()
```

Loptu sam potom oblikovao (doljnji dio koda na slici ispod) te joj zadao poziciju i pridoda ID u Igri(gornji dio koda na slici spod).

Autor: Dominik Babić
Mentor: Goran Boneta

```
    Lopta:
        id: lopta
        center: 300, 454

<Lopta>:
    size: 50,50
    canvas:
        Ellipse:
            source: 'source/lopta.png'
            pos: self.pos
            size: 50,50
            segments: 360
```

2.6 Funkcije vezane uz Touch

Touch je jedna od mogućnosti koje pruža Kivy, a ona služi za određivanje klikna, odnosno dodira. Touch ima vrijednosti True/False i koordinate. Funkcije vezane uz *touch* su on_touch_down (pokrene se kad se klik "spusti"), on_touch_move (pokrene se kad se isti taj klik miče) i on_touch_up (pokrene se kad se klik otpusti). Ja sam za ovu igru trebao samo dve od te funkcije. Postavio sam da s on_touch_down funkcijom program provjeri koji je igrač kliknut i je li njegov red. S funkcijom on_touch_up program izračunava udaljenost zadnje pozicije klika i središta igrača koji igra te mu prema tim izračunima pridoda brzinu.

Funkcije se nalaze unutar klase *Igra*.

```
#kad se klikne provjerava koji je igrac kliknut
def on_touch_down(self, touch):
    if self.igrac1.collide_point(*touch.pos):
        self.slijed = True
    elif self.igrac2.collide_point(*touch.pos):
        self.slijed = False
```

```
#kad se otpusti klik, provjerava zadnja pozicija klika te se računa brzina koja se nadodaje igraču
def on_touch_up(self, touch):
    if self.slijed and self.red % 2 == 1:
        self.igrac1.velocity_x = int((self.igrac1.center_x - touch.x)*3/4)
        self.igrac1.velocity_y = int((self.igrac1.center_y - touch.y)*3/4)
        self.slijed = BooleanProperty(None)
        self.red += 1
    elif self.red % 2 == 0 and not self.slijed:
        self.igrac2.velocity_x = int((self.igrac2.center_x - touch.x)*3/4)
        self.igrac2.velocity_y = int((self.igrac2.center_y - touch.y)*3/4)
        self.slijed = BooleanProperty(None)
        self.red += 1
```

2.7 Funkcija Restart

Funkcija koja vraća igrače i loptu na početnu poziciju. Pokreće se zvuk komentatora kako govori “Gol”.

```
def restart(self):
    self.goal.play()
    self.lopta.center = (300, 454)
    self.lopta.velocity = (0,0)
    self.igrac1.center = (300, 200)
    self.igrac1.velocity = (0,0)
    self.igrac2.center = (300, 708)
    self.igrac2.velocity = (0,0)
```

2.8 Funkcija Update

Glavna funkcija koja pokreće sve važne dijelove *Igre* i provjerava potrebne uvjete. Provjerava ako igrači ili lopta udaraju u rubove terena te ih odbija u slučaju da udare. Ako se ijedan igra.

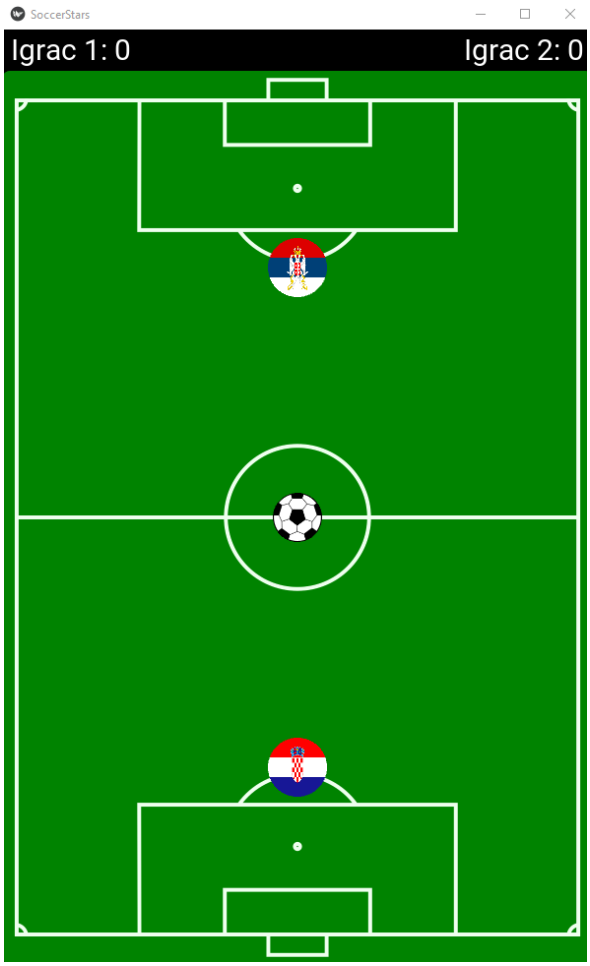
```
def update(self, dt):
    #pokretanje funkcije kretanja igraca, 60x u sekundi provodi micanje igraca
    #provjerava i provodi odbijanje između igrača i igrača s loptom
    if self.igrac1.velocity_y != 0 or self.igrac2.velocity_x != 0:
        self.igrac1.move()
        self.igrac1.player_collide(self.igrac2)
        self.igrac1.player_collide(self.lopta)
    if self.igrac2.velocity_y != 0 or self.igrac2.velocity_y != 0:
        self.igrac2.move()
        self.igrac2.player_collide(self.igrac1)
        self.igrac2.player_collide(self.lopta)
    if self.lopta.velocity_y != 0 or self.lopta.velocity_x != 0:
        self.lopta.move()
        self.lopta.ball_collide(self.igrac1)
        self.lopta.ball_collide(self.igrac2)

    #Odbijanje od zidova terena
    if (self.igrac1.x <= 0) or (self.igrac1.right >= 600):
        self.igrac1.velocity_x *= -1
        self.igrac1.move()
    if (self.igrac1.y <= 0) or (self.igrac1.top >= 908):
        self.igrac1.velocity_y *= -1
        self.igrac1.move()
    if (self.igrac2.x <= 0) or (self.igrac2.right >= 600):
        self.igrac2.velocity_x *= -1
        self.igrac2.move()
    if (self.igrac2.y <= 0) or (self.igrac2.top >= 908):
        self.igrac2.velocity_y *= -1
        self.igrac2.move()

    #Odbijanje lopte od zidova i restartanje igre u slučaju gola
    if  (self.lopta.x <= 0) or (self.lopta.right >= 600):
        self.lopta.velocity_x *= -1
        self.lopta.move()
    if (self.lopta.y <= 0):
        self.igrac2.score += 1
        self.restart()
    elif (self.lopta.top >= 908):
        self.igrac1.score +=1
        self.restart()
```

2.9 Završni izgled

Pokretanjem programa dobije se ovo sučelje:



3.SAMOKRITIKA

Zadovoljan sam ovim što sam napravio, iako ima svojih mana. Siguran sam da ću još raditi na tome da poboljšam program u svim mogućim aspektima. Problema je bilo tokom pisanja koda: manjak volje u nekim slučajevima (kad bi pomislio da ne mogu naći rješenje za probleme), manjak vremena s obzirom na druge obaveze (treninzi i škola).