GIMNAZIJA ANDRIJE MOHOROVIČIĆA RIJEKA

DOKUMENTACIJA

Try2Survive®

Andre Flego

Rijeka, 2020.

Sadržaj

1. UVOD

Ovaj dokument služi kao dokumentacija programa *Try2Survive®* čiji je autor Andre Flego. Zabranjeno je bilo kakvo kopiranje, distribuiranje i objavljivanje sadržaja programa bez odobrenja autora. Program je u potpunosti napisan od strane autora te za rješavanje problemskog djela programa nije korištena nikakva pomoć ostalih osoba.

Try2Survive jednostavna je igra u kojoj korisnik mora pomicati svog igrača i izbjegavati prepreke. Igrač je svemirski brod *Gianni*, a prepreke su asteroidi koji se kreću različitim brzinama. Igrač ima određenu razinu života, tzv. *health,* i rezultat igre, tzv. *score.* Svaki put kada je igrač pogođen asteroidom smanjuju se razina života i rezultat u ovisnosti o brzini asteroida. Što je asteroid brži to je veći gubitak razine života i rezultata igre.

Ideja za ovu igru nastala je prilikom gledanja Ratova zvijezda i razmišljanja o putovanju po svemiru te s kojim bi se sve problemima mogli susresti dok nemilosrdno haramo praznim beskonačnošću.

U dokumentaciji ću navesti sve klase te njihove atribute i metode, kao i ostale datoteke koje nisu klase, ali služe za učinkovito funkcioniranje programa. Objasnit ću što koja metoda radi, a kod većih blokova kôda ću priložiti sam kôd te malo detaljnije opisati na koji način se izvršava taj dio.

Screencast u kojem se može vidjeti kako funkcionira program dostupan je na ovom linku:

https://youtu.be/yNdF9BINLPA

2. TEHNIČKE INFORMACIJE

2.1. Tehnički preduvjeti

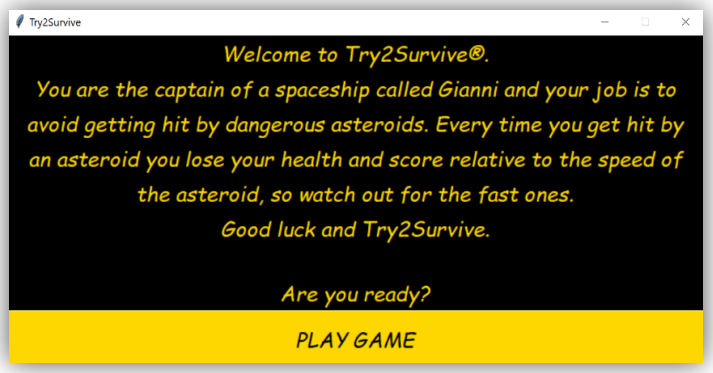
Računalo sa 64-bitnim operativnim sustavom Windows 7+, 4GB RAM-a, 1.8GhZ procesor.

2.2. Softverski preduvjeti

Program je optimiziran za pokretanje na Windows operativnim sustavima. Za pokretanje programa potrebno je imati instaliran *Python 2.7+* te *Python* modul *pygame 1.9.6+.* Program je pisan u PyCharm-u, stoga je najbolje pokretati program iz te aplikacije, no poslužit će i običan *IDLE* od *Pythona.*

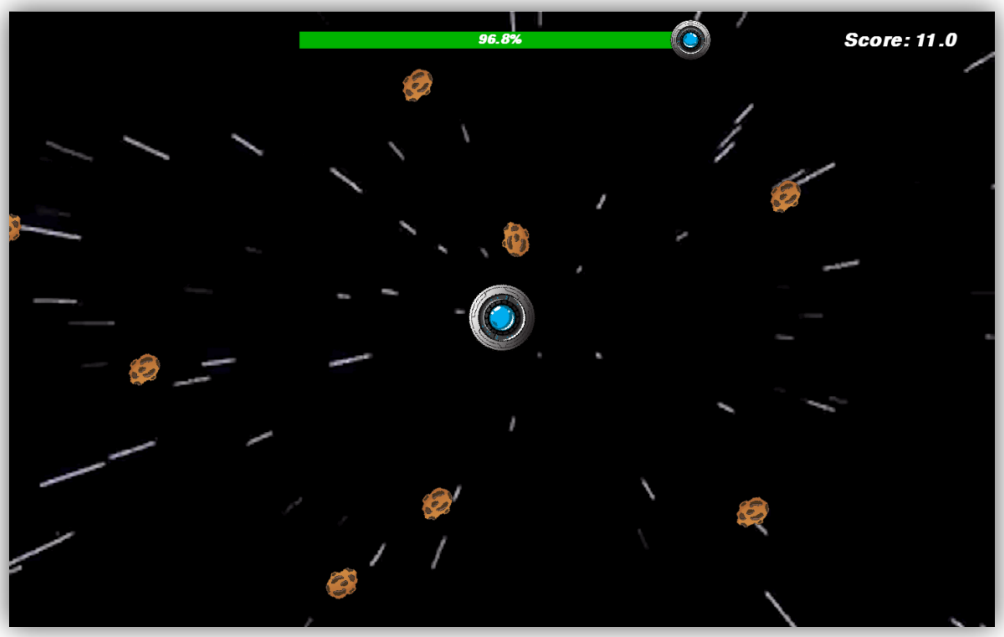
2.3. Opis rada

Prvo što korisnik vidi jest **početni skočni prozor**.



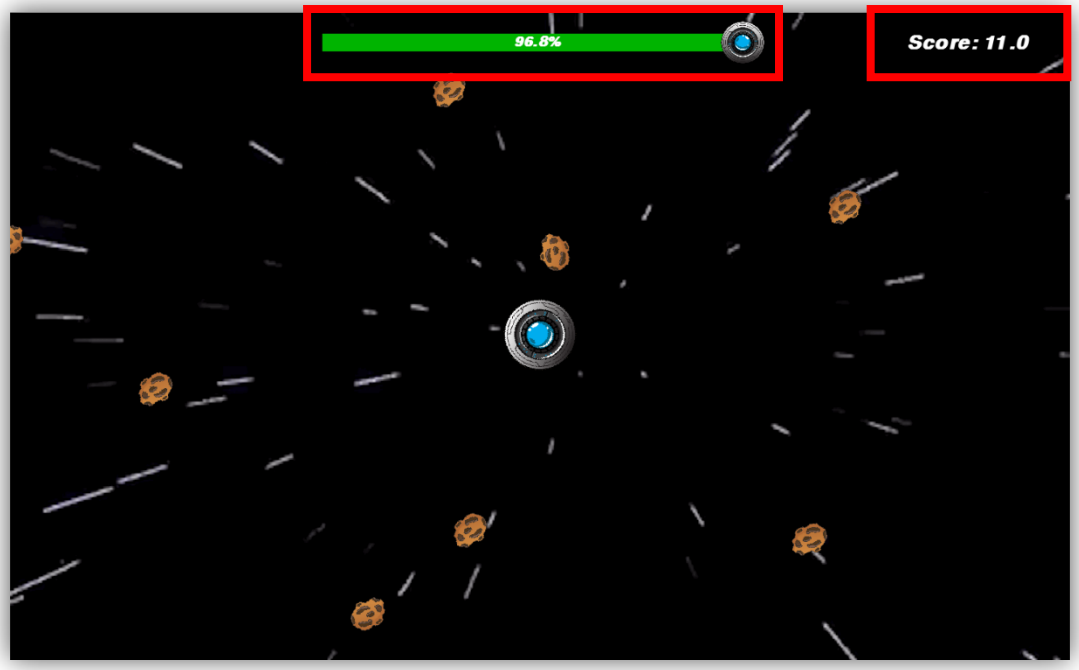
Korisnika pozdravlja **poruka dobrodošlice** i upoznaje ga s pričom igre. Klikom na gumb ***„Play game"*** pokreće se igra te se zatvara skočni prozor.

Zatim se pred korisnikom prikazuje **glavni prozor** u kojem se odvija cijela igra.
Prije početka igranja pred korisnikom se prikazuje **animacija ulaska igrača u prozor za igru**.
Potom asteroidi počinju popunjavati scenu te igranje počinje.

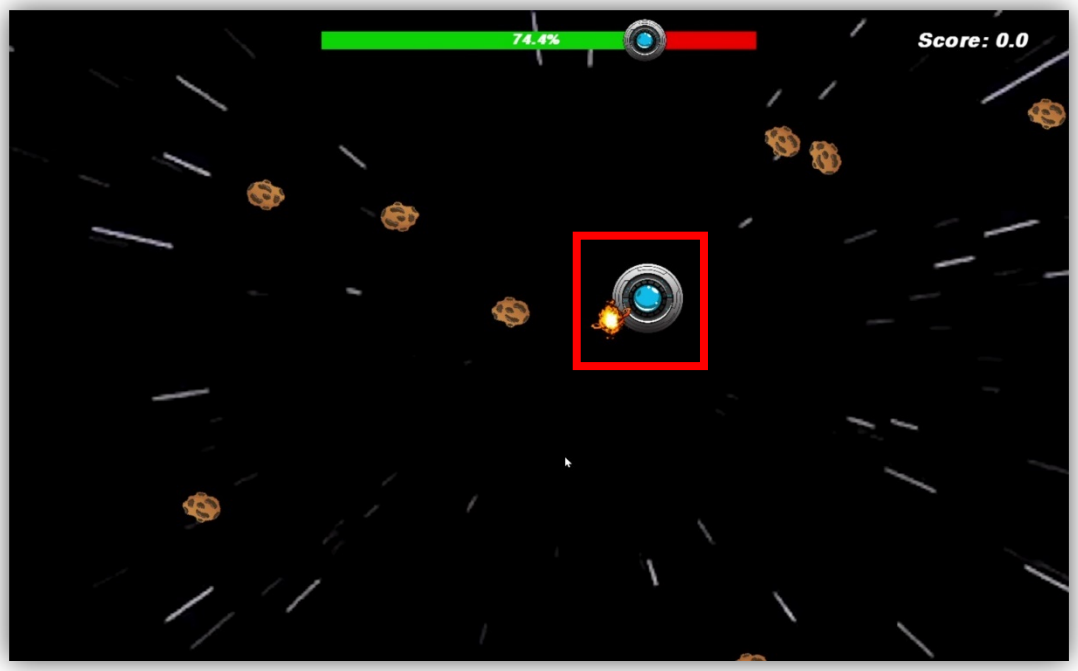


Korisnik koristi **tipke strelica za pomicanje** u smjerovima gore, dolje, lijevo i desno. Igrač je ograničen na pomicanje samo unutar rubova prozora tj. ne može napustiti vidljivi dio prozora za igru.

Pored samog igrača na prozoru su vidljivi i **razina života** te **rezultat igre**.

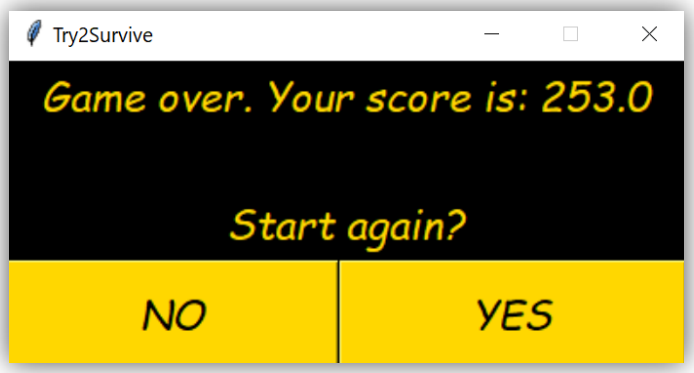


Ukoliko dođe do **sudara igrača i asteroida** tada se igraču smanjuju razina života i rezultat, ovisno o brzini asteroida, a na mjestu sudara vidljiva je **animacija eksplozije**.



S obzirom da igra nije vremenski ograničena korisnik može igrati dokle god razina života ne padne na 0. To je manje-više nemoguće jer s vremenom broj asteroida se povećava te postaje poprilično zahtjevno za pratiti situaciju u igri.

Kada razina života padne na 0 igra završava te se pojavljuje animacija za kraj igre i **završni skočni prozor** u kojem je vidljiv **konačan rezultat igre**.



Korisniku se u završnom skočnom prozoru nudi opcija za pokretanje nove igre. Klikom na gumb ***„Yes“*** pokreće se nova igra, a rezultat i razina života se vraćaju na početne vrijednosti. Klikom na gumb ***„No“*** zatvara se skočni prozor i igra završava.

U bilo kojem trenutku igre korisnik može pritisnuti gumb ***ESC*** te će se igra završiti bez prikazivanja završnog skočnog prozora.

3. KÔD

Kod se sastoji od **5 klasa: *Player, Enemy, Game, Starter, Ender*** i dvije pomoćne datoteke: ***Loader.py*** i ***Config*.py.** Klase ***Game*** i ***Ender*** importiraju pomoćnu datoteku ***Loader.py*** u kojoj se učitavaju sve potrebne datoteke poput slika, audio zapisa i sl. kako bi se centraliziralo učitavanje stavki na jedno mjesto. *(Više u poglavlju **3.6 Loader**)*

3.1. Player

Player je klasa kojom se kreira igrač, tj. instanca ove klase je igrač kojeg kontrolira korisnik. Instanca se kreira pozivanjem klase: ***Player().*** Klasa *Player* importira pomoćnu datoteku *Config.py* u kojoj su definirane varijable poput širine i visine prozora itd. *(Više u poglavlju **3.7 Config**)*

```
class Player:
    def __init__(self):
        self.width = 100
        self.height = 100
        self.x = windowWidth / 2
        self.y = windowHeight + self.height
        self.speed = 10
        self.image = pygame.transform.scale(playerImage, (self.width, self.height))
        self.imageAngle = 0
        self.hitbox = pygame.Rect(self.x - self.width / 2, self.y - self.height / 2, self.width, self.height)
        self.hitCount = 0
        self.health = 500
        self.maxHealth = 500
        self.score = 0
```

Atributi ove klase su:

- **width** – širina
- **height** – visina
- **x** – horizontalna koordinata
- **y** – vertikalna koordinata
- **speed** – brzina igrača
- **image** – slika igrača
- **imageAngle** – kut pod kojim je okrenuta slika igrača
- **hitbox** – granica oko igrača koja služi za određivanje je li igrač pogođen
- **hitCount** – brojač pogodaka
- **health** – razina života
- **maxHealth** – najveća razina života (početna razina)
- **score** – rezultat igre

Klasa *Player* ima sljedeće **metode** koje služe za normalno funkcioniranje igre:

- **getCoordinates()**
- **getCenterCoordinates()**
- **draw()**
- **moveUp()**
- **moveDown()**
- **moveRight()**

- moveLeft()
- values()
- isHit()

3.1.1. *getCoordinates()*
Vraća koordinate igrača.

3.1.2. *getCenterCoordinates()*
Vraća centralne koordinate igrača.

3.1.3. *draw()*
Crta igrača u prozoru.

```
def draw(self, window):
    image = pygame.transform.rotate(self.image, self.imageAngle)
    window.blit(image, (self.x - int(image.get_width() / 2), self.y - int(image.get_height() / 2)))
    self.imageAngle = (self.imageAngle + 10) % 360
    self.hitbox = pygame.Rect(self.x - self.width / 2, self.y - self.height / 2, self.width, self.height)
```

Metoda ***pygame.transform.rotate*** rotira sliku za neki kut te tu sliku dodjeljuje varijabli *image*. Zatim se slika *image* crta u *pygame* prozoru, kut poveća za 10˚, a koordinate atributa *hitbox* se sinkroniziraju s koordinatama igrača.

3.1.4. *moveUp()*
Pomiče igrača do gornjeg ruba pygame prozora.

3.1.5. *moveDown()*
Pomiče igrača do donjeg ruba *pygame* prozora.

3.1.6. *moveRight()*
Pomiče igrača do desnog ruba *pygame* prozora.

3.1.7. *moveLeft()*
Pomiče igrača do lijevog ruba *pygame* prozora.

3.1.8. *values()*
Vraća osnovne vrijednosti instance ove klase.

3.1.9. *isHit()*
Vraća Boolean je li igrač pogođen.

```
def isHit(self, enemy):
    if self.hitbox.colliderect(enemy.hitbox):
        self.score -= enemy.damage if self.score > enemy.damage else self.score
        self.hitCount += 1
        self.health -= enemy.damage if self.health > enemy.damage else self.health
        return True
    return False
```

Metoda ***colliderect*** nad atributom *hitbox* je ugrađena metoda koja provjerava jesu li dva objekta klase ***pygame.Rect*** preklopljena.

3.2. Enemy

Enemy je klasa kojom se kreira neprijatelj – asteroid. Instanca se kreira pozivanjem klase: ***Enemy()***. Klasa *Enemy* importira pomoćnu datoteku *Config.py* u kojoj su definirane varijable poput širine i visine prozora itd. *(Više u poglavlju **3.7 Config**)*

```
class Enemy:
    def __init__(self):
        self.x = randint(0, 3)
        # Neprijatelj dolazi s lijeve strane
        if self.x == 0:
            self.x1 = -32
            self.y1 = randint(0, windowHeight)
            self.x2 = windowWidth + 32
            self.y2 = randint(0, windowHeight)
        # Neprijatelj dolazi s gornje strane
        elif self.x == 1:
            self.x1 = randint(0, windowWidth)
            self.y1 = -32
            self.x2 = randint(0, windowWidth)
            self.y2 = windowHeight + 32
        # Neprijatelj dolazi s desne strane
        elif self.x == 2:
            self.x1 = windowWidth + 32
            self.y1 = randint(0, windowHeight)
            self.x2 = -32
            self.y2 = randint(0, windowHeight)
        # Neprijatelj dolazi s donje strane
        else:
            self.x1 = randint(0, windowWidth)
            self.y1 = windowHeight + 32
            self.x2 = randint(0, windowWidth)
            self.y2 = -32

        self.dx = (self.x2 - self.x1) / framerate
        self.dy = (self.y2 - self.y1) / framerate

        self.width = 50
        self.height = 50
        self.speedFactor = randint(3, 6)
        self.image = pygame.transform.scale(enemyImage, (self.width, self.height))
        self.imageAngle = 0
        self.hitbox = pygame.Rect(self.x1 - self.width / 2, self.y1 - self.height / 2, self.width, self.height)
        self.damage = 100 // self.speedFactor
```

Atributi ove klase su:

- **x** – nasumičan broj kojim se određuje smjer odakle dolazi neprijatelj
  - Ako je **x** jednak nuli tada će neprijatelj dolaziti s lijeve strane *pygame* prozora
  - Ako je **x** jednak jedan tada će neprijatelj dolaziti s gornje strane *pygame* prozora
  - Ako je **x** jednak dva tada će neprijatelj dolaziti s desne strane *pygame* prozora
  - Ako je **x** jednak tri tada će neprijatelj dolaziti s donje strane *pygame* prozora.
- **x1** – početna horizontalna koordinata
- **y1** – početna vertikalna koordinata
- **x2** – krajnja horizontalna koordinata
- **y2** – krajnja vertikalna koordinata

- **dx** – horizontalna udaljenost(u pikselima) koju neprijatelj pređe u jednom *frame-u*
- **dy** – vertikalna udaljenost(u pikselima) koju neprijatelj pređe u jednom *frame-u*
- **framerate** – varijabla definirana u Config.py
- **width** – širina
- **height** – visina
- **speedFactor** – nasumični broj koji određuje brzinu neprijatelja
- **image** – slika neprijatelja
- **imageAngle** – kut pod kojim je slika neprijatelja okrenuta
- **hitBox** – granica oko neprijatelja koja služi za određivanja je li igrač pogođen
- **damage** – vrijednost koju neprijatelj skida od razine života igrača kada ga pogodi

Klasa *Enemy* ima nekoliko **metoda** koje su potrebne kako bi igra normalno funkcionirala:

- getCoordinates()
- getCenterCoordinates()
- draw()
- move()

3.2.1. *getCoordinates()*

Vraća koordinate neprijatelja.

3.2.2. *getCenterCoordinates()*

Vraća centralne koordinate neprijatelja.

3.2.3. *draw()*

Crta neprijatelja u prozoru.

```
def draw(self, window):
    image = pygame.transform.rotate(self.image, self.imageAngle)
    window.blit(image, (self.x1 - int(image.get_width() / 2), self.y1 - int(image.get_height() / 2)))
    self.imageAngle = (self.imageAngle + 3) % 360
    self.hitbox = pygame.Rect(self.x1 - self.width / 2, self.y1 - self.height / 2, self.width, self.height)
```

Metoda ***pygame.transform.rotate*** rotira sliku za neki kut te tu sliku dodjeljuje varijabli *image*. Zatim se slika *image* crta u *pygame* prozoru, kut poveća za 3˚, a koordinate atributa *hitbox* se sinkroniziraju s koordinatama neprijatelja.

3.2.4. *move()*

Pomiče neprijatelja preko prozora po zadanoj putanji.

```
def move(self):
    if (0 < self.x1 + self.width and self.x1 < windowWidth) or
                (0 < self.y1 + self.height and self.y1 < windowHeight):
        self.x1 += self.dx / self.speedFactor
        self.y1 += self.dy / self.speedFactor
```

Pomiče neprijatelja ako je on unutar vidljivog dijela *pygame* prozora. Vertikalni i horizontalni pomak dijeli se s atributom *speedFactor* kako bi korisnik imao vremena za izbjeći neprijatelja.

3.3. Starter

Starter je klasa kojom pokrećemo igru, tj. inicijalizacijom te klase prikazuje se **početni skočni prozor** s dobrodošlicom i uputama za korisnika. Ova klasa koristi *tkinter* modul za prikazivanje prozora. Klasa *Starter* importira pomoćnu datoteku *Config.py* u kojoj su definirane varijable poput širine i visine prozora itd. *(Više u poglavlju **3.7 Config**)*

```
class Starter(Frame):
    def __init__(self, root):
        self.root = root
        self.root.title('Try2Survive')
        self.root.resizable(False, False)
        self.width = 800
        self.height = 380
        self.root.geometry(str(self.width) + 'x' + str(self.height) + '+' + str(windowWidth // 2 - self.width // 2)
                           + '+' + str(windowHeight // 2 - self.height // 2))
        super().__init__(self.root)

        self.font = ("Comic Sans MS", 18, "italic")
        self.instructions = ["Welcome to Try2Survive®.",
                             "You are the captain of a spaceship called Gianni and your job is to",
                             "avoid getting hit by dangerous asteroids. Every time you get hit by",
                             "an asteroid you lose your health and score relative to the speed of",
                             "the asteroid, so watch out for the fast ones.",
                             "Good luck and Try2Survive.",
                             "\nAre you ready?"]

        self.grid(column=1)
        self.createInterface()
        return
```

U inicijalizaciji se definira *tkinter* prozor, naslov prozora, njegove dimenzije i koordinate na ekranu, mogućnost mijenjanja veličine prozora. Zatim se broj stupaca u prozoru postavlja na 1, definira se font kojim će se prikazati upute u prozoru te same upute, a na kraju se poziva metoda ***createInterface()***.

3.3.1. *createInterface()*

```
def createInterface(self):
    for i in range(0, len(self.instructions)):
        label = Label(self.root, text=self.instructions[i], bg='black', fg='gold', font=self.font)
        label.grid(row=i, column=0, sticky=W + E)

    button = Button(self.root, text='PLAY GAME', bg='gold', fg='black',
                    font=self.font, command=self.buttonClicked, height='2')
    button.grid(row=i + 1, column=0, sticky=W + E)

    self.root.grid_columnconfigure(0, weight=1)
    self.root.grid_rowconfigure(i + 1, weight=1)
    return
```

For petlja prolazi kroz listu s uputama i postavlja svaki dio upute u poseban redak kako bi tekst upute stao u prozor, a ispod upute se dodaje gumb kojim se pokreće sama igra. Klikom na gumb poziva se metoda ***buttonClicked().***

3.3.2. *buttonClicked()*

```
def buttonClicked(self):
    self.root.destroy()
    game = Game()
    game.main()
```

Pozivom funkcije ***self.root.destroy*** zatvara se *tkinter* prozor, a nakon toga instancira se objekt klase *Game* i poziva ***main()*** metoda te klase kojom se pokreće igra.

3.4. Game

Game je klasa u kojoj se odvija 90% igre i sadržajno je najveći dio programa. Objekt ove klase instancira se pozivanje klase: ***Game().*** Klasa *Game* importira pomoćnu datoteku *Loader.py* u kojoj su učitane datoteke poput slika, audio zapisa i sl. *(Više u poglavlju **3.6 Loader**),* klase *Player, Enemy, Ender* i modul *tkinter*.

```
class Game():
    def __init__(self):
        self.clock = pygame.time.Clock()
        self.frameCounter = 0

        self.player = Player()
        self.playerImage = pygame.transform.scale(pygame.image.load('Assets/player.png'), (60, 60))

        self.explosionCount = 6
        self.explosionCoords = (0, 0)
        self.enemies = []
        self.maxEnemyNumber = 5

        self.run = True
        self.isGameOver = False
        self.isPlaying = False

        self.backgroundImageIndex = 0
```

Atributi ove klase su:

- **Clock –** instanca klase ***pygame.time.Clock()***, služi za postavljanje *framerate-a* igre
- **frameCounter –** brojač *frame-ova*
- **player –** igrač, instanca klase *Player*
- **playerImage –** slika igrača
- **explosionCount –** brojač za prikazivanje sličica eksplozije pri animaciji eksplozije
- **explosionCoords –** koordinate sudara igrača i neprijatelja
- **enemies –** lista prikazanih neprijatelja
- **maxEnemyNumber –** početni najveći broj neprijatelja koji istovremeno mogu biti u *pygame* prozoru
- **run –** uvjet za izvršavanje glavne petlje
- **isGameOver –** Boolean koji označava kraj igre
- **isPlaying –** Boolean koji označava je li igra u tijeku
- **backgroundImageIndex –** brojač koji služi za crtanje slika pozadine pri animaciji pozadine

Klasa *Game* ima 5 ključnih **metoda** pomoću kojih igra normalno funkcionira:

- ***main()***
- ***redrawWindow()***
- ***enemyCheck()***
- ***drawGameStatusInfo()***
- ***gameOver()***

3.4.1. *main()*

Ova metoda je ključna funkcija u cijelom programu. U njoj se izvršavaju sve pomoćne funkcije za crtanje igrača, neprijatelja, pozadine… Ona se može rastaviti na nekoliko manjih funkcijskih cjelina tj. blokova.

```
def main(self):
    pygame.time.delay(200)
    pygame.mixer.music.play(-1)
    while self.run:
        self.clock.tick(framerate)
        self.frameCounter += 1

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.run = False

        keys = pygame.key.get_pressed()
        if keys[pygame.K_LEFT]:
            self.player.moveLeft()
        if keys[pygame.K_RIGHT]:
            self.player.moveRight()
        if keys[pygame.K_UP]:
            self.player.moveUp()
        if keys[pygame.K_DOWN]:
            self.player.moveDown()
        if keys[pygame.K_ESCAPE]:
            self.run = False

        if self.player.y > windowHeight / 2 and not(self.isPlaying):
            self.player.moveUp()
        else:
            self.isPlaying = True

            if len(self.enemies) < self.maxEnemyNumber:
                enemy = Enemy()
                self.enemies.append(enemy)

            if self.maxEnemyNumber < 20 and not(self.frameCounter % 60):
                self.maxEnemyNumber += 0.5

        self.redrawWindow()

        if self.isGameOver:
            self.gameOver()
            break
```

```
def main(self):
    pygame.time.delay(200)
    pygame.mixer.music.play(-1)

    while self.run:
        self.clock.tick(framerate)
        self.frameCounter += 1
    (...)
```

U prvom djelu *main* metode pokreće se pozadinska glazba pozivanjem funkcije ***pygame.mixer.music.play(-1)*** i ulazi se u *while* petlju u kojoj se postavlja *framerate* igre tj. koliko puta u jednoj sekundi će se ponoviti glavna *while* petlja. Varijabla *framerate* definirana je u *Config.py* datoteci i ona iznosi 60 što znači da će se *while* petlja izvršiti 60 puta u sekundi (60FPS).

```
def main(self):
    (...)
    for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.run = False

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        self.player.moveLeft()
    if keys[pygame.K_RIGHT]:
        self.player.moveRight()
    if keys[pygame.K_UP]:
        self.player.moveUp()
    if keys[pygame.K_DOWN]:
        self.player.moveDown()
    if keys[pygame.K_ESCAPE]:
        self.run = False
    (...)
```

Zatim se u *for* petlji prolazi kroz tzv. *evente* koji predstavljaju neku akciju korisnika, kao npr. pritisak na tipku tipkovnice, klik mišem i sl. Provjerava se je li pritisnut *„X“* za zatvaranje prozor, ako je gumb pritisnut uvjet za izvršavanje glavne petlje postavlja se na *False* čime se prekida *while* petlja i zatvara se *pygame* prozor.

Nakon toga se pomoću funkcije ***pygame.key.get_pressed()*** dohvaća lista pritisnutih tipki na tipkovnici, mišu ili drugom uređaju te se, ovisno o pritisnutoj tipki, poziva određena funkcija. Npr. ako je pritisnuta tipka *strelica dolje* pozvat će se metoda ***moveDown()*** klase *Player* te će se igrač pomaknuti prema donjem rubu prozora za određen broj piksela. Također, ako je pritisnuta tipka *ESC* uvjet za izvršavanje glavne petlje postavlja se na *False* i time se prekida *while* petlja i zatvara se *pygame* prozor.

```
def main(self):
    (...)
    if self.player.y > windowHeight / 2 and not(self.isPlaying):
        self.player.moveUp()
    else:
        self.isPlaying = True
        if len(self.enemies) < self.maxEnemyNumber:
            enemy = Enemy()
            self.enemies.append(enemy)

        if self.maxEnemyNumber < 20 and not(self.frameCounter % 60):
            self.maxEnemyNumber += 0.5
    (...)
```

Ova provjera služi za animaciju ulaska igrača u vidljiv dio *pygame* prozora. Početne koordinate igrača su na donjoj sredini ekrana te se ulazak igrača animira tako da ga se pomiče prema gore sve dok ne dođe do sredine prozora.

Nakon animacije varijabla *isPlaying* se postavlja na *True* kako se više nebi ponavljala animacija igrača. Dok traje igra, izvršava se *else* dio ove petlje, tj. izvršavaju se nove provjere.

U prvoj se provjerava je li prikazan najveći mogući broj neprijatelja, ako nije u listu neprijatelja dodaje se novi neprijatelj koji se kasnije prikazuje u prozoru. U drugoj se provjeri najveći mogući broj neprijatelja povećava svakih 60 *frame-ova* (svake sekunde) za 0.5 sve dok ne dosegne vrijednost 20. Na taj se način broj istovremeno vidljivih neprijatelja povećava što se duže odvija igra.

```
def main(self):
    (...)
    self.redrawWindow()

    if self.isGameOver:
        self.gameOver()
        break
```

Ovo je posljednji dio *main* metode i u njemu se izvršavaju dvije funkcije. Prva koja se izvršava je ***redrawWindow()*** i u njoj se crtaju svi objekti na *pygame* prozoru. Nakon toga se provjerava je li igra gotova, ako je varijabla *isGameOver* postavljena na *True* poziva se funkcija ***gameOver()*** u kojoj se izvršava završni dio igre.

3.4.2. *redrawWindow()*

Ova metoda ključna je za prikazivanje tj. crtanje objekata na *pygame* prozor. Crtaju se igrač, neprijatelj, pozadina i statusna traka u kojoj su prikazani razina života i rezultat igre.

```
def redrawWindow(self):
    window.blit(backgroundImages[int(self.backgroundImageIndex // 1)], (0, 0))
    self.backgroundImageIndex = (self.backgroundImageIndex + 0.5) % 202

    self.player.draw(window)
    self.enemyCheck()
    self.drawGameStatusInfo()

    pygame.display.update()
```

Pozivanjem ove metode najprije se postavlja slika pozadine kojoj se pristupa iz liste slika pozadine, a nakon postavljanja pozadine uvećava se varijabla *backgroundImageIndex* koji je potreban da bi se pozadina vidjela kao animacija.

Zatim se prikazuje igrača te se pozivaju funkcije ***enemyCheck()*** i ***drawGameStatusInfo()***.

Na samom kraju poziva se funkcija ***pygame.display.update()*** koja osvježava prikaz *pygame* prozora tj. briše stare prikaze objekata i zamjenjuje ih novima prikazima. Npr. na ekranu mi vidimo kako se igrač pomiče, međutim, sve što se u programu događa jest brisanje igrača sa stare pozicije i crtanje igrača na novoj poziciji u prozoru.

3.4.3. *enemyCheck()*

Ova metoda je odgovorna za određivanje rezultata igre, razine igrača i prikazivanje eksplozije u trenutku sudara igrača i neprijatelja tj. u ovoj metodi odvija se provjera je li igrač pogođen. Metoda se može podijeliti u nekoliko blokova.

```
def enemyCheck(self):
    if self.explosionCount < 6:
        window.blit(explosion[self.explosionCount], self.explosionCoords)
        self.explosionCount += 1

    for enemy in self.enemies:
        if (enemy.x1 + enemy.width < 0 or enemy.x1 - enemy.width > windowWidth)
                or (enemy.y1 + enemy.height < 0 or enemy.y1 - enemy.height > windowHeight):
            self.enemies.pop(self.enemies.index(enemy))

        if self.player.isHit(enemy):
            self.isGameOver = self.player.health <= 0
            playerHitSoundEffect.play(0)
            self.explosionCount = 0

            self.explosionCoords = (self.player.hitbox.clip(enemy.hitbox).topleft[0] - 32,
                                    self.player.hitbox.clip(enemy.hitbox).topleft[1] - 32)
            self.enemies.pop(self.enemies.index(enemy))
        else:
            self.player.score += 0.01

        enemy.draw(window)
        enemy.move()
```

```
def enemyCheck(self):
    if self.explosionCount < 6:
        window.blit(explosion[self.explosionCount], self.explosionCoords)
        self.explosionCount += 1

    for enemy in self.enemies:
        if (enemy.x1 + enemy.width < 0 or enemy.x1 - enemy.width > windowWidth)
                or (enemy.y1 + enemy.height < 0 or enemy.y1 - enemy.height > windowHeight):
            self.enemies.pop(self.enemies.index(enemy))
    (...)
```

U početku ove metode provjerava se je li brojač *explosionCount* manji od 6. Taj brojač se koristi kod iteriranja kroz listu slika eksplozija kako bi se prikazala animacija eksplozije koje se sastoji od 6 sličica. Nakon prikaza svakog djela eksplozije brojač se uvećava za 1.

Zatim slijedi *for* petlja koja prolazi kroz listu neprijatelja i za svakog provjerava je li unutar vidljivog dijela *pygame* prozora, ako nije neprijatelj se briše iz liste neprijatelja i više se ne crta.

```
def enemyCheck(self):
    (...)
    if self.player.isHit(enemy):
        self.isGameOver = self.player.health <= 0
        playerHitSoundEffect.play(0)
        self.explosionCount = 0

        self.explosionCoords = (self.player.hitbox.clip(enemy.hitbox).topleft[0] - 32,
                                self.player.hitbox.clip(enemy.hitbox).topleft[1] - 32)
        self.enemies.pop(self.enemies.index(enemy))
    else:
        self.player.score += 0.01
    (...)
```

Nakon toga slijedi provjera je li igrač pogođen, ako je pogođen izvršava se više stvari. Prvo, kada razina života igrača padne na nulu varijabla *isGameOver* postavit će se na *True* čime će se prekinuti glavna *while* petlja i prikazati završni skočni prozor.

Zatim se pozivanjem metode ***play()*** nad objektom *playerHitSoundEffect* reproducira zvuk eksplozije, a vrijednost brojača *explosionCount* postavlja na nulu. Također, bilježe se koordinate sudara, a sam neprijatelj se briše iz liste neprijatelja kako se nebi više prikazivao jer je on nestao u sudaru.

Ako igrač nije pogođen rezultat igre se uvećava za 0.01.

```
def enemyCheck(self):
    (...)
    enemy.draw(window)
    enemy.move()
```

Na kraju metode crta se neprijatelj i nad njim se poziva metoda ***move()*** kako bi se prikazalo kretanje neprijatelja po *pygame* prozoru.

3.4.4. *drawGameStatusInfo()*

Ova metoda se poziva za crtanje statusne trake u kojoj je prikazana razina života igrača i trenutni rezultat igre.

```
def drawGameStatusInfo(self):
    damageBar = pygame.Rect(int(windowWidth // 2 - 300), 30, 600, 25)
    healthBar = pygame.Rect(int(windowWidth // 2 - 300), 30, int(600 * self.player.health /
                                                                 self.player.maxHealth), 25)
    pygame.draw.rect(window, red, damageBar)
    pygame.draw.rect(window, (0, 180, 0), healthBar)
    window.blit(self.playerImage, (healthBar.midright[0] - self.playerImage.get_width() // 2,
                        healthBar.centery - self.playerImage.get_height() // 2))
    healthText = fontGameStatusBar.render(str(round(self.player.health / 5, 1)) + "%", 100, white)
    window.blit(healthText, (damageBar.centerx - healthText.get_width() // 2,
                    damageBar.centery - healthText.get_height() // 2))

    text = font.render("Score: " + str(round(self.player.score, 0)), 100, white)
    window.blit(text, (windowWidth - text.get_width() - 50, 30))
```

```
def drawGameStatusInfo(self):
    damageBar = pygame.Rect(int(windowWidth // 2 - 300), 30, 600, 25)
    healthBar = pygame.Rect(int(windowWidth // 2 - 300), 30, int(600 * self.player.health /
                                                                 self.player.maxHealth), 25)
    pygame.draw.rect(window, red, damageBar)
    pygame.draw.rect(window, (0, 180, 0), healthBar)
    window.blit(self.playerImage, (healthBar.midright[0] - self.playerImage.get_width() // 2,
                        healthBar.centery - self.playerImage.get_height() // 2))
    (...)
```

Prvo se definiraju dva pravokutnika *healthBar* i *damageBar*. *damageBar* je pravokutnik crvene boje i on se ne mijenja tokom igre, a *healthBar* je pravokutnik zelene boje koji u početku prekriva *damageBar*, ali svakim sudarom neprijatelja i igrača njegova širina se smanjuje tako da se vidi dio *damageBar-a*. Širina *healthBar-a* proporcionalna je razini života. Također, na kraju *healthBar-a* crta se ikona igrača.

```
def drawGameStatusInfo(self):
    (...)
    healthText = fontGameStatusBar.render(str(round(self.player.health / 5, 1)) + "%", 100, white)
    window.blit(healthText, (damageBar.centerx - healthText.get_width() // 2,
                    damageBar.centery - healthText.get_height() // 2))
    (...)
```

Zatim se na sredinu pravokutnika crta tekst koji u obliku postotka prikazuje razinu života igrača.

```
def drawGameStatusInfo(self):
    (...)
    text = font.render("Score: " + str(round(self.player.score, 0)), 100, white)
    window.blit(text, (windowWidth - text.get_width() - 50, 30))
```

Na kraju se crta tekst s rezultatom igre u gornji desni kut *pygame* prozora.

3.4.5. *gameOver()*

Ovo je završna metoda klase *Game* koja se poziva kada razina života igrača padne na nulu.

```
def gameOver(self):
    pygame.mixer.music.stop()
    gameOverSoundEffect.play(0)
    k = 10
    delay = int(gameOverSoundEffect.get_length() * 1000 // k) - 75
    for i in range(0, k + 1):
        pygame.time.delay(delay)
        rect = pygame.Rect(0, 0, windowWidth, (i + 1) * windowHeight // k)
        pygame.draw.rect(window, black, rect)
        pygame.display.update()

    pygame.display.quit()
    pygame.quit()
    score = round(self.player.score, 0)
    del(self)
    ender = Ender(Tk(), score)
    ender.mainloop()
```

Pozivanjem funkcije ***pygame.mixer.music.stop()*** zaustavlja se pozadinska glazba te se pozivanjem metode ***play()*** nad objektom *gameOverSoundEffect* reproducira melodija za kraj igre.

Zatim se ulazi u *for* petlju kojom se prikazuje završna animacija. Petlja se ponavlja 11 puta, s time da je između svake iteracije pauza od *delay* milisekundi kako bi se uskladilo trajanje animacije sa melodijom za kraj igre.

Nakon toga pozivaju se metode ***pygame.display.quit()*** i ***pygame.quit()*** kojima se zatvara *pygame* prozor i deinicijaliziraju svi *pygame* moduli. Definira se varijabla *score* u koju se sprema rezultat igre zaokružen na jednu decimalu radi lakšeg prikaza u završnom skočnom prozoru.

Na samom kraju instancira se novi objekt klase *Ender* i poziva metoda ***mainloop()*** nad njim čime se prikazuje završni skočni prozor.

3.5. Ender

Ender je klasa čiji se objekti instanciraju pri završetku igre pozivanjem klase: ***Ender()***. Instanciranjem objekata klase *Ender* pokreće se završni dio programa koji se sastoji od **završnog skočnog prozora**. Ova klasa koristi *tkinter* modul za prikazivanje prozora. Klasa *Ender* importira pomoćnu datoteku *Loader.py* u kojoj su učitane datoteke poput slika, audio zapisa i sl. *(Više u poglavlju **3.6 Loader**).*

```
class Ender(Frame):
    def __init__(self, root, score):
        self.root = root
        self.root.title('Try2Survive')  # postavlja naslov prozora
        self.root.resizable(False, False)
        self.width = 400
        self.height = 180

        self.root.geometry(str(self.width) + 'x' + str(self.height) + '+' + str(windowWidth // 2 - self.width // 2)
                           + '+' + str(windowHeight // 2 - self.height // 2))
        super().__init__(self.root)

        self.font = ("Comic Sans MS", 18, "italic")
        self.instructions = ["Game over. Your score is: " + str(score),
                             "\nStart again?"]

        self.grid(column=2, sticky=N+S+E+W)
        self.createInterface()
        return
```

U inicijalizaciji se definira *tkinter* prozor, naslov prozora, njegove dimenzije i koordinate na ekranu, mogućnost mijenjanja veličine prozora. Zatim se broj stupaca u prozoru postavlja na 2, definira se font kojim će se prikazati upute u prozoru te same upute koje se sastoje od završne poruke i konačnog rezultata igre, a na kraju se poziva metoda ***createInterface()***.

3.5.1. createInterface()

```
def createInterface(self):
    for i in range(0, len(self.instructions)):
        label = Label(self.root, text=self.instructions[i], bg='black', fg='gold', font=self.font)
        label.grid(row=i, column=0, columnspan=2, sticky=N+S+W+E)
        self.root.grid_columnconfigure(0, weight=1)

    button1 = Button(self.root, text='NO', bg='gold', fg='black', font=self.font, command=self.closeWindow)
    button1.grid(row=i + 1, column=0, sticky=N + S + W + E)
    self.root.grid_columnconfigure(0, weight=1)

    button2 = Button(self.root, text='YES', bg='gold', fg='black', font=self.font, command=self.startAgain)
    button2.grid(row=i + 1, column=1, sticky=N + S + W + E)
    self.root.grid_columnconfigure(1, weight=1)
```

For petlja prolazi kroz listu s uputama i postavlja svaki dio upute u poseban redak kako bi tekst upute stao u prozor, a ispod upute se dodaju dva gumba, *„Yes“* i *„No“*, kojima se pokreće nova igra ili samo zatvara igra. Klikom na gumb *„Yes“* poziva se metoda ***startAgain()*** čime se pokreće nova igra, a klikom na gumb *„No“* poziva se metoda ***closeWindow()*** čime se zatvara *tkinter* prozor i sam program.

3.5.2. *startAgain()*

```
def startAgain(self):
    self.root.destroy()
    pygame.init()
    loadData()
    from Game import Game
    game = Game()
    game.main()
    return True
```

Pozivom funkcije ***self.root.destroy()*** zatvara se *tkinter* prozor te se s ***pygame.init()*** inicijalizira *pygame* modul. Nakon toga se poziva funkcija ***loadData()*** pomoćne datoteke *Loader.py* kojom se učitavaju potrebne datoteke za ponovno pokretanje igre.

Zatim se učita klasa *Game* kako bi se mogao instancirati novi objekt te klase, a pozivom metode ***main()*** nad novim objektom pokreće se nova igra.

Klasa *Game* importira se tek neposredno prije instanciranja novog objekta te klase jer kada bi se ta klasa učitavala na samom početku datoteke došlo bi do beskonačne petlje kružnog importiranja jer klasa *Game* importira klasu *Ender* koja ponovo importira klasu *Game* itd. što na kraju rezultira greškom u programu.

3.5.3. *closeWindow()*

```
def closeWindow(self):
    self.root.destroy()
    return False
```

Pozivom funkcije ***self.root.destroy()*** zatvara se *tkinter* prozor te se igra završava kao i sam program.

3.6. Loader

Loader je pomoćna datoteka u kojoj se učitavaju sve potrebne datoteke poput slika igrača, neprijatelja…, audio zapisa, fontova i sl. Importiranjem datoteke *Loader.py* u neku klasu automatski se izvršava i kod za učitavanje datoteka. Datoteka *Loader* učitava i pomoćnu datoteku *Config.py*. *(Više u poglavlju **3.7 Config**).*

```
# window = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
window = pygame.display.set_mode((windowWidth, windowHeight))
pygame.display.set_caption("Try2Survive")

backgroundImageNames = ['Assets/background/space' + str(i) + '.png' for i in range(1, 203)]
backgroundImages = [pygame.transform.scale(pygame.image.load(image).convert_alpha(),
                                (windowWidth, windowHeight)) for image in backgroundImageNames]

explosion = [pygame.image.load('Assets/explosion/explosion1.png'),
             pygame.image.load('Assets/explosion/explosion2.png'),
             pygame.image.load('Assets/explosion/explosion3.png'),
             pygame.image.load('Assets/explosion/explosion4.png'),
             pygame.image.load('Assets/explosion/explosion5.png'),
             pygame.image.load('Assets/explosion/explosion6.png')]

backgroundTrack = pygame.mixer.music.load('Assets/audio/backgroundTrack.wav')
pygame.mixer.music.set_volume(0.3)

playerHitSoundEffect = pygame.mixer.Sound('Assets/audio/explosion.wav')
playerHitSoundEffect.set_volume(0.8)

gameOverSoundEffect = pygame.mixer.Sound('Assets/audio/game_over.wav')
gameOverSoundEffect.set_volume(0.8)

font = pygame.font.SysFont("avenirnextttc", 40, True, True)
fontGameStatusBar = pygame.font.SysFont("avenirnextttc", 30, True, True)
```

Na početku se definira *pygame* prozor, njegove dimenzije *windowWidth, windowHeight* koje se nalaze u pomoćnoj datoteci *Config.py* i naslov prozora. Zatim se učitavaju slike pozadinske animacije i eksplozije, pozadinska glazba i zvučni efekti te se postavlja njihova glasnoća. Na samom kraju učitavaju se potrebni fontovi.

Također, isti kod je smješten u funkciju ***loadData()*** te datoteke kako bi se moglo učitati datoteke i pomoću funkcije. Razlog tomu je što jednom kad u neku klasu importiramo datoteku *Loader* nakon toga ju ne možemo više importirati pa samim time ne možemo ponovno učitati potrebne datoteke, stoga postoji funkcija *loadData()* čijim se pozivanjem učitavaju te datoteke. Ovo je nužno kod pokretanja nove igre nakon završetka stare.

3.7. Config

Config.py je pomoćna datoteka u kojoj se izvršava inicijalizacija *pygame* modula i postavljaju varijable *windowWidth i windowHeight te definiraju osnovne boje bijela, crna, crvena, zelena i plava.*

```
import pygame
pygame.init()

infoObject = pygame.display.Info()
windowWidth, windowHeight = infoObject.current_w, infoObject.current_h

framerate = 60

black = (0, 0, 0)
white = (255, 255, 255)
red = (255, 0, 0)
green = (0, 255, 0)
blue = (0, 0, 255)
```

Najprije se importira *pygame* modul te se zatim vrši njegova inicijalizacija. Zatim se pozivanjem funkcije ***pygame.display.Info()*** učitavaju informacije o ekranu te se iz njih dohvaćaju širina i visina ekrana na kojem se program pokreće. Postavlja se *framerate* igre i definiraju osnovne boje.