RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes un infromācijas tehnoloģijas fakultāte

Rinalds Mackevičs, Students;201RDB354

Ņikita Rjabovs, Students, 201DDB017

Valentīns Stepanovs, Students, 161RMC161

Andrejs Gavriļuks, Students, 211RDB145

Ņikita Hramčenko, Students, 211RDB396

RĪGA, 2022/2023. m.g.

**SATURS**

[**Spēles apraksts un anotācija** 3](#_Toc163387285)

[**1.** **Programmas kods:** 5](#_Toc163387286)

[1.1. Programmas koda apraksts 5](#_Toc163387287)

[1.2. Pilns kods 15](#_Toc163387288)

[**2.** **Minimax un Alfa-beta algoritms** 22](#_Toc163387289)

[2.1. Minimax algoritms 22](#_Toc163387290)

[2.2. Alfa-Beta algoritms 23](#_Toc163387291)

[**3.** **Koku ģenerators** 24](#_Toc163387292)

[3.1. Koda apraksts 24](#_Toc163387293)

[**4.** **Rezultātu analīze** 25](#_Toc163387294)

[4.2. GUI 25](#_Toc163387295)

# **Spēles apraksts un anotācija**

**Papildu prasības programmatūrai**

Spēles sākumā cilvēks-spēlētājs izvēlas, ar kuru skaitli diapazonā no 25 līdz 40 sākt spēli.

**Spēles apraksts**

Spēles sākumā ir dots cilvēka-spēlētāja izvēlētais skaitlis. Abiem spēlētājiem ir 0 punktu. Turklāt spēlē tiek izmantota spēles banka, kura sākotnēji ir vienāda ar 0. Spēlētāji veic gājienus pēc kārtas, reizinot pašreizējā brīdī esošu skaitli ar 2, 3 vai 4. Ja reizināšanas rezultātā tiek iegūts pāra skaitlis, tad no spēlētāja punktu skaita tiek atņemts 1 punkts, bet ja nepāra skaitlis – tad 1 punkts tiek pieskaitīts. Savukārt, ja tiek iegūts skaitlis, kas beidzas ar 0 vai 5, tad spēles bankai tiek pieskaitīts 1 punkts. Spēle beidzas, kad ir iegūts skaitlis, kas ir lielāks par vai vienāds ar 5000. Spēlētājs, pēc kura gājiena spēle beidzas, iztukšo banku, saviem punktiem pieskaitot bankas punktus. Uzvar spēlētājs, kam ir vairāk punktu spēles beigās. Ja punktu skaits ir vienāds, tad rezultāts ir neizšķirts.

Uzdevuma nostādne:

Spēles sākumā ir dots cilvēka-spēlētāja izvēlētais skaitlis. Abiem spēlētājiem ir 0 punktu. Turklāt spēlē tiek izmantota spēles banka, kura sākotnēji ir vienāda ar 0. Spēlētāji veic gājienus pēc kārtas, reizinot pašreizējā brīdī esošu skaitli ar 2, 3 vai 4. Ja reizināšanas rezultātā tiek iegūts pāra skaitlis, tad no spēlētāja punktu skaita tiek atņemts 1 punkts, bet ja nepāra skaitlis – tad 1 punkts tiek pieskaitīts. Savukārt, ja tiek iegūts skaitlis, kas beidzas ar 0 vai 5, tad spēles bankai tiek pieskaitīts 1 punkts. Spēle beidzas, kad ir iegūts skaitlis, kas ir lielāks par vai vienāds ar 5000. Spēlētājs, pēc kura gājiena spēle beidzas, iztukšo banku, saviem punktiem pieskaitot bankas punktus. Uzvar spēlētājs, kam ir vairāk punktu spēles beigās. Ja punktu skaits ir vienāds, tad rezultāts ir neizšķirts.

**Spēles apraksts:***Sākums un spēles parametri:*

* Spēles sākumā katram spēlētājam ir no 25 līdz 40 punktiem.
* Sākot spēli, banka ir vienāda ar 0.

*Spēles norise:*

Spēlētāji veic gājienus pēc kārtas.

* Katrs gājiens ir reizināšana ar 2, 3 vai 4 pašreizējā skaitļa.
* Ja rezultāts ir pāra skaitlis, spēlētājam tiek atņemts 1 punkts; ja nepāra, tiek pieskaitīts 1 punkts.
* Ja rezultāts ir skaitlis, kas beidzas ar 0 vai 5, bankai tiek pieskaitīts 1 punkts.

*Beigu nosacījumi:*

* Spēle beidzas, ja kāds no spēlētājiem sasniedz vai pārsniedz 5000 punktu robežu.
* Kad spēle beidzas, spēlētājs, kurš izdarījis pēdējo gājienu, iztukšo banku, pieskaitot tās punktus savam rezultātam.

*Uzvaras nosacījumi:*

* Uzvar spēlētājs ar visvairāk punktiem spēles beigās.
* Ja punktu skaits ir vienāds, rezultāts ir neizšķirts.

*Papildu piezīmes:*

* Spēles parametri un nosacījumi ir jāievēro, lai nodrošinātu spēles pareizu darbību un godīgu rezultātu.
* Spēles interfeisam jābūt intuitīvam un saprotamam, lai spēlētāji varētu viegli izsekot notiekošajam un veikt gājienus.
* Nepieciešams nodrošināt spēles drošību, lai novērstu neautorizētu piekļuvi vai manipulāciju ar spēles datiem.

Komandā bija apspriests un izvelēta valoda Python. Kods tiek ievietots GitHub platofrmā, kur var apskatīt šeit: <https://github.com/nickh-dev/team_58/tree/main/team58_ready_algorithm>

Gatavs spēles kods: <https://github.com/nickh-dev/team_58/blob/main/team58_ready_algorithm/team58_game_final.py>

Gatavs koka ģenerešanas kods: <https://github.com/nickh-dev/team_58/blob/main/team58_ready_algorithm/treegenerator.py>

Prakstiskais darbs tiek organizēts izmantojot vadlīnijas, kuras tiek ievietotas estudijas. Vietne: <https://estudijas.rtu.lv/mod/hvp/view.php?id=3931512#h5pbookid=21768&chapter=h5p-interactive-book-chapter-0abb5d65-8838-4dd1-a620-50dd1459110d&section=0>

# **Programmas kods:**

## Programmas koda apraksts

import pygame

import sys

import random

* pygame: Šī bibliotēka nodrošina spēles izveidi un attēlu apstrādi Python vidē. Tā ir plaši izmantota spēļu izstrādē.
* sys: Šī bibliotēka nodrošina piekļuvi sistēmas funkcijām un resursiem, piemēram, iziet no programmas vai piekļūt argumentiem, kas padoti programmai no komandrindas.
* random: Šī bibliotēka nodrošina funkcijas, kas saistītas ar gadījuma skaitļu ģenerēšanu. Tā var būt noderīga, ja nepieciešams veikt gadījuma izvēli vai izveidot gadījuma secības.

pygame.init()

Lai Pygame bibliotēka būtu gatava lietošanai, šis kods to inicializē. Visi pygame moduļi, kas nepieciešami spēles izveidei, tiek inicializēti, kad tiek izsaukts pygame.init(). Tam ir logrīku pārvaldība, audio, vizuālie materiāli un citi elementi, kas nepieciešami videospēļu izveidei. Pygame.init() vienmēr ir jāizsauc pirms pygame lietošanas, lai pārliecinātos, ka bibliotēka ir pareizi inicializēta un sagatavota lietošanai.

screen\_width, screen\_height = 800, 900

screen = pygame.display.set\_mode((screen\_width, screen\_height))

pygame.display.set\_caption('Game with Alpha-Beta Pruning')

Šis kods nosaka spēles galvenā logrīka izmēru un nosaukumu, kā arī izveido un iestata to. Ekrāna platuma un augstuma dati tiek saglabāti pirmajā rindā. Pēc tam tiek izveidots galvenais ekrāns, izmantojot iepriekš norādītās proporcijas, izmantojot metodi pygame.display.set\_mode(). Pygame.display.set\_caption() visbeidzot maina galvenā logrīka nosaukumu. Šis nosaukums, kas bieži tiek rādīts logrīka virsrakstjoslā, var pateikt jebko par spēli vai tās saturu.

# Define colors

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

LIGHT\_GREY = (200, 70, 200)

DARK\_GREY = (30, 30, 30)

BUTTON\_COLOR = (160, 160, 160)

BUTTON\_HOVER\_COLOR = (230, 230, 230)

font = pygame.font.Font(None, 36)

Tiek definētas krāsas.

#Draws a button with text on the screen and detects clicks

def draw\_button(text, position, active\_color, inactive\_color):

    mouse = pygame.mouse.get\_pos()

    click = pygame.mouse.get\_pressed()

    button\_rect = pygame.Rect(position[0], position[1], 150, 50)

    pygame.draw.ellipse(screen, inactive\_color if button\_rect.collidepoint(mouse) else active\_color, button\_rect)

    pygame.draw.rect(screen, inactive\_color if button\_rect.collidepoint(mouse) else active\_color, button\_rect)  # Основное тело кнопки

    text\_surf = font.render(text, True, BLACK)

    text\_rect = text\_surf.get\_rect(center=(position[0] + 75, position[1] + 25))

    screen.blit(text\_surf, text\_rect)

    return button\_rect.collidepoint(mouse) and click[0] == 1

Šī funkcija nosaka klikšķus un parāda pogu ar tekstu. Pirmkārt, tas izmanto pygame.mouse.get\_pos() un pygame.mouse.get\_pressed(), lai iegūtu peles pozīciju un klikšķa notikumu datus. Pygame.Rect() pēc tam tiek izmantots, lai izveidotu taisnstūri, kas apzīmē pogu. Pygame.draw.ellipse() un Pygame.draw.rect() tiek izmantotas, lai izveidotu šo taisnstūri ekrānā, pamatojoties uz to, vai peles rādītājs atrodas uz pogas. Turklāt tiek uzzīmēts pogas teksts, tiek konstatēts klikšķis un tiek atgriezts pareizais rezultāts, kad pele atrodas virs pogas.

#Displays the main menu and handles the algorithm selection.

def main\_menu():

    screen.fill(DARK\_GREY)

    chosen\_algorithm = None

    running = True

    while running:

        start\_button\_clicked = draw\_button('START', (325, 100), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR)

        quit\_button\_clicked = draw\_button('EXIT', (325, 550), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR)

        if draw\_button('Minimax', (325, 250), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

            chosen\_algorithm = "minimax"

        elif draw\_button('Alpha-Beta', (325, 350), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

            chosen\_algorithm = "alphabeta"

        elif draw\_button('Random', (325, 450), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

            chosen\_algorithm = "random"

        if start\_button\_clicked and chosen\_algorithm is not None:

            start\_number = start\_menu()

            game\_loop(start\_number, chosen\_algorithm)

            running = False

        if quit\_button\_clicked:

            pygame.quit()

            sys.exit()

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                running = False

        pygame.display.flip()

Šī funkcija pārvalda algoritma izvēli un parāda galveno izvēlni. Izmantojot screen.fill(), pirmais solis ir aizpildīt ekrānu ar tumši pelēku nokrāsu. Tālāk tiek izveidotas divas pogas: viena spēles palaišanai un otra programmas izslēgšanai. Turklāt ir pieejami vairāki algoritmi, katrs ar pogu. Galvenā cilpa tiek pārtraukta, ja tiek nospiesta viena no pogām un tiek izvēlēts algoritms. Tādā gadījumā tiek izsaukti start\_menu() un game\_loop(). Programmatūra beidzas, ja tiek nospiesta poga "EXIT". Beidzot notiek notikumu apstrāde, un pygame.display.flip() atsvaidzina ekrānu.

#Allows the player to choose a starting number

def start\_menu():

    input\_number = ''

    running = True

    while running:

        screen.fill(BLACK)

        title\_text = font.render('Choose a starting number (25 to 40): ' + input\_number, True, WHITE)

        screen.blit(title\_text, (200, 250))

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_BACKSPACE:

                    input\_number = input\_number[:-1]

                elif event.key == pygame.K\_RETURN:

                    if 25 <= int(input\_number) <= 40:

                        return int(input\_number)

                    else:

                        input\_number = ''

                else:

                    if event.unicode.isdigit() and len(input\_number) < 2:

                        input\_number += event.unicode

        pygame.display.flip()

Izmantojot šo funkciju, spēlētājs var izvēlēties savu sākotnējo numuru. Pirmais solis ir izveidot tukšu ievades virkni ar nosaukumu ievades\_numurs un iestatīšana, kas darbojas uz True. Pēc tam spēle nonāk cilpā, kurā ekrānā parādās teksts ar melnu fonu, un lietotājam tiek piedāvāts izvēlēties sākuma skaitli no 25 līdz 40. Notikumi tiek pārvaldīti, lai noteiktu, vai spēlētājs ir ievadījis datus, nospiežot ciparu taustiņus, ENTER vai atpakaļatkāpe. Funkcija beidzas un tiek atgriezts vesels skaitlis, ja atskaņotājs ir norādījis derīgu numuru. Lietojumprogramma beidzas, ja tiek nospiesta poga "QUIT". Visbeidzot, pygame.display.flip() apgriež ekrānu.

#Calculates points and updates the bank based on the current number.

def update\_points\_and\_bank(number):

    points = 0

    bank = 0

    if number % 2 == 0:

        points -= 1

    else:

        points += 1

    if number % 10 == 0 or number % 10 == 5:

        bank += 1

    return points, bank

Pamatojoties uz pašreizējo spēles rezultātu, šī funkcija aprēķina punktus un atjauno banku. Bankas mainīgie un punkti vispirms tiek iestatīti uz 0. Pēc tam tiek noteikts, vai pašreizējais skaitlis ir nepāra vai pāra, un attiecīgi piešķir punktus (-1 vai +1). Pēc tam tiek noteikts, vai skaitlis beidzas ar 5 vai to var dalīt ar 10 bez atlikuma. ja var, banka tiek palielināta par 1. Galu galā banka un aprēķinātie punkti tiek atgriezti kā divu elementu rezultāti.

#Displays the end game screen showing final scores

def display\_end\_game\_screen(player\_points, ai\_points, game\_bank, log\_messages):

    screen.fill(BLACK)

    result\_text = f"Player points: {player\_points}, AI points: {ai\_points}, Bank: {game\_bank}"

    result\_msg = font.render(result\_text, True, WHITE)

    screen.blit(result\_msg, (screen\_width // 2 - result\_msg.get\_width() // 2, 20))

    log\_start\_y = 60

    for i, log\_message in enumerate(log\_messages[-15:], start=1):

        log\_text = font.render(log\_message, True, WHITE)

        screen.blit(log\_text, (50, log\_start\_y + i \* 20))

    pygame.display.flip()

    waiting = True

    while waiting:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                waiting = False

                pygame.quit()

                sys.exit()

            elif event.type == pygame.KEYDOWN and event.key == pygame.K\_RETURN:

                waiting = False

Šī funkcija parāda spēles beigu ekrānu, kurā ir redzami gala rezultāti. Pirmkārt, ekrāns.fill(BLACK) tiek izmantots, lai aizpildītu ekrānu ar melnu fonu. Spēlētāja punktu, datora punktu un bankas galīgie rezultāti pēc tam tiek ģenerēti kā teksts. Izmantojot screen.blit(), šis teksts tiek centrēts un parādīts ekrānā. Nākamais žurnāla ziņojums parāda jaunākos 15 žurnāla ierakstus, kas atbilst spēles notikumiem. Pygame.display.flip() atjauno pēdējo ekrānu. Pēc tam tiek izveidota cilpa, kas gaida, līdz lietotājs nospiež taustiņu ENTER, lai izietu no pēdējā ekrāna, vai nospiež krustiņu, lai beigtu spēli. Programmatūra beidzas, kad tiek nospiesta poga "QUIT".

#Evaluates the current state of the game for AI.

def evaluate(current\_number, ai\_points, player\_points, game\_bank):

    if current\_number >= 5000:

        return float('inf')

    else:

        A = 1

        B = 1

        C = 0.001

        return A \* (ai\_points - player\_points) + B \* game\_bank - C \* abs(5000 - current\_number)

Šī funkcija novērtē spēles pašreizējo situāciju, ņemot vērā datora intelektu (AI). Ja pašreizējais skaitlis ir 5000 vai lielāks, tiek atgriezta pozitīva bezgalība, kas apzīmē lielāko vērtību, kas nepieciešama, lai uzvarētu spēlē. Lai novērtētu scenāriju un izvēlētos optimālo rīcības virzienu, tiek izmantota aptuvenā novērtēšanas funkcija, kurā tiek ņemti vērā spēlētāja, bankas un datora punkti, ja to skaits ir mazāks par 5000. Koeficienti A, B un C ir izmanto šajā funkcijā, lai nodrošinātu noteiktu spēles faktoru nozīmi.

# minimax algorithm

def minimax(number, depth, ai\_points, player\_points, game\_bank, is\_maximizing):

    if number >= 5000 or depth == 0:

        return evaluate(number, ai\_points, player\_points, game\_bank), None

    if is\_maximizing:

        best\_score = float('-inf')

        best\_move = None

        for multiplier in [2, 3, 4]:

            new\_number = number \* multiplier

            new\_ai\_points, new\_game\_bank = update\_points\_and\_bank(new\_number)

            score, \_ = minimax(new\_number, depth-1, ai\_points + new\_ai\_points, player\_points, game\_bank + new\_game\_bank, False)

            if score > best\_score:

                best\_score = score

                best\_move = multiplier

        return best\_score, best\_move

    else:

        best\_score = float('inf')

        best\_move = None

        for multiplier in [2, 3, 4]:

            new\_number = number \* multiplier

            new\_player\_points, new\_game\_bank = update\_points\_and\_bank(new\_number)

            score, \_ = minimax(new\_number, depth-1, ai\_points, player\_points + new\_player\_points, game\_bank + new\_game\_bank, True)

            if score < best\_score:

                best\_score = score

                best\_move = multiplier

        return best\_score, best\_move

Minimax metode tiek realizēta ar šo funkciju. Lai novērtētu katru kustību, kas ir iespējama līdz noteiktam spēles koka struktūras dziļumam, tiek izmantots rekursīvs risinājums. Novērtēšanas funkcija tiek izsaukta, lai novērtētu spēles statusu, kad tiek sasniegts maksimālais dziļums vai beigu stāvoklis — vērtība, kas lielāka par 5000. Kustība ar vislielāko vērtību tiek izvēlēta, ja algoritms maksimizē; ja tas tiek samazināts, tiek atlasīta kustība ar zemāko vērtību. Pamatojoties uz vērtējumu un pašreizējo spēles stāvokli, par katru izspēli tiek piešķirts noteikts punktu skaits. Galu galā funkcija nodrošina gājienu, kas atbilst vislabākajam rezultātam.

# alphabeta algorithm

def alphabeta(number, depth, alpha, beta, ai\_points, player\_points, game\_bank, is\_maximizing):

    if number >= 5000 or depth == 0:

        return evaluate(number, ai\_points, player\_points, game\_bank), None

    if is\_maximizing:

        best\_score = float('-inf')

        best\_move = None

        for multiplier in [2, 3, 4]:

            new\_number = number \* multiplier

            # Assuming update\_points\_and\_bank returns AI point change and bank change

            new\_ai\_points, new\_game\_bank = update\_points\_and\_bank(new\_number)

            score, \_ = alphabeta(new\_number, depth-1, alpha, beta, ai\_points + new\_ai\_points, player\_points, game\_bank + new\_game\_bank, False)

            if score > best\_score:

                best\_score = score

                best\_move = multiplier

            alpha = max(alpha, score)

            if beta <= alpha:

                break  # Apply alpha clipping

        return best\_score, best\_move

    else:

        best\_score = float('inf')

        best\_move = None

        for multiplier in [2, 3, 4]:

            new\_number = number \* multiplier

            # Bank update

            new\_player\_points, new\_game\_bank = update\_points\_and\_bank(new\_number)

            score, \_ = alphabeta(new\_number, depth-1, alpha, beta, ai\_points, player\_points + new\_player\_points, game\_bank + new\_game\_bank, True)

            if score < best\_score:

                best\_score = score

                best\_move = multiplier

            beta = min(beta, score)

            if beta <= alpha:

                break  # Apply beta clipping

        return best\_score, best\_move

Alfabēta-beta metode tiek realizēta ar šo funkciju. Tā ir salīdzināma ar minimax metodi, taču tā arī apgriež alfa-beta, lai samazinātu novērtējamo stāvokļu skaitu. Algoritms izmanto rekursiju, lai novērtētu katru kustību, kas var tikt veikta līdz noteiktam spēles koka struktūras dziļumam. Algoritms maina alfa un beta vērtības katrā rekursīvajā izsaukumā, lai noteiktu, vai ir iespējama atzarošana. Metode aptur turpmāku izpildi un neanalizē kustību, kuras vērtība pārsniedz pašreizējo alfa-beta diapazonu. Galu galā funkcija dod gan atbilstošo kustību, gan labāko novērtējumu.

def game\_loop(start\_number,algorithm):

    current\_number = start\_number

    player\_points = 0

    ai\_points = 0

    game\_bank = 0

    is\_player\_turn = True

    algorithm\_name = "Minimax" if algorithm == "minimax" else "Alpha-Beta" if algorithm == "alphabeta" else "Random"

    log\_messages = []

    running = True

    while running:

        screen.fill(BLACK)

        # Tekst

        info\_text = font.render(f'Pionts: {current\_number} || Player: {player\_points} | AI: {ai\_points} ({algorithm\_name}) || Bank {game\_bank}', True, WHITE)

        screen.blit(info\_text, (50, 50))

        # log

        log\_y\_start = 450

        for message in log\_messages[-5:]:

            log\_text = font.render(message, True, WHITE)

            screen.blit(log\_text, (50, log\_y\_start))

            log\_y\_start += 30

        # player turn

        if is\_player\_turn and current\_number < 5000:

            choice\_made = False

            while not choice\_made:

                screen.fill(BLACK)

                screen.blit(info\_text, (50, 50))

                log\_y\_start = 500

                for message in log\_messages[-5:]:

                    log\_text = font.render(message, True, WHITE)

                    screen.blit(log\_text, (50, log\_y\_start))

                    log\_y\_start += 30

                for event in pygame.event.get():

                    if event.type == pygame.QUIT:

                        pygame.quit()

                        sys.exit()

                button\_y\_offset = 0

                button\_spacing = 55

                if draw\_button('2', (screen\_width // 2 - 75, 300 + button\_y\_offset), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

                    multiplier = 2

                    choice\_made = True

                button\_y\_offset += button\_spacing

                if draw\_button('3', (screen\_width // 2 - 75, 300 + button\_y\_offset), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

                    multiplier = 3

                    choice\_made = True

                button\_y\_offset += button\_spacing

                if draw\_button('4', (screen\_width // 2 - 75, 300 + button\_y\_offset), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

                    multiplier = 4

                    choice\_made = True

                pygame.display.flip()

            pygame.time.wait(500)

            new\_number = current\_number \* multiplier

            points, bank = update\_points\_and\_bank(new\_number)

            log\_messages.append(f"Player {multiplier}, Rezult: {new\_number} (P: {points}, Bank: {bank})")

            current\_number = new\_number

            player\_points += points

            game\_bank += bank

        # AI turn

        elif not is\_player\_turn and current\_number < 5000:

            if algorithm == "minimax":

                \_, multiplier = minimax(current\_number, 3, ai\_points, player\_points, game\_bank, True)

            elif algorithm == "alphabeta":

                \_, multiplier = alphabeta(current\_number, 3, float('-inf'), float('inf'), ai\_points, player\_points, game\_bank, True)

            elif algorithm == "random":

                multiplier = random.choice([2, 3, 4])

            new\_number = current\_number \* multiplier

            points, bank = update\_points\_and\_bank(new\_number)

            log\_messages.append(f"AI  {multiplier}, Result: {new\_number} (P: {points}, Bank: {bank})")

            current\_number = new\_number

            ai\_points += points

            game\_bank += bank

        # check for end

        if current\_number >= 5000:

            if not is\_player\_turn:

                ai\_points += game\_bank

                log\_messages.append(f"Game over: AI reached {current\_number}. Bank points ({game\_bank}) added to AI.")

            else:

                player\_points += game\_bank

                log\_messages.append(f"Game over: Player reached {current\_number}. Bank points ({game\_bank}) added to Player.")

            break

        # If the game continues, the turn changes

        is\_player\_turn = not is\_player\_turn

    display\_end\_game\_screen(player\_points, ai\_points, game\_bank, log\_messages)

Primārā spēles cilpa, kas mainās starp spēlētāju un datoru, līdz tiek izpildīts spēles beigu nosacījums (kad skaitlis sasniedz 5000), tiek īstenots ar šo kodu. Pirmkārt, žurnāla ziņojumiem tiek izveidoti tukši saraksti un tiek iestatīti spēles iestatījumi. Kamēr skriešana ir patiesa, cilpa nekad nebeigsies. Informācijas teksts un iepriekšējie pieci žurnāla ziņojumi tiek parādīti ekrānā pēc katras cilpas iterācijas. Gadījumā, ja ir pienākusi spēlētāja kārta, viņš var izvēlēties, kuru skaitli reizināt ar pašreizējo skaitli, izmantojot interaktīvo pogu izvēlni. Datora pārvietošanas gadījumā tiek izvēlēts nejaušs skaitlis vai tiek izmantots minimax vai alfa-beta algoritms. tiek pievienoti jauni žurnāla ziņojumi, tiek aprēķināti jauni punkti un bankas utt. Kad spēle beidzas, tiek attēlots beigu ekrāns ar galīgajiem rezultātiem un žurnāla ziņojumiem.

def main():

    main\_menu()

Šī funkcija ir spēles galvenā funkcija, kas tiek izsaukta, lai sāktu spēli. Tā vienkārši izsauc main\_menu() funkciju, kas atbild par galvenā izvēlnes attēlošanu un galveno algoritmu izvēli. Šī funkcija palīdz spēlei uzsākties un vadīt spēles plūsmu no sākuma.

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Šis ir standarta Python kods, kas pārbauda, vai šis fails tiek palaists kā galvenais fails (main script). Ja tas tiek palaists kā galvenais fails, tad tiek izsaukta main() funkcija, kas uzsāk spēli. Tas nodrošina, ka, ja fails tiek importēts kā modulis citā programmā, main() funkcija netiek automātiski izsaukta, bet tikai tad, ja tas tiek palaists tieši kā galvenais skripts.

## Pilns kods

import pygame

import sys

import random

pygame.init()

screen\_width, screen\_height = 800, 900

screen = pygame.display.set\_mode((screen\_width, screen\_height))

pygame.display.set\_caption('Game with Alpha-Beta Pruning')

# Define colors

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

LIGHT\_GREY = (200, 70, 200)

DARK\_GREY = (30, 30, 30)

BUTTON\_COLOR = (160, 160, 160)

BUTTON\_HOVER\_COLOR = (230, 230, 230)

font = pygame.font.Font(None, 36)

#Draws a button with text on the screen and detects clicks

def draw\_button(text, position, active\_color, inactive\_color):

    mouse = pygame.mouse.get\_pos()

    click = pygame.mouse.get\_pressed()

    button\_rect = pygame.Rect(position[0], position[1], 150, 50)

    pygame.draw.ellipse(screen, inactive\_color if button\_rect.collidepoint(mouse) else active\_color, button\_rect)

    pygame.draw.rect(screen, inactive\_color if button\_rect.collidepoint(mouse) else active\_color, button\_rect)

    text\_surf = font.render(text, True, BLACK)

    text\_rect = text\_surf.get\_rect(center=(position[0] + 75, position[1] + 25))

    screen.blit(text\_surf, text\_rect)

    return button\_rect.collidepoint(mouse) and click[0] == 1

#Displays the main menu and handles the algorithm selection.

def main\_menu():

    screen.fill(DARK\_GREY)

    chosen\_algorithm = None

    running = True

    while running:

        start\_button\_clicked = draw\_button('START', (325, 100), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR)

        quit\_button\_clicked = draw\_button('EXIT', (325, 550), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR)

        if draw\_button('Minimax', (325, 250), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

            chosen\_algorithm = "minimax"

        elif draw\_button('Alpha-Beta', (325, 350), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

            chosen\_algorithm = "alphabeta"

        elif draw\_button('Random', (325, 450), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

            chosen\_algorithm = "random"

        if start\_button\_clicked and chosen\_algorithm is not None:

            start\_number = start\_menu()

            game\_loop(start\_number, chosen\_algorithm)

            running = False

        if quit\_button\_clicked:

            pygame.quit()

            sys.exit()

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                running = False

        pygame.display.flip()

#Allows the player to choose a starting number

def start\_menu():

    input\_number = ''

    running = True

    while running:

        screen.fill(BLACK)

        title\_text = font.render('Choose a starting number (25 to 40): ' + input\_number, True, WHITE)

        screen.blit(title\_text, (200, 250))

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_BACKSPACE:

                    input\_number = input\_number[:-1]

                elif event.key == pygame.K\_RETURN:

                    if 25 <= int(input\_number) <= 40:

                        return int(input\_number)

                    else:

                        input\_number = ''

                else:

                    if event.unicode.isdigit() and len(input\_number) < 2:

                        input\_number += event.unicode

        pygame.display.flip()

#Calculates points and updates the bank based on the current number.

def update\_points\_and\_bank(number):

    points = 0

    bank = 0

    if number % 2 == 0:

        points -= 1

    else:

        points += 1

    if number % 10 == 0 or number % 10 == 5:

        bank += 1

    return points, bank

#Displays the end game screen showing final scores

def display\_end\_game\_screen(player\_points, ai\_points, game\_bank, log\_messages):

    screen.fill(BLACK)

    result\_text = f"Player points: {player\_points}, AI points: {ai\_points}, Bank: {game\_bank}"

    result\_msg = font.render(result\_text, True, WHITE)

    screen.blit(result\_msg, (screen\_width // 2 - result\_msg.get\_width() // 2, 20))

    log\_start\_y = 60

    for i, log\_message in enumerate(log\_messages[-15:], start=1):

        log\_text = font.render(log\_message, True, WHITE)

        screen.blit(log\_text, (50, log\_start\_y + i \* 20))

    pygame.display.flip()

    waiting = True

    while waiting:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                waiting = False

                pygame.quit()

                sys.exit()

            elif event.type == pygame.KEYDOWN and event.key == pygame.K\_RETURN:

                waiting = False

#Evaluates the current state of the game for AI.

def evaluate(current\_number, ai\_points, player\_points, game\_bank):

    if current\_number >= 5000:

        return float('inf')

    else:

        A = 1

        B = 1

        C = 0.001

        return A \* (ai\_points - player\_points) + B \* game\_bank - C \* abs(5000 - current\_number)

# minimax algorithm

def minimax(number, depth, ai\_points, player\_points, game\_bank, is\_maximizing):

    if number >= 5000 or depth == 0:

        return evaluate(number, ai\_points, player\_points, game\_bank), None

    if is\_maximizing:

        best\_score = float('-inf')

        best\_move = None

        for multiplier in [2, 3, 4]:

            new\_number = number \* multiplier

            new\_ai\_points, new\_game\_bank = update\_points\_and\_bank(new\_number)

            score, \_ = minimax(new\_number, depth-1, ai\_points + new\_ai\_points, player\_points, game\_bank + new\_game\_bank, False)

            if score > best\_score:

                best\_score = score

                best\_move = multiplier

        return best\_score, best\_move

    else:

        best\_score = float('inf')

        best\_move = None

        for multiplier in [2, 3, 4]:

            new\_number = number \* multiplier

            new\_player\_points, new\_game\_bank = update\_points\_and\_bank(new\_number)

            score, \_ = minimax(new\_number, depth-1, ai\_points, player\_points + new\_player\_points, game\_bank + new\_game\_bank, True)

            if score < best\_score:

                best\_score = score

                best\_move = multiplier

        return best\_score, best\_move

# alphabeta algorithm

def alphabeta(number, depth, alpha, beta, ai\_points, player\_points, game\_bank, is\_maximizing):

    if number >= 5000 or depth == 0:

        return evaluate(number, ai\_points, player\_points, game\_bank), None

    if is\_maximizing:

        best\_score = float('-inf')

        best\_move = None

        for multiplier in [2, 3, 4]:

            new\_number = number \* multiplier

            new\_ai\_points, new\_game\_bank = update\_points\_and\_bank(new\_number)

            score, \_ = alphabeta(new\_number, depth-1, alpha, beta, ai\_points + new\_ai\_points, player\_points, game\_bank + new\_game\_bank, False)

            if score > best\_score:

                best\_score = score

                best\_move = multiplier

            alpha = max(alpha, score)

            if beta <= alpha:

                break

        return best\_score, best\_move

    else:

        best\_score = float('inf')

        best\_move = None

        for multiplier in [2, 3, 4]:

            new\_number = number \* multiplier

            new\_player\_points, new\_game\_bank = update\_points\_and\_bank(new\_number)

            score, \_ = alphabeta(new\_number, depth-1, alpha, beta, ai\_points, player\_points + new\_player\_points, game\_bank + new\_game\_bank, True)

            if score < best\_score:

                best\_score = score

                best\_move = multiplier

            beta = min(beta, score)

            if beta <= alpha:

                break

        return best\_score, best\_move

def game\_loop(start\_number,algorithm):

    current\_number = start\_number

    player\_points = 0

    ai\_points = 0

    game\_bank = 0

    is\_player\_turn = True

    algorithm\_name = "Minimax" if algorithm == "minimax" else "Alpha-Beta" if algorithm == "alphabeta" else "Random"

    log\_messages = []

    running = True

    while running:

        screen.fill(BLACK)

        # Tekst

        info\_text = font.render(f'Pionts: {current\_number} || Player: {player\_points} | AI: {ai\_points} ({algorithm\_name}) || Bank {game\_bank}', True, WHITE)

        screen.blit(info\_text, (50, 50))

        # log

        log\_y\_start = 450

        for message in log\_messages[-5:]:

            log\_text = font.render(message, True, WHITE)

            screen.blit(log\_text, (50, log\_y\_start))

            log\_y\_start += 30

        # player turn

        if is\_player\_turn and current\_number < 5000:

            choice\_made = False

            while not choice\_made:

                screen.fill(BLACK)

                screen.blit(info\_text, (50, 50))

                log\_y\_start = 500

                for message in log\_messages[-5:]:

                    log\_text = font.render(message, True, WHITE)

                    screen.blit(log\_text, (50, log\_y\_start))

                    log\_y\_start += 30

                for event in pygame.event.get():

                    if event.type == pygame.QUIT:

                        pygame.quit()

                        sys.exit()

                button\_y\_offset = 0

                button\_spacing = 55

                if draw\_button('2', (screen\_width // 2 - 75, 300 + button\_y\_offset), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

                    multiplier = 2

                    choice\_made = True

                button\_y\_offset += button\_spacing

                if draw\_button('3', (screen\_width // 2 - 75, 300 + button\_y\_offset), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

                    multiplier = 3

                    choice\_made = True

                button\_y\_offset += button\_spacing

                if draw\_button('4', (screen\_width // 2 - 75, 300 + button\_y\_offset), BUTTON\_COLOR, BUTTON\_HOVER\_COLOR):

                    multiplier = 4

                    choice\_made = True

                pygame.display.flip()

            pygame.time.wait(500)

            new\_number = current\_number \* multiplier

            points, bank = update\_points\_and\_bank(new\_number)

            log\_messages.append(f"Player {multiplier}, Rezult: {new\_number} (P: {points}, Bank: {bank})")

            current\_number = new\_number

            player\_points += points

            game\_bank += bank

        # AI turn

        elif not is\_player\_turn and current\_number < 5000:

            if algorithm == "minimax":

                \_, multiplier = minimax(current\_number, 3, ai\_points, player\_points, game\_bank, True)

            elif algorithm == "alphabeta":

                \_, multiplier = alphabeta(current\_number, 3, float('-inf'), float('inf'), ai\_points, player\_points, game\_bank, True)

            elif algorithm == "random":

                multiplier = random.choice([2, 3, 4])

            new\_number = current\_number \* multiplier

            points, bank = update\_points\_and\_bank(new\_number)

            log\_messages.append(f"AI  {multiplier}, Result: {new\_number} (P: {points}, Bank: {bank})")

            current\_number = new\_number

            ai\_points += points

            game\_bank += bank

        # check for end

        if current\_number >= 5000:

            if not is\_player\_turn:

                ai\_points += game\_bank

                log\_messages.append(f"Game over: AI reached {current\_number}. Bank points ({game\_bank}) added to AI.")

            else:

                player\_points += game\_bank

                log\_messages.append(f"Game over: Player reached {current\_number}. Bank points ({game\_bank}) added to Player.")

            break

        # If the game continues, the turn changes

        is\_player\_turn = not is\_player\_turn

    display\_end\_game\_screen(player\_points, ai\_points, game\_bank, log\_messages)

def main():

    main\_menu()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# **Minimax un Alfa-beta algoritms**

## Minimax algoritms

Viens no populārākajiem algoritmiem lēmumu pieņemšanā, īpaši spēļu teorijā, ir Minimax algoritms. Nulles summas spēlēs, kur viens spēlētājs cenšas palielināt savu lietderību, bet otrs to samazināt, šis paņēmiens tiek izmantots, lai noteiktu labāko rīcības veidu. Šajās spēlēs ir iekļautas daudzas labi zināmas spēles, piemēram, tic tac toe, dambrete un šahs.

Minimax algoritms izmanto rekursīvu dziļuma meklēšanas paņēmienu, lai atrastu labāko kustību. Visas iedomājamās kustības spēles koka struktūrā tiek mainītas līdz noteiktam dziļumam kopā ar visiem iespējamiem spēles stāvokļiem. Novērtēšanas funkcija tiek izmantota, lai noteiktu spēles pašreizējo stāvokli, kad ir sasniegta dziļuma robeža vai beigu stāvoklis. Ja stāvoklis ir galīgs vai sasniegts dziļuma limits, tiek izvērtēta spēles stāvokļa vērtība.

Spēlēs, kurās divi pretinieki veic secīgas kustības, cenšoties palielināt paši savu laimestu un samazināt pretinieka peļņu, izvēles noteikšanai tiek izmantots Minimax algoritms. Datorprogrammu izstrāde, kas var konkurēt ar cilvēkiem vai citām datorsistēmām, ir izplatīts spēļu teorijas un mākslīgā intelekta pielietojums.

Viena no spēles tehniskajām daļām šajā kodā ir minimax algoritms. Izmantojot minimizēšanas un maksimizācijas paņēmienu, algoritms nosaka optimālo gājienu un izvēlas to, kas samazina spēlētāja punktu skaitu, vienlaikus palielinot datora. Rezultātā minimax algoritms ir ļoti svarīgs spēles rezultātam, jo tas palīdz datoram pieņemt optimālos lēmumus, ņemot vērā konkrētos apstākļus.



<https://static.javatpoint.com/tutorial/ai/images/mini-max-algorithm-in-ai-step1.png>

## Alfa-Beta algoritms

Alfa-beta metode ir pilnveidota minimax algoritma versija, kas, ierobežojot pārbaudāmo spēļu stāvokļu skaitu, efektīvi samazina nevajadzīgu stāvokļu izpēti. Lai gan tas ir balstīts uz minimax algoritmu, tas nodrošina efektīvāku meklēšanas pieeju, lai samazinātu bezjēdzīgas darbības un uzlabotu algoritma efektivitāti.

Minimax metodes fāzes tiek veiktas, izmantojot alfa-beta algoritmu, kā arī turpmākas optimizācijas. Metode ierobežo labāko un sliktāko prognozēto rezultātu diapazonu, izmantojot divus parametrus: "alfa" un "beta". Programma lieki nepēta alternatīvas kustības, kurām nav ietekmes uz alfa vai beta vērtībām, ja kāda no prognozētajām iespējām izrādās labāka vai sliktāka par pašreizējo vērtību.

Pieņemot lēmumus, bieži tiek izmantota alfa-beta metode, īpaši mākslīgā intelekta un spēļu teorijā. Tas var ievērojami palielināt datora spēju spēlēt dažādas spēles pret cilvēkiem vai citām datorprogrammām. Tas ir īpaši labs, lai noteiktu labākos gājienus spēlēs ar lielu stāvokļu skaitu.

Šis kods veic datora kustības "Numuru spēlē", izmantojot alfa-beta algoritmu, nevis minimax stratēģiju. Tas samazina resursu daudzumu, kas nepieciešams, lai atrastu ideālo atbildi, un palīdz ātrāk noteikt atbilstošās kustības, ņemot vērā pašreizējo spēles stāvokli. Rezultātā alfa-beta algoritms uzlabo datora spēju konkurēt ar cilvēku, piedāvājot praktisku atbildi spēļu laikā.

### **Koku ģenerators**

Kamēr nav sasniegts iepriekš noteikts maksimālais dziļums vai spēle ir beigusies, spēles koka izveidošanai tiek izmantota katra iespējamā kustību kombinācija. Tas ļauj algoritmam izpildīt ideālo kustību, kas jāveic, lai iegūtu augstāko iespējamo punktu skaitu.

Izmantojot minimax tehniku, lai izveidotu spēļu koku, izstrādātāji var noteikt labākos iespējamos spēļu risinājumus.

## Koda apraksts

from graphviz import Digraph

Digraph klase, kas ļauj vizualizēt grafiku, tiek importēta ar šo kodu no graphviz pakotnes. Ja jums ir nepieciešams grafiski attēlot dažādas diagrammas vai datu struktūras.

def minimax\_visualize(current\_number, depth, graph, parent\_name="Root", action=None):

    # Generate a node name without using the "-" symbol

    node\_name = f"{parent\_name}\_{action}\_{current\_number}" if action else f"Start\_{current\_number}"

    node\_label = f"{action}\*{current\_number}" if action else f"Start: {current\_number}"

    graph.node(node\_name, label=node\_label)

    if action:  # If the action is specified, add an edge to the graph

        graph.edge(parent\_name, node\_name)

    # Stop condition: reached maximum depth or number exceeded 5000

    if depth == 0 or current\_number >= 5000:

        return

    for next\_multiplier in [2, 3, 4]:

        next\_number = current\_number \* next\_multiplier

        minimax\_visualize(next\_number, depth-1, graph, node\_name, f"\*{next\_multiplier}")

Minimax algoritms tiek parādīts kā koks, izmantojot funkciju minimax\_visualize, ko nosaka šis kods. Katrs koka mezgls apzīmē noteiktu stāvokli, kā arī iespējamos ceļus, lai sasniegtu šo stāvokli. Nodrošinot vecāku mezglu un darbības, kas jāveic, lai sasniegtu jauno mezglu, funkcija rekursīvi veido un pievieno diagrammai mezglus.

def create\_decision\_tree(start\_number=25, depth=3):

    graph = Digraph(comment='Decision Tree for Game')

    minimax\_visualize(start\_number, depth, graph)

    # Specify the output format 'pdf' and enable viewing after generation

    graph.render('decision\_tree', view=True, format='pdf')

Funkcija create\_decision\_tree, ko definē šis kods, izmanto minimax algoritmu, lai izveidotu spēles lēmumu koku. Izmantojot Dot valodu, tas izveido grafiku ar detalizētu informāciju par katra mezgla pašreizējo stāvokli un potenciālajiem darbības virzieniem. Pēc tam tas ģenerē grafiku PDF formātā. Galu galā tas palaiž saražotā koka attēlojumu.

create\_decision\_tree(25, 10)

Funkciju create\_decision\_tree šis kods izsauc ar meklēšanas dziļumu 10 un sākuma skaitli 25. Tas nozīmē, ka lēmumu koks tiks izveidots, izmantojot metodi minimax, kas meklē desmit līmeņu dziļumā, sākot ar skaitli 25. Galu galā lēmums koka vizualizācija tiks izgatavota un prezentēta kā PDF fails.

## Pilnais kods

from graphviz import Digraph

def minimax\_visualize(current\_number, depth, graph, parent\_name="Root", action=None):

    node\_name = f"{parent\_name}\_{action}\_{current\_number}" if action else f"Start\_{current\_number}"

    node\_label = f"{action}\*{current\_number}" if action else f"Start: {current\_number}"

    graph.node(node\_name, label=node\_label)

    if action:

        graph.edge(parent\_name, node\_name)

    if depth == 0 or current\_number >= 5000:

        return

    for next\_multiplier in [2, 3, 4]:

        next\_number = current\_number \* next\_multiplier

        minimax\_visualize(next\_number, depth-1, graph, node\_name, f"\*{next\_multiplier}")

def create\_decision\_tree(start\_number=25, depth=3):

    graph = Digraph(comment='Decision Tree for Game')

    minimax\_visualize(start\_number, depth, graph)

    graph.render('decision\_tree', view=True, format='pdf')

create\_decision\_tree(25, 10)

# **Rezultātu analīze**

## 4.2. GUI