1. Quiz + Ranking (podstawy)

Cel: Stringi, wektory, I/O, pliki.

1. Stwórz tablicę pytań i odpowiedzi.
2. Zadaj je w pętli.
3. Porównaj odpowiedzi po .trim().
4. Dodaj scoring.
5. Dodaj losowanie pytań (rand).
6. Dodaj możliwość ponownego uruchomienia gry.
7. Zapisuj wynik do pliku.
8. Dodaj imię gracza.
9. Wyświetl ranking z pliku (posortowany).
10. Dodaj tryb trudny (czas na odpowiedź).

2. Mini parser matematyczny

Cel: Parsowanie stringów, match, własne funkcje.

1. Odczytaj prosty string 2+3.
2. Rozdziel liczby i operator (split).
3. Dodaj obsługę + - * /.
4. Obsłuż błędne dane (wynik Result).
5. Obsłuż dzielenie przez zero.
6. Dodaj potęgowanie ^.
7. Dodaj obsługę spacji.
8. Dodaj obsługę wielu operacji (2+3*4).
9. Dodaj obsługę nawiasów (2+3)*4.
10. Dodaj testy jednostkowe parsera.

3. Szyfr Cezara

Cel: operacje na znakach, char, Unicode.

1. Wczytaj tekst i klucz.
2. Zaszyfruj litery A–Z.
3. Dodaj deszyfrowanie.
4. Obsłuż małe litery.
5. Obsłuż znaki niealfabetyczne.
6. Obsłuż polskie litery.
7. Dodaj tryb brute-force (próba wszystkich kluczy).
8. Dodaj zapis zaszyfrowanego tekstu do pliku.
9. Dodaj deszyfrowanie z pliku.
10. Dodaj automatyczne zgadywanie klucza (częstotliwość liter).

4. Symulator banku

Cel: struct, impl, mutowalność.

1. Stwórz struct Konto { imie, saldo }.
2. Dodaj metodę wpłać().
3. Dodaj metodę wypłać().
4. Dodaj metodę sprawdź_saldo().
5. Dodaj menu: wybór operacji.
6. Obsłuż brak środków.
7. Dodaj zapis stanu kont do pliku JSON.
8. Dodaj odczyt przy starcie.
9. Dodaj wielu użytkowników.
10. Dodaj przelewy między kontami.

5. Symulator cząsteczek 2D

Cel: mini projekt.

1. Zdefiniuj strukturę Particle z polami: pozycja (x, y), prędkość (vx, vy).
2. Stwórz wektor Vec<Particle> z kilkoma losowymi cząsteczkami.
3. Zdefiniuj rozmiar „świata” (np. szerokość 80, wysokość 24).
4. Zaimplementuj metodę update(&mut self, dt: f32, width: u32, height: u32) dla Particle, która:
    a. przesuwa cząsteczkę o vx * dt, vy * dt,
    b. odbija od ścian zmieniając znak prędkości przy krawędzi.
5. Napisz funkcję render(particles: &Vec<Particle>), która tworzy bufor ASCII (Vec<Vec<char>>), wypełnia go spacjami, a w miejscach cząsteczek rysuje np. *.
6. W main stwórz pętlę symulacji:
    a. czyść ekran (\x1b[2J\x1b[H),
    b. aktualizuj wszystkie cząsteczki,
    c. renderuj,
    d. std::thread::sleep(Duration::from_millis(100)).
7. Dodaj różne prędkości startowe dla każdej cząsteczki.
8. Dodaj kolizje między cząsteczkami – jeśli dwie zajmują to samo pole, zmieniają kierunek.
9. Dodaj grawitację (stała siła w dół zmieniająca vy).
10. Dodaj tryb interaktywny: klawiszami zmieniasz liczbę cząsteczek albo pauzujesz symulację.
11. Po projekcie 17 wróć do projektu dodając grafikę 2D.

6. Symulator kolejek

Cel: VecDeque, kolejki, loop.

1. Stwórz kolejkę klientów.
2. Dodaj opcję dołączenia klienta.
3. Dodaj obsługę klienta (usuwanie z kolejki).
4. Dodaj możliwość podglądu kolejki.
5. Dodaj losowe przychodzenie klientów (rand).
6. Dodaj obsługę wielu stanowisk (kilka kolejek).
7. Dodaj licznik obsłużonych klientów.
8. Dodaj średni czas oczekiwania.
9. Dodaj tryb symulacji (działa w czasie rzeczywistym).
10. Dodaj zapis logów do pliku.

7. Notatnik JSON

Cel: serde_json, pliki, Vec<struct>.

1. Stwórz struct Note { title, body }.
2. Dodaj opcję dodawania notatek.
3. Dodaj opcję wyświetlania notatek.
4. Dodaj zapis do pliku JSON.
5. Dodaj odczyt przy starcie.
6. Dodaj usuwanie notatki.
7. Dodaj edytowanie notatki.
8. Dodaj wyszukiwanie po tytule.
9. Dodaj filtrowanie po słowach.
10. Dodaj eksport do .txt.

8. Generator muzyki (MIDI / wave)

Cel: tablice, pliki binarne, matematyka dźwięku.

1. Wygeneruj tablicę sinusoidy.
2. Zapisz jako .wav.
3. Dodaj możliwość wyboru nuty.
4. Dodaj kilka częstotliwości jednocześnie.
5. Dodaj gamę muzyczną.
6. Dodaj akordy.
7. Dodaj prostą melodię z tablicy nut.
8. Dodaj zapis melodii do pliku.
9. Dodaj losowy generator melodii.
10. Dodaj tryb „Markov chain composer".

9. Chat w konsoli

Cel: std::net, TCP.

1. Uruchom serwer TCP.
2. Obsłuż jednego klienta.
3. Dodaj wysyłanie wiadomości.
4. Dodaj odbieranie wiadomości.
5. Dodaj wielu klientów (wątki).
6. Dodaj nazwy użytkowników.
7. Dodaj zapis logów do pliku.
8. Dodaj prostą komendę /users.
9. Dodaj możliwość prywatnych wiadomości.
10. Dodaj opcję zamykania serwera.

10. Algorytmy ewolucyjne (sztuczna ewolucja)

Cel: geny = stringi, losowość, mutacje.

1. Wygeneruj losowy string.
2. Oceń podobieństwo do hasła docelowego.
3. Dodaj mutacje (zmiana znaku).
4. Dodaj krzyżowanie 2 „osobników".
5. Dodaj populację N osobników.
6. Iteruj pokolenia.
7. Zapisz najlepsze do logu.
8. Dodaj selekcję najlepszych.
9. Dodaj parametr mutacji.
10. Pokaż, jak szybko algorytm znajdzie hasło.

11. Ewolucja dróg do celu (algorytm genetyczny)

Cel: pathfinding, geny jako sekwencja kroków.

1. Zbuduj prostą planszę ASCII z przeszkodami.
2. Zdefiniuj genom = lista ruchów (↑↓←→).
3. Napisz funkcję fitness = odległość od mety.
4. Wygeneruj populację losowych genomów.
5. Oceń ich fitness.
6. Wybierz najlepszych (selekcja).
7. Zaimplementuj krzyżowanie genomów.
8. Dodaj mutacje (losowe zmiany ruchów).
9. Iteruj pokolenia, pokaż progres na ekranie.
10. Zatrzymaj, gdy osobnik dotrze do mety.

12. Mini baza wiedzy z logiką (expert system)

Cel: HashMap, parser, prosty inference engine.

1. Zdefiniuj fakty (np. „kot ma futro”).
2. Dodaj reguły („jeśli X ma futro → X jest ssakiem”).
3. Zaimplementuj silnik wnioskowania.
4. Wczytaj fakty z pliku.
5. Dodaj możliwość dodawania nowych faktów.
6. Dodaj interaktywny tryb pytań.
7. Dodaj zapis nowych faktów do JSON.
8. Dodaj sprawdzanie sprzeczności.
9. Dodaj możliwość wnioskowania łańcuchowego.
10. Dodaj tryb quizu (AI pyta gracza).

13. Gra: Life (Conway’s Game of Life)

Cel: macierze, iteracje, symulacja.

1. Stwórz siatkę 20x20.
2. Wypełnij losowo.
3. Zasady: żywa komórka z 2–3 sąsiadami → żyje.
4. Martwa komórka z 3 sąsiadami → ożywa.
5. Iteruj w pętli.
6. Wyświetlaj w ASCII.
7. Dodaj możliwość ręcznego ustawienia komórek.
8. Dodaj licznik generacji.
9. Dodaj zapis i odczyt stanu z pliku.
10. Dodaj zatrzymywanie i restart symulacji.

14. Mini silnik szachowy (tekstowy)

Cel: plansza, ruchy, walidacja.

1. Stwórz planszę 8x8.
2. Dodaj figury jako symbole.
3. Dodaj ruch pionka.
4. Dodaj ruch wieży.
5. Dodaj ruch gońca.
6. Dodaj ruch hetmana.
7. Dodaj ruch skoczka.
8. Dodaj ruch króla.
9. Dodaj sprawdzanie poprawności ruchu.
10. Dodaj sprawdzanie szacha.

15. Async web scraper

Cel: async/await, concurrency, reqwest.

1. Pobierz stronę www.
2. Wypisz tytuł (<title>).
3. Wyciągnij wszystkie linki (<a href>).
4. Obsłuż błędy.
5. Dodaj async pobieranie wielu stron.
6. Dodaj limit równoległych zadań.
7. Dodaj zapis do pliku CSV.
8. Dodaj filtrowanie linków (np. tylko .pdf).
9. Dodaj progres.
10. Dodaj testy wydajności (ile stron/min).

16. Sztuczne życie w świecie 2D

Cel: symulacja biologiczna, genomy, zachowania.

1. Stwórz planszę 2D z losowym jedzeniem.
2. Zdefiniuj osobnika: pozycja + geny (prędkość, zasięg widzenia, „charakter").
3. Dodaj ruch losowy.
4. Dodaj zbieranie jedzenia = punkty fitness.
5. Po każdej turze słabsze osobniki giną.
6. Najlepsi rozmnażają się (krzyżowanie genów).
7. Dodaj mutacje (np. większa prędkość, krótsze życie).
8. Iteruj pokolenia, pokazuj statystyki.
9. Dodaj drapieżników (osobniki, które zjadają inne).
10. Pokaż ewolucję w czasie → ekosystem sam się rozwija.

17. Symulator rzutów pocisków

Cel: symulacja fizyczna.

1. Zdefiniuj parametry początkowe: prędkość v0, kąt theta (w radianach), grawitacja g.
2. Oblicz składowe prędkości:
    a. vx = v0 * cos(theta)
    b. vy = v0 * sin(theta)
3. Stwórz zmienne x, y = (0,0) jako pozycję pocisku.
4. Ustal krok czasu dt (np. 0.1 s).
5. W pętli: t += dt, x += vx * dt, y += vy * dt - 0.5 * g * dtˆ2, vy -= g * dt.
6. Zapisuj kolejne (x,y) w Vec<(f32,f32)>.
7. Gdy y <= 0, przerwij pętlę (pocisk spadł).
8. Stwórz bufor ASCII: siatkę 2D (np. szerokość 80, wysokość 20).
9. Zaznacz punkty toru lotu * w siatce.
10. Wypisz bufor w konsoli – zobaczysz parabolę.

18. Snake

Cel: projekt końcowy.

1. Zdefiniuj strukturę Point { x: i32, y: i32 }.
2. Stwórz Vec<Point> reprezentujący ciało węża – pierwszy element to głowa.
3. Ustal początkowy kierunek (np. enum Direction { Up, Down, Left, Right }).
4. W pętli gry:
    - dodaj nową głowę (na podstawie kierunku),
    - usuń ostatni segment ogona,
    - wstaw nową głowę na początek wektora.
5. Losuj Point dla „jedzenia” – upewnij się, że nie nachodzi na ciało węża.
6. Jeśli głowa = jedzenie: nie usuwaj ogona (wąż rośnie).
7. W render() twórz bufor ASCII z:
    - spacjami dla pustego,
    - @ dla głowy,
    - dla ciała,

    *dla jedzenia.
8. Obsłuż sterowanie – np. crate crossterm pozwala na odczyt strzałek bez Entera.
    - Sprawdź warunki końca gry:
    - wąż wyszedł poza planszę,
9. głowa weszła w swoje ciało.
10. Dodaj drobne rozszerzenia:
    - licznik punktów,
    - rosnący poziom trudności (szybsze ticki),
    - restart gry po przegranej.