Dinozaury w lesie — projekt zaliczeniowy (*Programowanie*)

MAJ/CZERWIEC 2026

Przygotowanie (21–28 maja)

Wstęp

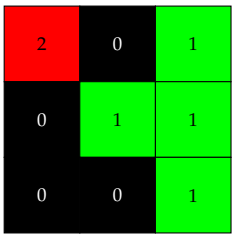
DAWNO, dawno temu... No dobrze, wcale nie tak dawno temu, bo jedyne około 100 milionów lat temu, w ostatnim okresie ery mezozoicznej, tzn. w (nietablicowej) kredzie żyły sobie nieświadome swego bliskiego spotkania z meteorytem dinozaury... *Tylko dlaczego około 100 milionów lat temu?* — zapytałby ktoś — *Wszak dinozaury powstały jeszcze wcześniej*. Ponieważ wtedy, 100 milionów lat temu, rosły już lasy, które były dość podobne do współczesnych. I część z ówczesnych dinozaurów żyła w lesie. W lesie bywało całkiem fajnie, było gdzie się schować i były roślinki do zjedzenia (mniam!). Niestety, w lesie zdarzały się też pożary — wywoływane na przykład uderzeniem pioruna. Pożary bywały różne — małe i duże, krótkotrwałe i te dłuższe. Psuły one samopoczucie dinozaurom. Aż w końcu postanowiły one coś zrobić z tym problemem. Powołano więc Dinozaurzą Straż Pożarną, której zadaniem było natychmiastowe gaszenie pożarów. Do pewnego czasu pomogło to naszym starym bohaterom, jednak któregoś razu pożar ogarnął cały las i spaliło się wszystko w okolicy. Problemem, jak się później okazało, było zbyt szybkie wcześniejsze dogaszanie małych pożarów. W ten sposób las zagęścił się do stopnia wcześniej nieznanego i wystarczyła mała iskra, aby w krótkim czasie pożar niepowstrzymywany żadnymi leśnymi odstępami w formie polan rozprzestrzenił się do samych krańców dinozaurzej puszczy.

Dinozaurom na szczęście udało się zbiec do sąsiedniego lasu, gdzie na Mezozoicznym Uniwersytecie im. Dinozaura Cudownego na Wydziale Leśnictwa, Roślinności, Spania i Jedzenia w Zakładzie Dinoobliczeń powołano specjalną grupę badawczą, która miała zająć się problemem opracowania odpowiedniej strategii gaszenia pożarów. Pytanie było bardzo proste — jak często i do jakiego stopnia należy gasić pożary, aby dinozaury mogły uchronić się od tych najgorszych? Problem do rozwiązania jednak nie był trywialny i wymagał wykonania długotrwałych obliczeń na powstałych co dopiero zaawansowanych programowalnych maszynach matematycznych.

Podstawy teorii dinoobliczeń

NASZE inteligentne dinozaury z Zakładu Dinoobliczeń podeszły do problemu bardzo profesjonalnie i po przejrzeniu zawartości uniwersyteckiej biblioteki stworzyły odpowiedni model obliczeniowy. Aby wykonać symulację wzrostu i pożaru lasu dinozaury postanowiły zaprezentować las z użyciem kwadratowej siatki złożonej z małych kwadratowych pól, które nazwane zostały *komórkami*. Każdej komórce dinozaury postanowiły przypisać jedną z trzech wartości liczbowych: 0, 1 lub 2. Wartość zero oznacza, że w danej komórce nic nie ma. Wartość jeden odpowiada komórce, w której znajduje się drzewo. Wartość dwa znajduje się w komórce objętej obecnie pożarem.

Jak te wartości zmieniają się w czasie? Na początku cała siatka składa się z samych komórek zerowych — jest to bezkresna (o czym za chwilę) przestrzeń, na której nic nie rośnie. Następnie, w każdej komórce może z pewnym prawdopodobieństwem urosnąć drzewo. W modelu dinozaurów jest to proces niezależny od otoczenia, tzn. nie ma wpływu, czy w sąsiedztwie danej komórki są już jakieś drzewa, czy też ich nie ma. Jest to pewne uproszczenie, choć nie jest ono pozbawione podstaw.

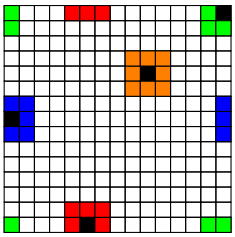


Rysunek 1: Fragment siatki obliczeniowej. Centralna komórka reprezentuje drzewo, które sąsiaduje z trzema innymi drzewami oraz z jedną komórką, która płonie. Oznacza to, że w kolejnej aktualizacji siatki, centralna komórka również się zapali.

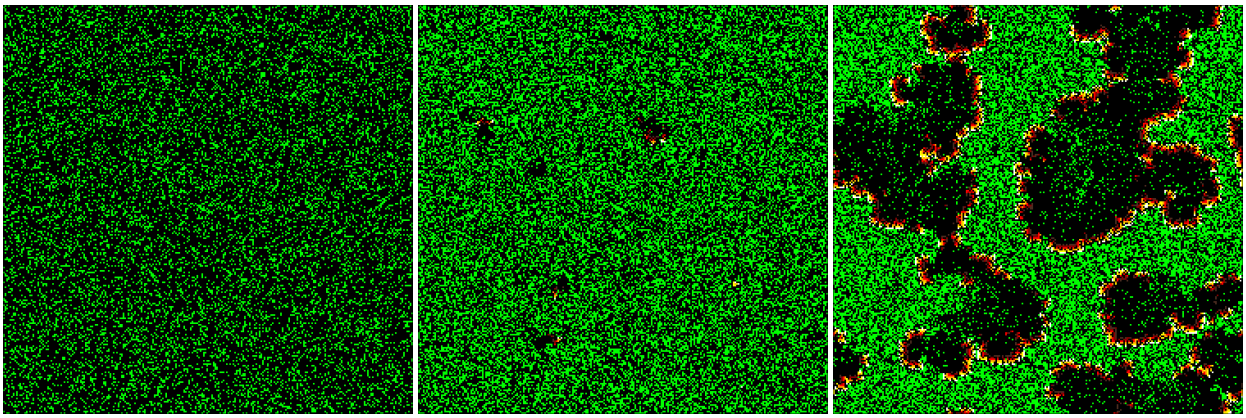
W jaki sposób dochodzi do zapłonu drzewa? Może to zajść na dwa sposoby. Pierwszym jest uderzenie pioruna — w każdą komórkę, w której znajduje się drzewo, może z pewnym prawdopodobieństwem uderzyć piorun. Drugim sposobem jest zapalenie się drzewa od ognia znajdującego się w jednej z ośmiu sąsiadów danej komórki. Sąsiadami komórki, według modelu dinozaurów są komórki z lewej, z prawej, z góry, z dołu a także po kierunkach „ukośnych", tzn. z lewej od góry, z prawej od góry, z lewej od dołu i z prawej od dołu. Można to zobaczyć na rysunku 1.

Oznacza to, że wartości pola mogą zmieniać się cyklicznie: zero (pusty obszar) po jakimś czasie przechodzi w wartość jeden (urosło drzewo), ta w pewnym momencie przechodzi w wartość dwa (w drzewo trafił piorun lub zapaliło się od sąsiedniego drzewa), a ta z kolei — w wartość zero (drzewo się spaliło i nic nie zostało). Ważne jest to, że kolejny stan siatki obliczeniowej zależy wyłącznie od stanu poprzedniego (aby zbudować siatkę obliczeniową o numerze $n+1$ należy sprawdzać stany komórek na siatce o numerze n).

Pozostał jeszcze jeden problem do rozwiązania, aby opis symulacji był kompletny. Co mianowicie zrobić z komórkami znajdującymi się na krawędziach układu? *Gdyby uznać, że te komórki są rzeczywiście na skraju siatki obliczeniowej* — argumentowały dinozaury — *to pożar na obrzeżach nie zachowuje się w naturalny sposób i to może negatywnie wpływać na słuszność wniosków wypływających z symulacji*. Nasi sprytni bohaterowie postanowili więc



Rysunek 2: Periodyczne warunki brzegowe. Dla każdej komórki o kolorze czarnym zaprezentowano jej sąsiedztwo jednolitym kolorem (czerwonym, zielonym, niebieskim i pomarańczowym).



Rysunek 3: Trzy stopklatki z przykładowej symulacji. Lewa: faza wzrostu lasu (niewielkie pożary mogły już zgasnąć samoistnie). Środkowa: początek dużego pożaru. Prawa: pożar w skali całego lasu zapoczątkowany w poprzedniej stopklatce. Kolory: czarny – obszar pusty, zielony – drzewo, biały – komórka obecnie zajęta pożarem, żółty/pomarańczowy/czerwony/bordowy/ciemny brąz – komórka pusta, która niedawno się paliła (im ciemniejszy kolor, tym dawniej był w niej pożar).

wprowadzić „periodyczne warunki brzegowe", które powodują, że granica siatki przestaje stanowić barierę. Komórka znajdująca się na krawędzi w rzeczywistości sąsiaduje również z komórkami znajdującymi się po przeciwległym brzegu siatki. Widać to na rysunku 2. W taki oto sposób niewielkim kosztem obliczeniowym można w pewnym sensie symulować nieskończone przestrzenie.

Animacje powstałe z symulacji wykonanych z użyciem opisanego modelu mogą wyglądać dość efektownie. Jeśli chcesz, obejrzyj takie animacje w postaci wideo — są one dostępne na stronie https://www.fuw.edu.pl/~ttark/www/dydaktyka/publiczne/programowanie/projekty/2025-2026/wideo/. Stopklatki z przykładowej symulacji znajdują się na rysunku 3, gdzie dla wygody obserwacji wprowadzono dodatkowe kolory informujące o komórkach, które płonęły chwilę wcześniej. Dzięki temu łatwiej obserwuje się rozwój frontu pożaru.

Na podstawie wyników symulacji dinozaury mogły opracować strategię gaszenia pożarów. Wykonały one całą serię obliczeń dla różnych wartości prawdopodobieństw wzrostu drzew i uderzenia pioruna. Na tej podstawie dinozaury wyciągnęły wnioski i opracowały odpowiednią strategię gaszenia pożarów, aby już nigdy nie doszło do katastrofy. Strategia została przekazana Dinozaurzej Straży Pożarnej, a grupa badawcza zajęła się tematyką badawczą spania i jedzenia...

Obwieszczenie Zakładu Dinoobliczeń nr 1

Obwieszczenie: Rozwiązując ten projekt pamiętaj, że będzie on sprawdzany w systemie operacyjnym GNU/Linux z wykorzystaniem kompilatora GNU C++ (g++). Do rozwiązania projektu podczas zajęć wykorzystaj program Emacs lub Vim, lub inny edytor, który nie zawiera narzędzi sztucznej inteligencji. Podczas pracy na ćwiczeniach możesz korzystać z materiałów znajdujących się na stronie wykładu i ćwiczeń w systemie Kampus, w tym z oficjalnych „ściąg", oraz ze strony https://cppreference.com/.

__Nota o autorstwie:__ Utwór poetycki jest dziełem hybrydowym, który powstał przy częściowym wsparciu modeli językowych AI. W pozostałych częściach dokumentu rola narzędzi AI ograniczyła się do drobnej korekty redakcyjnej i nie wpłynęła na merytoryczną zawartość projektu.

Pieśń Dinozaurów — rock-opera

(tajemnicza i budująca napięcie uwertura)

Tylko my, dinozaury, jesteśmy w stanie uratować ten las
(chór) *Tylko dinozaury!*
Wszystkie leśne zwierzęta pokładają nadzieję w nas
(chór) *Tylko dinozaury!*
Nie mamy wyjścia — działajmy szybko i skutecznie
(chór) *Tylko dinozaury!*
Wysiłek nasz przeogromny, wszak nie jest tu bezpiecznie
(chór) *Tylko dinozaury!*

(krótka, samotna, mocna i dramatyczna partia gitary elektrycznej)

Gdy pożar niszczy norkę Twą — jest źle!
Gdy spiżarnia Twa w dym obraca się — obudź się!
Nie uciekaj w mrok, nie uciekaj w las
przed ogniem schroń się wczas!

(długa partia instrumentalna przechodzi od masywnych dramatycznych tonów do łagodnych brzmień niosących nadzieję)

(głos prowadzący, szeptem na tle łagodnych klawiszy)
I choć ogień był straszny, a dym słońce skradł...
(chór – cicho) *Dinozaury!*
(głos – crescendo) *Ocalimy norki, spiżarnie i cały ten świat!*
(chór – mocno) *Dinozaury!*

Programujmy dniem i nocą, by las zielony trwał
(chór) *Dinoobliczenia!*
Niech C++ nas prowadzi przez płomieni krwawy szał
(chór) *Dinoobliczenia!*
I choć meteoryt czeka, by zakończyć naszą erę...
(chór – nagła pauza)
... to dziś gasimy sprawnie, z dino-pasją i szczerze!

(potężne uderzenie perkusji, gitara wchodzi na najwyższe obroty w triumfalnym riffie)

Więc do maszyn! Do obliczeń! Niech procesor rwie!
(chór) *Dinozaury!*
Uratujemy las, nim w popiół zmieni się!
(chór – patetycznie) *TYLKO DINOZAURY!*

(długi, wybrzmiewający akord, nagłe cięcie i kojący szum lasu)

Ciąg dalszy nastąpi...

Zadanie #1 (28 maja–11 czerwca)

Twoim zadaniem programistycznym na dzisiejsze zajęcia będzie napisanie funkcji pomocniczej (tzw. funkcji swobodnej, tzn. niezwiązanej z żadną klasą) i dwóch klas. Pamiętaj, aby odpowiednią funkcjonalność zawrzeć w plikach nagłówkowych (o rozszerzeniu *.h*) oraz implementacji (o rozszerzeniu *.cpp*). Dbaj o czytelność kodu (odpowiednie formatowanie, w tym wcięcia). Pamiętaj też o Obwieszczeniu nr 1 ogłoszonym w poprzednim tygodniu oraz nr 2 zawartym niżej.

Funkcja *success*

Zaimplementuj funkcję *success()*, która przyjmuje jeden argument o nazwie *success_probability* określający prawdopodobieństwo zwrócenia wartości *true*. Funkcja zwraca wartość logiczną *true* z prawdopodobieństwem *success_probability*, natomiast *false* — z prawdopodobieństwem *1 - success_probability*.

Jeśli chcesz, skorzystaj z następującej wskazówki. W funkcji *success* wylosuj wartość typu *double* z przedziału *[0, 1)*. Jeśli wylosowana wartość jest mniejsza niż *success_probability*, zwróć wartość *true*. W przeciwnym razie zwróć wartość *false*.

Zauważ, że zawsze prawdziwe są poniższe asercje:

```
1  assert(success(1.0));
2  assert(!success(0.0));
```

Do losowania możesz użyć klasycznej funkcji *rand()* z pliku nagłówkowego *<cstdlib>* albo nowoczesnej funkcjonalności zawartej w pliku nagłówkowym *<random>*.

Funkcja *success()* posłuży w kolejnej części projektu do symulowania losowości wzrostu drzew oraz uderzeń piorunów.

Klasa *cell*

Klasa opisuje pojedynczą komórkę symulacji.

Dane składowe klasy:

- *int state_* – przechowuje stan komórki (wartość *0* – komórka pusta/spalona, wartość *1* – drzewo, wartość *2* – pożar).

Pamiętaj, aby dane składowe klasy były prywatne.

Funkcje składowe klasy:

- Konstruktor domyślny *cell()* inicjuje stan komórki wartością *0*.
- Metoda *increment()* zwiększa wartość stanu komórki cyklicznie o jeden, tzn. jeśli np. początkowo *state_ == 0*, to po wywołaniu tej metody *state_ == 1*, a jeśli np. początkowo *state_ == 2*, to po wywołaniu tej metody *state_ == 0*. Innymi słowy, jest to operacja modulo *3*.
- Metoda *get_state()* zwraca wartość stanu komórki.

Na podstawie klasy *cell* w kolejnej części projektu zbudowana zostanie siatka obliczeniowa (las), reprezentująca stan przestrzenny symulacji.

Klasa *pbc_neighbors*

Klasa opisuje sąsiedztwo komórki na siatce kwadratowej z periodycznymi warunkami brzegowymi.

Dane składowe klasy:

- *std::size_t n_* – przechowuje rozmiar liniowy kwadratowej siatki symulacji (tzn. siatki o wymiarach *n_* wierszy na *n_* kolumn).

Pamiętaj, aby dane składowe klasy były prywatne.

Funkcje składowe klasy:

- Konstruktor *pbc_neighbors()* przyjmuje jeden argument o nazwie *n*, gdzie *n* określa rozmiar liniowy kwadratowej siatki symulacji.
- Metoda *get_n()* zwraca rozmiar liniowy kwadratowej siatki symulacji.
- Metoda *get_list()* przyjmuje dwa argumenty o nazwach *i* oraz *j* i zwraca kontener ze współrzędnymi sąsiadów komórki o współrzędnych *(i, j)*. Zwracanym kontenerem może być np. *std::vector* parametryzowany typem *std::pair*, np. *std::vector<std::pair<std::size_t, std::size_t>>*. Współrzędne sąsiadów komórki o współrzędnych *(i, j)* dane są poniżej:
  - *(prev(i), next(j))*,
  - *(i, next(j))*,
  - *(next(i), next(j))*,
  - *(prev(i), j)*,
  - *(next(i), j)*,
  - *(prev(i), prev(j))*,
  - *(i, prev(j))*,
  - *(next(i), prev(j))*.

  gdzie funkcja:

  - *prev(k)* zwraca *get_n() - 1* jeśli *k == 0* oraz *k - 1* w przeciwnym razie,
  - *next(k)* zwraca *0* jeśli *k == get_n() - 1* oraz *k + 1* w przeciwnym razie.

  Możesz porównać współrzędne sąsiadów z rysunkiem 2. Funkcje *prev()* oraz *next()* możesz zaimplementować jako prywatne metody klasy.

Klasa *pbc_neighbors* będzie odpowiedzialna w kolejnej części projektu za badanie sąsiedztwa danej komórki w celu zdecydowania, czy ogień rozprzestrzeni się na sąsiednie drzewa.

Obwieszczenie Zakładu Dinoobliczeń nr 2 🌲

Obwieszczenie: Każde rozwiązanie, nawet nieukończone, przed zakończeniem dzisiejszej pracy na zajęciach, należy spakować jako archiwum o rozszerzeniu *.zip*, *.tar*, *.tar.gz* lub *.tar.xz*, a następnie wysłać przez platformę Kampus do oceny. W przypadku, gdyby nie udało Ci się dokończyć pracy na zajęciach, uzupełnij pozostałą część w formie pracy domowej i przynieś na kolejne zajęcia.

Ciąg dalszy nastąpi...

Zadanie #2 (11 czerwca)

Tym razem do napisania będzie jedna klasa, jedna funkcja swobodna (operator) i program (funkcja *main()*) wykorzystujący całość dotychczasowej pracy. Pamiętaj, aby odpowiednią funkcjonalność zawrzeć w plikach nagłówkowych (o rozszerzeniu *.h*) oraz implementacji (o rozszerzeniu *.cpp*). Dbaj o czytelność kodu (odpowiednie formatowanie, w tym wcięcia). Zapoznaj się też z Obwieszczeniem nr 3 umieszczonym dalej.

Klasa *grid*

Klasa opisuje kwadratową siatkę obliczeniową złożoną z komórek typu *cell*, funkcjonującą w oparciu o periodyczne warunki brzegowe.

Dane składowe klasy:

- *std::size_t n_* — przechowuje rozmiar liniowy kwadratowej siatki symulacji,
- *pbc_neighbors pbc_* — obiekt pomocniczy do wyznaczania sąsiedztwa przy periodycznych warunkach brzegowych,
- *double p_growth_* — przechowuje prawdopodobieństwo wzrostu drzewa,
- *double p_fire_* — przechowuje prawdopodobieństwo spontanicznego zapłonu (np. poprzez uderzenie pioruna),
- *std::vector<std::vector<cell>> grid_* — dwuwymiarowy kontener przechowujący komórki siatki symulacji.

Pamiętaj, aby dane składowe klasy były prywatne.

Funkcje składowe klasy:

- Konstruktor *grid()* przyjmuje trzy argumenty:
  - *n* — rozmiar liniowy kwadratowej siatki symulacji,
  - *p_growth* — prawdopodobieństwo wzrostu drzewa,
  - *p_fire* — prawdopodobieństwo spontanicznego zapłonu.
- Metoda *get_n()* zwraca rozmiar liniowy kwadratowej siatki symulacji.
- Metoda *at()* przyjmuje dwa argumenty o nazwach *i* oraz *j* i zwraca obiekt typu *cell* znajdujący się na siatce obliczeniowej na współrzędnych *(i, j)*.
- Metoda *fire_nearby()* przyjmuje dwa argumenty o nazwach *i* oraz *j* i zwraca wartość typu *bool* równą *true* jeśli w sąsiedztwie komórki o współrzędnych *(i, j)* znajduje się choć jedna komórka reprezentująca pożar oraz równą *false* w przeciwnym razie. Skorzystaj tutaj z obiektu *pbc_*, aby pobrać listę współrzędnych sąsiadów.
- Metoda *step()* wykonuje krok symulacji według poniższego schematu:

```
void step() {
  auto next_grid = grid_;
  for (std::size_t i = 0; i < n_; ++i) {
    for (std::size_t j = 0; j < n_; ++j) {
      auto state = grid_[i][j].get_state();
      if (state == 0) {
        // Jesli success(p_growth_), to w next_grid[i][j] rosnie drzewo.
      } else if (state == 1) {
        // Jesli success(p_fire_) lub fire_nearby(i, j),
        // to w next_grid[i][j] wybucha pozar.
      } else {
        // Komorka plonela (state == 2), wiec next_grid[i][j] zostaje
        // ugaszona.
      }
    }
  }
  grid_ = next_grid;
}
```

  Pamiętaj, że wszystkie decyzje o zmianie stanu są podejmowane na podstawie „starej" siatki obliczeniowej (*grid_*), a wynik jest zapisywany w „nowej" (*next_grid*).

Pamiętaj, aby dane składowe klasy były prywatne.

Funkcja *operator<<(std::ostream& os, grid& g)*

Funkcja *operator<<* przyjmuje dwa argumenty: strumień wyjściowy *std::ostream&* oraz obiekt klasy *grid*. Funkcja wypisuje stan siatki na strumień w formie macierzy i zwraca referencję do strumienia *os*. Przykład wyjścia dla siatki o rozmiarze 4×4:

```
0 0 0 1
0 0 1 1
0 1 1 1
1 1 1 2
```

W powyższym przykładzie:

- wartość *2* (prawy dolny róg) reprezentuje pożar,
- wartość *1* reprezentuje drzewa,
- wartość *0* reprezentuje obszar pusty/spalony.

Wewnątrz funkcji użyj dwóch zagnieżdżonych pętli *for*. Aby pobrać stan komórki, wykorzystaj metodę *g.at(i, j).get_state()*. Pamiętaj o dodaniu znaku nowej linii *std::endl* (lub *'\n'*) po każdym wypisanym wierszu siatki, aby zachować czytelny układ macierzowy.

Program — funkcja *main()*

Poniżej znajduje się szkielet funkcji *main()*. Uzupełnij brakujący fragment, tworząc plikowy strumień wyjściowy (*std::ofstream*), do którego będą zapisywane kolejne stany siatki obliczeniowej.

```
// Tutaj powinny sie znalezc tylko pliki naglowkowe.

int main() {
  // Tutaj mozesz umiescic kod inicjujacy generator liczb pseudolosowych.
  std::size_t grid_size = 5;
  double p_growth = 0.1;
  double p_fire = 0.05;
  std::size_t n_steps = 100;
  grid g{grid_size, p_growth, p_fire};
  // Tutaj powinien znalezc sie plikowy strumien wyjsciowy o nazwie file.
  for (std::size_t i = 0; i < n_steps; ++i) {
    file << g;
    g.step();
    file << '\n';
  }
}
```

Obwieszczenie Zakładu Dinoobliczeń nr 3 🌲

Obwieszczenie: Na koniec zajęć należy wysłać swoje rozwiązanie — nawet nieukończone — w formie archiwum do oceny przez platformę Kampus. W przypadku konieczności uzupełnienia kodu, poprawioną pracę należy przesłać przez platformę Kampus po raz kolejny najpóźniej do godziny 23:59 dnia dzisiejszego.

Koniec! (Meteorytem zajmiemy się później.)