Program symuluje dwuwymiarowy automat komórkowy reprezentujący pożar lasu. Został napisany w języku Python z wykorzystaniem tablic numpy, w których każde pole odpowiada danemu stanowi. Wizualizacja jest przedstawiona za pomocą biblioteki Pygame. Obie biblioteki należy samodzielnie zainstalować (np. w terminalu PyCharm, Visual Studio Code itp. za pomocą polecenia **pip install numpy pygame**)

Stany (cyfry odpowiadają wartości w tablicy numpy):

* Pusta komórka [0]
* Małe drzewo [8]
* Średnie drzewo [9]
* Duże drzewo [1]
* Drzewo, w które uderzył piorun(samozapłon) [2]
* Drzewo na 1 stopniu spalania [5]
* Drzewo na 2 stopniu spalania [6]
* Spalone drzewo [7]
* Ogień [3]
* Jezioro [4]

Reguły przejścia między stanami:

* Jeżeli komórka jest pusta (0), to z pewnym prawdopodobieństwem (GROW\_CHANCE) wyrasta w niej małe drzewo (8).
* Jeżeli w komórce jest duże drzewo (1), to z pewnym prawdopodobieństwem (BOLT\_STRIKING\_CHANCE) uderza w nie piorun (2).
* Jeżeli w drzewo uderzył piorun (2), to w następnej iteracji zamienia się on w drzewo na 1 stopniu spalania (5).
* Jeżeli w komórce jest drzewo na 1 stopniu spalania (5), to w następnej iteracji przechodzi ono w drzewo na 2 stopniu spalania (6).
* Jeżeli na polu jest drzewo na 2 stopniu spalania (6), to na następną iterację przechodzi ono w ogień (3).
* Jeżeli w komórce jest ogień (3), to w następnej iteracji przechodzi on w spalone drzewo (7) i rozpoczyna odliczanie do regeneracji drzewa.
* Jeżeli sąsiedztwie drzewa (1) znajduję się ogień (3) to staje się ono drzewem na 1 stopniu spalania (5) z prawdopodobieństwem (FIRE\_SPREAD\_CHANCE).
* Jeżeli w komórce jest spalone drzewo (7), to po określonej liczbie iteracji (TREE\_REGENERATION\_TIME) z pewnym prawdopodobieństwem (TREE\_REGENERATION\_CHANCE) zaczyna się ono odnawiać (8), a w przeciwnym wypadku komórka staje się pusta (0).
* Jeżeli w komórce jest małe drzewo (8), to na następną iterację przechodzi ono w średnie drzewo (9).
* Jeżeli na w komórce jest średnie drzewo (9), to na następną iterację przechodzi ono w duże drzewo (1).
* Jeżeli w komórce jest jezioro (4), pozostaje ono bez zmian.

Program rozpoczyna się od inicjalizacji bibliotek i ustawienia parametrów. Oznaczają one kolejno:

**WIDTH, HEIGHT** - Wysokość i szerokość tablicy numpy

**TILE\_SIZE** - Wielkość pojedynczego pola w którym wyświetlany jest obiekt (potrzebny do wizualizacji w Pygame)

**TREE\_DENSITY** - Gęstość lasu (potrzebny do obliczenia liczby początkowych drzew)

**GROW\_CHANCE** - Szansa na wyrośnięcie drzewa w pustym miejscu

**BOLT\_STRIKING\_CHANCE** - Szansa na uderzenie pioruna w drzewo (samozapłonu)

**FIRE\_SPREAD\_CHANCE** - Szansa na rozprzestrzenienie się ognia na drzewo

**LAKES\_DENSITY** - Gęstość jezior (potrzebny do obliczenia liczby jezior)

**TREE\_REGENERATION\_TIME** - Ilość iteracji po których drzewo odnawia się

**TREE\_REGENERATION\_CHANCE** - Szansa na odnowę drzewa

**PAUSE\_LENGTH** - Długość pauzy pomiędzy iteracjami w sekundach

Następnie ładuje grafiki reprezentujące różne stany drzewa (dorosłe, palące się, spalone, małe, średnie) i inne elementy (piorun, jezioro).

Kolejna funkcja sprawdza czy niektóre parametry są odpowiednio ustalone jako wartości od 0 do 1. Jeżeli tak nie jest wyrzuca odpowiedni komunikat.

Główna funkcja **fireForestSim()** wywołując funkcję **createNewForest()** tworzy początkowy stan lasu. Opiera się o wcześniej podane parametry, a koniec inicjuje pożar w losowej komórce tablicy. Do przechowywania czasu regeneracji spalonego drzewa inicjowana jest pusta tablica numpy.

Następnie funkcja wchodzi w nieskończoną pętlę, w której symuluje kolejne iteracje pożaru. Do przechowywania nowego stanu lasu ponownie użyta jest tablica numpy. W każdej iteracji, dla każdego pola na mapie, na podstawie jego aktualnego stanu i losowej szansie, obliczany jest jego następny stan. Do obliczenia prawdopodobieństwa użyta została wbudowana metoda **random.random()**, która generuje liczbę od 0 do 1.

Większość instrukcji if opiera się tylko na sprawdzeniu stanu i przejściu na następny. Trochę bardziej skomplikowane są warunki odnowienia spalonego drzewa i podpalenia od sąsiedniego. Ten pierwszy wykorzystuje tablicę **regeneration\_time**, która odlicza czas za pomocą wartości której komórka odpowiada komórce danego stanu w tablicy lasu. Z każdą iteracją jej wartość zmniejsza się o 1, a gdy osiągnie 0 obliczana jest szansa na odnowę drzewa. Jeżeli wartość będzie mniejsza niż wcześniej podana w parametrach spalone drzewo zmieni się w małe drzewo, w przeciwnym wypadku zostanie zastąpione pustą komórką.

Natomiast drugi bardziej zaawansowany przypadek polega na sprawdzeniu przez program dla każdej komórki w której znajduję się drzewo, czyli wartości 1, czy któryś z 8 sąsiadów jest ogniem. Za pomocą 2 pętli for sprawdzamy najpierw czy wyszliśmy poza zakres tablicy, jeżeli nie to dla każdej sąsiadującej komórki zawierającej ogień, czyli o wartości 3, obliczana jest szansa na rozprzestrzenienie pożaru. Wartość mniejsza lub równa zmiennej **FIRE\_SPREAD\_CHANCE** podpala drzewo.

Warunkiem zakończenia programu jest spalenie wszystkich drzew. Aby to robić zdefiniowana została wcześniej lista **not\_burned\_trees**, która zawiera wartości odpowiadające drzewom, które są żywe. Następnie warunek **if not** **np.any(np.isin(forest, not\_burned\_trees))** sprawdza, czy w tablicy lasu nie występuje ani razu żadna z cyfr zawartych w liście. Jeżeli tak jest program zostaje przerwany i w konsoli wyświetlony zostanie odpowiedni komunikat.

Do wizualizacji symulacji pożaru lasu program wykorzystuje bibliotekę Pygame przeznaczonej do tworzenia gier komputerowych oraz aplikacji multimedialnych w języku Python.

Grafiki użyte w programie:

Jezioro – autorstwa gamer247 ze strony itch.io

Piorun - autorstwa juicy\_fish ze strony Freepik.com

Drzewa – autorstwa Freepik ze strony Freepik.com

Ogień - autorstwa macrovector\_official ze strony Freepik.com

Funkcja **displayForest(forest)** odpowiada za wyświetlanie obrazków na ekranie na podstawie wartości w tablicy **forest**. Dwie pętle for przeszukują tablicę i za pomocą metody **screen.blit()** rysują i skalują załadowane grafiki. Co każdą iterację w funkcji **fireForestSim()** okno programu zostaje zaktualizowane o nowy stan tablicy **forest.** Ten sposób wizualizacji o wiele bardziej ułatwia wyobrażenie działania automatu niż gdyby użyć do tego celu np. pojedynczych kolorów. Każdy ze stanów można od razu rozpoznać bez posiłkowania się instrukcją.

Wnioski

Program skutecznie symuluje automat komórkowy reprezentujący pożar lasu. Oprócz podstawowych stanów jak drzewo, płonące drzewo, spalone drzewo uwzględnia wiele dodatkowych jak np. 3 stany rośnięcia i spalania drzewa służące lepszemu odwzorowaniu dynamiki rozprzestrzeniania się pożaru czy jezioro które może stanowić blokadę dla ognia.

Dla poprawy symulacji program uwzględnia szansę na odnowienie drzewa np. po niewielkim pożarze. Zaimplementowana została również sytuacja kiedy pożar zostanie wywołany przez uderzenie pioruna. Do dostosowania symulacji do swoich potrzeb wystarczy tylko modyfikacja odpowiednich parametrów.

Wizualizacja w Pygame jest jasna i czytelna dla ludzi, od razu wiadomo w jakim stanie znajduję się każda komórka automatu. Modyfikacja lub rozszerzenie automatu o kolejne stany nie stanowi problemu, wystarczy tylko załadować nowe obrazy i przypisać je do odpowiednich wartości w tablicy numpy.

Sam program nadaje się do dalszego rozwoju. Można by na przykład dodać możliwość dostosowania czasu wyrośnięcia drzewa poprzez dodanie dodatkowej tablicy numpy, albo dodać inną szansę na podpalenie się małego i średniego drzewa. Dodanie wiatru od którego zależałby kierunek rozprzestrzeniania pożaru też byłoby dobrym pomysłem.