Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Informatyka w Multimediach

Generator mapy świata i symulator historii

Bartłomiej Jędrzej Moroz
Numer albumu 304076

promotor
dr hab. inż. Tomasz Martyn

WARSZAWA 2024

Generator mapy świata i symulator historii

Streszczenie. Komputerowe symulowanie historii polega na modelowaniu procesów rozwoju cywilizacji, próbując odtworzyć dzieje ludzkości lub eksplorować alternatywne scenariusze. Oprogramowanie o takiej tematyce wykorzystywane jest jako źródło rozrywki lub narzędzie edukacyjne. Niniejsza praca magisterska poświęcona jest analizie tego niszowego tematu, zaprojektowaniu własnego modelu rozwoju i implementacji programu symulatora i powiązanego z nim generatora terenu. Przegląd literatury dziedziny obejmuje sprecyzowanie zadania generacji i symulacji, analizę porównawczą metod proceduralnych, symulacyjnych, wykorzystujących uczenie maszynowe i innych technik generowania terenu i klimatu oraz porównanie istniejących komercyjnych i darmowych rozwiązań pozwalających na symulowanie historii. Własne rozwiązanie zostało opisane w trzech częściach. Pierwsza z nich prezentuje wysoce konfigurowalny model i związane z nim definicje, założenia, parametry i algorytmy. Druga część nakreśla ogólną architekturę projektu opartą o wzorzec ECS (entity-component-system), a trzecia przybliża sposób działania modułów generatora i symulatora. Stworzone programy zostały zbadane pod względem szybkości działania i skalowalności rozmiaru świata. Oprócz tego omówiono wyniki generacji i symulacji, porównując je z odpowiednimi rzeczywistymi danymi i zjawiskami. Praca jest dopiero pierwszym krokiem w tym kierunku badań. Zakres i głębia symulacji może być stopniowo zwiększana w przyszłości poprzez reprezentowanie kolejnych aspektów cywilizacji.

Słowa kluczowe: proceduralna generacja terenu, symulacja komputerowa, entity-component-system

World map generator and history simulator

Abstract. Simulating history involves modeling the development of civilization to recreate the history of humanity or explore alternative timelines. It is commonly used for entertainment or educational purposes. The purpose of this master's thesis is to explore this niche topic, design a history model, and use it to develop a simulator and a supplementary terrain generator. The conducted literature review involves outlining the task at hand, a survey of procedural, simulation, machine learning, and other terrain and climate generation methods, and a comparison of available free and commercial history simulators. After that, a new, highly customizable simulation model is proposed in the following three sections. The mathematical definition is discussed first, then the ECS (entity-component-system) based general project design, and finally the working details of the generator and simulator modules. Both applications are examined regarding performance, scalability with world size, and output correctness. The results of select generator and simulator runs are discussed using real-world data to compare with the phenomena they are supposed to model. This thesis is just the beginning of this line of research, as the scope and depth of the simulation can be gradually expanded with additional systems being incorporated into the model.

Keywords: procedural terrain generation, computer simulation, entity-component-system

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Politechnika Warszawska

Warszawa, 11.9.2024 r.
miejscowość i data

Bartłomiej Moroz
imię i nazwisko studenta

304076
numer albumu

Informatyka
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

Bartłomiej Moroz
czytelny podpis studenta

Spis treści

1. Wstęp

Historia jest specyficzną nauką pod tym względem, że dotyka wyłącznie przeszłości i polega na próbie połączenia faktów w ciąg przyczynowo-skutkowy *a posteriori*. Niemożliwe jest ustalenie, co ma wpływ na teraźniejszość lub co się będzie działo w przyszłości. Ten fakt fascynuje ludzi i inspiruje do zadawania pytań „co by było, gdyby", o czym świadczy popularność tekstów kultury i gier komputerowych z gatunku historii, historii alternatywnej lub historii fikcyjnych światów. Komputerowa symulacja historii polega na modelowaniu i odtwarzaniu procesów rozwoju gospodarczego, terytorialnego, wojskowego, technologicznego, kulturowego i religijnego ludzkości. Niektóre z nich, takie jak ekonomia, logistyka czy wojskowość mogą być dokładnie symulowane, podczas gdy inne często muszą być realizowane w bardzo abstrakcyjny sposób ze względu na jakościowy charakter historii, antropologii i innych dziedzin nauki.

Istniejące rozwiązania różnie podchodzą do tematu: dedykowane symulatory starają się w jak najbardziej zaawansowany i organiczny sposób tworzyć historię, gry strategiczne stawiają na czynny udział użytkownika w procesie, podczas gdy „gry w boga" (ang. *god games*) próbują pogodzić oba podejścia, jednocześnie często dodając element nadprzyrodzony. Każdy sposób ma swoje wady i zalety, które wynikają z różnych definicji celu i zakresu symulacji. Oprogramowanie tego typu jest wykorzystywane głównie w celach rozrywkowych, edukacyjnych i przy wspomaganiu procesów twórczych. Bardziej ogólne modele są przede wszystkim wartościowym i szeroko dostępnym materiałem pomocniczym przy nauczaniu historii [1], podczas gdy te bardziej specjalistyczne funkcjonują jako rekonstrukcje historyczne [2].

Symulatorom często towarzyszą generatory terenu – programy pozwalające tworzyć wirtualne światy o różnym stopniu realistyczności [3], które potem są wykorzystane jako środowiska testowe. Generatory umożliwiają przygotowywanie scenariuszy o charakterystykach geograficznych odbiegających od tych występujących na Ziemi, np. innym wieku gór, klimacie, kształcie kontynentów. Metody generowania terenu są podzbiorem metod proceduralnej generacji treści (ang. *procedural content generation*, PCG), które znajdują zastosowanie szczególnie przy tworzeniu wizualizacji, animacji i gier komputerowych.

1.1. Cel i motywacja pracy

Wyznaczono dwa główne cele pracy. Pierwszym z nich jest opracowanie teoretycznego modelu generacji świata i symulacji historii, na który składa się opis reprezentacji świata i ludzkiej cywilizacji, rodzajów wykorzystanych danych, modelowanych procesów naturalnych i społecznych, proponowanych algorytmów i konfigurowalnych parametrów. Drugi cel polega na zaprojektowaniu, implementacji i omówienia działania pary programów: generatora i symulatora, których działanie jest oparte o zaproponowany model. Programy powinny być wydajne, maksymalnie konfigurowalne, możliwie kompatybilne z zewnętrznymi źródłami danych, dobrze opisane, posiadać interfejs graficzny i być dostępne na otwartej licencji. Zakres generatora obejmuje generowanie kontynentów, ukształtowania powierzchni terenu i danych meteorologicznych pozwalających na klasyfikację klima-

tyczną terenów. Zakres symulatora obejmuje rozwój naukowy, kulturowy, podział terytorium, działania dyplomatyczne i wojskowe ludzkości podzielonej na niezależne państwa.

Praca obejmuje także przegląd literatury dotyczącej metod generacji terenu, które w porównaniu do symulacji są bardzo popularnym i aktywnie rozwijanym tematem (szczególnie w ostatnich latach w kontekście generatywnych modeli uczenia maszynowego), porównanie architektur silników do gier/multimediów wykorzystywanych przez symulatory oraz komentarz odnośnie samej historii i wykorzystania metod symulacyjnych w pokrewnych dziedzinach.

Powody do napisania tej pracy i postawienia takich celów wynikają z dwóch obserwacji. Motywacja do opracowania i opisania teoretycznego modelu symulacji historii wynika z faktu, praktycznie nie ma istniejących prac naukowych na dokładnie ten temat. Sama koncepcja komputerowej symulacji historii zaczęła się pojawiać w publikacjach w ciągu ostatnich kilkunastu lat, ale w całości dotyczy rozważań na temat *zastosowania i przydatności takiego oprogramowania* [1], [2], [4]–[6] (w większości odnosząc się do symulacyjnych gier komputerowych o tematyce historycznej), a nie *podstaw teoretycznych zagadnienia* lub *opisów technicznych implementacji*. Niniejsza praca jest więc próbą eksploracji tematu od strony naukowej i inżynierskiej. Drugi cel, czyli stworzenie działającego generatora i symulatora, wynika z obserwacji odnośnie stanu istniejącego oprogramowania. Popularne i najbardziej rozbudowane symulatory są nieopublikowane, płatne lub darmowe, ale bez udostępnionego kodu źródłowego lub chociaż opisu działania. Z drugiej strony, znalezione podczas przeglądu rozwiązań symulatory o otwartej licencji cierpią na zbyt ambitny zakres planowanych funkcjonalności w stosunku do włożonej pracy, są w bardzo wczesnej fazie rozwoju lub porzucane przed zrealizowaniem większości założeń.

1.2. Zawartość pracy

Praca składa się z powyższego wstępu, pięciu rozdziałów i podsumowania. Rozdział drugi zawiera opis tematu, sprecyzowanie zadania oraz analizę dostępnych metod i istniejących rozwiązań związanych z tematem. Rozdział trzeci jest w całości poświęcony proponowanemu modelowi generacji i symulacji, który następnie jest wykorzystany w implementacji programów. Rozdział czwarty opisuje postawione wymagania, wybrane technologie i narzędzia oraz architekturę całego projektu. Rozdział piąty pokazuje implementację i sposób działania poszczególnych modułów. Rozdział szósty zawiera dyskusję odnośnie wyników generacji i symulacji, badania wydajności i perspektywy dalszego rozwoju. Praca zakończona jest podsumowaniem zawierającym końcowe wnioski z pracy.

2. Analiza tematu

Generowanie świata jest dość popularnym i aktywnie rozwijanym (zarówno naukowo, jak i komercyjnie) tematem znajdującym zastosowanie przy produkcji gier komputerowych i innych mediów cyfrowych, podczas gdy symulatory historii powstają głównie w kręgach hobbistycznych lub w formie programów rozrywkowych. Z tego powodu te dwa zagadnienia są omówione oddzielnie i w inny sposób: Podrozdział 2.1 definiuje generowanie świata jako zadanie i opisuje wykorzystywane rozwiązania, Podrozdział 2.2 dokonuje krótkiego przeglądu metod symulacyjnych w dziedzinach nauki pośrednio związanych z historią, a Podrozdział 2.3 omawia istniejące projekty i różnice w podejściu do tematu.

2.1. Generowanie świata

Termin „świat" w tej pracy jest używany do określenia połączenia teoretycznego modelu przestrzeni (Paragraf 2.1.1, Paragraf 2.1.2), danych topograficznych (Paragraf 2.1.3, Paragraf 2.1.4) i danych klimatycznych (Paragraf 2.1.5), które razem reprezentują rzeczywiste lub syntetyczne środowisko naturalne.

Generowanie świata jest szczególnym przypadkiem proceduralnego generowania treści (ang. *procedural content generation*, PCG). PCG to określenie odnoszące się do metod tworzenia dowolnej treści opartych na systematycznym wykonywaniu ustalonych procedur [7], zwykle w formie algorytmów komputerowych[1]. Pozwala na automatyczne, masowe tworzenie treści takich jaki dźwięki, muzyka, obrazy, modele trójwymiarowe, animacje, teksty, plansze i mapy, przez co znajduje szczególne zastosowanie przy tworzeniu mediów cyfrowych takich jak wizualizacje trójwymiarowe, filmy, gry komputerowe i symulacje [8]. Proceduralna generacja może być używana w celu obniżenia kosztów produkcji lub aby dostarczyć treść w ilości niemożliwej do wytworzenia ręcznie [9], przykładowo w grach typu *sandbox* lub *rougelike*, których główną cechą jest umiejscawianie gracza w unikatowym, często nieskończenie wielkim świecie gry przy każdej rozgrywce.

Celem proceduralnego generowania *terenu* jest uzyskanie dwu- lub trójwymiarowej cyfrowej reprezentacji ukształtowania terenu o określonym wyglądzie, najczęściej przypominającego teren naturalnie spotykany na Ziemi. Generatory mogą być wykorzystywane do stworzenia drastycznie różnych środowisk, na przykład krajobrazów górskich[10]–[12], rozległych systemów jaskiń [13], dolin rzek[10], [11], pustynnych wydm [14], [15], całych kontynentów lub planet [16], [17]. Założenia odnośnie modelowanego terenu znacznie wpływają na dobór metod i sposób ich implementacji.

2.1.1. Model i skala terenu

Skala ma duży wpływ na rodzaj aspektów terenu i detali, które będą musiałby być odwzorowane. Generowanie terenu obejmującego małą powierzchnię (do kilku, kilkunastu kilometrów) skupia się na estetyce krajobrazu, symulacji różnych rodzajów erozji terenu i

[1] Chociaż PCG znalazła największe zastosowanie przy tworzeniu gier i grafice komputerowej, sama idea wywodzi się ze środowiska TTRPG (ang. *tabletop role-playing game*, stolikowe gry fabularne), które wykorzystywało modularne zestawy plansz i losowość przy konstruowaniu środowiska, w którym poruszali się gracze.

realistycznym tworzeniu form terenu (np. klify, pagórki, jaskinie, szczyty górskie), obiektów wodnych (jezior, rzek, wodospadów) i roślinności [10], [11].

Gdy teren ma obejmować dziesiątki czy setki kilometrów, podane obiekty i efekty erozji stają się zbyt małe lub kompletnie niewidoczne, więc nie mają dużego znaczenia przy generacji [3]. Zamiast tego ważne staje się wyznaczanie kształtu całych łańcuchów górskich, regionów nizinnych i wyżynnych, linii brzegowych, wysp. Należy też wziąć pod uwagę warunki klimatyczne, które zaczynają być coraz bardziej różnorodne wraz ze wzrostem skali.

Ze skalą planetarną związane są trzy unikatowe problemy. Po pierwsze, płyty tektoniczne mają duży wpływ na kształt kontynentów i formowanie się łańcuchów górskich – jeśli aktywność tektoniczna nie jest w jakiś sposób symulowana, doświadczony odbiorca może zauważyć sztuczność[16]. Po drugie, rzeczywiste planety mają kształt elipsoid[2], więc ich powierzchnia nie może być wiernie odwzorowana na mapie. Jest to znany problem z dziedziny kartografii [18]. Trzeci problem wyniki z drugiego: ciągłość na granicach mapy reprezentującej planetę musi być zachowana. Oznacza to, że powierzchnia musi być opisana trójwymiarowym układem współrzędnych kartezjańskich lub układem współrzędnych geograficznych. Alternatywnie można przyjąć model „płaskiej Ziemii" i reprezentację na prostej płaszczyźnie, który nie posiada wyżej wymienionych problemów, ale oznacza to, że generator ogranicza się do światów fantastycznych.

2.1.2. Reprezentacja i wizualizacja danych

Teren reprezentowany jest przez zbiór próbek w przestrzeni dwu- lub (rzadziej) trójwymiarowej. W pierwszym przypadku najczęściej jest definiowany jako tzw. mapa wysokości (ang. *heightmap*, nazywana także *digital elevation model* [19], *digital terrain model* [17], [19] i mylnie *digital surface model*), która zawiera punkty rozłożone na kwadratowej siatce i ich wysokości [8], [9], [11], [17], [19]. Mapa wysokości może mieć postać dwuwymiarowej tablicy lub obraz rastrowego, w którym każdy piksel odpowiada punktowi na siatce, a jego wartość jasności odpowiada wysokości w pewnej skali [20]. Taka forma jest popularna i łatwa w przetwarzaniu, lecz nie pozwala na istnienie więcej niż jednego punktu na tych samych koordynatach, a więc na reprezentację łuków lub jaskiń [3].

W przypadku reprezentacji trójwymiarowej stosuje się zazwyczaj chmury punktów lub modele wokselowe (najczęściej stosowane w grach komputerowych) [3]. Nie mają one takich ograniczeń, jak wcześniej wspomniane mapy wysokości, ale wymagają więcej pamięci i bardziej skomplikowanych algorytmów. Pozwalają także na dokładną wizualizację trójwymiarową w skali planetarnej.

Sposoby wizualizacji otrzymanego terenu są typowo związane z przyjętym modelem terenu i reprezentacją: mapy wysokości są wyświetlane jako kolorowy obraz lub trójwymiarowa siatka trójkątów przedstawiająca odkształcenia na płaszczyźnie [8], [10], [11], [17], światy o modelu kulistym są prezentowane jako siatka trójkątów-odkształceń na sferze [16],

[2] Wprawdzie Ziemia jest geoidą, a nie perfekcyjną kulą, ale takie przybliżenie jest powszechnie stosowane ze względu na praktycznie niedostrzegalną różnicę.

a modele wokselowe są wizualizowane jako modele 3D wykorzystując dowolne metody, głównie teselację wokseli lub *ray marching* [21].

2.1.3. Proceduralne metody generowania terenu

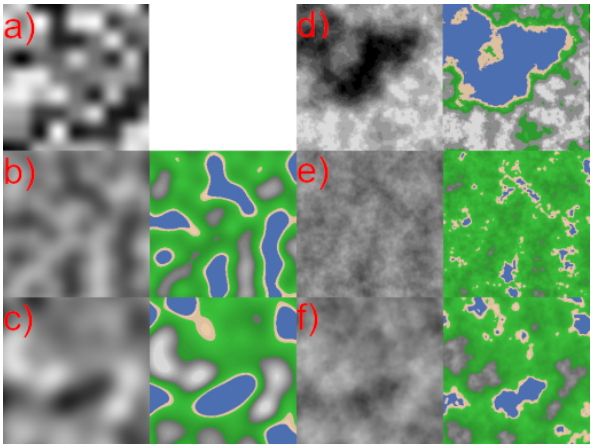
Najpopularniejszą i szybką metodą otrzymania mapy wysokości jest wykorzystanie spójnego szumu losowego (ang. *coherent noise*) [20], oryginalnie stworzonego do proceduralnej generacji tekstur wykorzystywanych w grafice komputerowej. Szum opisany jest funkcją zwracającą pseudolosowe wartości dla dowolnego punktu przestrzeni przy jednoczesnym zagwarantowaniu ogólnie postrzeganej „gładkości" obrazu. Najprostszy i najmniej atrakcyjny szum to tzw. *value noise*, który polega na przypisaniu losowej wartości punktom w regularnej kwadratowej siatce i bezpośrednim interpolowaniu wartości między punktami. Najbardziej popularny jest szum Perlina [22], który dla każdego punktu siatki generuje losowe gradienty (wektory) zamiast wartości. Wartości pomiędzy punktami są obliczane poprzez interpolowanie iloczynów skalarnych czterech sąsiadujących gradientów, dzięki czemu szum ma charakterystyczny, „organiczny" kształt. Następcą szumu Perlina jest szum simplex [23], w którym poprawiono wydajność i jakość szumu, a także usunięto artefakty mogące się pojawiać na granicach tekstury. Istnieją także inne modyfikacje i implementacje tych szumów takie jak OpenSimplex i SuperSimplex [24].

Aby osiągnąć lepszy rezultat wykorzystuje się szum fraktalny, czyli sumę kolejnych warstw wybranego podstawowego szumu (na przykład Perlina) o różnych częstotliwościach i amplitudach. Jeżeli amplituda kolejnych warstw jest odwrotnie proporcjonalna do częstotliwości, mówimy wtedy o szumie różowym (nazywanym także szumem 1/f) [25]. Jeżeli oprócz tego warunku częstotliwość rośnie dwukrotnie z każdą warstwą, taką warstwę nazywa się oktawą, a szum szumem Browna [26] (ang. *fractional Brownian noise* lub *fractional Brownian motion*, fBm). Efekt końcowy jest znacznie lepszy, ponieważ dodaje detale do obłego kształtu szumu Perlina/simplex. Wiąże się to z dodatkowym kosztem obliczeniowym, dlatego zazwyczaj nie wykorzystuje się więcej niż kilku warstw zwłaszcza, że zgodnie z definicją każda kolejna warstwa ma coraz mniejszy wpływ. Szum fraktalny można także mieszać z innymi rodzajami szumu przyjmującymi specyficzne kształty, takimi jak *billow noise* (przypominający chmury) i *ridge noise* (wysoki kontrast), co pozwala na generowanie miejscowo zróżnicowanych krajobrazów [3], [27].

Fraktale opisane L-systemami lub innymi gramatykami i metody rekurencyjne są często używane w metodach proceduralnej generacji treści [3], [8], [20]. Wynika to z obserwacji, że wiele naturalnych obiektów wykazuje samopodobieństwo, czyli podobieństwo fragmentu kształtu do jego całości [10], na przykład rzeki [10], ich dopływy, rośliny [8], kształty gór [28] albo systemy jaskiń [13].

Kolejnym często spotykanym algorytmem wykorzystującym samopodobieństwo jest algorytm diament-kwadrat (ang. *diamond-square*) [29]. Algorytm występuje w wielu wariantach [3] i jest dwuwymiarową modyfikacją algorytmu przemieszczenia środka odcinka (ang. *midpoint displacement*) [8]. Polega on na rekursywnym powtarzaniu dwóch kroków dla kolejnych nieprzetworzonych punktów: sumowaniu wartości punktów po przekątnej (krok „kwadrat") i wzdłuż osi (krok „diament"), za każdym razem dodając niewielką, ma-

lejącą ilość szumu i zmniejszając skalę. Algorytm diament-kwadrat potrzebuje znacznie więcej pamięci niż szumy, ponieważ wartość punktów zależy od wartości innych punktów, więc wszystkie muszą być zapamiętane, ale jest szybszy [20]. W przeciwieństwie do algorytmów na bazie szumu, wymagana jest inicjalizacja punktów początkowych, która może być dokonana losowo lub przez użytkownika, pozwalając mu wprowadzić punktowe sugestie odnośnie pożądanego terenu, na przykład lokalizacje czubków gór. Inną metodą wymagającą wprowadzenia danych wejściowych jest Uplift Model, który zamiast rekursji wykorzystuje specjalną funkcję uśredniania terenu wokół podanych szczytów, pozwalającą na tworzenie się płaskowyżów [30].



Rysunek 2.1. Przykładowe szumy. (a) Value noise, (b) Perlin, (c) simplex, (d) algorytm diament-kwadrat, (e) Perlin fBm, (f) simplex fBm. Po lewej znajdują się tekstury wartości, po prawej proste kolorowe mapa (bez value noise).

2.1.4. Inne metody generowania terenu

Aby poprawić wygląd istniejącej mapy wysokości można symulować zjawiska erozji powodujące degradację terenu wraz z upływem czasu. Wyróżnia się trzy rodzaje erozji: hydrologiczną [11], [31], termiczną [31] i wietrzną [14], [15]. Erozja hydrologiczna to proces odrywania i przemieszczania się fragmentów terenu spowodowany ruchem wody występujący podczas deszczu, w korytach rzek i nad brzegiem akwenów. Erozja termiczna polega na oddzielaniu się fragmentów skał lub topnieniu śniegów i czap lodowych pod wpływem temperatury. Erozja wietrzna występuje głównie na pustyniach, gdzie piasek jest porywany przez wiatr i formuje wydmy. Teren, który został poddany wpływowi tych procesów zazwyczaj jest gładszy i bardziej naturalny [3], a w przypadku erozji hydrologicznej pozwala na jednoczesne naturalne formowanie się rzek i jezior [11]. Symulacja jest realizowana za pomocą skończonej liczby cząsteczek wody, skał lub wiatru, które przemieszczają się z punktów powierzchni położonych wyżej do punktów położonych niżej, zabierając lub osadzając pewną ilość materiału. Jest to proces bardzo intensywny obliczeniowo, ponieważ łączy modyfikację terenu z symulacją dynamiki płynów [3]. Z tego powodu algorytmy symulacji erozji często muszą być realizowane przy użyciu kart graficznych [3], [11], [31].

Najbardziej realistyczną metodą generowania terenu jest symulacja ruchu płyt tektonicznych [3], [16], [32]. Symulowane są zjawiska takie jak zderzenia niesprężyste pomiędzy płytami (subdukcja, zderzenie kontynentów) oraz ocieranie się i oddalanie się płyt, które powodują formowanie się łańcuchów górskich, grzbietów śródoceanicznych, szelfów, uskoków, wulkanów, dolin ryftowych i innych form terenu [16]. Podobnie jak w przypadku erozji, dokładne symulowanie procesu wymaga znacznie większych zasobów sprzętowych, niż algorytmy szumu [3], ale może być w różnym stopniu uproszczone (z czym wiąże się niższa jakość wyników). Poszczególne płyty reprezentowane są jako chmury próbek lub wielokąty (np. otrzymane za pomocą teselacji Woronoja) z przypisanym wektorem prędkości. W obu przypadkach potrzebne jest pokrycie całej powierzchni świata próbkami, które nie jest trywialne, gdy założymy elipsoidalny model planety [16]. Wykorzystuje się tutaj próbkowanie metodą Fibonacci [33] albo przybliżenia próbkowania dyskiem Poissona [34], na przykład metodę najlepszego kandydata Mitchella [35] lub Fast CCVT [36]. W środowisku geomorfologów[3] nie ma zgody co do źródła ruchu płyt tektonicznych, podając wpływ prądów konwekcyjnych, siły grawitacji i ruchu obrotowego [38]. W konsekwencji wektory prędkości przypisuje się losowo [16] lub symulując ruchy konwekcyjne występujące wewnątrz płaszcza ziemskiego [32]. Właściwe generowanie terenu odbywa się iteracyjnie, przemieszczając punkty płyt, sprawdzając kolizje i odpowiednio wypiętrzając teren.

W ostatnich latach metody oparte o sztuczną inteligencję zyskały na popularności, w szczególności modele generatywne [3], [39]–[41]. Ponieważ teren można w łatwy sposób przedstawić w formie obrazu (jako mapę wysokości), problem generowania terenu można potraktować jako problem generowania tekstur, syntezy tekstur lub transferu stylu na podstawie szkicu użytkownika. Jako dane treningowe wykorzystuje się zdjęcia satelitarne [39], [40] lub mapy wysokości zeskanowanego rzeczywistego terenu [39], [41], które są typowo wysokiej jakości i publiczne dostępne, ponieważ wymagają bardzo specjalistycznego sprzętu, na który mogą sobie pozwolić zazwyczaj tylko instytucje naukowe [28]. Wśród prac dominują raczej sieci typu GAN (ang. *Generative Adversarial Networks*), ale istnieją także prace oparte na sieciach dyfuzyjnych [41].

Oprócz modeli uczenia maszynowego można korzystać z algorytmów genetycznych [42] i ewolucyjnych [43] modyfikujących początkowo nadaną powierzchnie terenu. Zaletą tych rozwiązań, w przeciwieństwie do klasycznego, proceduralnego podejścia jest interaktywność. Eksperymentuje się także z wykorzystaniem systemów wieloagentowych [44], gdzie wirtualny agent może niezależnie zmodyfikować ukształtowanie terenu poprzez podnoszenie lub opuszczanie poszczególnych punktów ziemi. Przy początkowej konfiguracji mapy jako tafli wody na poziomie morza, zadaniem agentów jest współpracować, aby wyłonić ląd. To nietypowe podejście na razie nie przynosi jednak lepszych efektów ani nie jest wydajniejsze, niż inne metody.

2.1.5. Metody generowania klimatu

Większość prac ogranicza się do generowania powierzchni terenu i nie uwzględniają kwestii zróżnicowania klimatu, który ma znaczny wpływ na odbiór stworzonego świata.

[3] Geomorfologia to nauka o formach rzeźby powierzchni Ziemi i procesach mających na nią wpływ [37].

Reprezentowanie klimatu zazwyczaj polega na przypisywaniu punktów świata do biomów, czyli kategorii ekologicznych opisujących regiony o określonym klimacie, szacie roślinnej i świecie zwierzęcym [45]. Istnieje wiele sposobów klasyfikacji w zależności od preferowanych konwencji i dostępnych zmiennych geograficznych i meteorologicznych, m.in. klasyfikacja Holdridge'a [46], Whittakera [47] czy Waltera [48]. Największy wpływ na klimat mają temperatura i poziom opadów atmosferycznych; niektóre klasyfikacje biorą dodatkowo pod uwagę potencjalną ewapotranspirację (ang. *potential evapotranspiration*, PET), czyli teoretyczne maksimum zdolności oddawania wody do atmosfery, ale raczej nie jest wykorzystywana w generatorach. Wynika to z faktu, że równania szacujące PET albo wykorzystują dużo specyficznych zmiennych (równanie FAO Penmana-Monteitha [49]), albo są zależne wyłącznie od temperatury (równanie Thornthwaite'a [50] lub Priestley'a–Taylora [51]).

Realistyczne symulowanie klimatu jest zupełnie niepraktyczne ze względu na liczbę czynników: zmiennych meteorologicznych, rozmiaru i lokalizacji ciał wodnych lub gór, wynikającego z nich przepływu prądów powietrznych i morskich, wzajemny wpływ temperatury i opadów. Oprócz tego, istniejące modele symulacji klimatu działają głównie w oparciu o dostępność rzeczywistych danych, działają w małej skali lub są skupione na przewidywaniu postępujących zmianach klimatycznych [52].

Najprostsza, sztuczna metoda przypisywania biomów jest podobna do prymitywnej metody tworzenia płyt tektonicznych. Polega na podziale świata na regiony (typowo wielokąty, wykorzystując teselację Woronoja) i losowym wyborze biomów dla całego regionu. Taki sposób znajduje zastosowanie w grach komputerowych z otwartym światem, gdzie gracz ma ograniczone pole widzenia i trudniej dostrzec arbitralnie wyznaczone granice. W innych przypadkach próbuje się szacować temperaturę i opady atmosferyczne w proceduralny sposób, na przykład na podstawie szerokości geograficznej, szumu lub losowego pola wektorów reprezentującego prądy powietrzne. Otrzymane dane mogą być dalej modyfikowane przybliżając efekty wybranych zjawisk zachodzących na Ziemi.

2.2. Symulacja historii

Słowo symulacja jest różnie definiowane w środowisku naukowym: „imitowanie działania rzeczywistego systemu" [53], „odtworzenie działania badanego procesu rzeczywistego na podstawie jego modelu matematycznego" [54], „maszyna dostarczająca spekulatywne lub warunkowe reprezentacje rzeczywistości" [2]. Symulacja komputerowa składa się z trzech elementów: modelu, parametrów i implementacji. Najważniejszym z nich jest model, czyli abstrakcyjna, matematyczna definicja rzeczywistego obiektu lub fenomenu. Wartość wyników symulacji zależy przede wszystkim od tego, jak dobrze model odwzorowuje rzeczywistość [53], [55]–[58].

Historia jest nauką zajmującą się badaniem przyczyn i skutków działań i wytworów ludzkich [59]. Symulacja historii integruje więc szereg symulacji tematycznych: rozwoju gospodarczego, terytorialnego, wojskowego, technologicznego, kulturowego i religijnego ludzkości. W praktyce nie wykorzystuje się istniejących profesjonalnych symulacji z danych dziedzin ze względu na ich sumaryczne wymagania sprzętowe i wzajemną niekompaty-

bilność. Mimo to, znajomość takich rozwiązań może pomóc przy tworzeniu własnych rozwiązań, będąc źródłem inspiracji i praktycznej wiedzy o dziedzinie.

2.2.1. Ekonomia i logistyka

W ekonomii (szczególnie ekonomii eksperymentalnej [53], [58]) i logistyce często wykorzystuje się modele matematyczne i badania ilościowe ze względu na możliwość wyrażania aspektów tych dziedzin w postaci danych liczbowych oraz dużą dostępność danych statystycznych. Takie środowisko znacznie ułatwia komputerowe symulowanie zjawisk ekonomicznych i łańcuchów logistycznych.

Symulacje szczególnie znajdują zastosowanie w przypadku złożonych procesów stochastycznych, które są ciężkie lub niemożliwe do opisania i optymalizowania używając wyłącznie modeli matematycznych [54], lub takich, których przeprowadzenie może się wiązać z wysokimi kosztami lub problemami natury etycznej [58]. Stosuje się je do planowania i optymalizowania procesów produkcji i dostarczania dóbr, przewidywania zachowania rynku lub zwrotów inwestycji.

Klasyczne metody symulacji to arkusze kalkulacyjne, programowanie liniowe, algorytmy harmonogramowania [54], [55], [60]. Polegają one na przedstawieniu problemu jako zbioru równań do rozwiązania lub sekwencyjnie liczonych wyrażeń matematycznych. Często spotykane jest użycie zamkniętego oprogramowania komercyjnego pozwalającego na modelowanie wybranych aspektów biznesowych [60]. Takie metody głównie znajdują zastosowanie kiedy zadaniem jest spełnienie jakiegoś celu przy zadanych ograniczeniach, np. planowania produkcji, wykazania zysku na danym poziomie.

Gdy obiektem zainteresowań jest zmienna natura rynku i zachowanie graczy (producentów i dostawców, emitentów i akcjonariuszy, sprzedawców i klientów), wykorzystuje się systemy wieloagentowe [53], [58], [60]. Symulacja polega na stworzeniu środowiska zawierającego agentów - programy, które autonomicznie podejmują decyzje i wchodzą w interakcję między sobą (w dyskretnych odstępach czasu). Jest to niejako przeniesienie ekonomicznych gier symulacyjnych i eksperymentów z udziałem osób [53] w przestrzeń wirtualną.

2.2.2. Wojskowość

Symulacje komputerowe są wykorzystywane w wojskowości przede wszystkim w celach dydaktycznych [61], ale także przy poważnym wspomaganiu planowania, analizie wpływu wszelakich czynników i weryfikacji założeń (szczególnie na szczeblu operacyjnym i strategicznym) [62]. Można je podzielić na dwie grupy: matematyczne modele konfliktów (np. Lanchestera [63], Collinsa [64], RAND [62]) i tzw. profesjonalne gry wojenne w cyfrowej formie [61], [62], [65].

Symulacje są także wykorzystywane w historii wojskowości, jako dodatkowe narzędzie przy analizie i porównywaniu źródeł opisujących historyczne bitwy [56], [57] oraz nauczaniu historii [56], [57], [61]. W takim przypadku symuluje się historyczne lub fikcyjne bitwy podobnie do rozgrywania gier wojennych na poziomie taktycznym. Projektuje się matema-

tyczny model walki odpowiedni dla epoki i wykorzystuje ludzkich i/lub komputerowych graczy wcielających się w role dowódców wybranego szczebla.

Do symulacji konfliktów wykorzystuje się programy lub zestawy zasad gry operacyjnej projektowane pod konkretny scenariusz. Nie istnieją algorytmy ogólnego zastosowania, ponieważ sposób prowadzenia walki jest bardzo różny w zależności od epoki i tego, w jakiej przestrzeni (ląd, powietrze, morze, przestrzeń kosmiczna) się odbywa. Nie wykorzystuje się także uczenia maszynowego ze względu na szereg czynników: stopień skomplikowania modelu, operacje wykonywane w przestrzeni trójwymiarowej i czasie jednocześnie, dużą rolę losowości i niepełności dostępnych informacji [64], [65], wymóg umiejętności krótko- i długoterminowego planowania oraz wpływu kwestii społecznych i psychologicznych [62], [64].

2.2.3. Inne dziedziny

Niestety, w niektórych dziedzinach nie wykorzystuje się symulacji. Powodem tego jest niedostateczna ilość dostępnych danych, jakościowy charakter badań czy wątpliwości środowiska naukowego wobec wartości symulacji komputerowych jako narzędzia badawczego [58]. Na przykład politologia i stosunki międzynarodowe, które czerpią z teorii gier, nie sięgają raczej po cyfrowe implementacje różnego rodzaju scenariuszy decyzyjnych, choć jest to kierunek potencjalnych badań.

W przypadku administracji i zarządzania często istnieje potrzeba podejmowania decyzji, które mogą podlegać optymalizacji matematycznej, na przykład podziału środków, dotacji i grantów tak, aby uzyskać najlepszy efekt. Do takich zadań można wykorzystywać modele sztucznej inteligencji, z czym się obecnie eksperymentuje [66].

W kwestiach kulturowych, językowych czy religijnych nie stosuje się poważnie symulacji ze względu na zdecydowanie jakościowy rodzaj badań. Niemniej jednak warte rozważenia może być wykorzystywanie systemów wieloagentowych, do modelowania interakcji pomiędzy ludźmi lub grupami, lub algorytmów ewolucyjnych do symulowania ewolucji kultur, doktryn, wierzeń i języków na przestrzeni wieków.

2.3. Istniejące rozwiązania

Istniejące rozwiązania łączące generowanie mapy świata i symulowanie historii można podzielić na trzy kategorie, w kolejności od najbardziej szczegółowych do najbardziej interaktywnych:

1. właściwe symulatory historii bez elementów grywalnych,
2. „gry w boga” (ang. *god games*, nazywane także „boskie gry”) pozwalające na jednoczesną symulację i ingerencję użytkownika w proces,
3. niektóre gry strategiczne, które pozwalają na bezpośrednie sterowanie jednym z rozwijających się społeczeństw.

2.3.1. Symulatory historii

Najbliższe tematowi tej pracy są programy skupiające się na samym aspekcie symulacji, w którym rola użytkownika ogranicza się do dostosowywania wyglądu mapy, warunków

początkowych i doboru parametrów. Są zaawansowane pod względem dokładności modeli świata i są w stanie wytworzyć różnorodne przebiegi historii.

Dwarf Fortress [67] to darmowa gra komputerowa, która posiada generator mapy świata i symulator historii jako jeden z głównych modułów. Pozwala tworzyć zaawansowane światy w stylu *high fantasy*, które gracz może eksplorować w innych trybach gry. Symulator obejmuje zakres do 250 lat, symulując rozwój średniowiecznych miast, królestw i władców zarówno rasy ludzkiej, jak i innych fantastycznych ras. Nazwy lokalizacji, postaci i obiektów są generowane na podstawie stałego słownika fikcyjnych języków. Po zakończeniu symulacji użytkownik może przeczytać wygenerowaną historię w formie encyklopedii. Wadami tej gry są niewielka liczba dostępnych parametrów symulacji, surowa oprawa graficzna i brak możliwości podglądania informacji przed zakończeniem działania. Gra jest rozwijana stale od 2006 roku, lecz kod źródłowy nie jest udostępniony.

Fantasy Worlds History Simulator [68] jest programem skupionym na symulowaniu powstawania, wzrostu i upadku cywilizacji przez tysiące lat na wczytywanych mapach powierzchni planet. Model historii obejmuje rozwój terytorialny, polityczny i naukowy państw, formowanie sojuszy i federacji oraz upadek i rozpad na nowe państwa pod wpływem zewnętrznych i wewnętrznych konfliktów. Projekt, rozwijany od 2020 roku, jeszcze nie jest dostępny publicznie, ale autor publikuje nagrania wideo z sesjami programu.

Worlds: History Simulator [69] to program pozwalający na wygenerowanie pseudo-realistycznej mapy świata (topografia, temperatura, opady i akweny wodne, biomy) i symulowanie rozwoju ludzkości. Rozwijany w latach 2020-2022, niestety może być traktowany tylko jako *proof of concept*, ponieważ pomimo ambitnych planów jest w stanie symulować wyłącznie pierwotne grupy plemienne. Program i kod są dostępne za darmo na licencji MIT.

WorldSim [70] jest projektem w domenie publicznej o podobnym celu, co *Dwarf Fortress*, lecz na znacznie wcześniejszym poziomie rozwoju, tak jak *Worlds: History Simulator*. Umożliwia zarówno generowanie lub wczytywanie uproszczonej mapy świata (podział ląd/woda i przypisany biom), jak i prymitywną symulację ruchomych grup plemiennych i nieruchomych osad. Rozwijany od 2018.

2.3.2. Gry w boga

Worldbox [71] to gra z rodzaju „boskich gier", które pozwalają na ingerencję użytkownika podczas symulacji. Świat jest generowany w prosty sposób, a gra skupia się na rozwoju ludzkości, która buduje osady, uprawia rolę, tworzy państwa, wypowiada wojny i zawiera sojusze. Każda z czynności jest realizowana przez program w prosty sposób, co ułatwia użytkownikowi zrozumienie dziejących się procesów kosztem realizmu. Gra została wydana w 2021 roku.

SimEarth [72] to kolejna „boska gra", wydana w 1990 roku. Skupia się na ewolucyjnym aspekcie tworzenia świata, jako że łączy symulacje rozwoju litosfery i biosfery planety, ewolucji fauny i flory i rozwoju cywilizacji stworzonej przez istoty rozumne (które nie muszą być ludźmi). Pozwala na uzyskanie pseudo-realistycznych światów, lecz podejście

do samej cywilizacji jest bardzo prymitywne, chociażby przez brak podziału na odrębne państwa czy kultury.

2.3.3. Gry strategiczne

Niektóre gry strategiczne z gatunku *grand strategy* lub *4X* (ang. *EXplore, EXpand, EXploit and EXterminate*, eksploruj, dokonuj ekspansji, eksploatuj, eksterminuj) zmieniają rolę użytkownika z pasywnego obserwatora w aktywnego uczestnika tworzącego historię, pozwalając w pełni kontrolować poczynania jednego z wybranych państw czy cywilizacji. W konsekwencji takie programy skupiają się bardziej na „rozgrywaniu" symulacji z jednej perspektywy, niż na zaawansowanej symulacji, chociaż mogą być zmodyfikowane, aby w rozgrywce brali udział tylko gracze komputerowi.

Seria *Cywilizacja* [73] to najsłynniejszy tytuł spośród wymienionych w tej pracy. Gry z tej serii pozwalają na kontrolowanie wybranej istniejącej w rzeczywistości cywilizacji osadzonej w proceduralnie wygenerowanym świecie. Zakres symulacji obejmuje czasy od neolitu aż po niedaleką przyszłość, pozwalając na rozwój ziem, nauki, religii, kultury i miast. Procesy symulowane w grach są uproszczone w celu zwiększenia dostępności gry i wartości edukacyjnej. W porównaniu z wcześniej wymienionymi programami, wadą jest fakt, że liczba cywilizacji (graczy) jest ustalona i niemożliwie jest wyłanianie się nowych ze starych.

Seria *Europa Universalis* [74] to seria gier skupiająca się na symulowaniu rzeczywistej historii świata od XV do XIX wieku. Jest unikatowa pod tym względem, że mapa świata, państwa i ich terytoria są z góry ustalone, a rozgrywka polega na podążaniu biegiem historii lub tworzeniem własnej, alternatywnej. Ze względu na węższy zakres czasowy i osadzenie w realiach historycznych, gry są bardziej realistyczne i szczegółowe, ale mniej elastyczne i przez to tworzące podobne do siebie scenariusze.

3. Model rozwiązania

W ramach pracy zaproponowano teoretyczny model świata, generatora i symulatora, na który składają się model przestrzeni i czasu, wybrane reprezentowane aspekty świata i historii, rodzaje potrzebnych informacji, sposób ich organizacji oraz zależności między nimi, proponowane algorytmy, funkcje i parametry, a także komentarze odnośnie praktycznej implementacji.

3.1. Generacja świata

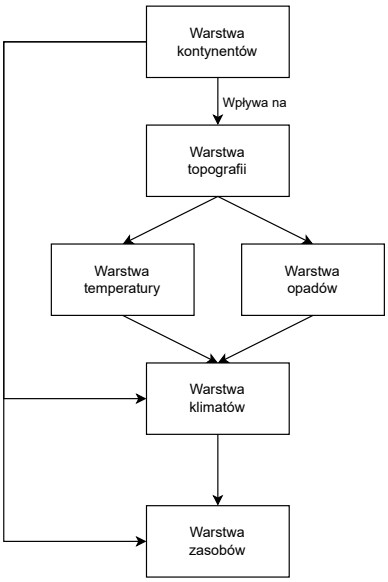
Świat składa się z dwuwymiarowej regularnej siatki punktów o dowolnym, ale skończonym rozmiarze. Współrzędne punktów odpowiadają szerokości i wysokości geograficznej, lecz modelują „płaską" powierzchnię, a nie elipsoidę. Podjęcie takiej decyzji powoduje oczywistą stratę możliwości prawdziwego odwzorowania skali planetarnej, lecz pozwala na jednoczesne poprawne operowanie w średniej skali, kontynentalnej czy regionalnej (Rysunek 3.2). Informacje o świecie przechowywane są w formie warstw zawierających dowolne dane dla każdego punktu. Dane zawarte w jednej warstwie mogą wpływać na generowanie lub interpretowanie danych w innej. Na potrzeby tej pracy zdefiniowano sześć następujących warstw (Rysunek 3.1):

- Warstwa kontynentów,
- Warstwa topografii,
- Warstwa temperatury,
- Warstwa opadów,
- Warstwa klimatu,
- Warstwa zasobów.

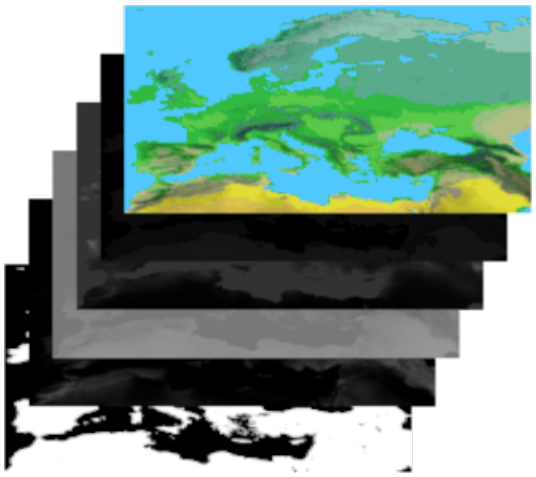
Uznano, że taki zestaw informacji jest wystarczający, by uzyskać akceptowalne wyniki generacji świata i późniejszej symulacji rozwoju cywilizacji. Kontynenty wyznaczają masy lądowe, topografia wpływa na temperaturę i opady, które z kolei decydują o klimacie, a ten określa dostępność zasobów potrzebnych do funkcjonowania cywilizacji. Przyjęty model świata jest na tyle elastyczny, że lista warstw może być w przyszłości dowolnie zredukowana lub rozszerzona (np. o warstwę tektoniczną lub prądów powietrznych i morskich) w zależności od zadania i poziomu szczegółowości. Szczegółowy opis implementacji generatora warstw opartej na algorytmach szumu znajduje się w Podrozdziale 5.1.

3.1.1. Warstwa kontynentów

Wyznacza linię brzegową kontynentów poprzez binaryzację powierzchni świata na lądy i wody w dowolny sposób. Implementacja modelu w tej pracy realizuje to za pomocą parametru światowego poziomu morza i wygenerowanej mapy pseudo-wysokości, gdzie punkty na mapie powyżej poziomu morza są oznaczane jako ląd. Nie jest to jednak jedyny sposób na wyznaczanie linii brzegowej – jako przykład można podać skonstruowanie, zniekształcenie i rasteryzację krawędzi diagramu Woronoja dla losowo wybranych punktów przestrzeni albo metody opisane w poprzednim rozdziale (generatywne modele sztucznej inteligencji, systemy wieloagentowe, symulacje tektoniczne).



Rysunek 3.1. Diagram warstw świata i zależności między nimi.



Rysunek 3.2. Warstwy świata na przykładzie mapy Europy.

3.1.2. Warstwa topografii

Opisuje rzeźbę powierzchni terenu używając wysokość punktów nad poziomem morza. Separacja podziału ląd/woda i właściwej topografii pozwala uniknąć zjawiska częstego w naiwnych generatorach terenu, jakim jest tworzenie gór zawsze na środku kontynentu, zawsze o podobnym do kontynentu kształcie, podczas gdy w rzeczywistości góry są w formie pasm i nie muszą się znajdować w głębi kontynentu (na przykład Andy w Ameryce Południowej). Należy jednak wprowadzić algorytm wygładzający punkty znajdujące się przy linii brzegowej, żeby nie dopuścić do powstawania losowych kilometrowych klifów, jeśli wybrzuszenie terenu w warstwie topografii nałoży się na styk lądu i wody w warstwie kontynentów. Model świata nie obejmuje morfologii terenów podwodnych, ponieważ nie mają wpływu na późniejsze symulowanie ludzkości.

3.1.3. Warstwa temperatury

Zawiera średnią roczną temperaturę powietrza na powierzchni, która jest wyznaczana na podstawie szerokości geograficznej. Temperatura jest określona parametrami dla dziewięciu równoleżników, pomiędzy którymi wartości są interpolowane:

- Równik (0° N),
- Zwrotniki (23° N i S23° S),
- Strefy umiarkowane (46° N i 46° S),
- Strefy podbiegunowe (69° N i 69° S),
- Bieguny (90° N i 90° S).

Model uwzględnia także wpływ wysokości terenu na temperaturę, czyli tzw. gradient temperatury (ang. *lapse rate*) [75]. Rozważany jest gradient wilgotnoadiabatyczny (MALR,

ang. *moist adiabatic lapse rate*), czyli spadek temperatury występujący w przypadku unoszącego się powietrza częściowo nasyconego wodą [76]. MALR jest zależny od temperatury, wilgotności i ciśnienia atmosferycznego i przyjmuje wartości w zakresie od 3.6° C/km do 9.2° C/km, średnio 6° C/km dla klimatów górskich [77]. W pracy został uproszczony do podawanej przez użytkownika stałej, domyślnie[4] wynoszącej wartość 5° C/km, w przybliżeniu odpowiadającą połowie wartości stałego gradientu suchoadiabatycznego 9.8° C/km (DALR, ang. *dry adiabatic lapse rate*).

3.1.4. Warstwa opadów

Zawiera średnią roczną ilość opadów atmosferycznych. Informacje o opadach są generowane na podstawie szerokości geograficznej w sposób analogiczny do temperatury. W przeciwieństwie do temperatury, takie uproszczenie powoduje mocno dostrzegalne różnice pomiędzy danymi wygenerowanymi a rzeczywistymi (Podrozdział 6.2), ponieważ ignoruje wpływ ważnych czynników, które bardzo trudno symulować, co omówiono w poprzednim rozdziale. Wysokości terenu ma także inny wpływ na ilość opadów, niż w przypadku temperatury. Poziom opadów rośnie do pewnej wysokości, nazywanej wysokością maksymalnych opadów (AMP, ang. *altitude of maximum precipitation* lub MAP, ang. *maximum altitude of precipitation*), a następnie zaczyna spadać wraz z dalszym wzrostem wysokości. Zjawisko to jest nazywane inwersją opadów. Podobnie jak MALR, AMP różni się w zależności od rzeźby terenu i warunków klimatycznych, osiągając wartości w przedziale 1000-5000 m n.p.m w zależności od badanych gór [78]–[80]. Zarówno AMP, jak i współczynnik liniowego spadku przybliżający to zjawisko, są parametrami dostępnymi dla użytkownika i wynoszą domyślnie[5] 2000 m i 1.5 mm/m.

3.1.5. Warstwa klimatu

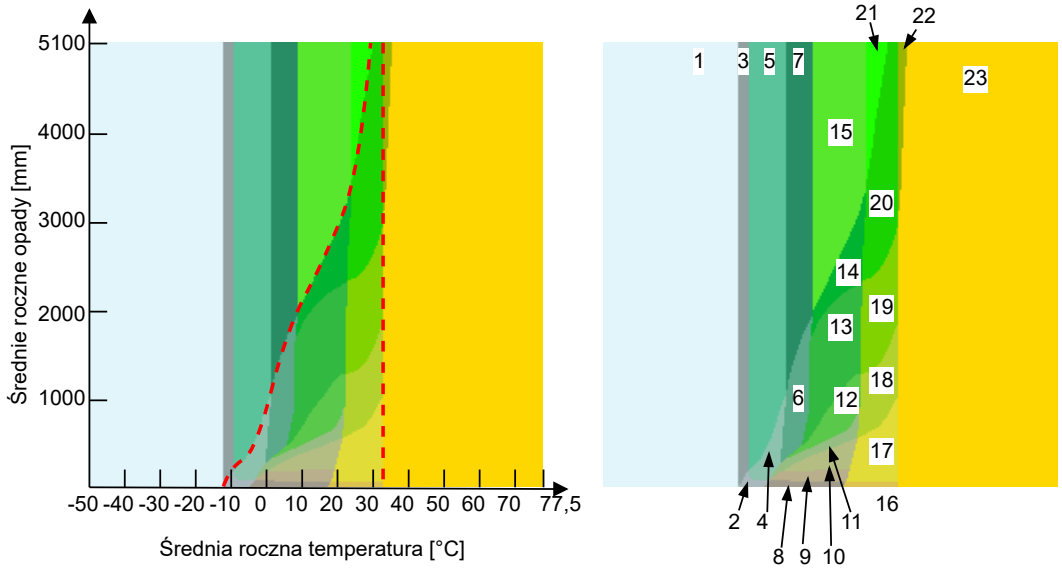
Przypisuje punkty świata do biomów. Klasyfikacja biomów stosowana w tej pracy została oparta o klasyfikację Whittakera [47], która definiuje biomy na podstawie temperatury i opadów atmosferycznych. Rysunek 3.3 przedstawia zaadoptowany diagram biomów wg. Whittakera, rozszerzony o dodatkowe i nieistniejące biomy tak, aby pokryć cały zakres użytych w implementacji wartości temperatury i opadów atmosferycznych. Oryginalne biomy zostały odseparowane czerwoną linią przerywaną. Biomy ponumerowane na rysunku zostały opisane poniżej (* oznacza dodany biom, nie występujący w pracy Whittakera):

1. Pustynia lodowa (ekstremalna)*,
2. Pustynia lodowa*,
3. Pustynia lodowa (mokra)*,
4. Tundra,
5. Tundra (mokra)*,
6. Tajga / Las borealny,
7. Tajga / Las borealny (mokry)*,
8. Zimna pustynia (sucha)*,

[4] Wartość dobrano eksperymentalnie, porównując wyniki generatora dla mapy Ziemi z rzeczywistymi danymi pomiarowymi (Podrozdział 6.2).

[5] Wartości dobrane arbitralnie dla gór o średniej wysokości.

9. Zimna pustynia,
10. Step,
11. Scrub,
12. Rzadki las strefy umiarkowanej,
13. Las strefy umiarkowanej,
14. Las deszczowy strefy umiarkowanej,
15. Las deszczowy strefy umiarkowanej (mokry)*,
16. Gorąca pustynia (sucha)*,
17. Gorąca pustynia,
18. Sawanna,
19. Las tropikalny,
20. Równikowy las deszczowy,
21. Równikowy las deszczowy (mokry)*,
22. Równikowy las deszczowy (ekstremalny)*,
23. Gorąca pustynia (ekstremalna)*,
24. Biom morski*, przypisany wyłącznie punktom leżącym poza lądem.



Rysunek 3.3. Wykres biomów w zależności od temperatury i opadów.

Do każdego biomu dodatkowo przypisano współczynnik „zamieszkania” (ang. *habitability*) opisujący łatwość osadzania się ludności w zakresie od 0 (niemożliwy do przeżycia) do 1 (pełen komfort). Współczynnik nie ma znaczenia przy generowaniu świata, ale jest wykorzystywany w symulacji przy wyborze punktów startowych i kolonizacji terytorium. Lista i granice biomów mogą być dowolnie modyfikowane przez użytkownika.

3.1.6. Warstwa zasobów

Posiada informacje o rodzajach i rozmiarach złóż zasobów naturalnych, które mogą być eksploatowane podczas symulacji, aby uzyskać zasoby potrzebne do rozwoju cywilizacji.

Każdy punkt może zawierać dowolną liczbę złóż zasobów, które mogą być przypisane losowo lub na podstawie biomu. Arbitralnie zdefiniowano domyślną listę złóż, która może być dowolnie rozszerzana w zależności od potrzeb użytkownika:

1. Grunty orne*,
2. Pastwiska,
3. Zwierzęta łowne*,
4. Tereny lesiste,
5. Łowisko ryb,
6. Złoże kamienia i gliny,
7. Złoże metali*,
8. Złoże węgla.

Złoża 1-4 występują tylko w niektórych biomach – im lepsze warunki rozwoju biosfery, tym większa szansa na pojawienie się tych złóż i tym większy ich rozmiar. Złoże nr 5, łowiska ryb występują tylko w biomie morskim. Reszta może pojawić się wszędzie, ponieważ reprezentuje surowce nieodnawialne. Złoża oznaczone * posiadają warianty, których eksploatacja przynosi bogactwo zamiast zasobów przemysłowych. Różnice pomiędzy zasobami zostaną omówione w następnym podrozdziale.

Po wygenerowaniu wszystkich warstw świat może być użyty jako środowisko symulacyjne.

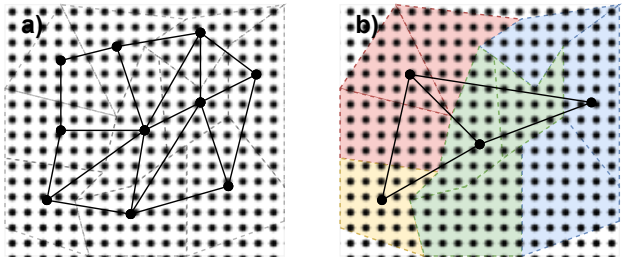
3.2. Symulacja historii

Symulacja polega na iteracyjnym wykorzystywaniu określonych funkcji i algorytmów nazywanych „systemami" do obliczenia wybranych aspektów stanu wszystkich modelowanych obiektów: regionów, państw i konfliktów. Ludzkość jest reprezentowana poprzez regiony, czyli zbiory punktów mapy świata wyznaczające pewne terytorium z przypisaną liczbą ludności, dostępnymi złożami zasobów i innymi statystykami opisanymi później. Każdy punkt mapy może należeć do co najwyżej jednego regionu. Regiony należą do państw (ang. *polity*), czyli bytów reprezentujących społeczeństwo o wspólnej puli zasobów, poziomie rozwoju nauki i kultury i możliwości podejmowania decyzji odnośnie polityki wewnętrznej i zagranicznej. Na podstawie powierzchni wyznaczanej przez zbiór punktów regionu można otrzymać kształt regionu i graf sąsiedztwa pomiędzy regionami i państwami (Rysunek 3.4). Trzecim rodzajem obiektu w symulacji jest konflikt, który reprezentuje tymczasowe starcie zbrojne dziejące się pomiędzy co najmniej dwoma państwami.

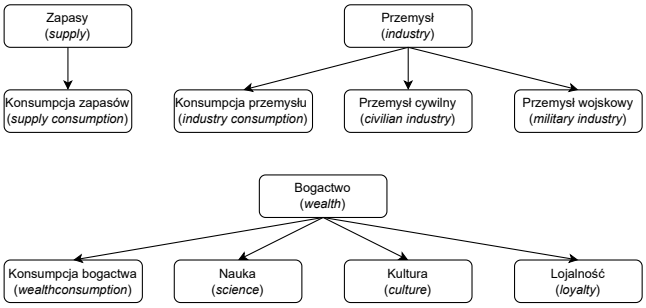
Założono dyskretny model czasu symulacji, w którym pojedynczy krok odpowiada miesiącowi w symulowanym świecie. Symulacja nie ma wyznaczonego warunku ani miejsca stopu – trwa, dopóki nie zostanie przerwana przez użytkownika. Poszczególne zmiany w modelu odbywają się co krok lub co dwanaście kroków (symulowany rok) w zależności od systemu. Kolejność wykonywania systemów jest ustalona[6]:

1. System ekspansji terytorialnej,
2. System wykrywania zasobów,

[6] Systemy oznaczone * wykorzystują krok roczny zamiast miesięcznego.



Rysunek 3.4. Podział na (a) regiony i (b) państwa nałożony na siatkę punktów świata. Linie przerywane oznaczają granice regionów, kolory przynależność do państw, a linie ciągłe i kropki reprezentują krawędzie i wierzchołki grafów sąsiedztwa.



Rysunek 3.5. Rodzaje zasobów.

3. System podziału pracy,
4. System ekonomii,
5. System urbanizacji*,
6. System rozwoju kultury*,
7. System rozwoju nauki*,
8. System wzrostu populacji,
9. System dyplomacji,
10. System konfliktów,
11. System polityki*.

W dużym skrócie, symulowane państwa produkują zasoby w zależności od złóż dostępnych na posiadanym terytorium i wykorzystują zasoby do wykonywania czynności pozwalających uzyskać jeszcze więcej zasobów:

- Zaopatrzenie – do utrzymania i zwiększenia liczby ludności,
- Przemysł – do zwiększenia powierzchni zajmowanego terytorium (pokojowo lub nie) lub rozwoju regionalnej infrastruktury,
- Bogactwo – do rozwoju nauki i kultury zwiększających wydajność funkcjonowania.

Decyzje odnośnie podziału zasobów, priorytetyzacji dziedzin nauk, tradycji, budynków, ścieżek rozwoju, stosunków międzynarodowych są podejmowane na podstawie profilu polityki państwa, która składa się z pięciu strategii (Tabela 3.5, Tabela 3.6) o zakresie wartości $[0,1]$:

- Ekspansjonizm – podejście do rozwoju terytorium: czy państwo powinno zajmować nowe tereny, czy ulepszać już posiadane,
- Agresywność – relacje z sąsiadami: czy państwo powinno dążyć do podbojów, czy do zawierania sojuszy,
- Rozwojowość – podział pracy: czy państwo powinno produkować więcej bogactwa używanego do rozwoju naukowego i kulturalnego, czy przemysłu,
- Militaryzm – podział zasobów: czy państwo powinno inwestować w przemysł wojskowy i lojalność, czy w przemysł cywilny i pokojowy rozwój,
- Progresywizm – wybranie ścieżki rozwoju: czy państwo skupia się na rozwoju naukowym, czy kulturowym,
- Praworządność – dążenie do wewnętrznej stabilności kosztem potencjalnie mniejszej produkcji.

Dobór wartości strategii jest więc najbardziej „wysokopoziomową" czynnością wywierającą wpływ na dzieje państwa.

Symbol	Nazwa	Do czego używany?
z_{sup}	Zaopatrzenie	Konsumpcja.
z_{ind}	Przemysł	Podział na konsumpcję, trybut, przemysł cywilny i wojskowy.
z_{wel}	Bogactwo	Podział na konsumpcję, trybut, naukę, kulturę, lojalność.
z_{civ}	Przemysł cywilny	Kolonizacja i urbanizacja.
z_{mil}	Przemysł wojskowy	Sprzęt wojskowy.
z_{sci}	Nauka	Rozwój dziedzin nauki.
z_{cul}	Kultura	Rozwój tradycji i dziedzictwa.
z_{loy}	Lojalność	Organizacja wojskowa.

Tabela 3.1. Lista rodzajów zasobów.

Symbol	Nazwa	Co zwiększa?
s_{geo}	Nauki o Ziemi	Wydajność ekstrakcji zasobów ze złóż.
s_{med}	Medycyna	Przyrost naturalny i wydajność szpitali.
s_{eng}	Inżynieria	Wydajność przemysłu cywilnego.
s_{mtl}	Metalurgia	Wydajność przemysłu wojskowego.
s_{phi}	Filozofia	Szybkość rozwoju kulturowego.
s_{mat}	Matematyka	Szybkość rozwoju naukowego.
s_{mgt}	Zarządzanie	Wydajność lojalności.
s_{law}	Prawo	Wydajność sądów (stabilność).
s_{lng}	Lingwistyka	Szybkość zmiany relacji zagranicznych.
s_{mil}	Wojskowość	Siła wyposażenia wojska.

Tabela 3.2. Lista dziedzin nauki.

Reszta rozdziału została poświęcona opisowi parametrów i sposobu działania poszczególnych systemów w formie tekstu i równań. Ponieważ na symulacje składa się łącznie ponad sto parametrów i kilkadziesiąt równań, w pracy konsekwentnie korzysta się z następującego schematu oznaczeń i symboli:

- τ – obecny krok symulacji,

Symbol	Nazwa	Co zwiększa?
h_{pnr}	Pionierstwo	Szybkość ekspansji terytorialnej.
h_{mon}	Monumentalność	Szybkość urbanizacji regionów.
h_{cur}	Ciekawość	Szybkość rozwoju naukowego.
h_{cre}	Kreatywność	Szybkość rozwoju kulturowego.
h_{pros}	Zamożność	Wydajność ekstrakcji zasobów ze złóż.
h_{aut}	Władza	Stabilność.
h_{dip}	Dyplomacja	Zwiększa punkty wkładu w konflikt, zmniejsza koszty trybutu.
h_{sup}	Supremacja	Siła organizacji wojska i buntowniczość.

Tabela 3.3. Lista tradycji.

Symbol	Nazwa	Czego przepustowość zwiększa?
b_{hos}	Szpital	Służby zdrowia.
b_{man}	Manufaktura	Przemysł cywilny.
b_{frg}	Kuźnia	Przemysł wojskowy.
b_{uni}	Uniwersytet	Rozwój nauki.
b_{amt}	Amfiteatr	Rozwój kultury.
b_{crt}	Sąd	Służby bezpieczeństwa.
b_{frt}	Forteca	Poziom fortyfikacji.

Tabela 3.4. Lista budynków.

Strategia	Nazwa	Nauka	Tradycja
l_{exp}	Ekspansjonizm	s_{med} vs s_{geo}	h_{mon} vs h_{pnr}
l_{cmp}	Agresywność	s_{lng} vs s_{mil}	h_{dip} vs h_{sup}
l_{mct}	Rozwojowość	-	-
l_{mil}	Militaryzm	s_{eng} vs s_{mtl}, s_{mgt}	-
l_{prg}	Progresywizm	s_{phi} vs s_{mat}	h_{cre} vs h_{cur}
l_{lgl}	Praworządność	$-$ vs s_{law}	h_{pros} vs h_{aut}

Tabela 3.5. Wpływ strategii na podział nauki i kultury.

Strategia	Nazwa	Budynek	Zasób
l_{exp}	Ekspansjonizm	b_{hos} vs $-$	-
l_{cmp}	Agresywność	b_{frt} vs $-$	-
l_{mct}	Rozwojowość	-	z_{ind} vs z_{wel}
l_{mil}	Militaryzm	b_{man} vs b_{frg}	z_{civ} vs z_{mil}, z_{loy}
l_{prg}	Progresywizm	b_{amt} vs b_{uni}	z_{cul} vs z_{sci}
l_{lgl}	Praworządność	$-$ vs b_{crt}	-

Tabela 3.6. Wpływ strategii na podział zasobów i budynków.

- $T = \lfloor \frac{\tau}{12} \rfloor$ – rok odpowiadający obecnemu krokowi,
- W, w – zbiór wszystkich punktów świata, pojedynczy punkt,
- $W_x(w, \tau)$ – właściwość x punktu świata w w kroku τ,
- P, p – zbiór wszystkich państw, pojedyncze państwo,
- $P_x(p, \tau)$ – właściwość x państwa p w kroku τ,
- R, r – zbiór wszystkich regionów, pojedynczy region,

- $R_x(r,\tau)$ – właściwość x regionu r w kroku τ,
- C, c – zbiór wszystkich konfliktów, pojedynczy konflikt,
- $C_x(c,p,\tau)$ – właściwość x uczestnika p w konflikcie c w kroku τ,
- $Z = Z' \cup Z'', z$ – zbiór wszystkich typów zasobów, pojedynczy typ zasobu,
- $Z' = \{z_{sup}, z_{ind}, z_{wel}\}$ – zbiór typów podstawowych zasobów,
- $Z'' = \{z_{civ}, z_{mil}, z_{sci}, z_{cul}, z_{loy}\}$ – zbiór typów zaawansowanych zasobów,
- D, d – zbiór wszystkich typów złóż zasobów, pojedynczy typ złoża zasobów,
- S, s – zbiór wszystkich dziedzin nauki, pojedyncza dziedzina nauki,
- H, h – zbiór wszystkich tradycji, pojedyncza tradycja,
- B, b – zbiór wszystkich typów kluczowych struktur, pojedynczy typ kluczowej struktury,
- L, l – zbiór wszystkich strategii, pojedyncza strategia,
- k_x – parametr x dostępny dla użytkownika,
- $k_{x,y}$ – parametr x zależny od y dostępny dla użytkownika,
- $Fun(x,y,z)$ – funkcja Fun (pisownia wielką literą) przyjmująca argumenty x, y, z, dostępna dla wszystkich systemów,
- $fun(x,y,z)$ – funkcja pomocnicza fun (pisownia małą literą) przyjmująca argumenty x, y, z, dostępna tylko dla wybranego systemu,

3.2.1. System ekspansji terytorialnej

System reprezentuje odkrywanie i kolonizowanie niezamieszkałych ziem. Zajmowanie terenu zwiększa ilość dostępnych złóż zasobów oraz pozwala nawiązać kontakt z innymi państwami. Model symulacji zakłada, że tylko sąsiednie punkty lądowe na mapie mogą być kolonizowane i nie pozwala na bezpośrednie zajmowanie odległych terenów, np. na innym kontynencie. Algorytm przedstawiony poniżej opisuje warunki i sposób przeprowadzenia ekspansji.

- k_{col} – parametr bazowego kosztu kolonizacji punktu terenu,
- k_{spwl} – parametr zwiększenia kosztu za liczbę regionów w państwie,
- $R_{col}(r,\tau)$ – pula punktów kolonizacji dla regionu r,
- $P_r(p,\tau)$ – zbiór wszystkich regionów państwa p,
- $W_{hab}(w)$ – współczynnik zamieszkania dla biomu przypisanego do punktu w.

Algorytm jest wykonywany dla każdego regionu $r \in R$:

1. Sprawdź, czy $R_{col}(r,\tau) \geq k_{col} + k_{spwl} \cdot |P_r(p,\tau)|$. Jeżeli nie, zakończ działanie.
2. Znajdź wszystkie punkty świata spełniające wszystkie poniższe warunki:
   - Punkt nie należy do żadnego regionu,
   - Punkt sąsiaduje z punktem należącym do tego regionu,
   - Punkt znajduje się na lądzie,
   - Punkt spełnia warunek $W_{hab}(w) > 0$.
3. Jeżeli żaden punkt nie spełnia warunków, zakończ działanie.
4. Każdemu ze znalezionych punktów przypisz wagę równą $W_{hab}(w)$,
5. Dokonaj losowego ważonego wyboru jednego punktu ze zbioru,

6. Dodaj wylosowany punkt w do zbioru punktów regionu, dodaj złoża zasobów tego punktu do sumy złóż zasobów regionu,
7. Zmniejsz $R_{col}(r,\tau)$ o $k_{col} \cdot W_{hab}(w)$.

3.2.2. System wykrywania zasobów

System wyznacza maksymalną ilość dostępnych zasobów na podstawie typu i rozmiaru wszystkich złóż zasobów znajdujących się na terenie regionu oraz poziomu infrastruktury i nauk, które dodatkowo zwiększają rozmiar złóż i wydajność ekstrakcji.

- k_{dev+} – parametr zwiększenia produkcji za każdy poziom rozwoju regionu,
- $k_{z,d}$ – parametr wydajności ekstrakcji zasobu z dla złoża zasobów d,
- $R_{dev}(r,\tau)$ – poziom rozwoju dla regionu r,
- $R_{dep}(r,d,\tau)$ – rozmiar złoża zasobów d dla regionu r,
- $P_r(p,\tau)$ – zbiór wszystkich regionów państwa p,
- $Science(p,s,T)$ – funkcja zwracająca mnożnik wynikający z nauki s dla państwa p,
- $P_{resmax}(p,z,\tau)$ – teoretyczne maksimum ilości zasobu z dla państwa p,
- $P_{res+}(p,\tau)$ – mnożnik wydajności ekstrakcji zasobów dla państwa p.

$$P_{res+}(p,\tau) = \sum_{r}^{P_r(p,\tau)} (R_{dev}(r,\tau) \cdot k_{dev+}) \cdot Science(p, s_{geo}, T) \tag{3.1}$$

$$P_{resmax}(p,z,\tau) = \left(\sum_{r}^{P_r(p,\tau)} \sum_{d}^{D} R_{dep}(r,d,\tau) \cdot k_{z,d} \right) \cdot P_{res+}(p,\tau), \quad \text{gdzie} \quad z \in Z' \tag{3.2}$$

3.2.3. System podziału pracy

Populacja państwa jest podzielona na ludność pracującą w trzech sektorach produkujących trzy podstawowe zasoby i na ludność służącą w wojsku. Algorytm wyznacza liczbę pracowników sektora na podstawie przewidywanych wartości zapotrzebowania społeczeństwa i teoretycznej ilości zasobu, które jest w stanie wyprodukować. Ponieważ zaopatrzenie jest zasobem reprezentującym żywność i inne artykuły spożywcze potrzebne do przeżycia, produkcja tego zasobu ma największy priorytet.

- $k_{eff,z}$ – parametr wydajności produkcji dla zasobu z,
- $P_{pop}(\tau)$ – liczba pracującej ludności w państwie p,
- $P_{res+}(p,\tau)$ – mnożnik wydajności ekstrakcji zasobów dla państwa p,
- $P_{tbin}(p,z,\tau)$ – ilość zasobu z otrzymanego przez państwo p w ramach trybutu,
- $P_{need}(p,z,\tau)$ – ilość zasobu z wymagana przez populację państwa p,
- $P_{split}(p,z,T)$ – procent ludności przeznaczony do produkcji zasobu z państwa p,
- $P_{jobsm}(p,z,\tau)$ – minimalna liczba ludności zatrudnionej w sektorze zasobu z państwa p, pokrywająca zapotrzebowanie,
- $Tradition(p,h,T)$ – funkcja zwracająca mnożnik wynikający z tradycji h dla państwa p,
- $P_{jobs}(p,z,\tau)$ – całkowita liczba ludności zatrudnionej w sektorze zasobu z państwa p.

$$coeff(p,z,\tau) = k_{eff,z} \cdot P_{res+}(p,\tau) \cdot Tradition(p,h_{pros},T) \tag{3.3}$$

$$P_{jobsm}(p,z_{sup},\tau) = min\left(\frac{P_{need}(p,z_{sup},\tau)}{coeff(p,z_{sup},\tau)}, P_{pop}\right) \tag{3.4}$$

$$P_{jobsm}(p,z_{ind},\tau) = min\left(\frac{P_{need}(p,z_{ind},\tau) - P_{tbin}(p,z_{ind},\tau)}{coeff(p,z_{ind},\tau)}, P_{pop}(\tau) - P_{jobsm}(p,z_{sup},\tau)\right) \tag{3.5}$$

$$P_{jobsm}(p,z_{wel},\tau) = min\left(\frac{P_{need}(p,z_{wel},\tau) - P_{tbin}(p,z_{wel},\tau)}{coeff(p,z_{wel},\tau)}, P_{pop}(\tau) - \sum_{y}^{\{z_{sup},z_{ind}\}} P_{jobsm}(p,y,\tau)\right) \tag{3.6}$$

Jeżeli wszyscy ludzie pracują, by w ogóle utrzymać społeczeństwo, system kończy obliczenia w tym momencie. W innym razie reszta ludności zdolnej do pracy będzie produkować nadwyżkę zasobów, które przyczynią się do rozwoju państwa. Na początku symulacji zdecydowana zasobów będzie przeznaczona na utrzymanie liczby ludności i spełnienie ich potrzeb, ale wraz ze wzrostem wydajności *per capita* wynikającym z rozwoju naukowego, kulturowego i budowy infrastruktury, coraz mniejszy procent ludności będzie musiał pracować na utrzymanie społeczeństwa.

$$P_{jobs}(p,z,\tau) = P_{split}(p,z,T) \cdot \left(P_{pop}(\tau) - \sum_{y}^{Z'} P_{jobsm}(p,y,\tau)\right), \quad \text{gdzie} \quad z \in Z' \tag{3.7}$$

3.2.4. System ekonomii

System ekonomii jest odpowiedzialny za przydzielanie wydobytych zasobów podstawowych na podstawie liczby ludności zatrudnionej w danym sektorze, wydajności ekstrakcji i dostępnych złóż zasobów. Zasoby podstawowe są następnie przekształcane w zasoby zaawansowane, które są wykorzystywane w innych systemach (Rysunek 3.5).

- $k_{eff,z}$ – parametr wydajności produkcji dla zasobu z,
- $k_{effcap,z}$ – parametr wydajności produkcji powyżej przepustowości dla zasobu z,
- $k_{tbratio}$ – parametr wielkości płatności trybutu jako procentu wyprodukowanych zasobów,
- k_{msth}, k_{lsth} – parametry czasu magazynowania zasobów przemysłu wojskowego i lojalności,
- $P_{split}(p,z,T)$ – procent zasobów przeznaczony do produkcji zasobu z państwa p,
- $P_{resmax}(p,z,\tau)$ – teoretyczne maksimum ilości zasobu z dla państwa p,
- $P_{need}(p,z,\tau)$ – ilość zasobu z wymagana przez populację państwa p,
- $P_{tbin}(p,z,\tau)$ – ilość zasobu z otrzymanego przez państwo p w ramach trybutu,
- $P_{trib}(p,\tau)$ – lista procentowych udziałów odbiorców trybutów, które płaci państwo p,

- $Tradition(p,h,T)$ – funkcja zwracająca mnożnik wynikający z tradycji h dla państwa p,
- $Science(p,s,T)$ – funkcja zwracająca mnożnik wynikający z nauki s dla państwa p,
- $Capacity(p,b,T)$ – funkcja zwracająca przepustowość typu budynku b dla państwa p,
- $P_{jobs}(p,z,\tau)$ – całkowita liczba ludności zatrudnionej w sektorze zasobu z państwa p,
- $P_{tbout}(p,z,\tau)$ – ilość zasobu z oddanego przez państwo p w ramach trybutu,
- $P_{res}(p,z,\tau)$ – ilość zasobu z wyprodukowanego przez państwo p w miesiącu τ,
- $P_{acc}(p,z,T)$ – ilość zasobu z skumulowana przez państwo p w roku T.

$$P_{res}(p,z,\tau) = max\left(\left(P_{jobs}(p,z,\tau)\cdot k_{eff,z}\cdot P_{res+}(p,\tau)\cdot Tradition(p,h_{pros},T)\right), P_{resmax}(p,z,\tau)\right) + P_{tbin}(p,z,\tau-1), \quad \text{gdzie} \quad z\in Z' \tag{3.8}$$

Alternatywnym źródłem zasobów przemysłowych i bogactwa są trybuty $P_{tbin}(p,z,\tau)$, czyli zasoby otrzymywane od innych państw po wygraniu wojny. Wartość trybutu jest równoważna wartości odejmowanej od puli zasobów płatnika ($P_{tbout}(p,z,\tau)$). Trybuty są opłacane przez ustaloną liczbę miesięcy.

$$P_{tbout}(p,z,\tau) = \sum_{x}^{P_{trib}(p,\tau)}\left(P_{res}(p,z,\tau)\cdot x\cdot k_{tbratio}\cdot k_{eff,z}\right), \quad \text{gdzie} \quad z\in\{z_{ind},z_{wel}\} \tag{3.9}$$

Oprócz ewentualnych trybutów wobec przeciwników, państwo musi także przekazać pewną ilość zasobów podstawowych (szczególnie zaopatrzenie) własnemu społeczeństwu w ramach konsumpcji realizowanej przez system wzrostu populacji. Pozostałe zasoby podstawowe są wykorzystane do produkcji zasobów zaawansowanych, która odbywa się z pełną wydajnością dopóki jest poniżej limitu przepustowości $Capacity(p,b,T)$ danego zasobu. Przepustowość jest określona jako suma poziomów regionalnych budynków odpowiedzialnych za dany zasób, np. uniwersytety i zasoby naukowe. Jeżeli ilość przeznaczonych zasobów przekracza przepustowość, nadmiar jest wykorzystywany ze znacznie niższą wydajnością. W ten sposób zwiększenie produkcji osiąga się nie poprzez samo zwiększenie zasobów wejściowych, ale też inwestowanie w kluczowe zakłady pracy.

$$yield(p,z,z^*,s,h,b,\tau) = max\left(\left(P_{res}(p,z^*,\tau) - P_{need}(p,z^*,\tau) - P_{tbout}(p,z^*,\tau)\right)\cdot P_{split}(p,z,T), 0\right) \cdot k_{eff,z}\cdot Science(p,s,T)\cdot Tradition(p,h,T) \tag{3.10}$$

$$
\begin{aligned}
P_{res}(p,z,\tau) = {} & min\big(yield(p,z,z^*,s,h,b,\tau), Capacity(p,b,T)\big) \\
& + max\big(\big(yield(p,z,z^*,s,h,b,\tau) - Capacity(p,b,T)\big), 0\big) \cdot k_{effcap,z}, \\
& \text{gdzie} \quad (z,z^*,s,h,b) \in \big\{(z_{civ}, z_{ind}, s_{eng}, -, b_{man}), (z_{mil}, z_{ind}, s_{mtl}, -, b_{frg}), \\
& (z_{sci}, z_{wel}, s_{mat}, h_{cur}, b_{uni}), (z_{cul}, z_{wel}, s_{phi}, h_{cre}, b_{amt}), (z_{loy}, z_{wel}, s_{mgt}, -, -)\big\}
\end{aligned} \tag{3.11}
$$

Wybrane typy zasobów są kumulowane przez okres czasu. W przypadku przemysłu cywilnego, punktów nauki i kultury powodowane jest to faktem, że systemy operujące na tych zasobach działają co rok, a nie co miesiąc. W przypadku przemysłu wojskowego i lojalności, zasoby te są magazynowane na określoną liczbę miesięcy k_{msth}, k_{lsth} na wypadek konfliktów.

$$
P_{acc}(p,z,T) = \sum_{i=1}^{12} P_{res}(p,z,12(T-1)+i), \quad \text{gdzie} \quad z \in Z' \tag{3.12}
$$

3.2.5. System urbanizacji

System urbanizacji odpowiada za dystrybucję państwowych zasobów przemysłu cywilnego pomiędzy regiony, które mogą wykorzystać je do ekspansji i urbanizacji. Ekspansja dzieli się na kolonizację terenu (obsługiwaną przez omówiony system ekspansji terytorialnej) i tworzenie nowych regionów (poprzez budowę miast) i generalnie służy do zwiększania potencjalnej liczby dostępnych złóż zasobów, podczas gdy urbanizacja zwiększa wydajność produkcji i statystyki społeczne.

- k_{devm} – parametr maksymalnego poziomu rozwoju regionu,
- k_{exps}, k_{devs} – parametry szybkości akumulacji punktów ekspansji / urbanizacji,
- k_{spwl} – parametr zwiększenia kosztu za liczbę regionów w państwie,
- k_{devl} – parametr kosztu rozwoju regionu w zależności od obecnego poziomu rozwoju,
- k_{cap} – parametr bazowej przepustowości dowolnego budynku,
- $k_{spd,b}$ – parametr szybkości budowy poziomu budynku b,
- $k_{bon,b}$ – parametr bonusu budynku b,
- k_{city} – parametr kosztu nowego regionu,
- k_{rsz} – parametr minimalnego rozmiaru regionu, który może stworzyć nowy region,
- $Capacity(p,b,T)$ – funkcja zwracająca przepustowość budynku b dla państwa p,
- $Science(p,s,T)$ – funkcja zwracająca mnożnik wynikający z nauki s dla państwa p,
- $Tradition(p,h,T)$ – funkcja zwracająca mnożnik wynikający z tradycji h dla państwa p,
- $R_{dev}(r,\tau)$ – poziom rozwoju dla regionu r,
- $R_{col}(r,\tau), R_{reg}(r,T)$ – pula punktów kolonizacji / budowy miasta dla regionu r,
- $R_{str}(r,b,T)$ – poziom budynku b dla regionu r,
- $R_{med}(r,T), R_{sec}(r,\tau)$ – służby medyczne / służby bezpieczeństwa w regionie r,
- $P_r(p,\tau)$ – zbiór wszystkich regionów państwa p,

- $P_{rege}(p,\tau), P_{regd}(p,\tau)$ – zbiór regionów państwa p, które mogą dokonać ekspansji / mogą się rozwinąć,
- $P_{pol}(p,l)$ – wartość strategii l państwa p,
- $P_{split}(p,b,T)$ – procent urbanizacji przeznaczony na budynek b w państwie p,
- $P_{acc}(p,z_{civ},T)$ – liczba punktów przemysłu cywilnego z_{civ} skumulowana przez państwo p w roku T.

$$pex(p,T) = P_{acc}(p,z_{civ},T)\cdot k_{exps}\cdot P_{pol}(p,l_{exp})\cdot Tradition(p,h_{pnr},T) \tag{3.13}$$

$$pub(p,T) = P_{acc}(p,z_{civ},T)\cdot k_{devs}\cdot (1-P_{pol}(p,l_{exp}))\cdot Tradition(p,h_{cre},T) \tag{3.14}$$

$$expn(p,T) = \begin{cases} \dfrac{pex(p,T)}{|P_{rege}(p,\tau)|}, & |P_{rege}(p,\tau)|>0 \\ 0, & |P_{rege}(p,\tau)|=0 \end{cases} \tag{3.15}$$

$$devn(p,T) = \begin{cases} \dfrac{pub(p,T)}{|P_{regd}(p,\tau)|}, & |P_{regd}(p,\tau)|>0 \\ 0, & |P_{regd}(p,\tau)|=0 \end{cases} \tag{3.16}$$

Obliczone punkty ekspansji są dzielone pomiędzy pulę punktów kolonizacji terenu i pulę punktów budowy miasta. Punkty urbanizacji pozwalają zwiększyć poziom rozwoju regionu (infrastruktury), który zwiększa dostępność regionalnych złóż zasobów. Koszt rozwoju regionu wzrasta wraz z aktualnym poziomem, aż osiągnie maksymalną możliwą wartość. Algorytm dystrybucji przydziela zasoby przemysłu cywilnego na ekspansję/urbanizację tylko tym regionom, które są w stanie zająć nowe terytorium/zbudować miasto/nie osiągnęły limitu rozwoju.

$$R_{col}(r,\tau) = R_{col}(r,\tau-1) + expn(p,T)\cdot P_{pol}(p,l_{exp}) \tag{3.17}$$

$$R_{reg}(r,T) = R_{reg}(r,\tau-1) + expn(p,T)\cdot (1-P_{pol}(p,l_{exp})) \tag{3.18}$$

$$R_{dev}(r,\tau) = min\left(k_{devm}, R_{dev}(r,\tau-1) + \frac{devn(p,T)\cdot P_{pol}(p,l_{exp})}{1+k_{devl}\cdot \lfloor R_{dev}(r,\tau)\rfloor}\right) \tag{3.19}$$

Żeby można było zbudować nowe miasto, region musi spełnić dwa warunki: musi mieć rozmiar co najmniej rsz punktów i co najmniej jeden punkt nie może się znajdować w bezpośrednim sąsiedztwie istniejącego miasta należącego do dowolnego państwa czy regionu. Jeżeli wiele punktów spełnia drugi warunek, lokalizacja jest wybierana losowo. Całkowity koszt budowy miasta wynosi $k_{city} + k_{spwl}\cdot |P_r(p,\tau)|$, czyli bazowy koszt i liniowa kara wynikająca z liczby regionów państwa. W momencie budowy miasta następuje aktualizacja granic wszystkich regionów państwa tak, aby każdy punkt państwa należał do regionu, do którego miasta znajduje się najbliżej. Wraz z tą zmianą odbywa się wewnątrzpaństwowa migracja ludności, w wyniku której populacja jest dzielona między regiony w zależności

od nowego rozmiaru. Większe regiony mają większą liczbę ludności, przez co gęstość zaludnienia jest podobna we wszystkich regionach państwa.

Punkty urbanizacji są wykorzystywane także do rozbudowywania kluczowych budynków regionu (Tabela 3.4), które zwiększają przepustowość zaawansowanych zasobów lub regionalnej służby zdrowia i bezpieczeństwa. Poziom budynków jest ograniczony poziomem rozwoju regionu.

$$str(r,b,T) = R_{str}(r,b,T-1) + P_{split}(p,b,T) \cdot k_{spd,b} \cdot \frac{devn(p,T) \cdot (1 - P_{pol}(p,l_{exp}))}{1 + k_{devl} \cdot \lfloor R_{dev}(r,\tau) \rfloor} \tag{3.20}$$

$$R_{str}(r,b,T) = min(\lfloor R_{dev}(r,\tau) \rfloor, str(r,b,T)) \tag{3.21}$$

$$regionCap(r,b,T) = R_{str}(r,b,T) \cdot k_{cap} \cdot k_{bon,b} \tag{3.22}$$

$$R_{med}(r,T) = regionCap(r,b_{hos},T) \cdot Science(p,s_{med},T) \tag{3.23}$$

$$R_{sec}(r,T) = regionCap(r,b_{crt},T) \cdot Science(p,s_{law},T) \tag{3.24}$$

$$Capacity(p,b,T) = \sum_{r}^{P_r(p,\tau)} regionCap(r,b,T) \tag{3.25}$$

3.2.6. System rozwoju kultury

Rozwój kulturowy pozwala zwiększyć wydajność produkcji, szybkość różnych rodzajów rozwoju i działalność międzynarodową poprzez zwiększanie poziomu „tradycji" (Tabela 3.3) za pomocą punktów kultury. Rozwój podlega regresji – poziom tradycji maleje z czasem, jeżeli nie inwestuje się w nią dostatecznie dużo punktów kultury.

- k_{sptd} – parametr szybkości rozwoju dowolnej tradycji,
- $k_{spd,h}$ – parametr szybkości rozwoju tradycji h,
- k_{dek} – parametr bazowego poziomu regresji dowolnej tradycji,
- k_{dekl} – parametr regresji dowolnej tradycji w zależności od poziomu,
- k_{maxt} – parametr maksymalnej wartości poziomu dowolnej tradycji,
- k_{botr} – parametr bonusu dowolnej tradycji w zależności od poziomu,
- $k_{bon,h}$ – parametr bonusu tradycji h,
- k_{gef} – parametr dzielnika dziedzictwa przy losowaniu wielkich wydarzeń,
- k_{gemax} – parametr maksymalnej szansy wystąpienia wielkiego wydarzenia,
- k_{gpp} – parametr prawdopodobieństwa, że wielkie wydarzenie będzie wielką osobą,
- k_{geb}, k_{gpb} – parametry poziomu dziedzictwa wynikającego z wielkiego dzieła / wielkiej osoby,
- k_{gel} – parametr czasu aktywności wielkiej osoby,
- $P_{split}(p,h,T)$ – procent rozwoju kultury przeznaczony na tradycję h państwa p,
- $P_{acc}(p,z_{cul},T)$ – liczba punktów kultury z_{cul} skumulowana przez państwo p w roku T,
- $P_{trd}(p,h,T)$ – poziom tradycji h dla państwa p,
- $P_{htg}(p,h,T)$ – poziom dziedzictwa tradycji h dla państwa p,
- $P_{htc}(p,h,T)$ – kumulacja punktów dziedzictwa tradycji h dla państwa p,

- $Tradition(p,h,T)$ – funkcja zwracająca mnożnik wynikający z tradycji h dla państwa p.

$$decay(p,h,T) = k_{dek} + \lfloor P_{trd}(p,h,T-1) \rfloor \cdot k_{dekl} \tag{3.26}$$

$$gain(p,h,T) = P_{acc}(p,z_{cul},T) \cdot P_{split}(p,h,T) \cdot k_{sptd} \cdot k_{spd,h} \tag{3.27}$$

W momencie, kiedy maksymalny poziom pewnej tradycji został osiągnięty, nadmiar zainwestowanych punktów staje się punktami „dziedzictwa" dla tej tradycji. Punkty dziedzictwa są kumulowane i zwiększają prawdopodobieństwo wystąpienia „wielkich wydarzeń".

$$P_{trd}(p,h,T) = clamp\big(\big(P_{trd}(p,h,T-1) + gain(p,h,T) - decay(p,h,T)\big), 0, k_{maxt}\big) \tag{3.28}$$

$$P_{htc}(p,h,T) = P_{htc}(p,h,T-1) + min\big(0, \big(P_{trd}(p,h,T-1) + gain(p,h,T) - decay(p,h,T)\big)\big) \tag{3.29}$$

$$Tradition(p,h,T) = 1 + \big(P_{trd}(p,h,T) + P_{htg}(p,h,T)\big) \cdot k_{botr} \cdot k_{bon,h} \tag{3.30}$$

Wielkie wydarzenia są losowymi wydarzeniami, które permanentnie zwiększają poziom tradycji ponad maksimum i reprezentują historyczne osiągnięcia danej kultury. Wydarzenia dzielą się na „wielkie dzieła", które nieznacznie zwiększają poziom i „wielkich osób", którzy znacznie zwiększają poziom tradycji przez pewien okres czasu ich aktywności. Po zakończeniu swojej aktywności bonus wynikający z działalności wielkiej osoby zmniejszany jest do poziomu wielkiego dzieła. W ten sposób usprawnienia wynikające z tradycji i dziedzictwa mogą rosnąć bez granic z biegiem czasu (tak, jak spuścizna kulturowa w rzeczywistości), ale jest to bardzo wolny proces. Algorytm wielkich wydarzeń jest wykonywany dla każdego państwa $p \in P$, dla każdej tradycji $h \in H$:

1. Jeżeli $P_{htg}(p,h,T) \leq 0$, zakończ działanie.
2. Oblicz prawdopodobieństwo wystąpienia wielkiego wydarzenia: $x = min\left(\frac{P_{htg}(p,h,T)}{k_{gef}}, k_{gemax}\right)$,
3. Dokonaj próby Bernouliego z prawdopodobieństwem x. Jeżeli wynik jest porażką, przejdź do punktu 5.
4. Dokonaj próby Bernouliego z prawdopodobieństwem k_{gpp}.
   - Jeżeli wynik jest sukcesem, dodaj nową wielką osobę, ustaw licznik czasu aktywności na k_{gel} i zwiększ $P_{htg}(p,h,T)$ o k_{gpb},
   - Jeżeli wynik jest porażką, dodaj nowe wielkie dzieło i zwiększ $P_{htg}(p,h,T)$ o k_{geb},
5. Wyzeruj $P_{htc}(p,h,T)$,
6. Dla każdej aktywnej wielkiej osoby zmniejsz licznik czasu aktywności o rok.
7. Jeżeli licznik czasu dowolnej osoby osiągnął 0, usuń osobę i zmniejsz $P_{htg}(p,h,T)$ o $k_{gpb} - k_{geb}$.

3.2.7. System rozwoju nauki

Podobnie jak rozwój kulturowy, rozwój naukowy pozwala zwiększyć wydajność produkcji, szybkość różnych rodzajów rozwoju i działalność międzynarodową poprzez zwiększanie poziomu różnych dziedzin nauki (Tabela 3.2) za pomocą punktów nauki. Rozwój naukowy podlega regresji i, w przeciwieństwie do tradycji, tempo rozwoju spowalnia wraz z aktualnym poziomem (wymaga coraz większych nakładów zasobów).

- k_{spdm}, k_{spdn} – parametry szybkości rozwoju głównego / pobocznego poziomu dowolnej dziedziny nauki,
- $k_{spd,s}$ – parametr szybkości rozwoju dziedziny nauki s,
- k_{dec} – parametr bazowego poziomu regresji dowolnej dziedziny nauki,
- k_{decl} – parametr regresji dowolnej dziedziny nauki w zależności od wartości głównego poziomu,
- k_{difl} – parametr szybkości rozwoju dowolnej dziedziny nauki w zależności od wartości głównego poziomu,
- k_{maxm}, k_{maxn} – parametry maksymalnej wartości głównego / pobocznego poziomu dowolnej dziedziny nauki,
- k_{bonm}, k_{bonn} – parametry bonusu głównego / pobocznego poziomu dowolnej dziedziny nauki,
- $k_{bon,s}$ – parametr bonusu dziedziny nauki s,
- $P_{split}(p, s, T)$ – procent rozwoju nauki przeznaczony na dziedzinę s państwa p,
- $P_{acc}(p, z_{sci}, T)$ – liczba punktów nauki z_{sci} skumulowana przez państwo p w roku T,
- $P_{scin}(p, s, T)$ – poboczny poziom rozwoju dziedziny nauki s państwa p,
- $P_{scim}(p, s, T)$ – główny poziom rozwoju dziedziny nauki s państwa p,
- $Science(p, s, T)$ – funkcja zwracająca mnożnik wynikający z nauki s dla państwa p.

$$decay(p, s, T) = k_{dec} + \lfloor P_{scim}(p, s, T-1) \rfloor \cdot k_{decl} \tag{3.31}$$

$$gain_n(p, s, T) = P_{acc}(p, z_{sci}, T) \cdot P_{split}(p, s, T) \cdot k_{spdn} \cdot k_{spd,s} \tag{3.32}$$

$$gain_m(p, s, T) = P_{acc}(p, z_{sci}, T) \cdot P_{split}(p, s, T) \cdot k_{spdm} \cdot k_{spd,s} \tag{3.33}$$

$$difficulty(p, s, T) = \begin{cases} 1 + \lfloor P_{scim}(p, s, T-1) \rfloor \cdot k_{difl}, & \lfloor P_{scim}(p, s, T-1) \rfloor \neq 0 \\ 1, & \lfloor P_{scim}(p, s, T-1) \rfloor = 0 \end{cases} \tag{3.34}$$

Rozwój naukowy jest podzielony na główne poziomy, które reprezentują przełomowe odkrycia i teorie oraz poboczne poziomy, które są odpowiednikiem procesu edukacji, doskonalenia wynalazków i znajdywania zastosowań teorii. Poziomy główne dają większe bonusy, ale są rozwijane wolniej i tylko wtedy, gdy poziom poboczny osiągnął tymczasową maksymalną możliwą wartość. Jeżeli maksymalny teoretyczny poziom nauki został już osiągnięty, punkty są przeznaczane wyłącznie na powstrzymywanie regresji. Brak możliwości nieskończonego rozwoju naukowego jest kompensowany większym bonusem wynikającym z poziomu wiedzy, niż tym wynikającym z kultury (dla domyślnych parametrów symulacji).

$$P_{scim}(p,s,T) = clamp\left(\frac{gain_m(p,s,T) - decay(p,s,T)}{difficulty(p,s,T)} + P_{scim}(p,s,T-1), 0, k_{maxm}\right) \tag{3.35}$$

$$P_{scin}(p,s,T) = clamp\left(\frac{gain_n(p,s,T) - decay(p,s,T)}{difficulty(p,s,T)} + P_{scin}(p,s,T-1), 0, k_{maxm} \cdot k_{maxn}\right) \tag{3.36}$$

$$Science(p,s,T) = 1 + \left(P_{scim}(p,s,T) \cdot k_{bonm} + P_{scin}(p,s,T) \cdot k_{bonn}\right) \cdot k_{bon,s} \tag{3.37}$$

3.2.8. System wzrostu populacji

Wzrost populacji jest typowo modelowany przy użyciu krzywej logistycznej, jeżeli górna granica wartości populacji jest stała (np. z powodu ograniczonych zasobów). W omawianym modelu symulacji górna granica nie jest znana, ponieważ państwa mogą zwiększyć z czasem limit dostępnych zasobów przez rozwój lub powiększenie terytorium. Zamiast tego, model zakłada wykładniczy charakter wzrostu w stylu $f(t+1) = f(t) \cdot (1+a)$, dodatkowo zależny od zaopatrzenia i poziomu zdrowia społeczeństwa.

- $k_{need,z}$ – parametr bazowej konsumpcji zasobu z na jednostkę populacji,
- $k_{neec,z}$ – parametr konsumpcji zasobu z na jednostkę populacji wynikającej z niestabilności,
- k_{help} – parametr maksymalnego poziomu kary za niedostateczną opiekę medyczną,
- k_{popm} – parametr minimalnej liczby ludności w regionie,
- k_{popg} – parametr bazowego miesięcznego przyrostu naturalnego,
- k_{crm} – parametr współczynnika przestępczości,
- k_{rbs} – parametr szybkości wzrostu buntu,
- $Science(p,s,T)$ – funkcja zwracająca mnożnik wynikający z nauki s dla państwa p,
- $Tradition(p,h,T)$ – funkcja zwracająca mnożnik wynikający z tradycji h dla państwa p,
- $R_{reb}(r,p,\tau)$ – siła buntu w regionie r,
- $R_{med}(r,T)$ – służby medyczne w regionie r,
- $R_{hel}(r,\tau)$ – średni poziom zdrowia w regionie r,
- $R_{sec}(r,T)$ – służby bezpieczeństwa w regionie r,
- $R_{stab}(r,p,\tau)$ – średni poziom stabilności w regionie r,
- $R_{pop}(r,\tau)$ – liczba ludności w regionie r,
- $P_{army}(p,\tau)$ – liczba żołnierzy w państwie p,
- $P_r(p,\tau)$ – zbiór wszystkich regionów państwa p,
- $P_{res}(p,z,\tau)$ – ilość zasobu z wyprodukowanego przez państwo p,
- $P_{need}(p,z,\tau)$ – ilość zasobu z wymagana przez populację państwa p.

$$R_{pop}(r,p,\tau) = max\left(k_{popm}, R_{pop}(r,p,\tau-1) \cdot (cov(p,\tau) + growth(r,p,\tau))\right) \tag{3.38}$$

$$P_{pop}(p,\tau) = \sum_{r}^{P_r(p,\tau)} R_{pop}(r,p,\tau) \tag{3.39}$$

Potrzeby ludności mają najwyższy priorytet przy podziale zasobów. Jeżeli państwo nie jest w stanie w całości zaspokoić konsumpcji zaopatrzenia populacji $cov(p,\tau)$, ludność bez zasobów umiera. Niezaspokojenie konsumpcji przemysłu i bogactwa nie powoduje negatywnych konsekwencji, funkcjonuje wyłącznie jako „koszt" posiadania dużej liczby ludności.

$$cov(p,\tau) = \begin{cases} \dfrac{P_{res}(p,z_{sup},\tau)}{P_{need}(p,z_{sup},\tau)}, & P_{need}(p,z_{sup},\tau) \neq 0 \\ 1, & P_{need}(p,z_{sup},\tau) = 0 \end{cases} \tag{3.40}$$

$$P_{need}(p,z,\tau) = \big(P_{pop}(p,\tau-1) - P_{army}(p,\tau-1)\big) \cdot \big(k_{need,z} + k_{neec,z} \cdot (1 - P_{stab}(p,\tau))\big) \tag{3.41}$$

$$growth(r,p,\tau) = R_{hel}(r,\tau) \cdot k_{popg} \cdot Science(p,s_{med},T) \tag{3.42}$$

Tempo wzrostu naturalnego jest modyfikowane przez współczynnik zdrowotności $R_{hel}(r,\tau)$, czyli stosunek punktów służby zdrowia (bezpośrednio wynikających z poziomu szpitali w regionie) do populacji regionu i poziom nauk medycznych państwa.

$$R_{hel}(r,\tau) = 1 - min\left(k_{help}, \frac{max\big(0, R_{pop}(r,\tau) - R_{med}(r,T)\big)}{R_{pop}(r,\tau)}\right) \tag{3.43}$$

$$P_{hel}(p,\tau) = \frac{\sum_{r}^{P_r(p)} \big(R_{hel}(r,\tau) \cdot R_{pop}(r,p,\tau)\big)}{P_{pop}(p,\tau)} \tag{3.44}$$

$$\tag{3.45}$$

Oprócz współczynnika zdrowotności obliczany także jest współczynnik stabilności $R_{stab}(r,p,\tau)$. Gdy stabilność wynosi mniej niż 100%, konsumpcja ludności wzrasta, ponieważ część zasobów jest stracona. Niestabilność wynika z naturalnego poziomu przestępczości (może być zmniejszony poprzez tradycję władzy) i z tymczasowego poziomu buntu, który występuje po aneksji regionu przeciwnika po wojnie. Stabilność jest zwiększana przez służby bezpieczeństwa (poziom sądów) analogicznie do szpitali. Jeżeli buntownicy są silniejsi od służb bezpieczeństwa, poziom buntu wzrasta z czasem, w przeciwnym wypadku spada.

$$crime(r,p,\tau) = \frac{R_{pop}(r,\tau)\cdot(k_{crm}+R_{reb}(r,p,\tau))}{Tradition(p,h_{aut},T)} \tag{3.46}$$

$$R_{reb}(r,p,\tau) = clamp\left(R_{reb}(r,p,\tau-1) - k_{rbs}\cdot(R_{stab}(r,p,\tau)-R_{reb}(r,p,\tau-1)),0,2\right) \tag{3.47}$$

$$R_{stab}(r,p,\tau) = 1 - max\left(0,\frac{crime(r,p,\tau)-R_{sec}(r,T)}{R_{pop}(r,p,\tau)}\right) \tag{3.48}$$

$$P_{stab}(p,\tau) = clamp\left(\frac{\sum_r^{P_r(p)}\left(R_{stab}(r,p,\tau)\cdot R_{pop}(r,p,\tau)\right)}{P_{pop}(p,\tau)},0,1\right) \tag{3.49}$$

3.2.9. System dyplomacji

Stosunki międzynarodowe są przedstawione w uproszczony sposób: sąsiadujące państwa posiadają wspólną wartość „relacji" $P_{rel}(p_a,p_b,\tau)$, którą polepszają lub pogarszają w zależności od poziomu strategii agresywności. Polepszenie relacji jest możliwe wyłącznie gdy obie strony do tego aktywnie dążą lub gdy mają dostatecznie neutralne nastawienie – w takim przypadku następuje nieznaczna, naturalna poprawa relacji. Wybrany zakres modelowanych stosunków międzynarodowych ogranicza się wyłącznie do kwestii wojskowych.

- k_{good} – parametr naturalnej poprawy relacji,
- k_{gsh} – parametr szybkości zmiany relacji,
- k_{gxp} – parametr spowolnienia szybkości zmiany radykalnych relacji,
- $Science(p,s,T)$ – funkcja zwracająca mnożnik wynikający z nauki s dla państwa p,
- $P_{pol}(p,l)$ – wartość strategii l państwa p,
- $P_{dip}(p,\tau)$ – nastawienie państwa p wobec innych państw,
- $P_{rel}(p_a,p_b,\tau)$ – wzajemne relacje państw p_a i p_b.

$$P_{dip}(p,\tau) = 2\cdot(0.5-P_{pol}(p,l_{cmp}))\cdot Science(p,s_{lng},T) \tag{3.50}$$

$$shift(p_a,p_b,\tau) = \begin{cases} \dfrac{P_{dip}(p_a,\tau)+P_{dip}(p_b,\tau)}{2}+k_{good}, & P_{dip}(p_a,\tau)\geq 0 \Leftrightarrow P_{dip}(p_b,\tau)\geq 0 \\ min(P_{dip}(p_a,\tau),P_{dip}(p_b,\tau))+k_{good}, & P_{dip}(p_a,\tau)\geq 0 \oplus P_{dip}(p_b,\tau)\geq 0 \end{cases} \tag{3.51}$$

$$P_{rel}(p_a,p_b,\tau) = clamp\left(P_{rel}(p_a,p_b,\tau-1)+shift(p_a,p_b,\tau)\cdot k_{gsh}\cdot(k_{gxp}-|P_{rel}(p_a,p_b,\tau)|),-1,1\right) \tag{3.52}$$

W zależności od poziomu relacji status między państwami jest klasyfikowany w następujący sposób:

- Jeżeli $P_{rel}(p_a,p_b,\tau)\in[k_{sall},1]$, państwa są sojusznikami:
  - — Państwo nigdy nie dołączy do wojny przeciwko sojusznikowi,
  - — Państwo zawsze dołączy do wojny obronnej sojusznika,
  - — Jeżeli państwo nie bierze udziału w innej wojnie, może dołączyć do innej wojny sojusznika:

  - — Wojny agresywnej, jeżeli główny oponent jest rywalem lub nie jest sąsiadem,
  - — Wojny pomocniczej, jeżeli główny oponent jest rywalem, a sojusznik sojusznika jest przyjacielem,
- Jeżeli $P_{rel}(p_a, p_b, \tau) \in [k_{sfnd}, k_{sall})$, państwa są przyjaciółmi:
  - — Państwo dołączy do wojny obronnej przyjaciela, jeżeli główny oponent jest rywalem lub nie jest sąsiadem,
- Jeżeli $P_{rel}(p_a, p_b, \tau) \in (k_{sriv}, k_{sfnd})$, państwa są neutralne,
- Jeżeli $P_{rel}(p_a, p_b, \tau) \in (k_{shst}, k_{sriv}]$, państwa są rywalami,
- Jeżeli $P_{rel}(p_a, p_b, \tau) \in [-1, k_{shst}]$, państwa są wrogami i wywołają wojnę pod warunkiem, że:
  - — Czas startowego pokoju światowego minął,
  - — Państwa nie są w stanie wojny ze sobą,
  - — Co najmniej jedna ze stron nie jest w stanie wojny z nikim,
  - — Poprzednie traktaty pokojowe pomiędzy państwami wygasły.

Gdy warunki do wywołania wojny są spełnione, państwo z wyższym poziomem strategii agresywności staje się agresorem (prowadzi wojnę ofensywną), a jego przeciwnik obrońcą (prowadzi wojnę obronną). Jeżeli państwo bardziej agresywne nie może wywołać wojny (ponieważ bierze już udział w innym konflikcie), role się odwracają. Sojusznicy mogą dołączyć do wojny (prowadzą wojnę „pomocniczą") w dowolnym momencie, nie tylko w miesiącu rozpoczęcia.

Dzięki możliwości dołączania się do wojen bezpośrednich, a nawet pośrednich sojuszników, system dyplomacji jest w stanie odtworzyć do pewnego stopnia zawiłość relacji pomiędzy sąsiadami i sieci sojuszy występujące w historii. Pozwala to na powstawanie wielopaństwowych konfliktów, w które mogą być zaangażowane całe rejony kontynentów. Dodatkowo sojusze sprawiają, że państwa pacyfistyczne rzadziej padają ofiarą wrogich sąsiadów, dzięki czemu strategia pokojowość-agresywność jest zbalansowana.

3.2.10. System konfliktów

Wojna jest zjawiskiem tak starym jak sama ludzkość. Konflikty zbrojne spowodowane były głównie pobudkami ekonomicznymi, rzadziej religijnymi, politycznymi lub etnicznymi. W modelu symulacji, konflikt jest najszybszym sposobem na zwiększenie liczby ludności, zasobów, terytoriów. W przeciwieństwie do różnych rodzajów rozwoju, wiąże się z nim bardzo duże ryzyko przegranej, która prowadzi do strat lub kompletnej asymilacji przez inne państwo. Co więcej, sukces wymaga inwestowania zasobów w przemysł wojskowy i lojalność, które w innym przypadku mogłyby przyspieszać pokojowy, długoterminowy rozwój. System konfliktów jest najbardziej złożonym systemem symulacji i pozwala na branie udziału państw w wielu konfliktach z różnymi uczestnikami jednocześnie. Pojedynczy krok konfliktu c wygląda następująco:

1. Wybierz akcję każdego uczestnika p konfliktu:
   - Jeżeli wyniszczenie osiągnęło limit, wybierz poddanie się,
   - Jeżeli mobilizacja nadal trwa, wybierz mobilizację i zmniejsz licznik mobilizacji,

- Jeżeli sprzęt lub organizacja wynosi 0, wybierz umocnienie,
- W innym razie wybierz dowolną akcję ze zbioru $\{a_{ass}, a_{mnv}, a_{chr}, a_{rly}, a_{sge}, a_{frt}\}$,

2. Sprawdź, czy konflikt się skończył:
   - Jeżeli obie strony się poddały, konflikt się kończy rozejmem,
   - Jeżeli jedna strona się poddała, konflikt kończy się wygraną dla przeciwników,
3. Zastosuj akcję każdego aktywnego uczestnika:
   a) Wylosuj liczbę losową z rozkładu jednorodnego $[-k_{rng}, k_{rng}]$,
   b) Otrzymaj posiłki sprzętu i organizacji od państwa, oblicz poziom fortyfikacji,
   c) Oblicz siły ataku i obrony sprzętu, organizacji i oblężenia na podstawie wybranej akcji (Tabela 3.7), poziomu technologii i tradycji wojskowej i liczby losowej,
   d) Oblicz wkład w konflikt na podstawie posiadanej siły,
4. Oblicz stosunki sumarycznego ataku i obrony obu stron,
5. Oblicz obrażenia zadane każdemu aktywnemu uczestnikowi,
6. Oblicz straty każdego aktywnego uczestnika:
   a) Zadaj obrażenia organizacyjne wojsku. Jeżeli poziom organizacji stał się ujemny, dodaj nadmiar obrażeń do obrażeń sprzętowych,
   b) Zadaj obrażenia sprzętowe wojsku. Oblicz wyniszczenie wynikające ze strat wojskowych.
   c) Jeżeli poziom sprzętu stał się ujemny, zadaj nadmiar obrażeń fortyfikacjom,
   d) Jeżeli poziom fortyfikacji stał się ujemny, zadaj nadmiar obrażeń cywilom. Oblicz wyniszczenie wynikające ze strat cywilnych.

Wojsko składa się ze sprzętu i organizacji. Sprzęt (fr. *matériel*) reprezentuje broń, amunicję, zasoby, personel, czyli fizyczne, rzeczywiste elementy sił zbrojnych. Organizacja jest kwantyfikacją abstrakcyjnych pojęć: pozycji taktycznej, zdolności do komunikacji i planowania, spójności, poziomu wytrenowania i morale (fr. *esprit de corps*). Wojsko potrzebuje zarówno sprzętu i organizacji, aby móc brać udział w konflikcie.

Akcje pozwalają tymczasowo zwiększyć wybrane aspekty wojska kosztem innych (np. atak vs obrona, sprzęt vs organizacja, Tabela 3.7). Ponieważ każdy uczestnik konfliktu wybiera akcje niezależnie od innych, wyniki walki są bardziej zróżnicowane i nieprzewidywalne w porównaniu do liniowych obliczeń strat stosowanych w literaturze (zakładając symetryczny konflikt, czyli zbliżony potencjał wojskowy obu stron). Dla każdej akcji istnieje akcja o przeciwnym działaniu: natarcie-manewr, szarża-zebranie, oblężenie-umocnienie, dzięki czemu wybór akcji ma podobną dynamikę do gry papier-kamień-nożyce.

W momencie dołączenia państwa do konfliktu rozpoczyna się kilkumiesięczny okres mobilizacji, podczas którego wojsko zwiększa swój rozmiar i nie bierze udziału w walce. Posiłki przybywają natychmiast w każdym miesiącu konfliktu. Państwo powołuje określony ułamek całkowitej populacji do udziału w każdym konflikcie, ale może rekrutować ograniczoną liczbę wojska w każdym miesiącu. Prędkość rekrutacji może być zwiększona zgodnie ze strategią militaryzmu państwa. Rozmiar posiłków sprzętowych jest ograniczony liczbą dostępnych rekrutów i zdolnością przemysłu wojskowego, a posiłki organizacyjne dodatkowo ograniczone są wyprodukowanymi punktami lojalności. Państwo może redukować

Symbol	Nazwa	S. atk.	S. obr.	O. atk.	O. obr.	Oblęż.
a_{ass}	Natarcie	$1+k_{cass}$	$1+k_{cass}$	$1-k_{cass}$	$1-k_{cass}$	0
a_{mnv}	Manewr	$1-k_{cmnv}$	$1-k_{cmnv}$	$1+k_{cmnv}$	$1+k_{cmnv}$	0
a_{chr}	Szarża	$1+k_{cchr}$	$1-k_{cchr}$	$1+k_{cchr}$	$1-k_{cchr}$	0
a_{rly}	Zebranie	$1-k_{crly}$	$1+k_{crly}$	$1-k_{crly}$	$1+k_{crly}$	0
a_{sge}	Oblężenie	k_{csge}	k_{csge}	k_{csge}	k_{csge}	k_{sgeb}
a_{frt}	Umocnienie	k_{cfrt}	k_{cfrt}	$C_{frtb}(c,p,\tau)$	$C_{frtb}(c,p,\tau)$	0
a_{mob}	Mobilizacja	N/A	N/A	N/A	N/A	N/A
a_{sur}	Poddanie się	N/A	N/A	N/A	N/A	N/A

Tabela 3.7. Lista akcji konfliktu i mnożników siły.

zatrudnienie przy produkcji wszystkich zasobów dopóki spełnia wymogi konsumpcji – nie może dopuścić do głodu z powodu zbyt dużej armii.

Siła wojska jest reprezentowana przez cztery główne zmienne: atak i obronę sprzętową oraz atak i obronę organizacyjną. Atak ma wpływ na obrażenia zadawane przeciwnikom, a obrona ma wpływ na otrzymywane obrażenia. Istnieją trzy typy obrażeń: sprzętowe, organizacyjne i oblężnicze. Wartości dwóch pierwszych typów są obliczane na podstawie liczby sprzętu atakującej strony, stosunku ataku do obrony (zarówno sprzętu, jak i organizacji) i parametru śmiertelności/wrażliwości. Obrażenia oblężnicze są specjalne, ponieważ mogą zostać zadane tylko przez akcję oblężenia, która jednocześnie obniża atak i obronę. Obie strony konfliktu zadają obrażenia jednocześnie, więc jednocześnie są atakującymi i obrońcami.

Zadane obrażenia są dzielone równomiernie pomiędzy aktywnych uczestników strony konfliktu. Obrażenia sprzętowe redukują kolejno sprzęt/personel, fortyfikacje (przedstawione jako suma regionalnych fortec) i ludność przeciwnika i zwiększają wyniszczenie, które prowadzi do poddania się. Obrażenia oblężnicze pomijają wojska i bezpośrednio uszkadzają fortyfikacje lub ludność. Obrażenia organizacyjne zmniejszają wyłącznie organizację, ale jeżeli organizacja przeciwnika stanie się ujemna, nadmiar jest przekształcony w zwiększoną ilość obrażeń sprzętowych, co reprezentuje załamanie się szyku i panikę.

Wyniszczenie nie wpływa na zdolności produkcyjne państw, ale jeżeli osiągnie limit, państwo musi się poddać. W przypadku, gdy obie strony przekraczają ten limit, konflikt kończy się rozejmem. Państwa po obu stronach konfliktu podpisują traktat pokojowy na określoną ilość lat, relacje między nimi są resetowane, a wojska biorące udział w konflikcie są demobilizowane: pozostały sprzęt jest dodawany do zapasów przemysłu wojskowego, organizacja do lojalności, a żołnierze stają się cywilami zdolnymi do pracy. Jeżeli tylko jedna ze stron się poddała, to jest ogłaszana przegranym, a przeciwnik zwycięzcą.

Zwycięzcy mają prawo zażądać regionów od przegranych, jeżeli z nimi graniczą. Stosunek wkładu w konflikt zwycięzcy do przegranego decyduje, ile regionów można zabrać. Jeżeli więcej niż jeden zwycięzca może zająć pewien region, otrzymuje go uczestnik z największym wkładem. Zajęte regiony rzadko kiedy są oddawane pokojowo, co jest reprezentowane przez obniżenie poziomu rozwoju i budynków w zdobytym regionie i bunt populacji, na który wpływają poziom tradycji wojskowej i strategia militaryzmu. W ten

sposób balansuje się zysk wynikający ze zdobytych złóż, ludności i budynków. Zwycięzcy, którzy nie otrzymali żadnego regionu mają prawo wyznaczyć trybut od każdego przegranego, gdzie procent udziałów zwycięzców w trybucie zależy od wkładu w konflikt. Przegrany może zmniejszyć wysokość trybutu na podstawie poziomu tradycji dyplomacji.

3.2.11. System polityki

Strategie profilu politycznego państwa mają określony czas trwania, po upływie którego są wybierane na nowo. Zarówno czas trwania, jak i wartości strategii są modelowane za pomocą rozkładu normalnego o parametryzowanej średniej i wartości oczekiwanej. W ten sposób reprezentowana jest zmiana władzy lub myśli politycznej w państwie, która często występuje w nieregularnych odstępach (np. w wyniki śmierci władcy, rewolucji, wyborów, powstania nowego nurtu). Współczynniki $P_{split}(p,s,T)$, $P_{split}(p,h,T)$, $P_{split}(p,z,T)$, $P_{split}(p,b,T)$ dzielą państwową pulę punktów nauki/kultury/zasobów/urbanizacji na wybrane dziedziny nauki/tradycje/zasoby zaawansowane/budynki na podstawie wartości strategii (Tabela 3.5, Tabela 3.6). Współczynniki danej kategorii są normalizowane, aby sumarycznie wynosiły 1.

$$P_{split}(p,s,T) = \begin{cases} l_{exp}, & s = s_{geo} \\ 1 - l_{exp}, & s = s_{med} \\ 1 - l_{mil}, & s = s_{eng} \\ l_{mil}, & s = s_{mtl} \\ 1 - l_{prg}, & s = s_{phi} \\ l_{prg}, & s = s_{mat} \\ l_{lgl}, & s = s_{law} \\ 1 - l_{cmp}, & s = s_{lng} \\ l_{cmp}, & s = s_{mil} \end{cases} \tag{3.53}$$

$$P_{split}(p,h,T) = \begin{cases} l_{exp}, & h = h_{pnr} \\ 1 - l_{exp}, & h = h_{mon} \\ l_{prg}, & h = h_{cur} \\ 1 - l_{prg}, & h = h_{cre} \\ 1 - l_{lgl}, & h = h_{pros} \\ l_{lgl}, & h = h_{aut} \\ 1 - l_{cmp}, & h = h_{dip} \\ l_{cmp}, & h = h_{sup} \end{cases} \tag{3.54}$$

$$P_{split}(p,z,T) = \begin{cases} 0, & z = z_{sup} \\ 2 - l_{mct}, & z = z_{ind} \\ 1 + l_{mct}, & z = z_{wel} \\ 2 - l_{mil}, & z = z_{civ} \\ 1 + l_{mil}, & z = z_{mil} \\ 1 + (1 - l_{prg}) \cdot (1 - l_{mil}), & z = z_{cul} \\ 1 + l_{prg} \cdot (1 - l_{mil}), & z = z_{sci} \\ 2(1 + l_{mil}), & z = z_{loy} \end{cases} \tag{3.55}$$

$$P_{split}(p,b,T) = \begin{cases} 1 - l_{exp}, & b = b_{hos} \\ 1 - l_{mil}, & b = b_{man} \\ l_{mil}, & b = b_{frg} \\ l_{prg}, & b = b_{uni} \\ 1 - l_{prg}, & b = b_{amt} \\ l_{lgl}, & b = b_{crt} \\ l_{cmp}, & b = b_{frt} \end{cases} \tag{3.56}$$

4. Architektura rozwiązania

Opisane w tej pracy modele generacji mapy świata i symulacji historii zaimplementowano w formie aplikacji desktopowych z graficznym interfejsem użytkownika. Niniejszy rozdział przybliża postawione wymagania, technologie i metody, które zostały użyte i ogólny zarys projektu. Szczegóły działania generatora i symulatora są przybliżone w następnym rozdziale.

4.1. Wymagania

- Generator powinien implementować cały model świata (Podrozdział 3.1): móc tworzyć co najmniej wszystkie warstwy opisane w modelu i pozwalać na edytowanie listy biomów i złóż zasobów,
- Generator powinien móc wczytać lub zapisać gotowe dane warstw,
- Generator powinien wizualizować stworzone warstwy świata,
- Symulator powinien implementować cały model historii (Podrozdział 3.2): być kompatybilny z modelem świata generatora i implementować wszystkie systemy opisane w modelu,
- Symulator powinien pozwalać na kontrolę upływu czasu w symulacji (zatrzymanie/wznowienie/przyspieszenie/spowolnienie),
- Symulator powinien pozwalać przygotowywać różne scenariusze początkowe,
- Symulator powinien wizualizować elementy symulacji w czasie rzeczywistym,
- Symulator powinien podawać szczegółowe informacje o stanie zaznaczonych elementów symulacji,
- Jak najwięcej zachowań i wartości obu programów powinno być konfigurowalnych przez parametry,
- Oba programy powinny pozwalać modyfikować parametry generacji/symulacji na żywo oraz wczytywać/zapisywać je do pliku,
- Programy powinny działać co najmniej na maszynach z systemem operacyjnym Windows 10,
- Programy powinny mieć graficzny interfejs użytkownika,
- Programy powinny być szybkie i wydajne na typowym komputerze osobistym. Generacja mapy o typowym rozmiarze nie powinna zajmować więcej, niż kilka sekund. Symulator powinien być w stanie przetworzyć setki lub tysiące lat w ciągu kilkunastu minut,
- Programy powinny być dobrze udokumentowane w formie instrukcji użytkownika.

4.2. Wybrane technologie

Projekt został zrealizowany w całości w języku Rust [81]. Dokonano takiego wyboru z następujących powodów:

Wydajność Rust jest językiem niskiego poziomu, kompilowanym do kodu maszynowego i zakładającym ręczne zarządzanie pamięcią [82]. Dzięki temu programy napisane w tym

języku są porównywalnie szybkie do tych w C i C++ w przeprowadzanych testach wydajnościowych [83]–[85] i zdecydowanie szybsze od programów wykorzystujących wysokopoziomowe języki, często wymagające interpretera kodu lub odśmiecania pamięci.

Bezpieczeństwo Głównym celem języka jest zapewnienie bezpiecznego dostępu do pamięci i innych zasobów [82], [86] oraz bezpiecznego wykorzystywania współbieżności [82], [87]. Restrykcyjne zasady dostępu do pamięci, wbudowana w kompilator statyczna analiza kodu i oficjalny linter pozwalają wykryć większość potencjalnych błędów lub nieoptymalny kod na etapie kompilacji. Gwarancja bezpieczności nie odbywa się kosztem wydajności zgodnie z zasadą „abstrakcji o zerowym koszcie“ (ang. *zero cost abstraction*) [88].

Dostępność i przenośność Wraz z językiem rozwijany jest system budowania i zarządzania zależnościami `cargo` [89]. Kompilator języka Rust `rustc` wykorzystuje wewnętrznie kompilator LLVM [90], dzięki czemu wspiera wiele platform sprzętowych i systemów operacyjnych [91][92] oraz format WebAssembly [93] obsługiwany przez przeglądarki internetowe. Pozwala także na kompilację wskrośną.

Implementacja projektu została stworzona w oparciu o Bevy [94]: otwartoźródłowy silnik do gier i symulacji, napisany w języku Rust i wykorzystujący wzorzec architektoniczny ECS. Silnik ten został wybrany ze względu na modularną budowę (możliwość usunięcia fragmentów silnika[7], z których się nie korzysta), bezpłatną, otwartą licencję (MIT lub Apache 2.0) i wydajność.

4.3. Silniki gier i wzorce architektoniczne

Praca ta wprawdzie nie dotyczy bezpośrednio gier komputerowych, ale silniki gier znajdują zastosowanie także przy tworzeniu oprogramowania, które też potrzebuje oprawy audiowizualnej i/lub interakcji z użytkownikiem w czasie rzeczywistym, ale którego głównym celem nie jest rozrywka sama w sobie[95], [96]. Jako przykład można podać symulatory wojskowe [61], [95], treningowe [95] i badawcze lub programy wizualizujące przestrzenie trójwymiarowe [97]. Silniki zapewniają gotowe do użycia, wydajne implementacje silników grafiki 2D i 3D, systemów dźwięku i symulacji fizyki[97]. Dzięki nim autorzy oprogramowania mogą się skupić wyłącznie na implementacji logiki, bez potrzeby poświęcania czasu na kwestie techniczne, często nieistotne w pracy. Może się z tym jednak wiązać wymóg korzystania z określonej budowy programu, sposobu organizacji danych, czy interakcji między elementami. Warto więc dodatkowo omówić dlaczego powstały, czym dokładnie są i jakie różnice pomiędzy popularnymi architekturami silników gier, jakie są ich wady i zalety.

Pierwsze gry komputerowe miały monolityczną budowę ze względu na niski stopień skomplikowania, ograniczenia sprzętowe i wszechobecność niskopoziomowych, proceduralnych języków programowania. Popularyzacja gier, wzrost wartości rynku i zwiększająca się dostępność komputerów osobistych w latach dziewięćdziesiątych pozwoliły na szereg innowacji technicznych i gatunkowych. Gry stały się zaawansowanymi projektami

[7] Dyrektywy preprocesora pozwalają wyłączyć fragmenty kodu z kompilacji.

informatycznymi wymagającymi wieloosobowych zespołów, łączącymi zagadnienia grafiki komputerowej (coraz częściej trójwymiarowej), dźwięku przestrzennego, komunikacji sieciowej, symulacji fizyki i sztucznej inteligencji [97], [98]. Silniki gier pozwoliły na ogromną reużywalność kodu poprzez separacje tych uniwersalnych zagadnień technicznych od logiki, unikatowej dla każdej gry. W obecnych czasach 750 *tysięcy* programów korzysta z silnika Unity [99].

Silniki do gier zajmują się wieloma powiązanymi ze sobą zagadnieniami i jednocześnie wymagają wysokiej wydajności jako systemy czasu rzeczywistego, więc podjęte decyzje architektoniczne mają duże znaczenie nie tylko przy rozszerzaniu i utrzymywaniu kodu, ale też przy ocenie spełnienia założeń projektowych [100]. Poniżej przedstawiono trzy typowe podejścia [101], [102].

4.3.1. Architektura obiektowa

Podejście wykorzystujące paradygmat programowania obiektowego jest najpopularniejsze przy tworzeniu gier [100] ze względu na łatwość znalezienia analogii w funkcjonowaniu świata rzeczywistego i tworzonego świata wirtualnego [103]. Świat gry składa się z obiektów, typowo instancji klas dziedziczonych po podstawowej klasie oferowanej przez silnik. Każdy obiekt zawiera w całości informacje o sobie zgodnie z zasadą enkapsulacji i komunikuje się ze światem zewnętrznym poprzez metody i interfejsy [103], [104] lub wydarzenia/komunikaty. Reużywalność kodu i semantyczne relacje pomiędzy typami obiektów realizuje się głównie poprzez dziedziczenie [100], (np. postać gracza dziedziczy po klasie implementującej ruch), czasem połączone z kompozycją [105], która jest rekomendowana jako komplementarna technika [106]. Umiejętne zastosowanie obiektowych wzorców projektowych przy implementacji pozwala na zmniejszenie skomplikowania kodu [98], [100], [106], [107]. Przykładem silnika wykorzystującego podejście obiektowe jest Godot [108][8].

4.3.2. Wzorzec Entity Component

Wzorzec Entity Component (czasem też nazywany Entity System, Component System lub *mylnie* Entity Component System) wywodzi się z podejścia obiektowego, ale preferuje dynamiczną kompozycje zamiast dziedziczenia jako główny sposób organizacji danych i zachowań obiektów [100]. Elementy gry są reprezentowane przez obiekty podmiotu (ang. *entity*) typowo zawierające tylko najprostsze informacje, takie jak pozycja w świecie. Reszta aspektów (funkcjonalności) obiektów zawarta jest w obiektach komponentach (np. komponent odpowiadający za ruch lub model 3D), które statycznie lub dynamicznie są dołączane do encji. W ten sposób unika się przerośniętej hierarchii klas i nadmiernej abstrakcji, jakie mogą powstać przy niedbałym rozwoju gry zgodnie z paradygmatem obiektowym [101], jednocześnie zachowując reużywalność kodu w formie komponentów.

[8] Dokładniej mówiąc, Godot znajduje się pomiędzy podejściem czysto obiektowym a Entity Component. Gra w tym silniku składa się z obiektów „węzłów" zorganizowanych w drzewo (ang. *Scene tree*) zgodnie z techniką kompozycji, ale w przeciwieństwie do EC nie ma rozróżnienia pomiędzy podmiotem (entity) a komponentem. Co więcej, wbudowana w silnik hierarchia klas węzłów jest specjalizowana poprzez dziedziczenie, a nie rozszerzana przez kompozycję.

Wzorzec ten jest wykorzystywany przez silniki takie, jak Unity [109], Unreal Engine [110] i CryEngine [111].

4.3.3. Wzorzec Entity-component-system

Ten wzorzec jest zazwyczaj prezentowany jako odrzucenie paradygmatu obiektowego na rzecz paradygmatu projektowania skupionego wokół danych (data-oriented design, DOD) [112]. Wywodzi się z krytyki programowania obiektowego jako podejścia nie wykorzystującego w pełni możliwości nowoczesnych procesorów, a przez to tworzącego powolne programy [103], [113]. DOD zakłada, że projektowanie systemów powinno być skupione na analizie i reprezentacji danych oraz definiowania transformacji tych danych biorąc pod uwagę możliwości i ograniczenia architektury sprzętowej[112], w przeciwieństwie do programowania obiektowego, które skupia się na modelowaniu domeny problemu i występujących procesów i interakcji [112].

W praktyce, w przypadku silników gier, DOD dąży przede wszystkim do jak najbardziej kompaktowej i ciągłej reprezentacji danych w pamięci, aby jak najlepiej wykorzystać ograniczoną, lecz szybką pamięć cache procesorów [100], [103], [113]. Drugim celem jest jak największe zrównoleglenie operacji [102], [112], jako że w ostatnich latach wzrost mocy obliczeniowej procesorów wynika bardziej ze zwiększania się liczby rdzeni, niż ze wzrostu szybkości [103].

Wzorzec ECS zakłada, że świat składa się z trzech typów elementów: podmiotów (entity), komponentów i systemów. Podmioty reprezentują elementy gry i zawierają tylko identyfikator poświadczający ich istnienie i unikatową tożsamość. Komponenty zawierają dane odnośnie jakiegoś aspektu elementów gry (np. pozycję, prędkość, wytrzymałość), ale nie mają wydzielonych metod. Komponenty są przechowywane razem w celu zachowania sąsiedztwa w pamięci. Instancje komponentów są skojarzone z podmiotami. Systemy to algorytmy masowo przetwarzające komponenty [101], [102], [114], uruchamiane przez silnik w kolejnych klatkach programu.

Sam wzorzec nie zakłada konkretnego sposobu implementacji podmiotów, komponentów i systemów [101]. Forma podmiotu, sposób przechowywania komponentów i realizacji zapytań różni się pomiędzy rozwiązaniami. Co więcej, samo wykorzystanie ECS nie daje gwarancji wydajności, jeśli implementacja nie podąża za zasadami DOD. Komponenty mogą być przechowywane w prosty sposób poprzez strukturę zawierającą tablice komponentów (SoA, ang. *structure of arrays*)[101], [102], [114], lecz profesjonalne implementacje silników ECS decydują się na własne rozwiązania. Przykładowo, Unity DOTS definiuje zbiór typów komponentów jako „archetyp" i wyszukuje komponenty wyłącznie wewnątrz odpowiedniego archetypu [103]. Alternatywą do SoA i archetypów jest wykorzystanie rzadkich tablic (ang. *sparse set array*) [114]. Bevy łączy te podejścia – wewnętrznie korzysta z zaawansowanej implementacji archetypów [114][9] i rzadkich tablic, ale API pozwala

[9] Problem z archetypami pojawia się, gdy komponenty mogą być dodawane do podmiotu lub usuwane podczas działania programu, przez co podmiot zmienia swój archetyp, a więc przydzielone miejsce w pamięci dla jego komponentów, co może być kosztowne. Bevy wykorzystuje tutaj strukturę przyspieszającą nazwaną „grafem archetypów", w którym archetypy są węzłami, a operacje dodania/usunięcia konkretnych komponentów krawędziami.

zarówno na dostęp do archetypów, jaki i do pojedynczych komponentów, z opcjonalnym filtrowaniem [115]. Niektóre silniki ECS używają zwykłych liczb jako identyfikatorów podmiotów [101], a inne wskaźniki albo niewielkie struktury [116]. Różnice wynikają z tego, że każde rozwiązanie stawia przed projektantem szereg innych wyzwań.

Podejście ECS może przypominać ideę relacyjnych baz danych. Podmioty pełnią funkcje rekordów (wierszy tabel), komponenty to kolumny, a systemy wykonują kwerendy do tabel, aby uzyskać dane do przetworzenia. Łatwo zauważyć, że archetypy funkcjonalnie odpowiadają bazom danych z pojedynczą tabelą. Taka obserwacja jest podstawą do sugestii, żeby zastosować popularną technikę stosowaną przy projektowaniu baz danych – normalizację – przy projektowaniu komponentów [112], [114]. Kolejne formy normalizacji pozwalają zoptymalizować ilość danych i operacji potrzebnych do reprezentacji i zmiany elementów gry.

Przykładami zastosowania wzorca ECS są rozszerzenia Unity DOTS [103] dla Unity i Godex [117] dla Godot oraz silniki takie, jak Bevy [94] i flecs [118]. Ogólne zainteresowanie ECS rośnie, przez co jest wykorzystywany także w innych rodzajach projektów: usługach chmurowych [119], symulacjach naukowych [120], edukacji (tzw. poważne gry, ang. *serious games*) [121], generalnym przetwarzaniu danych [114]. Niektóre badania wskazują na to, że programy wykorzystujące wzorzec ECS są wydajniejsze od korzystających wyłącznie z programowania obiektowego [107], [122], choć trudniejsze w implementacji [107].

W silniku Bevy rolę komponentów i systemów pełnią odpowiednio struktury i funkcje. Parametry funkcji definiują kwerendy określające do jakich komponentów system ma dostęp, opcjonalne filtry oraz globalne „zasoby” (implementacja ECS wzorca singleton). Silnik odpowiada za efektywne przygotowywanie zawartości kwerend oraz za szeregowanie i uruchamianie systemów zgodnie z ograniczeniami nałożonymi przez programistę, np. w określonej kolejności, tylko po spełnieniu predykatu lub po upłynięciu określonego czasu. Ciało funkcji może zawierać dowolne instrukcje, lecz dostęp do danych jest ograniczony wyłącznie do komponentów zdefiniowanych kwerend i globalnych zasobów. Poniżej przedstawiono przykładowy system, który implementuje ruch obiektów (Listing 4.1). Uruchomiony system iteruje po wszystkich elementach kwerendy, dodając składowe prędkości przemnożone przez czas od ostatniej klatki do pozycji.

```
#[derive(Component)]
struct Velocity {
    pub x: f32,
    pub y: f32,
}

fn update_pos(mut query: Query<(&mut Transform, &Velocity)>, time: Res<Time>) {
    for (mut transform, velocity) in query.iter_mut() {
        transform.translation.x += velocity.x * time.delta_seconds();
        transform.translation.y += velocity.y * time.delta_seconds();
    }
}
```

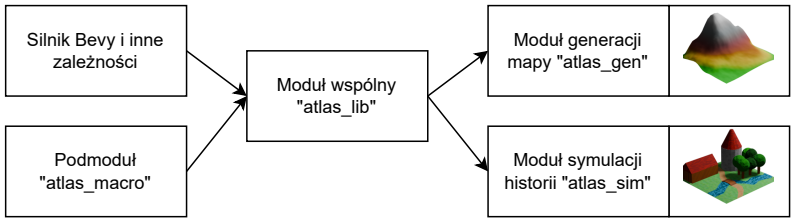
Listing 4.1. Przykładowy komponent i system w silniku Bevy.

4.4. Architektura projektu

Projekt składa się z czterech modułów[10] i ich zależności – dwóch bibliotek łączonych statycznie i dwóch plików wykonywalnych (Rysunek 4.1):

- `atlas_lib` – moduł zawierający kod współdzielony przez generator i symulator oraz większość zależności projektu, m. in. silnik Bevy z wtyczkami,
- `atlas_macro` – podmoduł `atlas_lib` zawierający makra (funkcje przetwarzające kod) pozwalające na automatyczną generację elementów GUI dla struktur danych,[11],
- `atlas_gen` – moduł wykonywalny służący do generacji mapy,
- `atlas_sim` – moduł wykonywalny służący do symulacji historii.

Moduł współdzielony jest odpowiedzialny za generowanie, układanie i wyświetlanie GUI, zarządzanie modelami, teksturami i materiałami, sterowanie kamerą w widoku wizualizacji, obsługę systemu wydarzeń wykorzystywanego do komunikacji pomiędzy odizolowanymi częściami programu oraz serializację i deserializację warstw świata i konfiguracji programu. Ponieważ generator i symulator to oddzielne, *de facto* niezależne programy, transfer danych pomiędzy nimi odbywa się poprzez pliki o określonym formacie. Parametry programów są zapisywane do tekstowych plików konfiguracyjnych w formacie TOML, a warstwy świata jako obrazy w formacie PNG. Dzięki temu generator może być używany do innych celów niż przygotowywanie map wyłącznie na potrzeby symulatora, a symulator może korzystać z danych z dowolnego źródła, jeżeli tylko są odpowiednio przetworzone, chociażby z rzeczywistych pomiarów Ziemi (przykład: Podrozdział 6.2 i Podrozdział 6.3).



Rysunek 4.1. Diagram modułów projektu.

5. Implementacja rozwiązania

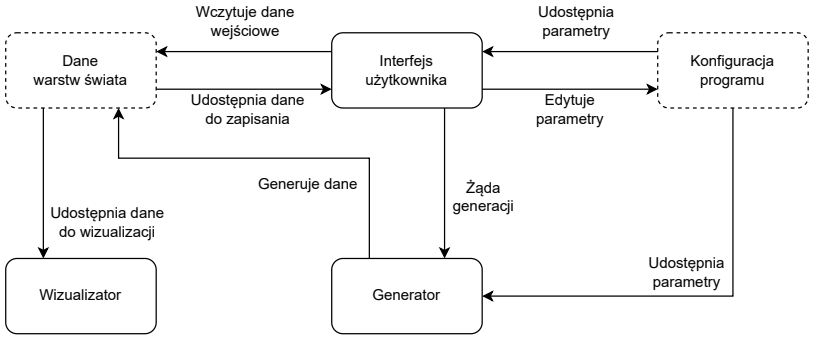
W poniższym rozdziale przedstawiono implementacje poszczególnych modułów realizujących teoretyczne modele, które opisano w Podrozdział 3.1 i Podrozdział 3.2, oraz interfejs użytkownika.

[10] Nomenklatura języka Rust może być tutaj myląca – mowa tutaj o „crate”, czyli strukturze organizacyjnej zawierającej pełen kod bibliotek lub plików wykonywalnych, a nie „module” który jest odpowiednikiem przestrzeni nazw.

[11] Powinien być częścią modułu współdzielonego, ale język Rust wymaga, żeby proceduralne makra znajdowały się w odrębnej bibliotece[87].

5.1. Moduł generatora mapy świata

Architektura i funkcjonalność Program składa się z pięciu głównych części: konfiguracji zawierającej parametry programu, GUI pozwalającego modyfikować parametry i wczytywać/zapisywać dane, danych warstw świata, właściwego generatora, który przetwarza dane warstwy zgodnie z parametrami i widoku wizualizującego dane (Rysunek 5.1). Komunikacja interfejs-generator i generator-wizualizator odbywa się za pomocą własnego mechanizmu kolejki zdarzeń, wykorzystując warunkowe uruchamianie systemów (nazwane w Bevy *run condition*). Konfiguracja i dane warstw świata są globalnie dostępnymi zasobami.



Rysunek 5.1. Zarys interakcji pomiędzy elementami generatora. Elementy wykonujące procedury są narysowane linią ciągłą, a elementy zawierające dane linią przerywaną.

W przeciwieństwie do symulatora, program praktycznie nie korzysta z podmiotów i komponentów – nie ma takiej potrzeby. Generacja warstw świata nie symuluje nic w czasie rzeczywistym, interakcja z użytkownikiem ogranicza się do elementów GUI i poruszania kamerą widoku wizualizacji. Pierwotnie rozważano implementacje każdego punktu świata jako oddzielnego podmiotu, lecz takie podejście zbyt obciążało zasoby sprzętowe (szczególnie przy ogromnym rozmiarze mapy, np. rozdzielczości 1000x1000 punktów), jednocześnie nie oferując znacznie lepszej organizacji kodu programu.

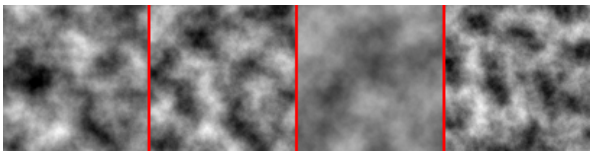
Model i dane Generator wykorzystuje model świata, które został opisany wcześniej (Podrozdział 3.1, Rysunek 3.1). Ponieważ następne warstwy zależą od poprzednich (przykładowo, warstwa temperatury zależy od topografii), każda zmiana danych warstwy (generacja lub wczytanie) automatycznie wywołuje kaskadę regeneracji kolejnych warstw. Dziedziny wartości danych warstw zostały obrane na podstawie dwóch warunków: powinny być w stanie reprezentować ekstrema pomiarowe występujące na Ziemi oraz powinny dać się łatwo enkodować jako jasność pikseli w jednokanałowym obrazie PNG z głębią 8 bitów na piksel (zakres wartości 0–255):

- Warstwa kontynentów – wartości poniżej 128 oznaczają obecność oceanu, a od 128 kontynentu,
- Warstwa topografii – jednostka wysokości odpowiada 40 metrom, dając całkowity zasięg wartości 0–10200 metrów,

- Warstwa temperatury – 100 jednostek temperatury odpowiada 0 °C, a różnica jednej jednostki opadów odpowiada różnicy 0.5 °C, dając całkowity zasięg -50–77.5 °C,
- Warstwa opadów – jednostka opadów odpowiada 20 milimetrom opadów, dając całkowity zasięg wartości 0–5100 mm,
- Warstwa klimatu – wartość odpowiada indeksowi przypisanego biomu w liście biomów,

W obecnej implementacji warstwa zasobów naturalnych jest wyjątkiem, jako że jest zapisywana jako część konfiguracji, a nie jako obraz. Każdy punkt może zawierać wiele rodzajów złóż zasobów o różnym rozmiarze – jest to za dużo informacji, by dało się je wydajnie zakodować w jednym bajcie (zwłaszcza, że użytkownik może zdefiniować dowolnie wiele typów złóż zasobów). Zamiast tego sąsiadujące punkty na mapie są grupowane w kwadratowe kawałki o określonym rozmiarze, nazwane *resource chunk*. Rozmiary złóż zasobów w kawałku są sumowane. Poprzez zapis tylko kawałków taka sama ilość informacji o *ilości* zasobów wykorzystuje mniej pamięci kosztem utraty informacji o *lokalizacji* tych zasobów wewnątrz kawałku. Rozmiar kwadratu wyznaczającego kawałek jest określany parametrem dostępnym dla użytkownika. W ten sposób, w zależności od potrzeb można zdecydować, jak bardzo zasoby powinny być „kompresowane".

Generacja – szumy Generator wykorzystuje głównie fraktalny gradientowy szum losowy jako źródło danych, który następnie jest przekształcany w sposób zależny od warstwy. Użytkownik ma do wyboru następujące algorytmy: Perlin [22], PerlinSurflet [123][12], OpenSimplex [24][13], SuperSimplex [24][14] (Rysunek 5.2). Wykorzystane w tej pracy implementacje algorytmów są dostępne w ramach biblioteki `noise-rs` [123].



Rysunek 5.2. Przykładowe szumy generatora. Od lewej do prawej: Perlin, PerlinSurflet, OpenSimplex, SuperSimplex. W każdym przypadku wykorzystano identyczne parametry i ziarno.

Szum może być opcjonalnie przetworzony podczas próbkowania za pomocą mnożenia i dodawania, które pozwalają zmniejszyć, zwiększyć lub przesunąć wyjściowy zakres wartości[15]. Kolejnym krokiem przetwarzania jest zastosowanie funkcji liniowej mapującej zakres wartości danych. Funkcja ta jest określona w przedziale $[0,1]$, posiada zakres wartości $[0,1]$ i jest zdefiniowana jako wielopunktowa interpolacja liniowa (Wzór 5.2). Parametry p_0 i p_1 to punktu sklejenia przedziałów, a v_0, v_1, v_2, v_3 to wartości odpowiadające argumentom 0, p_0, p_1 i 1.

[12] Implementacja algorytmu Perlina wykorzystująca funkcje falkowe zamiast interpolacji gradientów.
[13] Otwartoźródłowy algorytm oparty o algorytm simplex Perlina[23].
[14] Ulepszony wariant OpenSimplex.
[15] Analogicznie do aplikowania wag i odchyleń perceptronu przed zastosowaniem funkcji aktywacji w uczeniu maszynowym.

$$Lerp(x_0, x_1, t) = (1-t) \cdot x_0 + t \cdot x_1 \tag{5.1}$$

$$f(x) = \begin{cases} Lerp(v_0, v_1, \frac{x}{p_0}), & x \leq p_0 \\ Lerp(v_1, v_2, \frac{x-p_0}{p_1-p_0}), & p_1 \geq x \geq p_0 \\ Lerp(v_2, v_3, \frac{x-p_1}{1-p_1}), & x \geq p_0 \end{cases} \tag{5.2}$$

W ten sposób użytkownik jest w stanie manipulować rozkładem wartości wygenerowanych danych, co jest przydatne przy generowaniu topografii. Analiza zalet wynikających z zastosowania dwóch ostatnich kroków (mapy wpływu i mapowania) znajduje się w dalszej części pracy (Podrozdział 6.2).

Następnie wykonywane są przekształcenia specyficzne dla warstwy, które zostały opisane przy definicji modelu świata (Podrozdział 3.1). Warstwa kontynentów wykorzystuje parametr „poziomu morza", do sklasyfikowania, które punkty reprezentują ląd, tzn. czy wartość warstwy w punkcie przewyższa wysokość poziom morza. Jako, że warstwa topografii zawiera informacje o wysokości nad poziomem morza, wszystkie punkty morskie mają ustawianą wartość w tej warstwie na 0. Punkty lądowe w sąsiedztwie linii brzegowej (styku lądu i wody) mogą być opcjonalnie podane wpływowi „erozji", realizowanej za pomocą wygładzającego filtru jednorodnego o rozmiarze kernela podanym przez użytkownika. Warstwy temperatury i opadów dodają do szumu gradient na bazie szerokości geograficznej. Dodatkowo poziom temperatury i opadów jest obniżany wraz ze wzrostem wysokości terenu, symulując gradient wilgotnoadiabatyczny i zjawisko inwersji opadowej.

Proces generowania warstwy jest zwieńczony opcjonalną transformacją za pomocą skojarzonej z warstwą „mapy wpływu". Mapa wpływa to dodatkowe tablica o identycznym rozmiarze, która zawiera wagi wczytane lub wygenerowane przez użytkownika. Wagi te są wykorzystane jako zewnętrzna informacja pozwalająca modyfikować punkty w wybranych fragmentach przestrzeni świata. Ten fakt odróżnia ten krok od poprzednich przekształceń, które modyfikowały jednakowo wszystkie punkty przestrzeni, operując wyłącznie na ich zakresie wartości. Udostępniono trzy funkcje skalujące na podstawie mapy wpływu: $ScaleDown$ (Wzór 5.3), $ScaleUp$ (Wzór 5.4) i $ScaleUpDown$ (Wzór 5.5). Wartość w danym punkcie mapy oznaczono symbolem x, wagę odpowiadająca temu punktowi w, a s jest parametrem określającym siłę transformacji, podawanym przez użytkownika.

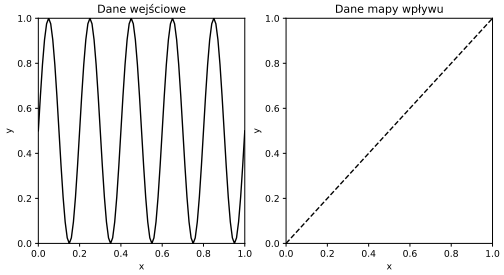
$$ScaleDown(x, w) = Lerp(0, x, 1 - (1-w) \cdot s) \tag{5.3}$$

$$ScaleUp(x, w) = Lerp(x, 1, w \cdot s) \tag{5.4}$$

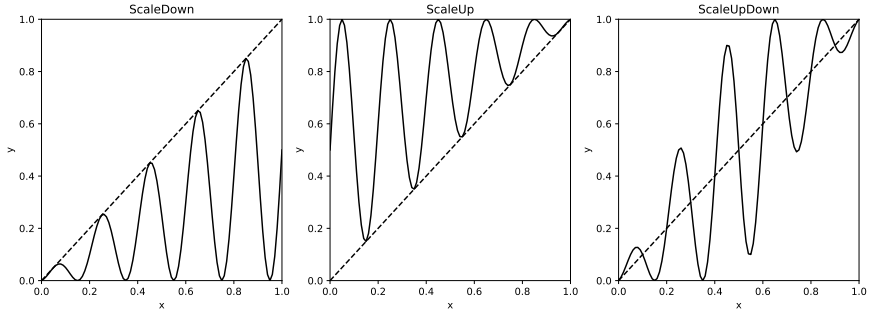
$$ScaleUpDown(x, w) = \begin{cases} Lerp(0, x, 2s \cdot (w-0.5) + 1) & (w-0.5), \cdot s \leq 0 \\ Lerp(x, 1, 2s \cdot (w-0.5)) & (w-0.5), \cdot s \geq 0 \end{cases} \tag{5.5}$$

Funkcja $ScaleDown$ skaluje wartość warstwy ku 0 wraz ze spadkiem wagi, $ScaleUp$

skaluje wartość ku 1 wraz ze wzrostem. $ScaleUpDown$ funkcjonuje jak $ScaleDown$ gdy $w \leq 0.5$, a $ScaleUp$ gdy $w \geq 0.5$, po uprzednim przeskalowaniu wartość wagi z powrotem do przedziału $[0,1]$. Rysunek 5.3 i Rysunek 5.4 obrazują działanie funkcji na przykładzie danych w jednym wymiarze.



Rysunek 5.3. Transformacja mapą wpływu w jednym wymiarze, cz. 1. Wykres po lewej przedstawia dane wejściowe, wykres po prawej dane mapy wpływu.

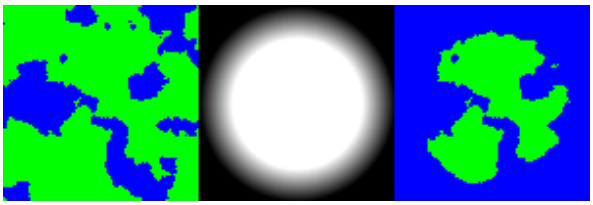


Rysunek 5.4. Transformacja mapą wpływu 1D, cz. 2: linia ciągła przedstawia wyjściowe dane, linia przerywana dane mapy wpływu, identyczne jak w cz. 1.

Głównym zamierzonym przypadkiem użycia mapy wpływu jest prymitywne modyfikowanie warstwy kontynentów bez potrzeby zapisywania warstwy i wykorzystania edytora obrazów Rysunek 5.5. Rysunek po lewej przedstawia nieprzetworzoną warstwę kontynentów, zieleń to kontynenty, a błękit to oceany. Na środku znajduje się mapa wpływu, gdzie czerń przedstawia wagę 0, a biel 1. Rysunek z prawej strony przedstawia rezultat transformacji – pseudo-wysokość punktów została przeskalowana w dół, poniżej poziomu morza, więc ląd został zatopiony. Program pozwala wypełnić mapy wpływu na cztery sposoby:

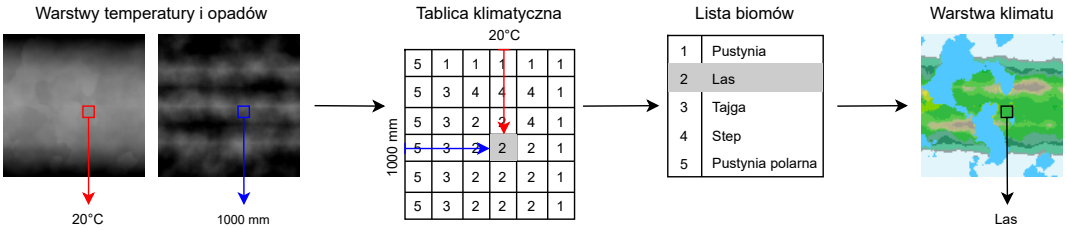
- wykorzystując szum,
- generując koło, w których wagi maleją wraz z odległością od środka koła,
- generując kształt pigułki – dwa koła połączone odcinkiem, wagi maleją wraz z odległością od środków kół i odległością od odcinka,
- wczytując 8-bitowy obraz PNG w odcieniach szarości.

Generacja – inne metody W przeciwieństwie do innych warstw, warstwa klimatu nie wykorzystuje szumu ani innych metod generowania nowych informacji. Jej zadaniem jest



Rysunek 5.5. Praktyczny przykład zastosowania mapy wpływu i metody ScaleDown.

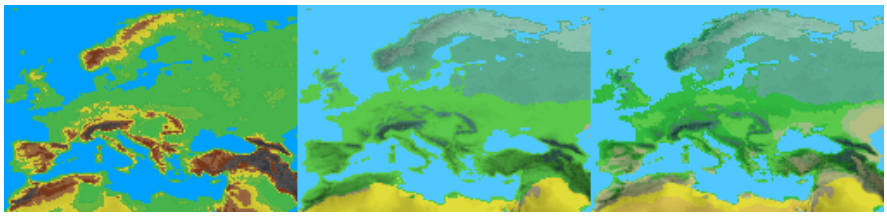
klasyfikacja punktów zawierających dane o temperaturze i opadach do biomów klimatycznych. Jest to zrealizowane przy pomocy dwuwymiarowej tablicy, w której kolumny i wiersze reprezentują dyskretne przedziały odpowiednio poziomu temperatury i ilości opadów atmosferycznych, a komórki zawierają indeksy do listy biomów zdefiniowanej w konfiguracji. Aby przypisać biom, wystarczy, że każdy punkt świata odczyta swoje wartości z warstw temperatury i opadów i użyje je jako numery kolumn i wierszy w tej tablicy. Rysunek 5.6 ilustruje powyższy proces. Domyślna tablica wykorzystana w tej pracy jest bezpośrednio oparta na diagramie biomów z rozdziału trzeciego (Rysunek 3.3). Program pozwala wczytać tablicę dostarczoną przez użytkownika w formie obrazu 256x256 pikseli, 8 bitów na piksel, w formacie PNG, gdzie jasność piksela odpowiada indeksowi biomu. Biom o indeksie 0 jest zarezerwowany dla specjalnego biomu morskiego, który jest przypisywany wtedy i tylko wtedy, gdy punkt nie znajduje się na lądzie.



Rysunek 5.6. Proces przypisywania biomów w warstwie klimatu.

Warstwa zasobów generowana jest poprzez wykonywanie dwóch testów dla każdego punktu, dla każdego dostępnego typu złoża zasobów. Pierwszy test obejmuje losowanie dostępności złoża za pomocą próby Bernoulliego. Jeśli próba zakończyła się sukcesem, następuje losowanie rozmiaru złoża za pomocą rozkładu normalnego. Określone typy złóż zasobów mogą być ograniczone do konkretnych biomów – w takim przypadku tylko punkty należące do tych biomów mogą próbować losować te złoża. Po wylosowaniu złóż informacje są grupowane w kawałki tak, jak zostało to opisane w akapicie „Model i dane".

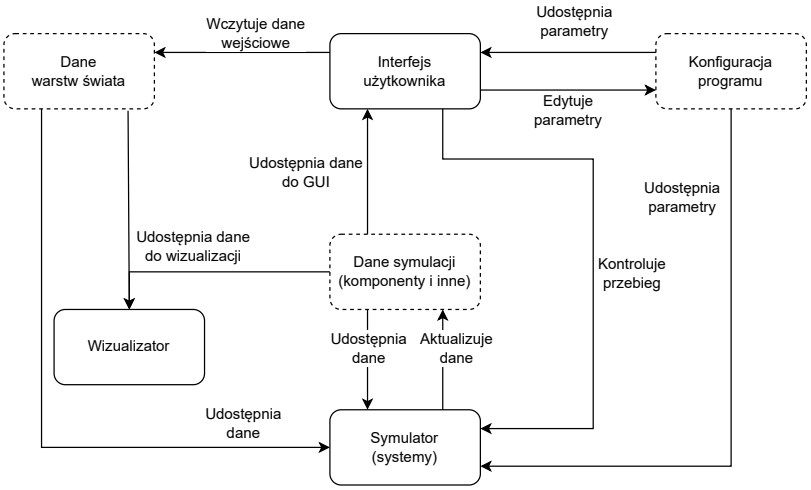
Po wygenerowaniu wszystkich warstw, generator jest w stanie dodatkowo stworzyć kolorowy podgląd mapy na trzy sposoby (Rysunek 5.7): kolorując warstwę topografii, kolorując warstwę klimatu w sposób szczegółowy lub uproszczony (z mniejszą paletą kolorów). Paleta kolorów klimatu może być zdefiniowana przez użytkownika jako część konfiguracji. Podgląd jest zapisywany jako zwykły obraz w modelu przestrzeni barw RGBA.



Rysunek 5.7. Wygenerowane warianty podglądu wczytanej mapy Europy, kolejno: podgląd topografii, uproszczony podgląd klimatu, szczegółowy podgląd klimatu. Zaciemnienia oznaczają różnicę wysokości.

5.2. Moduł symulatora historii

Architektura i funkcjonalność Program jest zbudowany podobnie do programu generatora. Składa się z konfiguracji zawierającej parametry programu, GUI pozwalającego modyfikować parametry i wczytywać/zapisywać dane, wczytanych danych warstw świata, widoku wizualizującego dane, danych symulacji i właściwego symulatora (Rysunek 5.8). Program działa w dwóch stanach: stanie inicjalizacji i stanie symulacji.



Rysunek 5.8. Zarys interakcji pomiędzy elementami symulatora. Elementy wykonujące procedury są narysowane linią ciągłą, a elementy zawierające dane linią przerywaną.

Stan inicjalizacji jest początkowym stanem programu i pozwala na wczytywanie warstw świata, wyświetlanie podglądu, konfigurowanie parametrów symulacji i ustaleniu warunków początkowych symulacji, takich jak liczba państw, początkowa liczba ludności, pozycje punktów startowych ludzkości.

Po zatwierdzeniu wszystkich zmian przez użytkownika program przechodzi do drugiego stanu i rozpoczyna działanie symulatora. W tym stanie modyfikacja warstw świata i parametrów programu lub powrót do poprzedniego stanu jest niemożliwy. Użytkownik może obserwować postępy symulacji, podglądać informacje o wybranych obiektach i kontrolować bieg czasu: wstrzymać/kontynuować lub przyspieszyć/spowolnić.

Symulator akceptuje dane warstw świata w formie opisanej wcześniej (Podrozdział 5.1) i bezpośrednio implementuje model rozwoju cywilizacji, który opisuje Podrozdział 3.2. W

przeciwieństwie do generatora, symulator faktycznie korzysta z wzorca ECS do reprezentacji elementów symulacji, które można podzielić na trzy grupy. Podmioty państw zawierają komponenty państw i są bytami wirtualnymi bez lokalizacji na mapie. Podmioty państw posiadają na własność podmioty regionów, które zawierają komponenty regionów, pozycję, powierzchnię i graficzną reprezentacje. Podmioty znaczniki są wyłącznie wizualne; nie zawierają żadnych istotnych danych i są wykorzystywane do oznaczania na mapie miejsc takich, jak stolice regionów czy punkty startowe państw. Przyszłe wersje implementacji mogą być poszerzone o podmioty reprezentujące konflikty, które obecnie są zawarte w globalnych zasobach.

Symulacja Symulacja nie może rozpocząć się bez uruchomienia algorytmu ustalającego warunki początkowe dla określonej przez użytkownika liczby punktów startowych. Każdy punkt reprezentuje oddzielne państwo i zawiera wylosowaną lokalizację na mapie, kolor i pierwotne wartości polityki oraz początkową liczbę ludności, która domyślnie jest jednakowa dla wszystkich punktów startowych. Algorytm wyboru lokalizacji jest podzielony na dwie części: przygotowanie wag do losowania i właściwe losowanie. Część przygotowująca tworzy dwie tablice: główną, zawierającą wagi dla każdego punktu mapy i pomocniczą, zawierającą średnią arytmetyczną wag dla każdego wiersza (Rysunek 5.9). Proces tworzenia tablic jest następujący:

1. Inicjalizuj główną tablicę o rozmiarze takim samym, co mapa świata,
2. Do każdego elementu tablicy przypisz wartość 1, jeśli odpowiadający punkt jest na lądzie albo 0, jeżeli jest na wodzie. Punkty z wagą 0 nie biorą udziału w losowaniu,
3. Dla każdego elementu tablicy oblicz wagę za pomocą metryki wykorzystującej współczynnik zamieszkania klimatu,
4. Utwórz pomocniczą tablicę o długości odpowiadającej liczbie wierszy głównej tablicy,
5. Dla każdego wiersza głównej tablicy zsumuj wartość elementów, podziel ją przez długość wiersza i przypisz ją do odpowiadającego elementu tablicy pomocniczej.

Wspomniana metryka jest wybierana przez użytkownika spośród pięciu dostępnych funkcji przyjmujących indeks rozważanego punktu i. Wartość współczynnika zamieszkania dla klimatu znajdującego się w punkcie o indeksie x oznaczono symbolem $C(x)$, a długość wiersza mapy świata symbolem w.

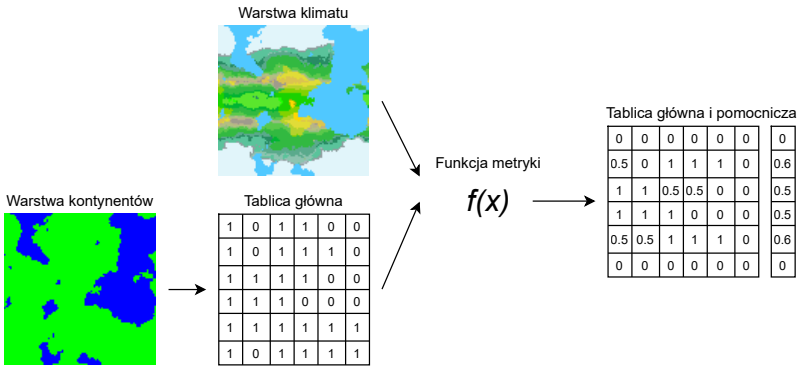
$$Uniform(i) = 1 \tag{5.6}$$

$$Weighted(i) = C(i) \tag{5.7}$$

$$WeightedSquared(i) = C(i)^2 \tag{5.8}$$

$$WeightedArea(i) = \begin{cases} \sum_{j=-1}^{1} \sum_{k=-1}^{1} C(i + j + k \cdot w) & C(i) \geq 0.1 \\ 0 & C(i) \leq 0.1 \end{cases} \tag{5.9}$$

$$WeightedSquaredArea(i) = \begin{cases} \sum_{j=-1}^{1} \sum_{k=-1}^{1} C(i+j+k\cdot w)^2 & C(i)^2 \geq 0.1 \\ 0 & C(i)^2 \leq 0.1 \end{cases} \tag{5.10}$$



Rysunek 5.9. Proces tworzenia tablicy głównej i pomocniczej.

Losowanie punktów startowych odbywa się poprzez ważone losowanie kolejno szerokości i długości geograficznej, czyli wiersza tablicy i pozycji w wierszu. Do losowania wiersza wykorzystane są wagi tablicy pomocniczej; losowanie pozycji w wierszu za to korzysta z wag tego wiersza tablicy głównej. Do wyboru losowego elementu ze zbioru wag wykorzystano implementację tzw. metody aliasów Walkera [124] z biblioteki `weighted_rand` [125], która wykorzystuje strukturę pomocniczą konstruowaną w czasie $\mathcal{O}(n)$ do próbkowania w czasie $\mathcal{O}(1)$. Algorytm losowania punktów leniwie konstruuje strukturę dla każdego wykorzystanego wiersza tablicy głównej i zapamiętuje ją w tablicy asocjacyjnej, przez co czas losowania jest znacznie amortyzowany. Jeżeli wylosowana pozycja jest wolna, to zostaje przydzielona do punktu startowego i oznaczona jako zajęta, a algorytm kontynuuje działanie dla następnego punktu startowego. Jeżeli pozycja jest zajęta przez inny punkt, losowanie jest ponawiane do 4 razy. Jeżeli pozycja nadal jest zajęta, algorytm oznacza punkt startowy jako nieprzypisany (nie bierze udziału w symulacji) i kontynuuje działanie. Jeżeli nie wszystkie punkty mają przypisaną lokalizację, po zakończeniu działania algorytmu użytkownik jest o tym informowany i może ponowić losowanie.

Po ustaleniu warunków początkowych program może przejść do następnego stanu, w którym przetwarza kolejne kroki symulacji z określoną przez użytkownika częstotliwością (maksymalnie 60 kroków na sekundę). Symulacja jest realizowana przez systemy implementujące poszczególne aspekty modelu cywilizacji. W większości systemy ograniczają się do zastosowania funkcji opisanych w modelu, dlatego komentowane będą tylko algorytmiczne fragmenty rozwiązania, w szczególności te związane z informacją przestrzenną.

Wyznaczanie granicy regionów wykorzystuje sąsiedztwo von Neumanna (cztery punkty sąsiadujące po bokach) do otrzymania zbioru sąsiadów każdego punktu regionu, z którego następnie są usuwane wszystkie punkty wewnątrz tego regionu. Granice regionów są wykorzystywane przy ustalaniu sąsiadujących państw poprzez sprawdzanie przynależności

punktów na granicy i znajdywaniu punktów nadających się do ekspansji, ograniczając się do punktów lądowych o odpowiednio wysokim współczynniku zamieszkania klimatu. Jeżeli region jest w stanie zająć nowe terytorium, jeden z punktów granicznych jest wybierany korzystając z metody Walkera, używając współczynników zamieszkania jako wag.

Tworzenie nowych regionów odbywa się przez podział istniejących regionów i jest procesem trzystopniowym. Pierwszy z nich to sprawdzenie spełnienia warunków: region rodzic musi być w stanie pokryć koszt zbudowania nowego miasta, jego rozmiar musi być odpowiednio duży i musi istnieć co najmniej jeden punkt wewnątrz regionu, który nie jest w sąsiedztwie Moore'a (osiem sąsiadów) żadnego miasta dowolnego państwa. Jeżeli warunki są spełnione, jeden z tych punktów wewnątrz regionu jest losowo wybierany jako lokalizacja nowego miasta z rozkładem jednostajnym. Następnie odbywa się wyznaczanie nowych powierzchni regionów i migracja ludności. Każdy punkt należący do państwa jest przypisywany do regionu, którego miasto znajduje się najbliżej, korzystając z metryki euklidesowej dla dwóch wymiarów. Do znalezienia najbliższego punktu z listy miast wykorzystano algorytm R*-drzewo [126], który pozwala na indeksowanie i wyszukiwanie obiektów w przestrzeni z wykorzystaniem struktury pomocniczej. R*-drzewo jest modyfikacją algorytmu R-drzewa [127] o wydajniejszych zapytaniach ($\mathcal{O}(\log n)$ zamiast $\mathcal{O}(\log_M n)$, gdzie M oznacza maksymalną liczbę elementów), lecz nieznacznie większym jednorazowym koszcie konstrukcji struktury. Praca korzysta z implementacji algorytmu z biblioteki `rstar` [128]. Po przypisaniu wszystkich punktów do regionów populacja państwa i poziom infrastruktury są ponownie dzielone między regiony proporcjonalnie do udziału nowej powierzchni regionu w powierzchni państwa.

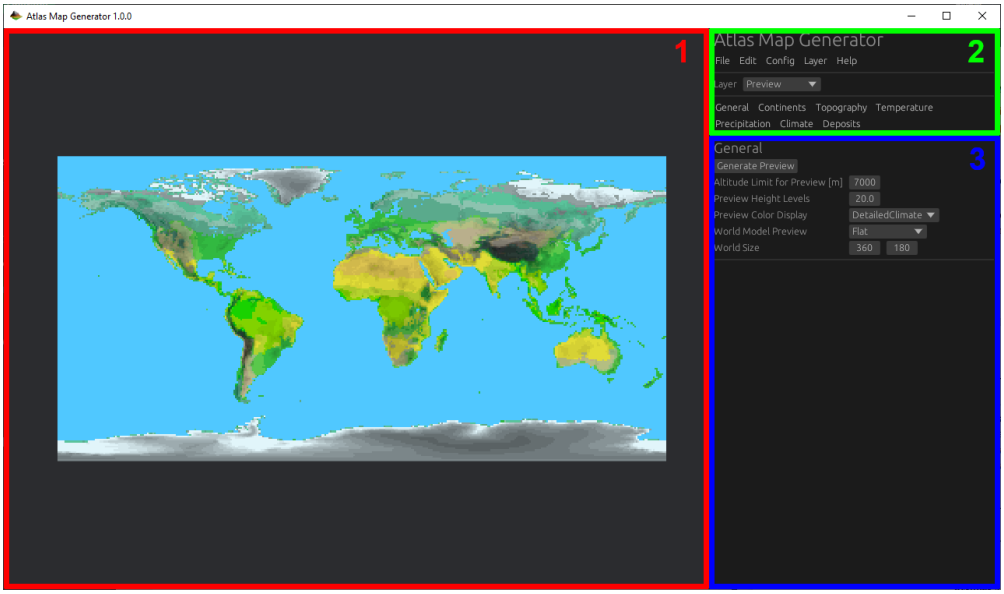
Wybór akcji podczas konfliktów odbywa się poprzez losowanie z wykorzystaniem metody Walkera.

5.3. Interfejs użytkownika

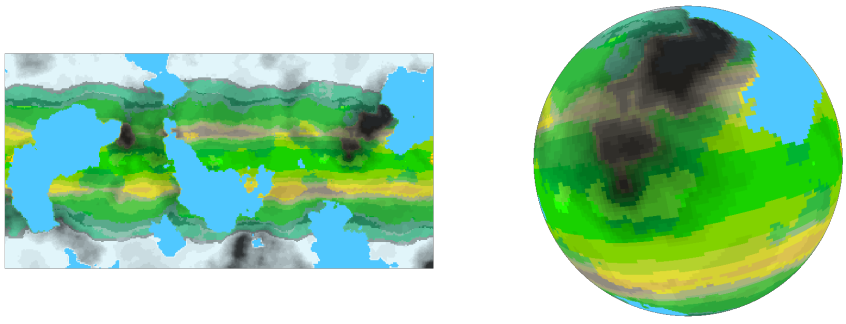
Użytkownik wchodzi w interakcję z programami wyłącznie poprzez interfejs graficzny. Oba programy korzystają z tego samego układu GUI (Rysunek 5.10).

Widok świata (numer 1, czerwona ramka) pozwala na interaktywną wizualizację wybranej warstwy świata w formie płaskiej mapy, która może być przesuwana, oddalana lub przybliżana za pomocą myszy. Program generatora pozwala także na rzutowanie warstwy na obrotowy trójwymiarowy model kuli (Rysunek 5.11). Nagłówek panelu bocznego (numer 2, zielona ramka) zawiera pasek menu, listę pozwalającą na zmianę podglądanej warstwy świata i zakładki do zmiany paneli. Program symulatora umiejscawia tutaj także elementy kontrolne do ustawiania prędkości symulacji i filtrów widoczności obiektów. Właściwy panel (numer 3, niebieska ramka) służy do edycji konfiguracji programu lub wyświetlania szczegółowych informacji o zaznaczonych obiektach w stanie symulacji.

Zarówno parametry generatora i symulatora, jak i warstwy świata mogą być edytowane „online" za pomocą GUI i „offline" jako pliki. Szczegółowy opis wszystkich opcji menu, paneli, plików konfiguracyjnych i akceptowanych formatów danych znajduje się w Załączniku 1 i Załączniku 2, instrukcjach użytkownika dla programów `atlas_gen` i `atlas_sim`.



Rysunek 5.10. Ogólny układ interfejsu użytkownika



Rysunek 5.11. Wizualizacja podglądu świata za pomocą mapy (po lewej) i kuli (po prawej).

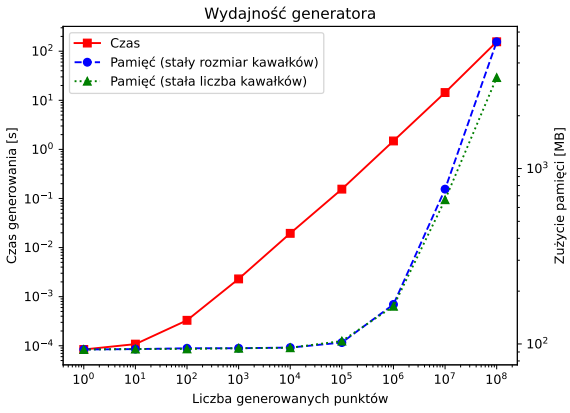
6. Wyniki

Stworzone implementacje generatora i symulatora zostały poddane badaniom wydajności i poprawności, weryfikując przyjęte założenia odnośnie modelu świata i historii. Rozdział zawiera opis przeprowadzonych badań, komentarz odnośnie otrzymanych wyników i propozycje dalszego rozwoju.

6.1. Badania wydajności

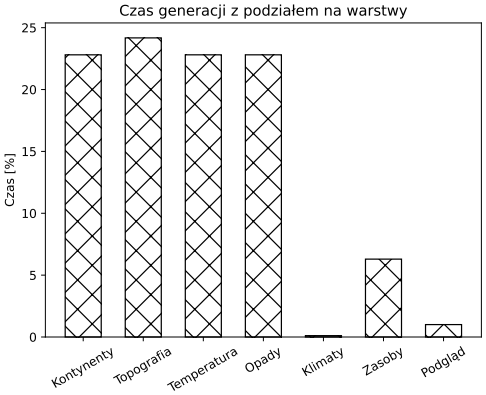
Implementacje generatora i symulatora zostały zbadane pod kątem wydajności: czasu potrzebnego do wygenerowania świata i szybkości symulacji. W obu przypadkach zmierzono także zużycie pamięci podczas działania. Wszystkie testy zostały uruchomione na maszynie z systemem operacyjnym Windows 10 Home (kompilacja KB5041580), procesorem Intel Core i5-7400 3 GHz i 16 GB pamięci RAM DDR4. Zebrane dane przedstawiają uśrednione wyniki uzyskane w pięciu pomiarach.

Czas trwania procesu generowania światów został zmierzony w zależności od rozmiaru mapy – dla każdego kolejnego pomiaru zwiększano całkowitą liczbę punktów 10 razy. Rysunek 6.1 przedstawia wyniki pomiarów dla typowych rozmiarów (do 1000x1000) i ekstremalnych (do 10000x10000). Zużycie pamięci jest mierzone dla dwóch scenariuszy: stałego rozmiaru kawałków złóż zasobów (wynoszącego domyślne 18 jednostek odległości) i stałej liczby kawałków (arbitralnie ustalonej na 20). Taki podział nie ma znaczenia dla małych i średnich światów, ale w przypadku ekstremalnych rozmiarów wariant ze stałą liczbą kawałków potrzebuje znacznie mniej pamięci (niecałe 2 GB mniej dla 10000x10000, czyli ok. 38% mniej), kosztem mniejszej precyzji lokalizacji zasobów. Na wykresie nie jest to widoczne, ale wzrost pamięci jest w rzeczywistości tak samo wykładniczy jak wzrost czasu. Wynika to z faktu, że do całkowitego zużycia pamięci wliczony jest stały koszt wynikający z działania programu (ok. 93 MB), który znacznie przewyższa koszt stworzenia mapy o małym i średnim rozmiarze. Ogólną wydajność generatora uznano za zadowalającą – dla typowych rozmiarów mapy cały proces odbywa się natychmiastowo lub prawie natychmiastowo.

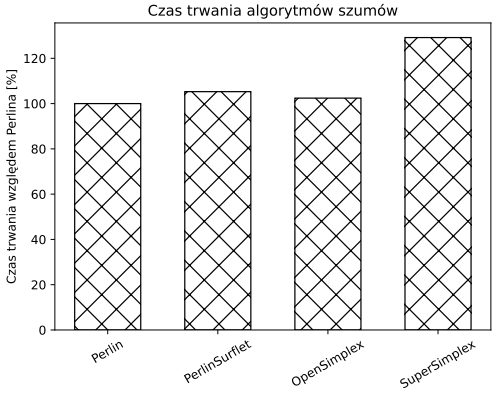


Rysunek 6.1. Wyniki pomiarów wydajności generatora.

Oprócz tego zmierzono czas generowania poszczególnych warstw świata (Rysunek 6.2) i porównano wydajność wykorzystywanych algorytmów generowania szumów (Rysunek 6.3). Zgodnie z intuicją, warstwy kontynentów, temperatury, opadów i topografii zajmują najwięcej czasu (każda ok. 22%) ze względu na wykorzystywanie algorytmów szumu. Szum Perlina, PerlinSurflet i OpenSimplex są podobnie szybkie, a SuperSimplex jest ok. 25% wolniejszy[16].



Rysunek 6.2. Porównanie czasu generacji warstw.



Rysunek 6.3. Porównanie czasu trwania algorytmów szumu.

Ponieważ symulacja odbywa się w czasie rzeczywistym, a nie na żądanie tak jak generacja, testy wydajności zostały przeprowadzone w inny sposób. Przygotowano trzy scenariusze wykorzystujące mapę Ziemi o rozmiarze 1000x500 z różną liczbą losowo położonych państw początkowych (100, 1000 i 10000) i sprawdzono, czy symulacja jest w stanie płynnie działać z maksymalną szybkością 160 lat na minutę. Przeprowadzone testy wykazały, że symulator jest w stanie spełnić ten warunek na maszynie testowej przy akceptowalnym obciążeniu systemu. Zebrane dane przedstawiono poniżej (Tabela 6.1).

Oprócz testów wydajnościowych przeprowadzono także profilowanie czasu wykorzystywanego przez poszczególne systemy symulacji (Rysunek 6.4). Ponad połowa czasu procesora jest wykorzystywana przez silnik Bevy do uruchamiania i obsługi kwerend systemów, zarządzania komponentami i wizualizowania świata. Większość pozostałego czasu jest spędzana w systemach przetwarzających przede wszystkim informacje przestrzenne: urbanizacja 13,51% (tworzenie i podział regionów), dyplomacja 10,44% (szukanie sąsiadów i określanie relacji), podział pracy i produkcja zasobów 9,67%, ekspansja 6,81% (wyznaczanie i rozszerzanie granic regionów).

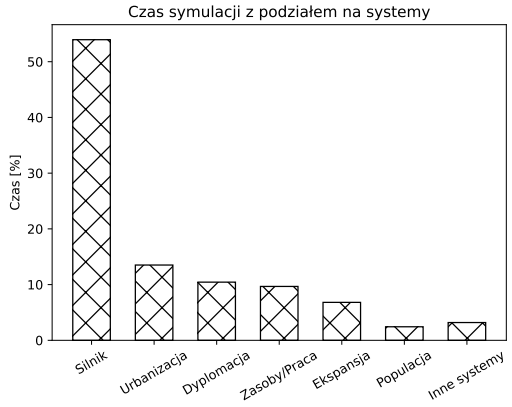
6.2. Analiza wyników generatora

Warstwy topografii, temperatury i opadów zostały porównane z rzeczywistymi danymi pomiarowymi Ziemi. W tym celu przygotowano mapę Ziemi w dwóch rozmiarach (360x180 i 1000x500) w formacie wykorzystywanym przez generator i symulator na podstawie sateli-

[16] Autor algorytmów OpenSimplex i SuperSimplex podaje, że powinny być porównywalnie szybkie, co oznacza, że implementacja tego algorytmu w wykorzystywanej bibliotece jest prawdopodobnie niedoskonała

Czas [min]	L. państw	L. regionów	Śr. CPU [%]	Pamięć [MB]
0	100	100	9	143
2	99	100	15	144
4	97	101	15	145
6	90	264	15	112
8	77	1353	18	184
0	1000	1000	13	218
2	713	1000	14	220
4	560	1003	17	220
6	273	3438	30	340
8	62	11974	25	807
0	10000	10000	31	870
2	862	10433	32	890
4	330	14090	33	1086
6	299	14217	33	1088
8	287	14334	28	1100

Tabela 6.1. Zużycie zasobów komputera podczas symulacji z maksymalną szybkością, trzy scenariusze testowe. Pamięć oznacza rozmiar zbioru roboczego programu (*working set*).

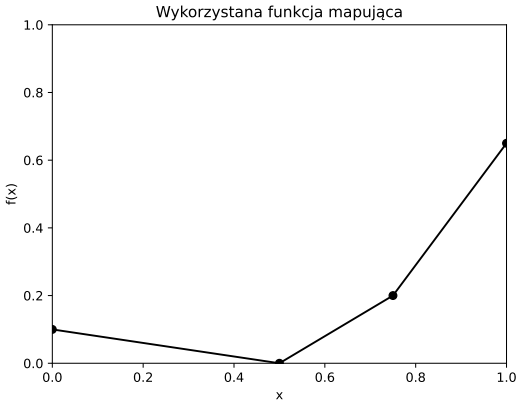


Rysunek 6.4. Porównanie czasu działania systemów i silnika.

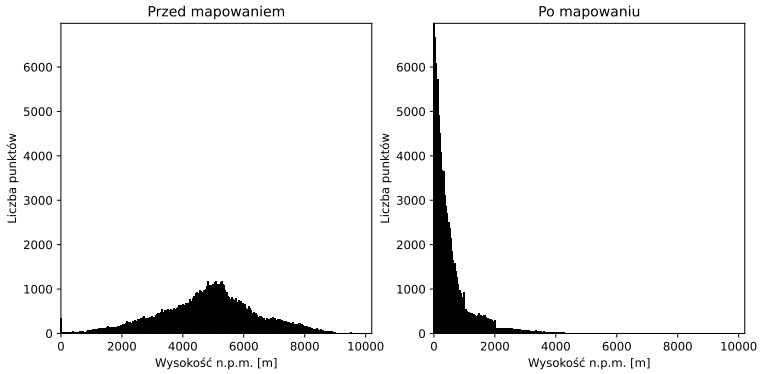
tarnej mapy wysokości wykonanej przez NASA [129] i zbioru danych klimatycznych CRU CL v2.0 [130].

W przypadku topografii porównano histogramy wysokości dla losowych map generatora z histogramem wysokości kontynentalnej części Ziemi (Rysunek 6.7, [131]). Aby uzyskać realistyczny rozkład wysokości wykorzystano mechanizm „funkcji mapowania” (Podrozdział 5.1), który pozwala manipulować wartościami punktów warstw za pomocą parametyzowanej funkcji. Rysunek 6.5 przedstawia taką funkcję, a Rysunek 6.6 histogramy wygenerowanych map wysokości przed i po zastosowanej korekcie. W ten sposób wygenerowany teren nie sprawia wrażenia całkowicie sztucznego: wysokie łańcuchy górskie i wyżyny formują „wyspy”, przestrzeń pomiędzy którymi jest wypełniona nizinami i dolinami.

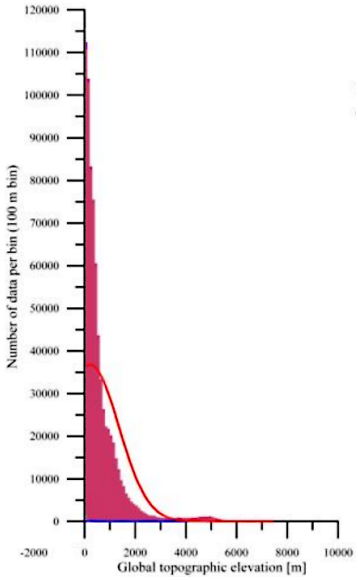
Zdolność generowania realistycznych warstw temperatury i opadów została sprawdzona



Rysunek 6.5. Jedna z zastosowanych funkcji mapowania topografii.



Rysunek 6.6. Histogramy wygenerowanej topografii dla mapy 300x300. Wykres po lewej przedstawia nieprzetworzone dane, a po prawej wyniki mapowania topografii funkcją (Rysunek 6.5) korygującą wysokość.

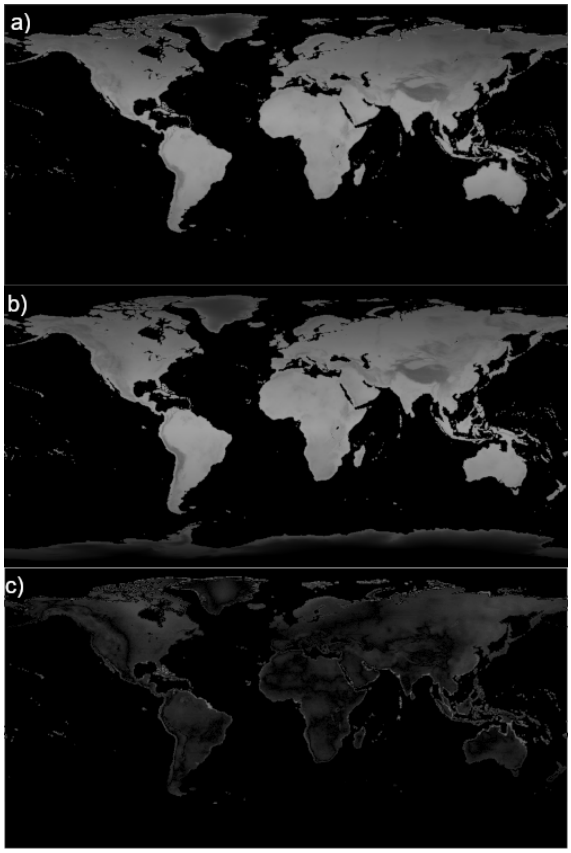


Rysunek 6.7. Histogram topografii Ziemi ograniczony do skorupy kontynentalnej i terenów powyżej poziomu morza. Źródło: [131].

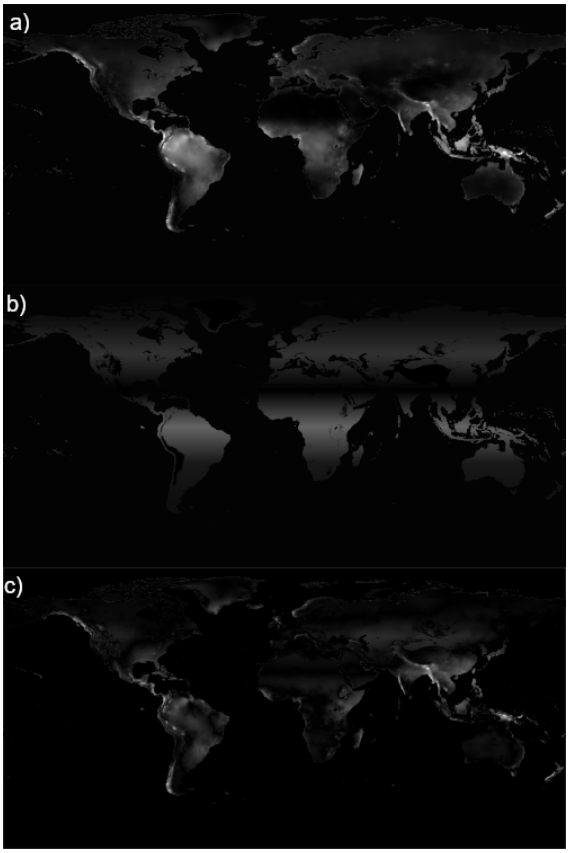
poprzez próbę odtworzenia rzeczywistych danych ze wspomnianego zbioru CRU CL. Warstwy kontynentów i topografii reprezentujących Ziemię zostały uprzednio wczytane przez generator, aby wykorzystać wiedzę o wysokości przy obliczaniu gradientu temperatury i opadów (Podrozdział 3.1). Parametry generatora zostały dobrane ręcznie na podstawie obserwacji i posiadanej wiedzy geograficznej i nie są optymalne. Rysunek 6.8 i Rysunek 6.9 przedstawiają wizualizacje warstw rzeczywistych danych, otrzymanych danych i obliczonego błędu (różnicy). Rysunek 6.10 i Rysunek 6.11 przedstawiają wyniki w formie histogramu błędu względnego. Tylko punkty znajdujące się na lądzie zostały wzięte pod uwagę.

W przypadku temperatury ponad 30% punktów na mapie ma bezbłędnie dopasowaną wartość, 90% punktów ma błąd względny na poziomie 5% lub niżej, a dla 98% punktów błąd

wynosi do 10%. Wyniki opadów są dostrzegalnie gorsze, ponieważ wprawdzie 30% punktów ma poprawną wartość, ale 90% punktów ma błąd do 10%, 98% aż do 15%, a nieliczne punkty mają kompletnie niepoprawną wartość. W obu przypadkach artefakty widoczne na linii brzegowej wiążą się z niedokładną reprezentacją konturu kontynentów wynikającą ze zastosowanego skalowania źródłowej mapy Ziemi, a nie algorytmów generatora. Warto też dodać, że referencyjny zbiór danych też nie jest w 100% poprawny, ponieważ dla dużej liczby punktów nie było dostępnych danych pomiarowych i wartości musiały być otrzymane w wyniku interpolacji [130].

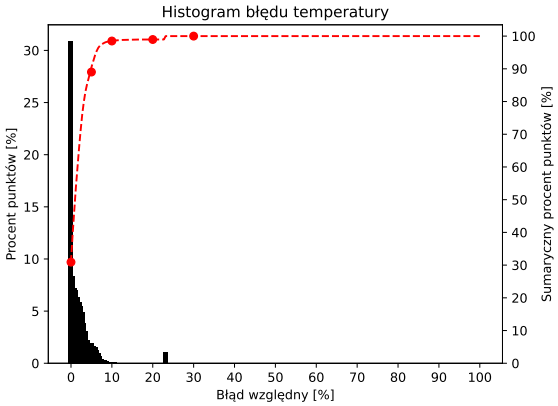


Rysunek 6.8. (a) Rzeczywiste i (b) wygenerowane dane temperatury oraz (c) bezwzględna różnica wartości. Obraz różnicy ma wielokrotnie zwiększoną ekspozycję dla czytelności.

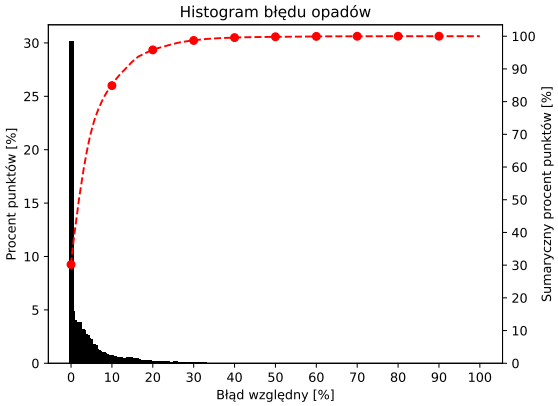


Rysunek 6.9. (a) Rzeczywiste i (b) wygenerowane dane opadów oraz (c) bezwzględna różnica wartości. Obraz różnicy tym razem *nie* wymaga zwiększonej ekspozycji.

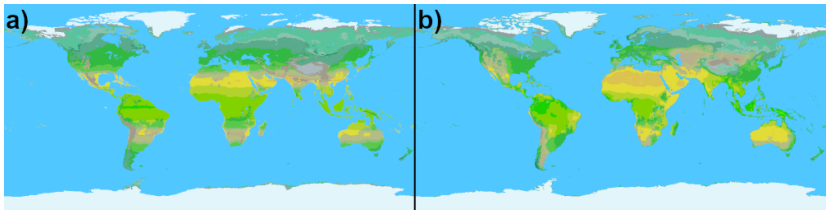
O ile w przypadku temperatury zastosowane metody generacji dają akceptowalne wyniki pod względem realizmu, tak w przypadku opadów atmosferycznych zdecydowanie widać, że są mało precyzyjnym przybliżeniem, jeżeli nie stosuje się kosztownej symulacji czynników meteorologicznych. Ponieważ jednak głównym celem programu jest stworzenie fikcyjnych światów, a nie 100% odtwarzanie danych na Ziemi, naturalnie wyglądające dane klimatyczne można łatwo osiągnąć poprzez zwiększenie roli szumu w procesie. Rysunek 6.12 pokazuje przykład podziału na biomy dla wygenerowanego i rzeczywistego klimatu, na którym można zauważyć, że pomimo regionalnych niezgodności (np. Maroko, Argentyna, Kazachstan) całokształt klimatu zachowuje pewne podobieństwo.



Rysunek 6.10. Histogram błędu temperatury.

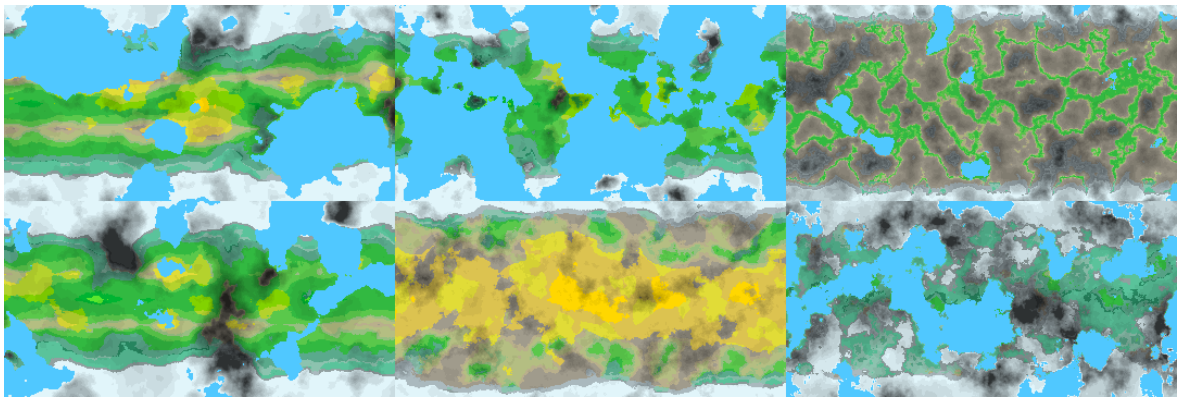


Rysunek 6.11. Histogram błędu opadów.



Rysunek 6.12. (a) Wygenerowany i (b) rzeczywisty klimat Ziemi.

Jako dodatek do tej sekcji, Rysunek 6.13 przedstawia przykładowe stworzone fikcyjne światy, pokazujące elastyczność generatora i różnorodność możliwych konfiguracji.



Rysunek 6.13. Wygenerowane fikcyjne światy.

6.3. Analiza wyników symulatora

Ze względu na brak dostępnych danych rzeczywistych, testy symulatora zostały przeprowadzone inaczej. Przygotowano specjalne małe scenariusze, podczas których badane są wybrane systemy lub modelowane zjawiska. Prezentacja aspektów symulacji odbywa się na podstawie omówienia zebranych danych statystycznych danego scenariusza. Poniżej przedstawiono sześć z wybranych scenariuszy.

Scenariusz 1: Rywalizacja (Rysunek 6.14). Dwa państwa zaczynają w takich samych warunkach. Dzięki bardziej agresywnej kolonizacji, państwo R ma pod koniec większą liczbę

ludności i zasobów, dlatego stać je na osiągnięcie wyższego poziomu rozwoju naukowego i kulturowego, niż państwo B. Widoczne wahania populacji pod koniec wynikają z częstych zmian priorytetyzowanych tradycji (które mają wpływ na wydajność produkcji zasobów) i wojen.

Scenariusz 2: Populacja (Rysunek 6.15). Liczba ludności rośnie, dopóki jest w stanie zwiększyć produkcję zasobów. Przed rokiem 1500 widać, że wzrost zaczął gwałtownie przyspieszać z powodu rozwoju naukowego, ale został zatrzymany przez ograniczoną liczbę zasobów dostępną na mapie. Zdrowotność przez większość czasu oscyluje wokół stałej wartości, ponieważ zwiększenie poziomu służby zdrowia jest negowane przez wzrost populacji; dopiero rozwój medycyny pozwala uciec z tego cyklu.

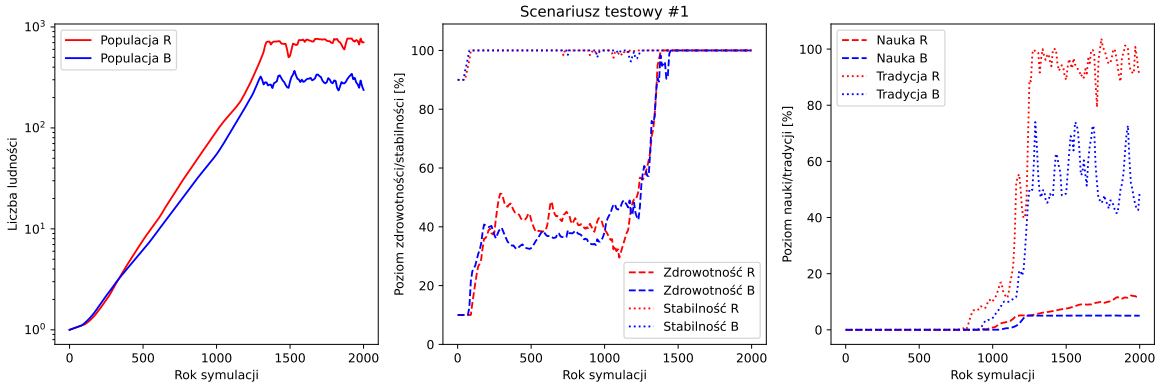
Scenariusz 3: Industrializacja (Rysunek 6.16). Początkowo zdecydowana większość ludności pracuje na utrzymanie społeczeństwa. Wraz z rozwojem ekonomii produkcja zasobów staje się wydajniejsza, dzięki czemu więcej ludzi może pracować w innych sektorach i przyczyniać się do dalszego rozwoju. Ok. roku 1350 można zauważyć, że rolnictwo przestaje być przewodnią gałęzią ekonomii.

Scenariusz 4: Rola klimatu (Rysunek 6.17). Cztery państwa startowe, każde z których zajmuje teren z inną grupą biomów. Biomy wpływają na ilość złóż zasobów i koszt kolonizacji, co przekłada się na tempo wzrostu liczby ludności. Państwo w strefie umiarkowanej jest najbardziej zaludnione przez cały czas, ale i tak okazjonalnie przegrywa konflikty, co widać w postaci krótkich nagłych spadków. W przypadku państw w innych klimatach populacja wygląda przez większość czasu podobnie, ze zmiennymi okresami dominacji w zależności od wygranych wojen i otrzymanych trybutów.

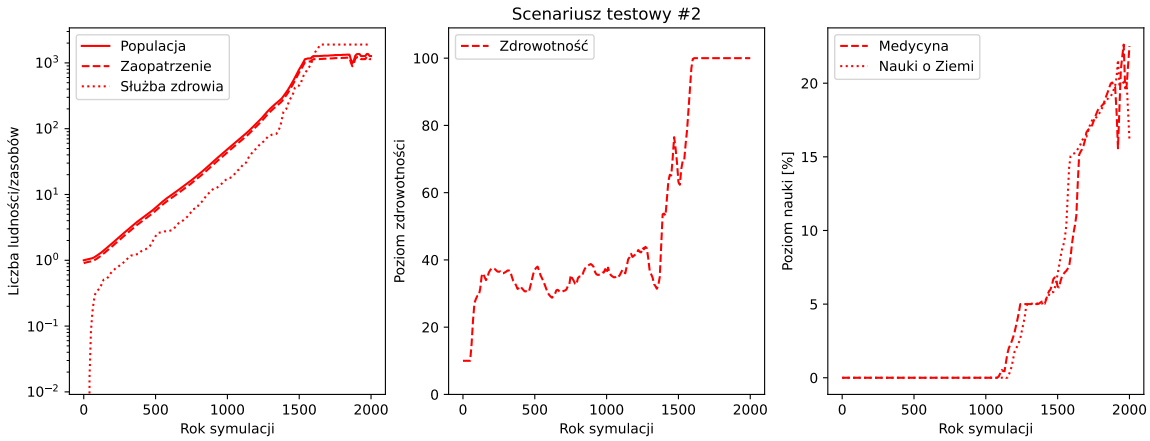
Scenariusz 5: Rozwój nauki (Rysunek 6.18). Rozwój nauki jest znacznie bardziej kosztowny i wolny od rozwoju ekonomicznego, ale wynikają z niego większe bonusy do produkcji. Nauka praktycznie zaczyna się rozwijać po roku 1100, kiedy to zaczyna być produkowane wystarczająco dużo punktów nauki, żeby „przebić" się przez barierę regresji. Poziom nauki rośnie w przybliżeniu liniowo dopóki produkcja punktów rośnie i matematyka (dziedzina odpowiedzialna za bonus do nauki) jest odpowiednio rozwijana – w przeciwnym wypadku rosnący z każdym poziomem koszt i regresja będą spowalniać lub zatrzymywać rozwój.

Scenariusz 6: Rozwój kultury (Rysunek 6.19). Rozwój kultury (tradycji) jest podobny do nauki, choć szybciej osiągany i bardziej wrażliwy na zmiany strategii państwa.

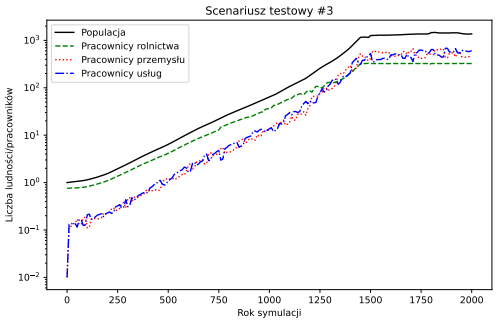
Dodatkowo załączono zrzuty ekranu wykonywane co 500 lat podczas przykładowych symulacji na mapie świata dla 100 państw (Rysunek 6.20) i Europy (Rysunek 6.21) dla 50 państw. Dla domyślnych parametrów programu symulacja może być podzielona na epoki charakteryzujące się różnym etapem i tempem rozwoju. Do 500 roku trwa powolny start – państwa głównie zwiększają populację i lokalne terytorium. Pomiędzy rokiem 500 i 1000 liczba ludności jest już na tyle wysoka, że państwa mogą przeznaczyć znaczną ilość zasobów na ekspansję i urbanizację, a w 1000 są już na tyle rozbudowane przemysłowo, że są w stanie szybko kolonizować otoczenie i jednoczyć sąsiadów. W ciągu kilkuset kolejnych lat rozwój przmysłowy, kulturowy i naukowy zaczyna na tyle przyspieszać, że cały ląd jest już zajęty, a najsilniejsze państwa zajmują kontynenty. Od tego czasu granice i populacja



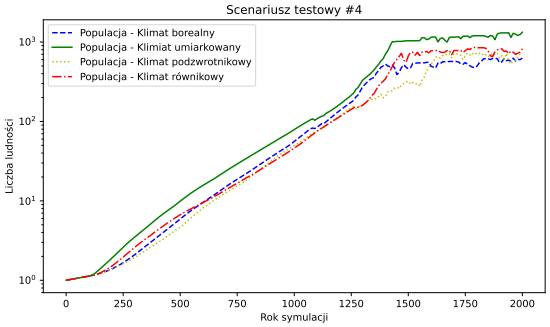
Rysunek 6.14. Scenariusz testowy 1: Rywalizacja.



Rysunek 6.15. Scenariusz testowy 2: Populacja.

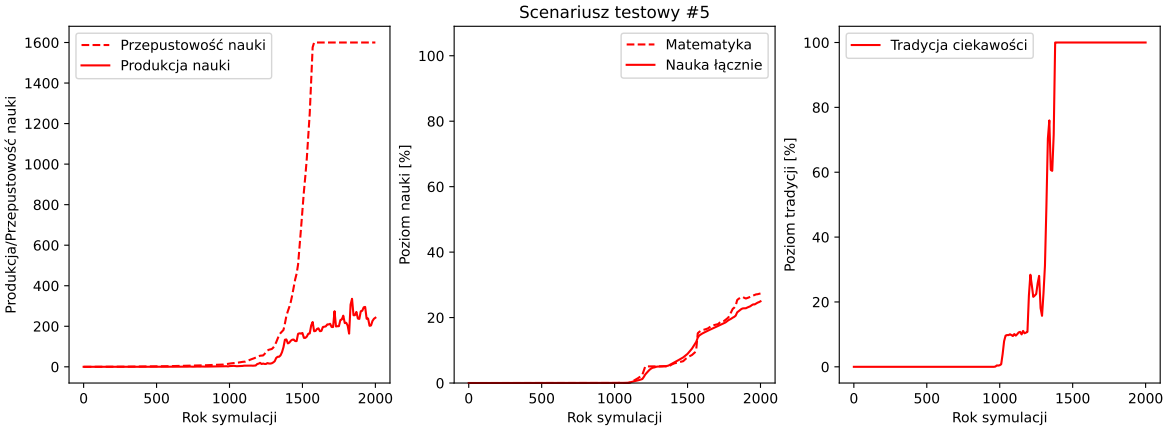


Rysunek 6.16. Scenariusz testowy 3: Industrializacja.

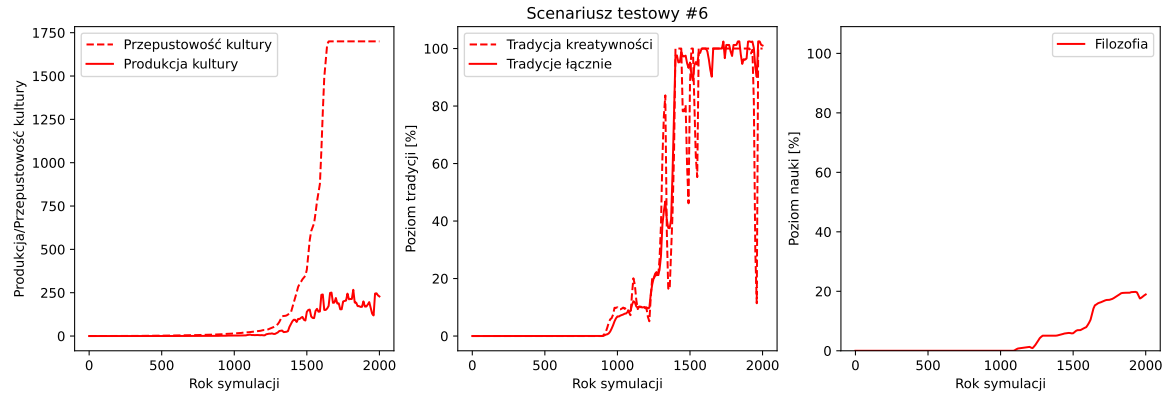


Rysunek 6.17. Scenariusz testowy 4: Rola klimatu.

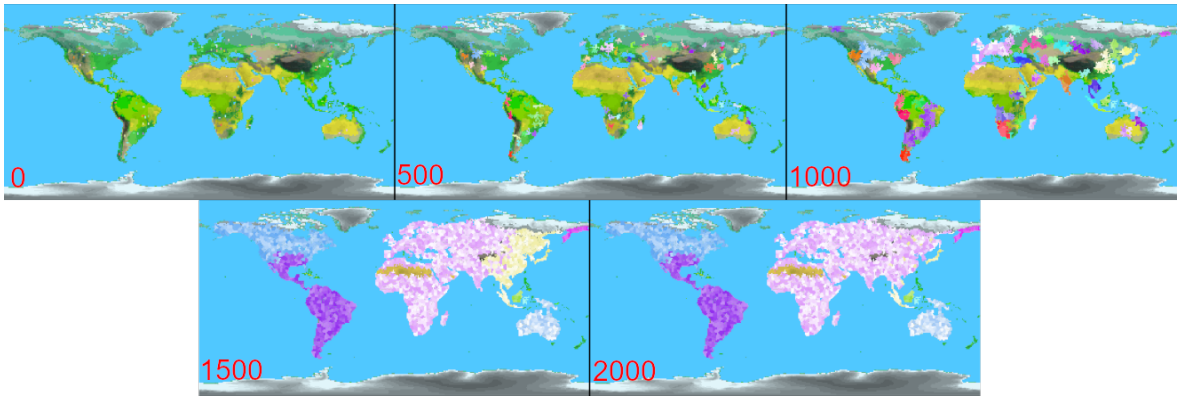
zaczynają się stabilizować, następuje stagnacja powodowana ograniczoną liczbą złóż zasobów, terytorium do budowy miast i porównywalną siłą militarną pozostałych państw. Okazjonalny wzrost spowodowany jest osiągnięciem nowego poziomu rozwoju naukowego lub częstym wielkim wydarzeniem kulturowym.



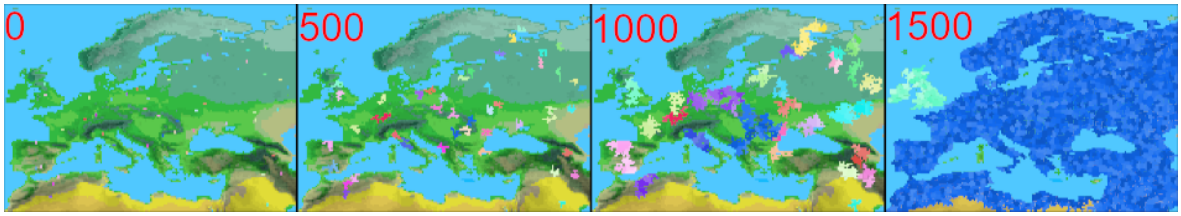
Rysunek 6.18. Scenariusz testowy 5: Rozwój nauki.



Rysunek 6.19. Scenariusz testowy 6: Rozwój kultury.



Rysunek 6.20. Symulacja na mapie świata, zrzut ekranu co 500 lat.



Rysunek 6.21. Symulacja na mapie Europy, zrzut ekranu co 500 lat.

6.4. Dalszy rozwój

Zarówno generator, jak i symulator mogą być w łatwy sposób rozszerzone dzięki podziale modelu świata na warstwy i modelu historii na systemy.

Większa selekcja szumów Dodanie kolejnych, bardziej specjalistycznych algorytmów szumu losowego takich jak *ridged noise* czy *billow noise* i wprowadzenie możliwości mieszania szumów razem pozwoliłoby użytkownikom tworzyć różne kształty i wzory, które obecnie są niemożliwe.

Symulacja prądów Obecnie najsłabszym punktem generatora jest warstwa opadów. Model świata mógłby być rozszerzony o warstwę prądów powietrznych i morskich, symulowanych np. polem losowanych wektorów, które wpływałyby na lokalną siłę opadów i temperaturę.

Kolonizacja morska Zaprezentowana wersja modelu historii zupełnie ignoruje morza i oceany. Jeżeli jakiś wyspa jest niezamieszkała na początku symulacji, to żadne państwo jej nie zajmie, ponieważ nie ma dostępu lądowego. Dodanie jakiegoś systemu żeglugi pozwoliłoby na zajmowanie odległych miejsc lub przeprowadzanie inwazji morskich.

Upadek państw Cykl wzrostu, szczytu potęgi i upadku towarzyszy imperiom od początku ludzkości. Zbyt rozrośnięte symulowane państwa powinny albo być jakoś osłabiane, albo rozpadać się na nowe państwa. Jest to potencjalnie najtrudniejsze z proponowanych usprawnień, ponieważ wymagałoby znacznego rozszerzenia obecnego współczynnika stabilności, zdrowotności i buntu oraz stworzenia algorytmu zdolnego spójnie wydzielić regiony tworzące nowe państwa od tych pozostających w starym państwie.

Wydajność Na podstawie testów wydajności oba programy uznano za dostatecznie szybkie, ale mogą być nadal usprawniane. Generowanie poszczególnej warstwy obecnie odbywa się jednowątkowo – wykorzystanie wielu wątków z podziałem mapy na fragmenty może przyspieszyć ten proces. Wybrane algorytmy symulatora, szczególnie te najbardziej obciążające program, mogą być poddane profilowaniu i optymalizacji. Można też rozważyć tryb „bezgłowy” (ang. *headless*), w którym program przeprowadzałby symulacje bez graficznego interfejsu użytkownika i tylko zapisywał statystyki, co pozwoliłoby znacznie zwiększyć szybkość symulacji.

7. Podsumowanie

W ramach tej pracy poddano analizie tematykę generowania mapy świata i symulowania z jej wykorzystaniem historii rozwoju ludzkości, zaproponowano własny model teoretyczny, dokonano jego implementacji i przeprowadzono badania wyników wydajności i poprawności.

Na podstawie przeglądu literatury opisano sposób działania, wady i zalety wykorzystywanych metod generacji terenu i klimatu. Przybliżone zostały problemy związane z realistycznym symulowaniem tematu tak szerokiego, jak historia i z jakich dziedzin nauki wykorzystujących symulacje można czerpać. Analizę wieńczy porównanie istniejących programów związanych z symulacją historii.

Zaproponowany model teoretyczny obejmuje wybrane reprezentowane aspekty świata i historii: teren, klimat, zasoby, populację, rozwój naukowy, kulturowy, ekonomiczny, stosunki międzynarodowe i konflikty zbrojne. Opisane są rodzaje potrzebnych informacji, zależności między nimi, proponowane algorytmy, funkcje i parametry.

Model zaimplementowano zgodnie z postawionymi wymaganiami w formie dwóch aplikacji: generatora i symulatora. Przedstawiono wybrane technologie, zarys całego projektu, opis wykorzystanej architektury *entity-component-system* w porównaniu z innymi architekturami silników gier oraz szczegóły i możliwości działania generatora, symulatora i wspólnego interfejsu użytkownika.

Implementacja generatora i symulatora zostały poddane badaniom wydajności i poprawności. Na podstawie wyników badań stwierdzono, że programy spełniły postawione wymogi odnośnie szybkości działania, jakości generowanych map i spójności symulacji. Zweryfikowano przyjęte założenia oraz określono najbliższe perspektywy dalszego rozwoju.

O ile tematyka generowania terenu jest dobrze zbadana w dostępnej literaturze, tak w przypadku metod symulacji historii praktycznie nie ma prac na ten temat, chociaż wskazuje się na rozrywkową, kreatywną i edukacyjną wartość takiego oprogramowania. Praca ta jest więc eksploracją tematu od strony praktycznej i próbą opracowania podejścia do tworzenia takiego typu symulatorów.

Bibliografia

[1] K. Squire, „Replaying History: Learning World History through playing Civilization III", praca dokt. Indiana University, sty. 2003.

[2] W. Uricchio, „Simulation, History, And Computer Games", w *Handbook of Computer Game Studies*, MIT Press, 2004, s. 327–338.

[3] E. Galin, É. Guérin, A. Peytavie i in., „A Review of Digital Terrain Modeling," *Computer Graphics Forum*, t. 38, 2019.

[4] M. W. Kapell i A. B. Elliott, red., „Playing with the past:", Bloomsbury Publishing, 2013, ISBN: 9781623567286.

[5] N. Lercari, „Simulating History in Virtual Worlds," w *Handbook on 3D3C Platforms: Applications and Tools for Three Dimensional Systems for Community, Creation and Commerce*, Y. Sivan, red. Springer International Publishing, 2016, s. 337–352, ISBN: 978-3-319-22041-3. DOI: 10.1007/978-3-319-22041-3_13.

[6] K. Weir i M. Baranowski, „Simulating History to Understand International Politics," *Simulation & Gaming*, t. 42, nr. 4, s. 441–461, 2011. DOI: 10.1177/1046878108325442.

[7] G. Smith, „An Analog History of Procedural Content Generation", w *International Conference on Foundations of Digital Games*, 2015.

[8] J. Freiknecht, „Procedural content generation for games", oprogramowanie, Dostęp zdalny (19.08.2024): https://madoc.bib.uni-mannheim.de/59000/, praca dokt. Uniwersytet w Mannheim, 2021.

[9] M. Blatz i O. Korn, „A Very Short History of Dynamic and Procedural Content Generation," *Game Dynamics: Best Practices in Procedural and Dynamic Game Content Generation*, s. 1–13, kw. 2017. DOI: 10.1007/978-3-319-53088-8_1.

[10] P. Prusinkiewicz i M. Hammel, „A Fractal Model of Mountains with Rivers," *Proceedings of Graphics Interface '93*, t. 30, s. 174–180, sty. 2002.

[11] X. Mei, P. Decaudin i B.-G. Hu, „Fast Hydraulic Erosion Simulation and Visualization on GPU", w *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, 2007, s. 47–56. DOI: 10.1109/PG.2007.15.

[12] K. Warszawski i S. Nikiel, „A proposition of erosion algorithm for terrain models with hardness layer," *Applied Computer Science*, t. 8, s. 76–84, 2014.

[13] I. Antoniuk i P. Rokita, „Procedural Generation of Underground Systems with Terrain Features using Schematic Maps and L-systems," *Challenges of Modern Technology*, t. 7, s. 8–15, wrz. 2016. DOI: 10.5604/01.3001.0009.5443.

[14] B. Benes i T. Roa, „Simulating Desert Scenery", w *International Conference in Central Europe on Computer Graphics and Visualization*, 2004.

[15] A. Paris, A. Peytavie, E. Guérin, O. Argudo i E. Galin, „Desertscape Simulation," *Computer Graphics Forum*, t. 38, nr. 7, s. 47–55, list. 2020. DOI: 10.1111/cgf.13815.

[16] Y. Cortial, A. Peytavie, E. Galin i E. Guérin, „Procedural Tectonic Planets," *Computer Graphics Forum*, t. 38, nr. 2, s. 1–11, 2019. DOI: 10.1111/cgf.13614.

[17] R. B. D. d'Oliveira i A. L. A. J. au2, „Procedural Planetary Multi-resolution Terrain Generation for Games", preprint, Dostęp zdalny (19.08.2024): https://arxiv.org/abs/1803.04612, 2018.

[18] D. Fenna, „Cartographic Science: A Compendium of Map Projections, with Derivations", Taylor & Francis, 2006, ISBN: 9780849381690.

[19] Q. Zhou, „Digital Elevation Model and Digital Surface Model," w *International Encyclopedia of Geography*. John Wiley & Sons, Ltd, 2017, ISBN: 9781118786352. DOI: 10.1002/9781118786352.wbieg0768.

[20] T. Rose i A. Bakaoukas, „Algorithms and Approaches for Procedural Terrain Generation - A Brief Review of Current Techniques", w *2016 8th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games)*, wrz. 2016. DOI: 10.1109/VS-GAMES.2016.7590336.

[21] O. Roosvall, „Procedural Terrain Generation Using Ray Marching", Dostęp zdalny (19.08.2024): https://www.diva-portal.org/smash/get/diva2:1059684/FULLTEXT01.pdf, praca inż. Blekinge Institute of Technology, 2022.

[22] K. Perlin, „An Image Synthesizer", w *ACM SIGGRAPH Computer Graphics*, t. 19, 1985, s. 287–296. DOI: 10.1145/325165.325247.

[23] K. Perlin, „Noise hardware", w *Real-Time Shading SIGGRAPH Course Notes*, M. Olano, red., 2001, rozd. 2.

[24] K. Spencer, „OpenSimplex 2", oprogramowanie, Dostęp zdalny (19.08.2024): https://github.com/KdotJPG/OpenSimplex2, 2019.

[25] M. Keshner, „1/f noise," *Proceedings of the IEEE*, t. 70, nr. 3, s. 212–218, 1982. DOI: 10.1109/PROC.1982.12282.

[26] B. B. Mandelbrot i J. R. Wallis, „Computer Experiments with Fractional Gaussian Noises: Part 3, Mathematical Appendix," *Water Resources Research*, t. 5, nr. 1, s. 260–267, 1969. DOI: 10.1029/WR005i001p00260.

[27] G. C. Backes i T. A. Engel, „Real-Time Massive Terrain Generation using Procedural Erosion on the GPU", w *Proceedings of SBGames*, Dostęp zdalny (19.08.2024): https://www.sbgames.org/sbgames2018/files/papers/ComputacaoShort/188264.pdf, 2018, s. 675–678.

[28] L. Polidori, J. Chorowicz, R. Guillande i in., „Description of terrain as a fractal surface, and application to digital elevation model quality assessment," *Photogrammetric Engineering and Remote Sensing*, t. 57, nr. 10, s. 1329–1332, 1991.

[29] A. Fournier, D. Fussell i L. Carpenter, „Computer rendering of stochastic models," *Communications of the ACM*, t. 25, nr. 6, s. 371–384, czer. 1982, ISSN: 00010782. DOI: 10.1145/358523.358553.

[30] J. R. Rankin, „The Uplift Model Terrain Generator," *International Journal of Computer Graphics & Animation*, t. 5, nr. 2, kw. 2015, ISSN: 22313281.

[31] B. Jákó i B. Tóth, „Fast Hydraulic and Thermal Erosion on GPU", w *Eurographics 2011 - Short Papers*, N. Avis i S. Lefebvre, red., The Eurographics Association, 2011. DOI: /10.2312/EG2011/short/057-060.

[32] N. McDonald, „Clustered Convection for Procedural Plate Tectonics", Dostęp zdalny (19.08.2024): https://nickmcd.me/2020/12/03/clustered-convection-for-simulating-plate-tectonics/, 2020.

[33] M. Alexa, „Super-Fibonacci Spirals: Fast, Low-Discrepancy Sampling of SO(3)", w *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, s. 8281–8290. DOI: 10.1109/CVPR52688.2022.00811.

[34] R. L. Cook, „Stochastic sampling in computer graphics," *ACM Transactions on Graphics*, t. 5, nr. 1, s. 51–72, sty. 1986, ISSN: 07300301. DOI: 10.1145/7529.8927.

[35] D. P. Mitchell, „Spectrally optimal sampling for distribution ray tracing," *ACM SIGGRAPH Computer Graphics*, t. 25, nr. 4, s. 157–164, lip. 1991, ISSN: 00978930. DOI: 10.1145/127719.122736.

[36] H. Li, D. Nehab, L.-Y. Wei, P. V. Sander i C.-W. Fu, „Fast capacity constrained Voronoi tessellation", w *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, Association for Computing Machinery, 2010, ISBN: 9781605589398. DOI: 10.1145/1730804.1730985.

[37] R. Huggett, „Fundamentals of Geomorphology", Taylor & Francis, 2011, s. 3, ISBN: 9780203860083.

[38] R. M. Palin i M. Santosh, „Plate tectonics: What, where, why, and when?" *Gondwana Research*, t. 100, s. 3–24, 2021, ISSN: 1342-937X. DOI: 10.1016/j.gr.2020.11.001.

[39] E. Panagiotou i E. Charou, „Procedural 3D Terrain Generation using Generative Adversarial Networks", w *SETN 2020: 11th Hellenic Conference on Artificial Intelligence*, Association for Computing Machinery, 2020, ISBN: 9781450388788. DOI: 10.48550/arXiv.2010.06411.

[40] G. Voulgaris, I. Mademlis i I. Pitas, „Procedural Terrain Generation Using Generative Adversarial Networks", w *2021 29th European Signal Processing Conference (EUSIPCO)*, 2021, s. 686–690. DOI: 10.23919/EUSIPCO54536.2021.9616151.

[41] Z. Hu, K. Hu, C. Mo, L. Pan i Z. Wang, „Terrain Diffusion Network: Climatic-Aware Terrain Generation with Geological Sketch Guidance", preprint, Dostęp zdalny (19.08.2024): https://arxiv.org/abs/2308.16725, sierp. 2023. DOI: 10.48550/arXiv.2308.16725.

[42] P. Walsh i P. Gade, „Terrain generation using an Interactive Genetic Algorithm", w *IEEE Congress on Evolutionary Computation*, 2010, s. 1–7. DOI: 10.1109/CEC.2010.5585913.

[43] J. Togelius, G. N. Yannakakis, K. O. Stanley i C. Browne, „Search-Based Procedural Content Generation: A Taxonomy and Survey," *IEEE Transactions on Computational Intelligence and AI in Games*, t. 3, nr. 3, s. 172–186, 2011. DOI: 10.1109/TCIAIG.2011.2148116.

[44] J. Doran i I. Parberry, „Controlled Procedural Terrain Generation Using Software Agents," *IEEE Transactions on Computational Intelligence and AI in Games*, t. 2, nr. 2, s. 111–119, 2010. DOI: 10.1109/TCIAIG.2010.2049020.

[45] V. Rull, „Quaternary Ecology, Evolution, and Biogeography", Elsevier Science & Technology, 2020, s. 67, ISBN: 9780128204733.

[46] L. Holdridge, „Life Zone Ecology", Tropical Science Center, 1964.

[47] R. H. Whittaker, „Communities and Ecosystems", Macmillan, 1975, ISBN: 9780024273901.

[48] S. Breckle, „Walter's Vegetation of the Earth", 4 wyd. Springer, 2002, ISBN: 9783540433156.

[49] R. G. Allen, L. S. Pereira, D. Raes, M. Smith i in., „Crop evapotranspiration - Guidelines for computing crop water requirements - FAO Irrigation and drainage paper 56", Food i Agriculture Organization of the United Nations, 1998, t. 300, rozd. 2, ISBN: 9251042195.

[50] C. W. Thornthwaite, „An Approach toward a Rational Classification of Climate," *Geographical Review*, t. 38, nr. 1, s. 55–94, 1948, ISSN: 00167428, 19310846. DOI: 10.2307/210739.

[51] C. H. B. Priestley i R. J. Taylor, „On the assessment of surface heat flux and evaporation using large-scale parameters," *Monthly weather review*, t. 100, nr. 2, s. 81–92, 1972. DOI: 10.1175/1520-0493(1972)100<0081:OTAOSH>2.3.CO;2.

[52] R. Srikanthan i T. A. McMahon, „Stochastic generation of annual, monthly and daily climate data: A review," *Hydrology and Earth System Sciences*, t. 5, nr. 4, s. 653–670, 2001. DOI: 10.5194/hess-5-653-2001.

[53] M. Łatuszyńska, „Symulacja komputerowa w ekonomii eksperymentalnej," *Zeszyty Naukowe Uniwersytetu Szczecińskiego. Studia Informatica*, t. 963, nr. 36, s. 5–18, 2015.

[54] T. Wiśniewski i A. Matuszczak, „Wykorzystanie symulacji do modelowania łańcucha dostaw," *Gospodarka Materiałowa i Logistyka*, t. 1, s. 2–10, 2017.

[55] I. Fechner i S. Krzyżaniak, „Symulacja szybkiej reakcji w łańcuchach dostaw na dynamiczne zmiany popytu," *Logistyka*, t. 4, s. 1807–1816, 2014.

[56] P. Sabin, „Lost Battles: Reconstructing the Great Clashes of the Ancient World", Hambledon Continuum, 2007, ISBN: 9781847251879.

[57] P. Sabin, „Simulating War: Studying Conflict Through Simulation Games", Continuum International Publishing, 2012, ISBN: 9781441185587.

[58] M. Łatuszyńska, „Symulacja komputerowa zamiast tradycyjnego eksperymentu ekonomicznego," *Studia i Prace WNEiZ*, t. 44, s. 33–48, sty. 2016.

[59] A. Munslow, „What history is," *History in Focus, Issue 2: What is History?*, paź. 2001, Dostęp zdalny (19.08.2024): https://archives.history.ac.uk/history-in-focus/Whatishistory/munslow6.html.

[60] P. Zwierzyński, „Program symulacyjny jako narzędzie projektowania łańcuchów dostaw", w *V Warsztaty Naukowe dla Doktorantów w Dyscyplinie Inżynieria Produkcji, Podlesice, 29-30 czerwca 2017 : Praca zbiorowa*, Politechnika Częstochowska Wydział Inżynierii Produkcji i Technologii Materiałów, sty. 2018, s. 148–155, ISBN: 9788363989569.

[61] Z. Świątnicki i P. Podgórny, „Gry wojenne jako narzędzie kształtowania bezpieczeństwa państwa. Analiza gier wojennych I i II generacji," *Rocznik Bezpieczeństwa Morskiego*, t. 25, s. 1–15, 2021.

[62] B. Connable, M. McNerney, W. Marcellino i in., „Will to Fight: Analyzing, Modeling, and Simulating the Will to Fight of Military Units", RAND Corporation, 2018, ISBN: 9781977400444.

[63] S. Wrigge, A. Fransén i L. Wigg, „The Lanchester Theory of Combat and Some Related Subjects. A Bibliography 1900-1993", Swedish National Defence Research Institute (FOA), Dostęp zdalny (19.08.2024): https://apps.dtic.mil/sti/tr/pdf/ADA302237.pdf, wrz. 1995.

[64] J. Fletcher, J. Apkarian, A. Roberts, K. Lawrence, C. Chase-Dunn i R. Hanneman, „War Games: Simulating Collins' Theory of Battle Victory," *Cliodynamics*, t. 2, nr. 2, s. 252–275, 2011.

[65] J. Setear i R. Corporation, „Simulating the Fog of War", (P (Rand Corporation)). RAND Corporation, 1989.

[66] O. A. Guerrero, D. Guariso i G. Castañeda, „Aid effectiveness in sustainable development: A multidimensional approach," *World Development*, t. 168, 2023.

[67] T. Adams, „Dwarf Fortress", Bay 12 Games, oprogramowanie, Dostęp zdalny (19.08.2024): https://www.bay12games.com/dwarves/, 2006.

[68] NFB, „Fantasy Worlds History Simulator", oprogramowanie, Dostęp zdalny (19.08.2024): https://www.youtube.com/@FWHSimulator, 2020.

[69] J. Pena, „Worlds: History Simulator", oprogramowanie, Dostęp zdalny (19.08.2024): https://drtardigrade.itch.io/worldhistorysim, 2020.

[70] R. Babij, „WorldSim", oprogramowanie, Dostęp zdalny (19.08.2024): https://github.com/RyanBabij/WorldSim, 2018.

[71] M. Karpenko, „WorldBox", oprogramowanie, 2012.

[72] Maxis, „SimEarth: The Living Planet", Maxis, oprogramowanie, 1990.

[73] Firaxis Games, „Sid Meier's Civilisation VI", 2K Games, oprogramowanie, 2016.

[74] Paradox Development Studio, „Europa Universalis IV", Paradox Interactive, oprogramowanie, 2013.

[75] C. Whiteman, „Mountain Meteorology: Fundamentals and Applications", Oxford University Press, 2000, ISBN: 9780195132717. https://books.google.pl/books?id=kZ7mCwAAQBAJ.

[76] J. Feiccabrino, W. Graff, A. Lundberg, N. Sandström i D. Gustafsson, „Meteorological Knowledge Useful for the Improvement of Snow Rain Separation in Surface Based Models," *Hydrology*, t. 2, s. 266–288, list. 2015. DOI: 10.3390/hydrology2040266.

[77] M. Córdova, R. Célleri, C. J. Shellito, J. Orellana-Alvear, A. Abril i G. Carrillo-Rojas, „Near-Surface Air Temperature Lapse Rate Over Complex Terrain in the Southern Ecuadorian Andes: Implications for Temperature Mapping," *Arctic, Antarctic, and Alpine Research*, t. 48, nr. 4, s. 673–684, 2016. DOI: 10.1657/AAAR0015-077.

[78] R. Chen, C. Han, J. Liu i in., „Maximum precipitation altitude on the northern flank of the Qilian Mountains, northwest China," *Hydrology Research*, t. 49, nr. 5, s. 1696–1710, lut. 2018, ISSN: 0029-1277. DOI: 10.2166/nh.2018.121.

[79] A. Napoli, A. Crespi, F. Ragone, M. Maugeri i C. Pasquero, „Variability of orographic enhancement of precipitation in the Alpine region," *Scientific reports*, t. 9, nr. 1, s. 13 352, wrz. 2019. DOI: 10.1038/s41598-019-49974-5.

[80] N. Wang, J. He, X. Jiang i in., „Study on the zone of maximum precipitation in the north slopes of the central Qilian Mountains," *Journal of Glaciology and Geocryology*, t. 31, s. 395–403, sty. 2009.

[81] N. D. Matsakis i F. S. Klock II, „The rust language", w *ACM SIGAda Ada Letters*, ACM, t. 34, 2014, s. 103–104.

[82] The Rust Programming Language, „Rust Programming Language", Dostęp zdalny (19.08.2024): https://www.rust-lang.org, 2019.

[83] I. Gouy, „The Computer Language Benchmarks Game", Dostęp zdalny (19.08.2024): https://benchmarksgame-team.pages.debian.net/benchmarksgame, 2024.

[84] Y. Zhang, Y. Zhang, G. Portokalidis i J. Xu, „Towards Understanding the Runtime Performance of Rust", w *ASE '22: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, Association for Computing Machinery, 2022, ISBN: 9781450394758. DOI: 10.1145/3551349.3559494f.

[85] hanabi1224, „Programming Language and compiler Benchmarks", Dostęp zdalny (19.08.2024): https://programming-language-benchmarks.vercel.app, 2024.

[86] T. S. McNamara, „Rust in Action : Systems Programming Concepts and Techniques", Shelter Island, New York: Manning, 2021.

[87] S. Klabnik i C. Nichols, „The Rust Programming Language, 2nd Edition," w No Starch Press, 2023, ISBN: 9781718503106.

[88] A. Turon, „Abstraction without overhead: traits in Rust", Dostęp zdalny (19.08.2024): https://blog.rust-lang.org/2015/05/11/traits.html, 2015.

[89] A. Crichton, S. Klabnik, C. Nichols i in., „The Cargo Book", Dostęp zdalny (19.08.2024): https://doc.rust-lang.org/cargo/.

[90] C. Lattner i V. Adve, „LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation", w *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*, ser. CGO '04, Palo Alto, California: IEEE Computer Society, 2004, s. 75–88, ISBN: 0769521029.

[91] The Rust Project Developers, „The rustc book", Dostęp zdalny (19.08.2024): https://doc.rust-lang.org/rustc/.

[92] Rust on Embedded Devices Working Group i in., „The Embedded Rust Book", Dostęp zdalny (19.08.2024): https://docs.rust-embedded.org/book/.

[93] „WebAssembly Core Specification", W3C, wer. 1.0, grud. 2019, Dostęp zdalny (19.08.2024): https://www.w3.org/TR/wasm-core-1/.

[94] Bevy Contributors, „Bevy Engine", wer. 0.13.0, oprogramowanie, Dostęp zdalny (19.08.2024): https://github.com/bevyengine/bevy, lut. 2024.

[95] M. Prensky, „"Simulations": Are They Games?" W *Digital Game-Based Learning*, McGraw-Hill, 2001, rozd. 8, ISBN: 0071363440.

[96] V. Narayanasamy, K. Wong, C. Fung i S. Rai, „Distinguishing games and simulation games from simulators," *Computers in Entertainment (CIE)*, t. 4, kw. 2006. DOI: 10.1145/1129006.1129021.

[97] B. Sobota i E. Pietriková, „The Role of Game Engines in Game Development and Teaching," w *Computer Science for Game Development and Game Development for Computer Science*, B. Sobota i E. Pietriková, red., IntechOpen, 2023, rozd. 5. DOI: 10.5772/intechopen.1002257.

[98] A. Ampatzoglou i A. Chatzigeorgiou, „Evaluation of object-oriented design patterns in game development," *Information and Software Technology*, t. 49, nr. 5, s. 445–454, 2007, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2006.07.003.

[99] Unity Technologies, „Unity Gaming Report 2022", Dostęp zdalny (19.08.2024): https://create.unity.com/gaming-report-2022, 2022.

[100] R. Nystrom, „Game Programming Patterns", Genever Benning, list. 2014, ISBN: 0990582906.

[101] T. Hollmann, „Development of an Entity Component System Architecture for Realtime Simulation", Dostęp zdalny (19.08.2024): https://kola.opus.hbz-nrw.de/files/1932/thesis-2019-09-16-final.pdf, praca inż. Universität Koblenz-Landau, 2019.

[102] T. Härkönen, „Advantages and Implementation of Entity-Component-System", Dostęp zdalny (19.08.2024): https://trepo.tuni.fi/handle/123456789/27593, praca mgr. Tampere University, 2019.

[103] B. Will, N. Lever, S. McGreal, D. K. Andersen i L. Gibert, „Introduction to the Data-Oriented Technology Stack for advanced Unity developers", Dostęp zdalny (19.08.2024): https://unity.com/resources/introduction-to-dots-ebook, maj 2024.

[104] B. Stroustrup, „What is "Object-Oriented Programming"?", w *ECOOP' 87 European Conference on Object-Oriented Programming*, 1987.

[105] J. Linietsky, „Why isn't Godot an ECS-based game engine?", Dostęp zdalny (19.08.2024): https://godotengine.org/article/why-isnt-godot-ecs-based-game-engine/, 2021.

[106] E. Gamma, R. Helm, R. Johnson i J. Vlissides, „Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994, rozd. 1, ISBN: 0201633612.

[107] K. Fedoseev, N. Askarbekuly, E. Uzbekova i M. Mazzara, „A Case Study on Object-Oriented and Data-Oriented Design Paradigms in Game Development", University of Innopolis, lip. 2020. DOI: 10.13140/RG.2.2.16657.66405.

[108] J. Linietsky, A. Manzur i in., „Godot Engine", wer. 4, oprogramowanie, Dostęp zdalny (19.08.2024): https://godotengine.org, 2023.

[109] Unity Technologies, „Unity", wer. 5, oprogramowanie, Dostęp zdalny (19.08.2024): https://www.unity.com, 2015.

[110] Epic Games, „Unreal Engine", wer. 5, oprogramowanie, Dostęp zdalny (19.08.2024): https://www.unrealengine.com, 2022.

[111] Crytek, „CryEngine”, wer. V, oprogramowanie, Dostęp zdalny (19.08.2024): https://www.cryengine.com, 2016.

[112] R. Fabian, „Data-oriented design: software engineering for limited resources and short schedules”, wrz. 2013, Dostęp zdalny (19.08.2024): https://www.dataorientedesign.com/dodbook/, ISBN: 9781916478701.

[113] T. Albrecht, „Pitfalls of Object Oriented Programming”, GCAP ’09, Dostęp zdalny (19.08.2024): https://harmful.cat-v.org/software/OO_programming/_pdf/Pitfalls_of_Object_Oriented_Programming_GCAP_09.pdf, 2009.

[114] B. V. Compton, „An Investigation of Data Storage in Entity-Component Systems”, Dostęp zdalny (19.08.2024): https://scholar.afit.edu/etd/5353, praca mgr. Air Force Institute of Technology, 2022.

[115] Bevy Contributors, „Bevy Documentation, bevy::ecs::system::Query”, wer. 0.13.0, Dostęp zdalny (19.08.2024): https://docs.rs/bevy/0.13.0/bevy/ecs/system/struct.Query.html, 2024.

[116] Bevy Contributors, „Bevy Documentation, bevy::ecs::entity::Entity”, wer. 0.13.0, Dostęp zdalny (19.08.2024): https://docs.rs/bevy/0.13.0/bevy/ecs/entity/struct.Entity.html, 2024.

[117] A. Catania i in., „Godex”, oprogramowanie, Dostęp zdalny (19.08.2024): https://github.com/GodotECS/godex, 2021.

[118] S. Martens i in., „flecs”, wer. 4, oprogramowanie, Dostęp zdalny (19.08.2024): https://www.flecs.dev/, 2024.

[119] J. Zhu, „ECS Overview and Architecture”, Dell Technologies, raport tech. 2022.

[120] M. Jensen, F. Dignum, L. Vanhée, C. Păstrăv i H. Verhagen, „Agile Social Simulations for Resilience,” w *Social Simulation for a Crisis*, F. Dignum, red. Springer, czer. 2021, s. 379–408, ISBN: 9783030763961. DOI: 10.1007/978-3-030-76397-8_14.

[121] M. Muratet i D. Garbarini, „Accessibility and Serious Games: What About Entity-Component-System Software Architecture?”, w *Games and Learning Alliance 9th International Conference, GALA 2020*, Springer, grud. 2020, s. 3–12, ISBN: 9783030634636. DOI: 10.1007/978-3-030-63464-3_1.

[122] W. Faryabi, „Data-oriented Design approach for processor intensive games”, Dostęp zdalny (19.08.2024): http://hdl.handle.net/11250/2575669, praca mgr. Norwegian University of Science i Technology, 2018.

[123] N. Whitney i in., „Noise-rs: Procedural Noise Generation library for Rust”, wer. 0.8.2, oprogramowanie, Dostęp zdalny (19.08.2024): https://github.com/razaekel/noise-rs, 2022.

[124] A. J. Walker, „An Efficient Method for Generating Discrete Random Variables with General Distributions,” *ACM Transactions on Mathematical Software*, t. 3, nr. 3, s. 253–256, wrz. 1977. DOI: 10.1145/355744.355749.

[125] ichi-h i in., „weighted_rand”, wer. 0.4.2, oprogramowanie, Dostęp zdalny (19.08.2024): https://github.com/ichi-h/weighted_rand, 2021.

[126] N. Beckmann, H.-P. Kriegel, R. Schneider i B. Seeger, „The R*-tree: an efficient and robust access method for points and rectangles”, w *Proceedings of the 1990*

ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, 1990, s. 322–331, ISBN: 0897913655. DOI: 10.1145/93597.98741.

[127] A. Guttman, „R-trees: a dynamic index structure for spatial searching", w *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, Association for Computing Machinery, 1984, s. 47–57, ISBN: 0897911288. DOI: 10.1145/602259.602266.

[128] S. Altmayer i in., „rstar", wer. 0.12.0, oprogramowanie, Dostęp zdalny (19.08.2024): https://github.com/georust/rstar, 2018.

[129] J. Allen, „Visible Earth: Topography", NASA's Earth Observatory, Dostęp zdalny (19.08.2024): https://visibleearth.nasa.gov/images/73934/topography, lip. 2005.

[130] M. New, D. Lister, M. Hulme i I. Makin, „A high-resolution data set of surface climate over global land areas," *Climate Research*, t. 21, nr. 1, s. 1–25, maj 2002, ISSN: 1616-1572. DOI: 10.3354/cr021001.

[131] C. Vérard, „Statistics of the Earth's Topography," *Open Access Library Journal*, t. 4, nr. 6, s. 1–50, 2017. DOI: 10.4236/oalib.1103398.

Wykaz symboli i skrótów

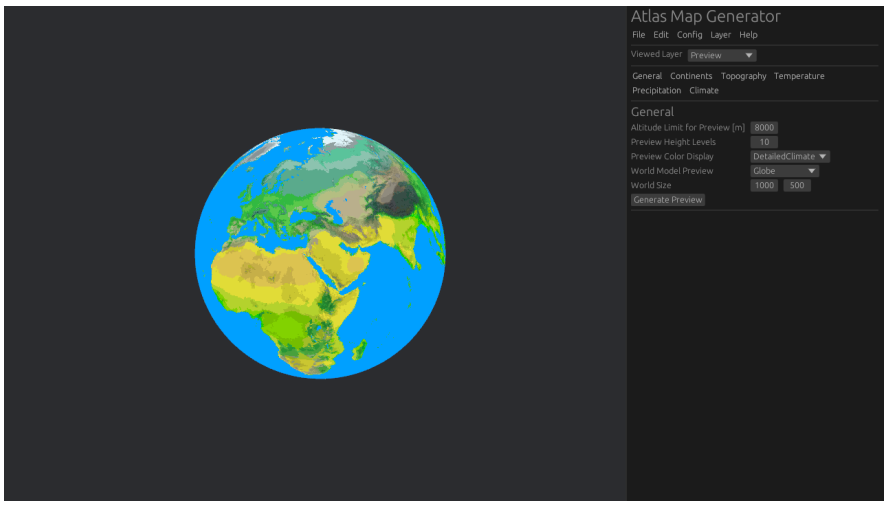
DOD – Data-Oriented Design (pl. projektowanie skupione wokół danych)
EC – Entity Component [system] (pl. [system] komponentów podmiotów)
ECS – Entity-Component-System (pl. podmiot-komponent-system)
GUI – Graphical User Interface (pl. graficzny interfejs użytkownika)
PCG – Procedural Content Generation (pl. proceduralna generacja treści)

Spis rysunków

Spis tabel

Spis listingów

Załącznik 1. Instrukcja użytkownika programu `atlas_gen` w języku angielskim



Window Layout

The application layout consists of two parts: map viewport (to the left) and sidebar (to the right). The sidebar is divided into three parts: title & menu bar, panel tabs and current panel.

Menu Bar

File - Import World Displays a folder dialog. When a directory is chosen, the following data is loaded from that directory from files:

- configuration - `atlasgen.toml`,
- preview layer - `preview.png`,
- continental layer - `continents.png`,
- initial topography layer - `topography.png`,
- final topography layer (with sea cutoff and coastal erosion applied) - `realtopography.png`,
- temperature layer - `temperature.png`,
- precipitation layer - `precipitation.png`,
- climate layer - `climate.png`,
- climate map - `climatemap.png`.

File - Export World Displays a folder dialog. When a directory is chosen, world data is saved as files in that directory. See previous action “Import World” for a list of all relevant files.

File - Exit Exits the application.

Edit - Reset Current Panel Resets all data in the currently viewed sidebar panel to their default values.

Config - Save Configuration Displays a file dialog. When a file name is entered or an exisiting file is chosen, the application configuration data is saved to that path in TOML format.

Config - Load Configuration Displays a file dialog. When a file is chosen, the application configuration is read from that file if it is in TOML format. Default values will be used if not all configuration data are present in the TOML file.

Config - Reset Configuration Resets the application configuration data to their default values.

Layer - Load Layer Data Displays a file dialog. When a file is chosen and it is in the correct image format (PNG 8-bit RGBA color sRGB for the map preview, PNG 8-bit greyscale otherwise) and has matching resolution, data of the currently viewed map layer will be replaced with that in the image.

Layer - Save Layer Data Displays a file dialog. When a file name is entered or an exisiting file is chosen, data of the currently viewed map layer will be saved to that path as a PNG image. See previous section for image format details.

Layer - Clear Layer Data Clears (batch sets to 0) data of the currently viewed map layer.

Layer - Render Layer Image Displays a file dialog. When a file name is entered or an exisiting file is chosen, the preview of the currently viewed map layer will be saved to that path as an image. The image format will be PNG 8-bit RGBA color sRGB.

Help - About Displays a window with information about the program.

Generic Generation Settings

Note: names in parentheses (`example`) are sections or keys in the TOML configuration file that refer to the discussed parameters accessible via the GUI.

Noise Algorithm (`algorithm`) This program uses two dimensional fractal noise as the basis of the generation process, with sample data scaled to [0.0; 1.0] range. There are 3 base noise algorithms available:

- Perlin (`perlin`)
- PerlinSurflet (`perlinsurflet`)
- OpenSimplex (`opensimplex`)
- SuperSimplex (`supersimplex`)
- FromImage (`fromimage`) - a special case. No data is generated, useful for when users supply exisitng data and don't want it to be overriden.

The following parameters can be used to customize generation output:

- Seed (`seed`) - the seed of random numbers. Changing the seed gives completely different results,
- Detail (number of octaves) (`detail`) - number of layers of noise overlaid on top of each other. Affects the overall level of detail. High level of zoom (low scale) requires higher detail for good results. High detail makes generation slower,

- Scale (frequency) (`frequency`) - frequency of sampling. High scale gives more zoomed out results,
- Neatness (lacunarity) (`neatness`) - multiplier to frequency for consecutive layers of noise. Small changes give different results, large increase will make shapes less defined,
- Roughness (persistance) (`roughness`) - power of amplitude for consecutive layers of noise. Low values give blurry shapes, high values give rough, high contrast shapes but also increase value,
- Bias (`bias`) - offset to the output *value* in [-1.0; 1.0] range,
- Offset (`offset`) - horizontal and vertical offset of the output. Offset should be scaled when frequency and lacunarity is changed.

Interpolation (`midpoint`) Output (in [0.0; 1.0] range) can be further modifed using three segment (four control point) linear interpolation. The start and end points have fixed position of 0.1 and 1.0 respectively, while two middle points (`midpoint_position` and `midpoint2_position`) can be freely moved on the X axis. Value (Y axis) can be customized for all 4 points (`start`, `midpoint`, `midpoint2` and `end`). This allows great flexibility in manipulating output ranges and distributions, i.e.:

- Scaling values,
- Reversing values,
- Creating steep slopes,
- Approximating nonlinear transformations.

Latitudinal Interpolation (`latitudinal`) Temperature and precipitation use noise algorithms only as supplementary (optional) source of data. The main source of data for these layers is latitude (map position on Y axis) based linear interpolation, with 9 fixed control points loosely based on circles of lattitude:

- Equator (0 degrees) (`equator_value`),
- Tropics (23 degrees) (`north_tropic_value` and `south_tropic_value`),
- Temperate zones (46 degrees) (`north_temperate_value` and `south_temperate_value`),
- Arctic/Antarctic (69 degrees) (`north_arctic_value` and `south_arctic_value`),
- Poles (90 degrees) (`north_pole_value` and `south_pole_value`),

An additional custom seeting is available, "Non-Linear Tropic Bias" (`non_linear_tropics`). When enabled, the interpolation between tropics and temperate zones becomes non-linear (in favor of tropics) which might produce better results when creating dry tropical deserts.

Influence Shape (`influence_shape`) Each layer can be optionally affected by a special layer called the "influence" map. It is a separate layer which can scale map data up or down, with intensity controlled by "Influence Strength" setting (`influence_strength`), depending on chosen mode (`influence_mode`):

- Scale down (`scaledown`) - influence values below 1.0 will scale data at that point down. This can be used to erase features outside the point of interest, i.e. to remove land at map corners,

- Scale up (`scaleup`) - influence values above 0.0 will scale data up. This can be used to emphasize features of specific locations,
- Scale down / up (`scaleupdown`) - influence values above 0.5 will scale data up, while below 0.5 will scale data down. This is a combination of the previous two modes.

Influence map can be generated in many ways, with most of them supporting interpolation as well (`midpoint`):

- None (`none`) - influence map is not applied,
- Circle (`circle`) - creates a circle defined by position (`offset`, offset from map center) and radius (`radius`). Points inside the circle have assigned value equal distance from circle center divided by its radius,
- Strip (`strip`) - creates a segment with a circle at each end. The following parameters are available: Segment center (`offset`, offset from map center), segment length (`length`), angle between the segment and horizontal axis (`angle`), segment thickness (`thickness`, also acts as circle radius), option to flip the strip horizontally (`flip`),
- Fbm (`fbm`) - standard noise algorithm (`algorithm`),
- FromImage (`fromimage`) - influence map is preserved as is, i.e. when loaded from file.

Panel Tabs

Note: names in parentheses (`example`) are sections or keys in the TOML configuration file that refer to the discussed parameters accessible via the GUI.

General (`[general]`) Configuration for the map in general as well as preview.

The following can be configured:

- Altitude limit for preview (`altitude_limit`) - Controls the altitude maximum for preview altitude shading. If above 0, tiles will become darker as they come closer to the maximum,
- Preview height levels (`height_levels`) - Controls how many discrete shading levels should be shown in the preview,
- Preview color display (`color_display`) - Controls how the tiles are colored when generating previews:
  - Topography (`topography`) - altitude based color palette (green-yellow-brown-grey),
  - Simplified climate (`simplifiedclimate`) - climate (biome) based color palette, using simplified biome colors,
  - Detailed climate (`detailedclimate`) - climate (biome) based color palette.
- Preview world model (`preview_model`) - Controls if the world map should be previewed as a flat map or as a globe,
- World size (`world_size`) - Horizontal (longitudinal) and vertical (latitudinal) size of the world, in tiles.

Continents (`[continents]`) Configuration for continents generation. Each map tile can be either a water tile (value <= 127) or a land tile (value > 127).

The following can be configured:

- Sea level (`sea_level`) - Height of the global sea level as a fraction (0.0-1.0 range). Layer data (normalised) below this value will be marked as water, otherwise it will be land,
- Standard noise algorithm with quad point interpolation (`algorithm`),
- Standard influence shape (`influence_shape`).

Topography (`[topography]`) Configuration for topography (height map). Each map tile contains altitude data. Altitude ranges from 0 (sea level) to 255 (10200 meters), with one altitude unit equal to 40 meters. Topography is affected by continental data - water tiles are forced to sea level altitude.

The following can be configured:

- Coastal erosion range (`coastal_erosion`) - Controls how far from the coast (in tiles) coastal erosion affects height. The closer to coast, the stronger the reduction in height is. Acceptable value range is from 0 (disabled) to 20. Note: long range erosion slows down the generation,
- Standard noise algorithm with quad point interpolation (`algorithm`),
- Standard influence shape (`influence_shape`).

Temperature (`[temperature]`) Configuration for temperature map. Each map tile contains temperature data (mean annual at surface level). Temperature ranges from 0 (-50 degrees Celsius) to 255 (+77,5 degrees), with one temperature unit equal to 0.5 degrees Celsius and value of 100 equal to 0 degrees.

The following can be configured:

- Moist adiabatic lapse rate (MALR) (`lapse_rate`) - Controls how much temperature lowers as altitude rises. Expressed in Celsius per kilometer,
- Latitudinal settings (values in degrees Celsisus) (`latitudinal`),
- Noise strength - Scales down noise algorithm output (`algorithm_strength`),
- Standard noise algorithm with quad point interpolation (`algorithm`),
- Standard influence shape (`influence_shape`).

Precipitation (`[precipitation]`) Configuration for precipitation (rainfall and snowfall) map. Each map tile contains precipitation data (mean annual). Precipitation ranges from 0 (0 mm) to 255 (5100 mm), with one precipitation unit equal to 20 milimeters of water.

The following can be configures:

- Altitude of maximum precipitation (`amp_point`) - Controls the *minimum* altitude at which precipitation begins to lower. Expressed in meters,
- Precipitation drop (`drop_per_height`) - Controls how much precipitation lowers as altitude rises. Expressed in milimeters per meter,
- Latitudinal settings (values in mm) (`latitudinal`),
- Noise strength (`algorithm_strength`) - Scales down noise algorithm output,
- Standard noise algorithm with quad point interpolation (`algorithm`),
- Standard influence shape (`influence_shape`).

Climate (`[climate]`) Configuration for climate assigning. Each map tile has assigned an index of a biome from biome list, based on temperature at precipitation at that location. The exact mapping is controlled by `climatemap.png` file (PNG 8-bit greyscale, 255x255 pixels), which is essentailly a two dimenstional lookup table:

- Horizontal axis - temperature, from left to right,
- Vertical axis - precipitation, from top to bottom,
- Value at point - index of a biome in the biome list.

There are two preview modes for this layer:

- Simplified color,
- Detailed color,

Each biome (`biomes`) has a name (`name`) and the following properties:

- Color (`color`) - Color to use for this climate in the detailed climate preview mode. Each biome should have a unique color,
- Color (simplified view) (`simple_color`) - Color to use in the simplified climate preview mode. Similar biomes should share colors,
- Resources (`deposits`) - List of resource deposit IDs that this biome provides with given probability.
- Habitability (`habitability`) - Weight used in the Atlas History Simulator for assinging starting locations and border expansion costs.

Note: adding or removing biomes from the list is possible only via config file.

Deposits (`[deposits]`) Configuration for resource deposits. World resources are grouped into square chunks of specified size (`chunk_size`), with each tile contributing resource deposits to its chunk. Deposits are not visualized in any way and have no impact on layers of world generation, but are essential to the Atlas History Simulator.

Each resource deposit type (`types`) has a name (`name`) and the following properties:

- Random Deposit Chance (`gen_chance`) - Probability of appearing in a tile regardless of biome.
- Deposit Average Size (`gen_average`) - Mean of the deposit size normal distribution. Larger deposits provide more resources.
- Deposit Size Deviation (`gen_deviation`) - Deviation of the deposit size normal distribution.
- Supply Points (`supply`) - Amount of supply resources provided per deposit size.
- Industry Points (`industry`) - Amount of industry resources provided per deposit size.
- Wealth Points (`wealth`) - Amount of wealth resources provided per deposit size.

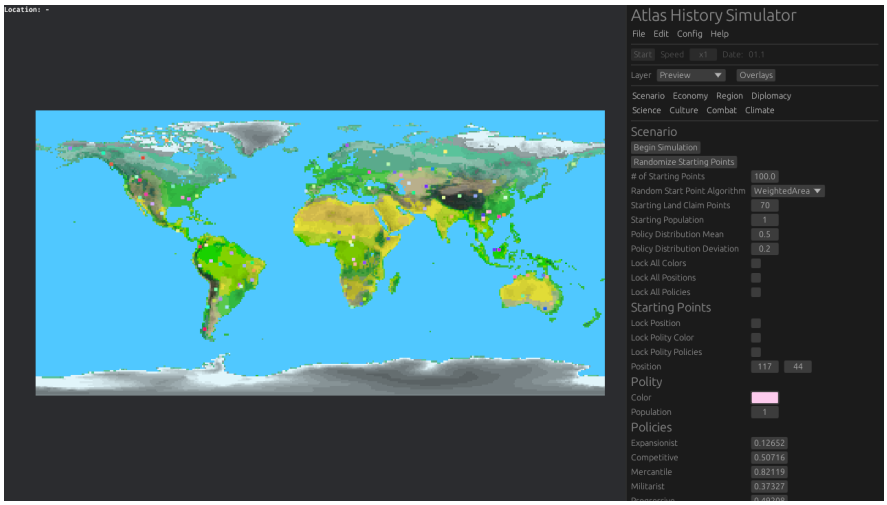
Each deposit chunk (`chunks`) has the following properties:

- Land Tile Count (`tile_count`) - Number of continental tiles within the chunk.
- Deposits (`deposits`) - List of deposit types contained and their total size in the chunk.

Tips

- No configuration changes will take effect until you press the “Generate Layer” button for the respective panels.
- Numerical input boxes also act like sliders. Dragging on horizontal axis will decrease or increase value.
- You can drag the edge of the sidebar to adjust its width.
- You can zoom in or out of the map using the mouse wheel.
- You can drag the plane in flat preview mode and rotate the globe in globe preview mode while pressing the right mouse button,
- If you prefer to work with text files over the GUI, you can save the default configuration and edit its TOML file, then load it in Atlas Map Generator and only generate layers.

Załącznik 2. Instrukcja użytkownika programu `atlas_sim` w języku angielskim



Window Layout

The application layout consists of two parts: map viewport (to the left) and sidebar (to the right). The sidebar is divided into three parts: title & menu bar, panel tabs and current panel.

Menu Bar

File - Import Generated World Displays a folder dialog. When a directory is chosen, the following data is loaded from that directory from files:

- configuration - `atlassim.toml`,
- preview layer - `preview.png`,
- continental layer - `continents.png`,
- initial topography layer - `topography.png`,

- final topography layer (with sea cutoff and coastal erosion applied) - `realtopography.png`,
- temperature layer - `temperature.png`,
- precipitation layer - `precipitation.png`,
- climate layer - `climate.png`.

File - Exit Exits the application.

Edit - Reset Current Panel Resets all data in the currently viewed sidebar panel to their default values.

Config - Save Configuration Displays a file dialog. When a file name is entered or an exisiting file is chosen, the application configuration data is saved to that path in TOML format.

Config - Load Configuration Displays a file dialog. When a file is chosen, the application configuration is read from that file if it is in TOML format. Default values will be used if not all configuration data are present in the TOML file.

Config - Reset Configuration Resets the application configuration data to their default values.

Help - About Displays a window with information about the program.

Panel Tabs (Setup)

Note: names in parentheses (`example`) are sections or keys in the TOML configuration file that refer to the discussed parameters accessible via the GUI.

Scenario (`[scenario]`) Configuration for specific simulation scenario setup, such as amount of starting polities, population, colors.

- # of Starting Points (`num_starts`) - number of starting points/polities in the simulation,
- Random Start Point Algorithm (`random_point_algorithm`) - algorithm that should be used to automatically set positions (map tiles) of starting points. There are 4 random algorithms available:
  - Uniform (`"uniform"`) - tiles have a uniform chance,
  - Weighted (`"weighted"`) - tiles are weighted by their habitability score,
  - WeightedArea (`"weightedarea"`) - tiles are weighted by average hability score of 3x3 area,
  - WeightedSquared(`"weightedsquared"`) -tiles are weighted by habitability squared,
  - WeightedSquaredArea(`"weightedsquaredarea"`) - tiles are weighted by average habitability squared of 3x3 area.
- Starting Land Claim Points (`starting_land_claim_points`) - amount of land claim points initially awarded to each polity,
- Starting Population (`start_pop`) - amount of starting population of each polity,

- Policy Distribution Mean/Deviation (`policy_mean/policy_deviation`) - mean/deviation of the normal distribution of policy values,
- Lock All Colors/Positions/Policies (`lock_colors/lock_positions/lock_policies`) - disable randomization of all polity colors/positions/policies.

Additionally, each starting point/polity (`start_points`) can be manually adjusted:

- Lock Position/Polity Color/Polity Policies (`position_locked/color_locked/policy_locked`) - disable randomization of this polity's position/color/policies,
- Position (`position`) - map coordiantes of this polity starting position,
- Color (`color`) - RGB888 color for map visualization,
- Population (`population`) - initial population,
- Policies (`policies`) - list of initial policy values.

Available policies:

- Expansionist - region expansion vs region development,
- Competitive - aggresive stance vs peaceful stance,
- Mercantile - wealth production vs industrial production,
- Militarist - military investment vs civilian investment,
- Progressive - research investment vs culture investment,
- Legalist - stability investment.

Economy (`[rules.economy]`) Configuration for simulation rules concerning the economy and population:

- Monthly Base Pop Growth (`pop_growth`) - regional population will grow by this multiplier each month (if at full supply and health),
- Maximum Healthcare penalty (`max_health_penalty`) - maximum penalty that can be applied to population growth due to insufficient healthcare,
- Supply/Industry/Wealth Need per Pop (`base_supply_need/base_industry_need/base_wealth_need`) - amount of supply/industry/wealth one unit of population consumes per month,
- Supply/Industry/Wealth Loss to Chaos (`chaos_supply_loss/chaos_industry_loss/chaos_wealth_loss`) - amount of supply/industry/wealth one unit of population consumes additionally due to insufficient stability,
- Base Crime Rate (`crime_rate`) - percentage of population that always causes crime (lowers stability),
- Rebelion Speed (`rebelion_speed`) - multiplier to how fast rebelion (low stability due to occupation) increases/decreases,
- Military Industry Storage (`military_stash`) - months worth of military industry output that can be stored for later use,
- Loyalty Storage (`loyalty_stash`) - months worth of loyalty output that can be stored for later use.

For each resource type (in `resources`):

- Supply - consumed by population,
- Industry (in general),

- Civilian industry - used for expansion and development,
- Military industry - used in combat,
- Wealth (in general),
- Research - used for scientific advancement,
- Culture - used for cultural advancement,
- Loyalty - used in combat,
- Industry tributes,
- Wealth tributes,

the following parameters can be adjusted:

- Efficiency (`efficiency`) - how many units of this resource one unit of population can produce monthly,
- Efficiency Over Capacity (`over_cap_efficiency`) - efficiency of resource production when over resource capacity.

Region (`[rules.region]`) Configuration for simulation rules concerning the regional development:

- Minimum Size to Split (`min_split_size`) - minimum size in tiles that a region must reach in order to split into two regions,
- New City Cost (`new_city_cost`) - cost of establishing a new city/region,
- Land Claim Cost (`land_claim_cost`) - cost of claiming a new map tile,
- Expansion Cost Increase Per Region (`sprawl_penalty`) - flat increase to land claim and new city cost for each polity region,
- Base Expansion Speed (`base_exp_speed`) - conversion modifier from civilian industry to land claim points and new city points,
- Base Development Speed (`base_dev_speed`) - conversion modifier from civilian industry to development points and strucutre points,
- Development Level Cost (`dev_level_cost`) - percent increase in development cost per existing development level,
- Maximum Development Level (`max_dev_level`) - maximum development & structure level,
- Development Bonus (`dev_bonus`) - bonus to supply/industry/wealth production per development level,
- Base Capacity (`base_capacity`) - capacity provided per structure level.

For each structure (in `structures`):

- Hospital - provides regional health power,
- Manufacture - provides civilian industry capacity,
- Forge - provides military industry capacity,
- University - provides research capacity,
- Amphitheater - provides culture capacity,
- Couthouse - provides reegional security power,
- Fortress - provides fortification level,

the following parameters can be adjusted:

- Strength (`strength`) - multiplier to base capacity,
- Cost (`cost`) - multiplier to development speed.

Diplomacy (`[rules.diplomacy]`) Configuration for simulation rules concerning inter-polity diplomacy and policies:

- Initial Peace (`initial_peace_length`) - length in months of the forced peace period at the start of the simulation,
- Truce Length (`truce_length`) - length in months of a truce period after a conflict ends,
- Policy Change Time Mean/Deviation (`policy_time_mean/policy_time_dev`) - mean/deviation of the normal distribution of time for policy change (in years),
- Relations Change Speed (`relations_speed`) - conversion modifier from competitive policy value to mutual relations value,
- Base Relations Improvement (`base_good_shift`) - bias towards positive mutual relations,
- Ally/Friend threshold (`ally_threshold/friend_threshold`) - minimum relations threshold to reach to be considered allies/friends,
- Rival/Enemy threshold (`rival_threshold/enemy_threshold`) - maximum relations threshold to reach to be considered rivals/enemies,
- Region Claim Difficulty (`claim_difficulty`) - multiplier to loser's conflict contribution score when making regional claims,
- Tribute Length (`tribute_time`) - length of tribute payments in months,
- Economy to Tribute (`tribute_ratio`) - percent of resources per lost conflict.

Science (`[rules.science]`) Configuration for simulation rules concerning scientific research:

- Major/Minor Level Speed (`speed_major/speed_minor`) - conversion multiplier from science resource to progress in major/minor field level,
- Maximum Major/Minor Level (`max_level_major/max_level_minor`) - maximum possible major/minor field level,
- Major/Minor Level Bonus (`bonus_major/bonus_minor`) - multiplier to bonuses provided by a field per major/minor level,
- Base Decay (`base_decay`) - base progress decay for each field,
- Major Level Decay (`level_decay`) - additional progress decay per field major level,
- Major Level Difficulty Increase (`level_difficulty`) - percent of progress cost increased per major level,

For each scientific field (in `fields`):

- Geoscience - increased supply/industry/wealth production,
- Medicine - increased healthcare,
- Engineering - increased civilian industry,
- Metallurgy - increased military industry,
- Philosophy - increased culture,
- Mathematics - increased research,

- Management - increased loyalty,
- Law - increased stability,
- Linguistics - increased inter-polity relation strength,
- Military Tech - increased combat strength,

the following parameters can be adjusted:

- Strength (`strength`) - multiplier to bonuses provided by this field,
- Speed (`speed`) - multiplier to research speed.

Culture (`[rules.culture]`) Configuration for simulation rules concerning cultural activities:

- Base Speed (`base_speed`) - conversion multiplier from culture resource to progress in a tradition,
- Base Decay (`base_decay`) - base progress decay for each tradition,
- Maximum Level (`max_level`) - maximum possible tradition level,
- Level Bonus (`level_bonus`) - multiplier to bonuses provided by a tradition per level,
- Level Decay (`level_decay`) - additional progress decay per tradition level,
- Overflow Culture to Heritage Ratio (`heritage_ratio`) - ratio of tradition porgress overflow converted to heritage points,
- Great Event Heritage Cost (`great_event_heritage`) - accumulated heritage is divided by this to get great event probability,
- Great Event Max Chance (`great_event_chance_max`) - maximum possible great event probability regardless of accumulated heritage,
- Great Person Chance (`great_person_chance`) - probability that a great event will be a great person,
- Great Work Bonus (`great_work_bonus`) - tradition levels added per great work,
- Great Person Bonus (`great_person_bonus`) - tradition levels added per active great person,
- Great Person Duration (`great_person_duration`) - duration of great person activity in months,

For each tradition type (in `traditions`):

- Pioneering - used in expansion,
- Monumentality - used in development,
- Curiosity - used in scientific advancement,
- Creativity - used in cultural advancement,
- Prosperity - used in production,
- Authority - used for stability,
- Diplomacy - used in inter-polity relations,
- Supremacy - used in combat,

the following parameters can be adjusted:

- Strength (`strength`) - multiplier to bonuses provided by this field,
- Speed (`speed`) - multiplier to research speed.

Combat (`[rules.combat]`) Configuration for simulation rules concerning conflicts:

- Action Weights (Attacker/Deffender) (`action_weights_attacker/action_weights_defender`) - weights of combat actions (assault/maneouver/charge/rally/siege/fortify) for attacker/defender,
- Assault Bonus (`assault_bonus`) - multiplier to material attack and defence when assaulting,
- Maneouver Bonus (`maneouver_bonus`) - multiplier to morale attack and defence when maneouvring,
- Charge Bonus (`charge_bonus`) - multiplier to any attack and penalty to any defence when charging,
- Rally Bonus (`rally_bonus`) - multiplier to any defence and penalty to any attack when rallying,
- Siege Bonus (`siege_bonus`) - conversion factor of any attack into siege damage when laying siege,
- Siege Penalty (`siege_penalty`) - penalty to any attack and defence when laying siege,
- Fortify Bonus (`fortify_bonus`) - multiplier to fortification level when fortifying,
- Fortify Penalty (`fortify_penalty`) - penalty to any attack when fortifying,
- Military Size Ratio (`military_size`) - military size can be up to this fraction of total population, per active conflict,
- Base Mobilization Speed (`base_mobilization`) - fraction of recruitable population that is mobilized into military per month,
- Mobilization From Militarist Policy (`militarist_mobilization`) - bonus to mobilization speed based on militarist policy,
- Initial Mobilization Time (`mobilization_build_up`) - months to spend mobilizing troops before committing to a conflict,
- Combat Randomness (`randomness`) - random strength multiplier will be in [1.0 - this; 1.0 + this] range,
- Material/Morale Damage Fatality/Fragility (`fatality/fragility`) - conversion factor from total material/morale strength to material/morale damage,
- Material/Morale Advantage Power (`material_advantage/morale_advantage`) - exponent of material/morale advantage ratio,
- Morale Breakdown Multiplier (`breakdown`) - conversion factor from excess morale damage to additional material damage when suffering a breakdown,
- Morale to Material Ratio Cap (`morale_cap`) - accumulated morale can this times larger than material,
- Equipment to Manpower Ratio (`equipment_manpower_ratio`) - each unit of material (manpower) requires this many units of military industry,
- Damage to Fort Ratio (`fort_damage`) - conversion factor from excess material damage to fort damage,
- Monthly Attacker/Defender Attrition (`base_attacker_attrition/base_defender_attrition`) - base monthly attrition increase for attacker/defender,

- Attrition From Combat Damage (`combat_attrition`) - attrition multiplier to quotient of material loss to total population,
- Attrition From Civilian Damage (`civilian_attrition`) - attrition multiplier to quotient of unabsorbed damage to total population,
- Civilian Damage From Combat Modifier (`civilian_damage`) - civilian damage multiplier to quotient of unabsorbed damage to total population,
- Maximum Civilian Damage Ratio (`civilian_damage_max`) - maximum percent of civilian damage dealt per month,
- Base Rebel Rate in Claimed Regions (`base_rebel_rate`) - base rebel rate in claimed regions,
- Rebel Damage to Infrastructure (`rebel_structure_damage`) - multiplier to one-time development damage in rebelious regions,

Climate (`[climate]`) Climate settings carried over from Atlas Map Generator. Each map tile has assigned an index of a biome from biome list.

There are two preview modes for this layer:

- Simplified color,
- Detailed color,

Each biome (in `biomes`) has a name (`name`) and the following properties:

- Color (`color`) - Color to use for this climate in the detailed climate preview mode. Each biome should have a unique color,
- Color (simplified view) (`simple_color`) - Color to use in the simplified climate preview mode. Similar biomes should share colors,
- Resources (`deposits`) - List of resource deposit IDs that this biome provides with given probability. Note: this field can be empty, as all resources have already been assigned in Atlas Map Generator or manually.
- Habitability (`habitability`) - Weight used for assinging starting locations and border expansion costs.

Note: adding or removing biomes from the list is possible only via config file.

Panel Tabs (Running)

Selected Information about the currently selected region:

- regional civilian population,
- total public security power & public health power,
- regional stability & healthcare (as % of population covered),
- number of map tiles occupied by region,
- accumulated land claim points & new city points,
- development level of the region,
- list of regional key structures and their levels,
- list of resource deposits available in this region.

Polity General information about the currently selected polity:

- the internal polity ID and map color,
- total number of regions,
- total civilian population,
- average stability & healthcare of all regions (as % of population covered),
- date of the next change of policies,
- list of current policies,
- list of neighbours: their polity ID and mutual relation value (above 0 is positive, below 0 is negative),
- list of pending tributes: number of monthly payments left, receiving polity ID and percent of tribute fund to pay per payment.

Economy Information about economy output and employment structure of the currently selected polity:

- accumulated yearly civilian industry,
- output of each resource this month,
- amount of industry and wealth tribute paid this month,
- total polity population (civilian + military),
- population count employed per sector: military, supply, industry, wealth.

Science Information about scientific progress of the currently selected polity:

- accumulated yearly research points,
- list of scientific fields with their major (left value) and minor (right value) levels.

Culture Information about cultural progress of the currently selected polity:

- accumulated yearly culture points,
- list of traditions with their primary level (left value) and great event level (right value),
- list of accumulated heritage for traditions (measured in culture points),
- list of created great works and great people containing date of creation and associated tradition.

Combat List of all conflicts that the currently selected polity is a part of and detailed information about them:

- conflict start date,
- primary attacker polity ID,
- primary defender polity ID,
- list of all conflict members, divided into attackers and defenders:
  - polity ID and map color,
  - months left to mobilize troops before entering combat,
  - material and morale strength committed to the conflict,
  - current attrition rate,
  - total contribution to conflict,
  - strength of all polity fortifications left,
  - chosen combat action for this month and engagement status.

Tips

- Numerical input boxes also act like sliders. Dragging on horizontal axis will decrease or increase value.
- You can drag the edge of the sidebar to adjust its width.
- You can zoom in or out of the map using the mouse wheel.
- You can drag the map around while the right mouse button is pressed.
- If you prefer to work with text files over the GUI, you can save the default configuration and edit its TOML file, then load it in Atlas History Simulator and only generate layers.

Załącznik 3. Instrukcja budowania i uruchamiania projektu w języku angielskim

Building the project The MSRV (Minimum Supported Rust Version) for this project is 1.76. The project uses standard build profiles: development and release. You can build the project with `cargo` using the following commands:

```
1 cargo build            # development profile
2 cargo build --release  # release profile
```

The project makes use of build scripts to insert icons into the compiled executables as resources. Make sure that files `atlas_gen/icon.ico` and `atlas_sim/con.ico` are present when compiling the project.

Running the project The project can be quick-run with `cargo` if the source code is available:

```
1 cargo run --bin=atlas_gen  # run the generator
2 cargo run --bin=atlas_sim  # run the simulator
```

If you have compiled executable files instead of source code, you can run them directly with no arguments:

```
1 ./bin=atlas_gen  # run the generator
2 ./bin=atlas_sim  # run the simulator
```

While the simulator can be run whenever from wherever, the generator expects a valid climate map file named `climatemap.png` to be present in the same directory the executable is.

Załącznik 4. Skrypt do konwersji zbioru danych CRU CL v2.0 do formatu wejściowego `atlas_gen`

```
# A script that turns temperature and precipitation data from CRU CL v2.0 dataset
# into a map layer usable with Atlas map generator.

import pandas as pd
import numpy as np
from PIL import Image
from math import floor

CONVERT_TEMP = False
CONVERT_PRECIP = False

DEFAULT_TEMP = 10
DEFAULT_PRECIP = 1000

TARGET_WIDTH = 1000
TAGRET_HEIGHT = 500

WIDTH = 360 * 6
HEIGHT = 180 * 6

def convert_temp(tmp: pd.DataFrame) -> pd.DataFrame:
    new = pd.DataFrame(DEFAULT_TEMP, index=range(HEIGHT), columns=range(WIDTH))
    for _, row in tmp.iterrows():
        # Map real latitude and longitude to one where map origin
        # is in the top left corner.
        lat, long = -(row[0] - 90.0), (row[1] + 180.0)
        y, x = floor((lat / 180) * HEIGHT), floor((long / 360) * WIDTH)
        # Get mean temperature of all months.
        avg = row[2:].mean()
        new.iloc[y, x] = avg
    # Each degree Celsius is 2 temperature units, with 0 C being 100 Tu.
    new = new.applymap(lambda x: min(round(x * 2 + 100), 255))
    return new

def convert_precip(pre: pd.DataFrame) -> pd.DataFrame:
    new = pd.DataFrame(DEFAULT_PRECIP, index=range(HEIGHT), columns=range(WIDTH))
    for _, row in pre.iterrows():
        # Map real latitude and longitude to one where map origin
        # is in the top left corner.
        lat, long = -(row[0] - 90.0), (row[1] + 180.0)
        y, x = floor((lat / 180) * HEIGHT), floor((long / 360) * WIDTH)
        # Get total annual precipitation.
        total = row[2:14].sum()
        new.iloc[y, x] = total
    # Each precipitation unit is 20mm of precipitation.
    new = new.applymap(lambda x: min(round(x / 20), 255))
    return new

def project_and_save(df: pd.DataFrame, name: str):
    # Note: since maps are assumed to use orthographic projection,
    # there's no need to map longitude and latitude.
```

```
52      img = Image.fromarray(df.to_numpy(dtype=np.uint8), mode="L")
53      # Resize to output dimensions using the highest quality resampler.
54      img = img.resize((TARGET_WIDTH, TAGRET_HEIGHT), Image.Resampling.BICUBIC)
55      img.save(name)
56
57  def main():
58      # Convert temperature data.
59      if CONVERT_TEMP:
60          tmp = pd.read_csv("grid_10min_tmp.dat", dtype=float)
61          tmp = convert_temp(tmp)
62          project_and_save(tmp, "tmp.png")
63      else:
64          print("Skipping temperature data.")
65      # Convert precipitation data.
66      if CONVERT_PRECIP:
67          pre = pd.read_csv("grid_10min_pre.dat", dtype=float)
68          pre = convert_precip(pre)
69          project_and_save(pre, "pre.png")
70      else:
71          print("Skipping precipitation data.")
72
73
74  if __name__ == "__main__":
75      main()
```