**Akademia Górniczo-Hutnicza**
**im. Stanisława Staszica w Krakowie**

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki


PRACA INŻYNIERSKA

DOROTA WOJTAŁOW, JACEK ZŁYDACH

# SYMULACJA ROZPRZESTRZENIANIA SIĘ DYMU I OGNIA W OPARCIU O NIEHOMOGENICZNE AUTOMATY KOMÓRKOWE

PROMOTOR:
dr inż. Jarosław Wąs

Kraków 2010

## OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE, I NIE KORZYSTAŁEM ZE ŹRÓDEŁ INNYCH NIŻ WYMIENIONE W PRACY.

.....

PODPIS

**AGH**
**University of Science and Technology in Krakow**


DOROTA WOJTAŁOW, JACEK ZŁYDACH

# SIMULATION OF FIRE AND SMOKE BY USING NON-HOMOGENEOUS CELLULAR AUTOMATA

SUPERVISOR:
Jarosław Wąs Ph.D

Krakow 2010

Serdecznie dziękujemy …

# Spis treści

# 1. Wstęp

## 1.1. Temat pracy

Tematem pracy jest stworzenie symulacji rozprzestrzeniania się dymu i ognia w oparciu o niehomogeniczne automaty komórkowe. Zakres pracy obejmuje stworzenie symulacji rozchodzenia się dymu i ognia na podstawie automatów komórkowych wraz z jej wizualizacją, a także walidację stworzonego modelu. Celem pracy jest pokazanie możliwości niehomogenicznych automatów komórkowych jako narzędzia umożliwiającego odzwierciedlenie rzeczywistego rozprzestrzeniania się dymu i ognia podczas pożaru.

## 1.2. Geneza tematu

W dobie wszechobecnej urbanizacji i ciągłego budownictwa, wraz ze wzrostem świadomości dotyczącej bezpieczeństwa pożarowego oraz zaangażowania w jego zagwarantowaniu pojawiła się potrzeba możliwości modelowania i obserwacji rozprzestrzeniania się ognia w zamkniętych budynkach. Wspomniane symulacje pożarów wykazują szereg zastosowań. Są z powodzeniem wykorzystywane w śledztwach. Dają możliwość odtworzenia przebiegu zdarzeń i porównania z wynikami oględzin. Umożliwiają zbadanie prototypu budynku pod kątem gwarancji bezpieczeństwa pożarowego. Ułatwiają projektowanie systemów oddymiania. W połączeniu z modelami ewakuacji ludzi stanowią kompleksowy system ułatwiający tworzenie bezpiecznych budowli.

W ostatnich latach powstał szereg programów umożliwiających wizualizację symulacji rozchodzenia ognia. Opracowane dotychczas rozwiązania swoje działanie opierają na metodach numerycznej dynamiki płynów (ang. Computational Fluid Dynamics). Niewątpliwą zaletą numerycznego podejścia jest dokładność wyników. Głównymi wadami jest złożoność obliczeń i stopień komplikacji modelu. Niehomogeniczne automaty komórkowe umożliwiają znaczne uproszczenie modelu. Uproszczenie modelu powoduje z kolei redukcję złożoności obliczeń czyniąc automaty komórkowe szczególnie dogodną metodą w przypadku tworzenia prototypów oraz symulacji czasu rzeczywistego.

## 1.3. Teza pracy

*Wykorzystując zasady tworzenia automatów komórkowych oraz w oparciu o prawa fizyki można w realistyczny sposób przy użyciu niehomogenicznych automatów komórkowych zamodelować zjawiska rozchodzenia się ognia i dymu podczas pożaru.*

W celu wykazania powyższej tezy przeprowadzono następujące działania:

- Przeprowadzono badania dotyczące zjawisk fizycznych zachodzących podczas pożaru.
- Zidentyfikowano czynniki mające kluczowy wpływ na kształt i charakter pożaru.
- Zaproponowano model niehomogenicznego automatu komórkowego odzwierciedlającego prawa fizyczne.

- Zaimplementowano opracowany algorytm wraz z wizualizacją wyników oraz możliwością edycji danych wejściowych oraz kontroli symulacji w czasie rzeczywistym.
- Dokonano weryfikacji jakościowej zaproponowanego modelu.

## 1.4. Realizacja projektu

Praca została zrealizowana jako wolnostojąca aplikacja komputerowa napisana w języku Java. Do renderowania grafiki trójwymiarowej została użyta biblioteka graficzna Java3D. Aplikacja została przetestowana z wykorzystaniem biblioteki JUnit4.

## 1.5. Zawartość pracy

Praca składa się z [iluś] rozdziałów. W pierwszym rozdziale znajdują się podstawy teoretyczne, związane zarówno z modelowanymi zjawiskami fizycznymi jak i użytym algorytmem. Rozdział Modele symulacji zawiera propozycje zweryfikowanych modeli rozprzestrzeniania się dymu i ognia zaprojektowanych w oparciu o niehomogeniczne automaty komórkowe. Rozdział Implementacja przedstawia sposób realizacji projektu, napotkane problemy oraz ich rozwiązania. Opisuje możliwości graficznego interfejsu użytkownika oraz sposób korzystania z niego.

# 2. Teoria

Kluczowym elementem, niezbędnym do prawidłowego zamodelowania pożaru, jest zrozumienie czym jest ogień oraz poznanie zjawisk, jakim podlega. Niniejszy rozdział zawiera krótki wstęp teoretyczny, przedstawiający zjawiska fizyczne niezbędne do zrozumienia istoty pożaru i prawidłowego jego zamodelowania.

## 2.1. Czym jest ogień?

Ogień nie jest substancją. Powstaje on jako produkt reakcji chemicznej zachodzącej między paliwem i tlenem. Obserwowalną postać ognia, czyli to co widzimy i nazywamy ogniem, tworzy światło powstałe w wyniku ruchu rozgrzanego powietrza. Jest to jednak tylko jeden z aspektów tego złożonego procesu. Elementami koniecznymi do powstania i podtrzymania ognia są:

- tlen
- paliwo
- ciepło

Paliwem może być ciecz, ciało stałe lub gaz. Paliwo samo w sobie nie ulega spalaniu. Ogrzewa się ono pod wpływem ciepła pochodzącego np. z zapałki lub otrzymanego od innego nagrzanego ciała. Po osiągnięciu odpowiedniej temperatury, paliwo ulega procesowi dekompozycji. Jednym z produktów dekompozycji są opary. W przypadku jednego z najbardziej popularnych paliw - drewna - oprócz oparów w wyniku dekompozycji otrzymujemy węgiel i popiół. Spalanie drewna przedstawia reakcja 2.1:

$$H_2CO + O_2 + heat- > CO_2 + CO + C + N_2 + H_2O \tag{2.1}$$

Kiedy opary osiągną odpowiednią temperaturę, tzw. temperaturę zapłonu (w przypadku drewna wynosi ona ok. $300°C$), oraz gdy ich stężenie w powietrzu jest odpowiednie, może dojść do zapłonu. Dochodzi do niego w wyniku kontaktu z otwartym ogniem, iskrą lub na skutek osiągnięcia przez opary tzw. temperatury samozapłonu. Wynikiem zapłonu jest spalanie oparów. Jak widać, głównym substratem reakcji spalania są opary powstające w wyniku ogrzania paliwa. W przypadku niektórych paliw jest to jedyny reagent. Jednym z przykładów jest benzyna, która w wyniku ogrzania w całości zamienia się w opary ulegające spalaniu. W przypadku drewna, poza oparami spalaniu ulega także węgiel. Jest to jednak reakcja bardzo powolna. Na szczególną uwagę zasługuje fakt wytwarzania energii cieplnej w procesie spalania, co powoduje samoistne podtrzymanie ognia. Płomień, będący wizualną postacią spalania, ogrzewa sąsiadujące cząsteczki paliwa, dzięki czemu nie gaśnie.

Bardzo istotnym reagentem w procesie spalania jest tlen. Atomy gazów - oparów powstałych w wyniku podgrzania paliwa - w wyniku zapłonu łączą się z tlenem. Aby mogło dojść do reakcji spalania bardzo ważne jest zachowanie odpowiednich proporcji między substratami reakcji. Wielkość nazwana Lower Explosive Limit (LEL) określa minimalne stężenie oparów w powietrzu, konieczne aby mogło dojść do zapłonu. Odpowiednio, Upper Explosive Limit (UEL) oznacza maksymalne stężenie oparów, powyżej którego nie dojdzie do zapłonu. Przykładowo: dla tlenku węgla $LEL = 12$, natomiast $UEL = 75$, co oznacza, że jego stężenie w powietrzu musi wynosić $12\%$ do $75\%$, aby mogło dojść do jego

zapalenia. $LEL$ oraz $UEL$ określają także pośrednio wymaganą ilość tlenu. Dla większości paliw ilość tlenu wymaganego do zapłonu wynosi ok. $15\%$.

## 2.2. Propagacja ciepła

Jednym z czynników niezbędnych do podtrzymania ognia jest ciepło. Ciepło podczas pożaru jest propagowane na trzy różne sposoby:

- Przewodnictwo
- Konwekcja
- Radiacja

### 2.2.1. Przewodnictwo

Przewodnictwo cieplne jest procesem, który polega na wymianie ciepła pomiędzy nierównomiernie ogrzanymi ciałami będącymi w kontakcie. Zachodzi ono we wszystkich stanach skupienia - ciałach stałych, cieczach i gazach. Przebieg i skala tego zjawiska są zróżnicowane w zależności od stanu skupienia. Najczęściej mówimy o przewodnictwie w ciałach stałych. W cieczach i gazach dominującym zjawiskiem staje się konwekcja, przez co przewodnistwo ma pomijalnie mały wpływ na przekazywanie ciepła. niezorganizowanym, przypadkowym ruchom i ich dyfuzji. W ciałach stałych przenoszenie ciepła odbywa się na dwa sposoby:

- dzięki drganiom atomów
- poprzez ruch elektronów

Celem omawianego przewodnictwa jest osiągnięcie równowagi cieplnej. Podczas przewodnictwa ciepło jest zawsze przenoszone od ciała o większej temperaturze do ciała o niższej. Zgodnie z zasadą zachowania energii, głoszącą że w układzie izolowanym suma wszystkich energii jest stała, ilość energii uzyskanej przez ciało chłodniejsze jest równa ilości energii oddanej przez cieplejszy obiekt. Energia przenoszona jest wraz z ruchem cząsteczek wewnętrznych. Nie wszystkie ciała przewodzą ciepło w takim sam sposób. Zależność między ilością ciepła przewodzonego przez ciało, a jego zmianą temperatury najlepiej opisuje prawo Fouriera. Przyjmuje ono następującą postać:

$$q(r,t) = -k * gradT \tag{2.2}$$

gdzie: k - współczynnik przewodzenia ciepła $[W/(m*K)]$ T - temperatura $[K]$ q - natężenie strumienia ciepła $[W/(m^2)]$ Prawo Fouriera oznacza, że gęstość strumienia ciepła przekazywana w jednostce czasu przez jednostkową powierzchnię jest proporcjonalna do gradientu temperatury. Minus we wzorze wynika ze wspomnianego wyżej kierunku przepływu ciepła: od ciała cieplejszego do zimniejszego. Strumień ciepła jest mierzony w kierunku zgodnym z jego przepływem, zatem przyrost temperatury będzie miał wartość ujemną.

Do dobrych przewodników ciepła należą przede wszystkim:

- metale - w szczególności:
  - srebro
  - miedź
  - złoto
  - aluminium

Źle przewodzą ciepło:

| Materiał | Temp. $[C]$ | Współ. przewodnictwa $[W/(m * K)]$ |
|---|---|---|
| Beton | 20 | 0.84-1.3 |
| Drewno | - | 0.1-0.17 |
| Szkło crown | 20 | 0.22-0.29 |
| Azbest | 20 | 0.16-0.37 |
| Guma wulkanizowana | 20 | 0.22-0.29 |
| Miedź | 20 | 400 |
| Stal | 20 | 10 |
| Ołów | 20 | 30 |
| Powietrze | 20 | 0.025 |

Tablica 2.1: Współczynniki przewodnictwa materiałów

- drewno
- papier
- ciecze
- gazy

Zła przewodność cieczy i gazów wynika z istoty procesu przewodnictwa w tych stanach skupienia. Za wysoką wartość współczynnika przewodnictwa odpowiada ruch elektronów. Dlatego też, we wszystkich dialektrykach wartość ta przyjmuje wartości z przedziału $[0,001 - 3][W/(m * K)]$, podczas gdy w metalach może sięgać ona nawet $400[W/m * k]$

Na uwagę zasługuje też fakt, że przewodność metali maleje wraz ze wzrostem ich temperatury. Tabela 2.1 zawiera współczynniki przewodnictwa przykładowych materiałów, które zostały wykorzystane przy testowaniu algorytmu symulacji.

### 2.2.2. Konwekcja

Konwekcja, zwana też unoszeniem lub wnikaniem, jest to zgodnie ze szkolną definicją sposób przewodnictwa ciepła polegający na "unoszeniu pobranej energii cieplnej przez cząsteczki substancji i dzięki swojej wędrówce przekazywaniu energii innym cząsteczkom". Konwekcja zachodzi we wszystkich płynach, czyli zarówno cieczach jak i gazach. Nie zachodzi natomiast w ciałach stałych. Konwekcja ze względu na połączenie w sobie dwóch zjawisk:

- przekazywania ciepła
- ruchu płynów

jest zjawiskiem niezwykle skomplikowanym do teoretycznego ujęcia. Przenoszenie ciepła w konwekcji zachodzi wskutek ruchu płynu, tak więc warunkiem niezbędnym do wystąpienia zjawiska konwekcji jest ruch ośrodka. Można wyróżnić dwa podstawowe typy konwekcji, dzielące zjawisko wnikania ze względu na przyczynę ruchu ośrodka:

- konwekcja naturalna - w tym przypadku ruch płynu wywołany jest różnicami gęstości substancji znajdujących się w polu grawitacyjnym
- konwekcja wymuszona - ruch płynu spowodowany jest działaniem urządzeń zewnętrznych (wentylatorów, pomp)

Przykładem konwekcji naturalnej jest unoszenie się ciepłego powietrzna w pomieszczeniu. Ogrzane powietrze zmniejsza swoją gęstość, w wyniku czego unosi się do góry. Jego miejsce wypełnia zimne

powietrze, które w kolejnym etapie ulega ogrzaniu rozpoczynając kolejny cykl wędrówki powietrza. Ruchy powietrza wywołane zjawiskiem konwekcji tworzą tzw. prądy konwekcyjne. Konwekcja naturalna jest typem konwekcji występującym podczas pożaru.

### 2.2.3. Radiacja

Radiacja, czyli inaczej promieniowanie, jest to sposób rozchodzenia ciepła w postaci fal elektromagnetycznych. Najważniejszym aspektem przewodnictwa ciepła przez promieniowanie jest możliwość wymiany ciepła między ciałami nie stykającymi się fizycznie ze sobą. W bardzo niskich temperaturach ilość przekazywanego przy pomocy radiacji ciepła jest tak mała, że zjawisko to może być pomijane. Wzrost znaczenia promieniowania następuje wraz ze wzrostem temperatury ciał wymieniających ciepło. Przyjmuje się, że radiacja zachodzi dla ciał o temperaturach wyższych od $0$ K. Promieniowanie jest rodzajem wymiany energii, która nie wymaga żadnego nośnika. Każde ciało emituje fale. W normalnych warunkach większość promieniowania zachodzi przy udziale fal podczerwonych. Należy jednak pamiętać że w radiacji mogą brać udział także fale świetlne czy ultrafioletowe. Poza emisją promieniowania każde ciało reaguje także na fale wysyłane przez innych. Dla każdego ciała jesteśmy w stanie określić wartości trzech współczynników opisujących reakcję ciała na wiązkę promieniowania. Należą do nich:

- Absorpcyjność czyli pochłanialność
- Refleksyjność czyli odbijalność
- Przepuszczalność

Radiacja następuje we wszystkich kierunkach aż do momentu zablokowania drogi promieni przez ciało pochłaniające je. Większość ciał stałych o rozmiarach większych od kilku mikrometrów nie przepuszcza promieniowania. Na wspomnianej głębokości pod powierzchnią ciała następuje całkowita absorpcja promieniowania cieplnego. Ponadto ciała stałe mogą przepuszczać fale tylko o określonej długości. Przykładem jest szkło, które przepuszcza jedynie fale świetlne. Ilość promieniowania emitowanego przez ciała szare można obliczyć ze wzoru 2.3

$$\dot{Q_{emit}} = \sigma * \varepsilon * A_s * T_s^4 \tag{2.3}$$

gdzie

- $\varepsilon \in (0, 1)$ - emisyjność powierzchni. $\varepsilon = 1$ - dla ciała doskonale czarnego. Określa jak bardzo dane ciało jest podobne do ciała doskonale czarnego.
- $\sigma = 5.67 * 10^-8[W/(m^2 * K^4])$ - stała promieniowania
- $A_s[m^2]$ - powierzchnia
- $T_s[K]$ - temperatura

Przeanalizujmy przykład promieniowania między rzeczywistymi obiektami znajdującymi się w pewnym pomieszczeniu, np. stół w pokoju. W przypadku jednej powierzchni zamkniętej w innej (w omawianym przypadku wewnętrzną powierzchnią będzie powierzchnia stołu, natomiast zewnętrzną ściany pokoju) zakłada się, że wewnętrzna powierzchnia ńie opromieniowuje samej siebie". Innymi słowy całe promieniowania ciała wewnętrznego przechodzi do powierzchni zewnętrznej. W drugim kierunku następuje tylko częściowe przejście energii z ciała zewnętrznego do wewnątrz. Ponadto, w przypadku gdy otaczająca powierzchnia jest znacząco większa od powierzchni wewnętrznej i obie powierzchnie są oddzielone gazem, który nie promieniuje (powietrze) zjawisko promieniowania zachodzi równolegle ze zjawiskiem konwekcji i oba te zjawiska należy wziąć pod uwagę równocześnie. W takim przypadku wymianę ciepła można określić za pomocą wzoru 2.4

$$\dot{Q_{cak}} =_{cak} *\sigma * A_s * (T_s^4 - T_{\inf^4}) \tag{2.4}$$

gdzie:

- $_{cak}$ $-$ $cakowitywspczynnikwymianyciepaT_{\inf -temperaturapowietrzawznacznejodlegoci}$ Omówione powyżej procesy powodują, że promieniowanie jest zjawiskiem szczególnie skomplikowanym.

### 2.2.4. Zależność temperatury od dostarczonej energii

Opisane w podrozdziałach 2.2.1, 2.2.2, 2.2.3 metody obrazują różne sposoby przekazywania energii między cząsteczkami materii. Po ich poznaniu należy zadać sobie pytanie w jaki sposób ta energia wpływa na temperaturę substancji? Wielkością reprezentującą zależność między dostarczoną energią a temperaturą substancji jest ciepło właściwe. Ciepło właściwe jest wielkością charakterystyczną dla materiału i informuje ono o tym ile ciepła należy dostarczyć aby ograć 1kg substancji o $1^{\circ}C$.Opisaną powyżej zależność przedstawia wzór 2.5

$$c = Q/(m * \Delta T) \tag{2.5}$$

gdzie:

- c - ciepło właściwe $[J/(kg * K)]$
- m - masa ciała $[kg]$
- T - temperatura $[K]$

Znając ilość ciepła dostarczonego do ciała w wyniku procesów przekazywania energii oraz dokonując przekształcenia powyższego wzoru można w bardzo prosty sposób policzyć zmianę temperatury badanego ciała.

# 3. Automaty komórkowe

W rozdziale tym wyjaśniono pojęcie automatu komórkowego, przedstawiono formalną definicję automatów komórkowych oraz ich najczęstsze zastosowania. Podczas omawiania automatów zamieszczono ogólny algorytm symulacji z wykorzystaniem automatów komórkowych. Zasadę działania prostych automatów przedstawiono na przykładzie gry Life. Szczególną uwagę podczas opisu automatów komórkowych poświęcono niehomogenicznym automatom komórkowym, gdyż to one zostały wykorzystane do symulacji pożaru.

## 3.1. Definicja automatu komórkowego

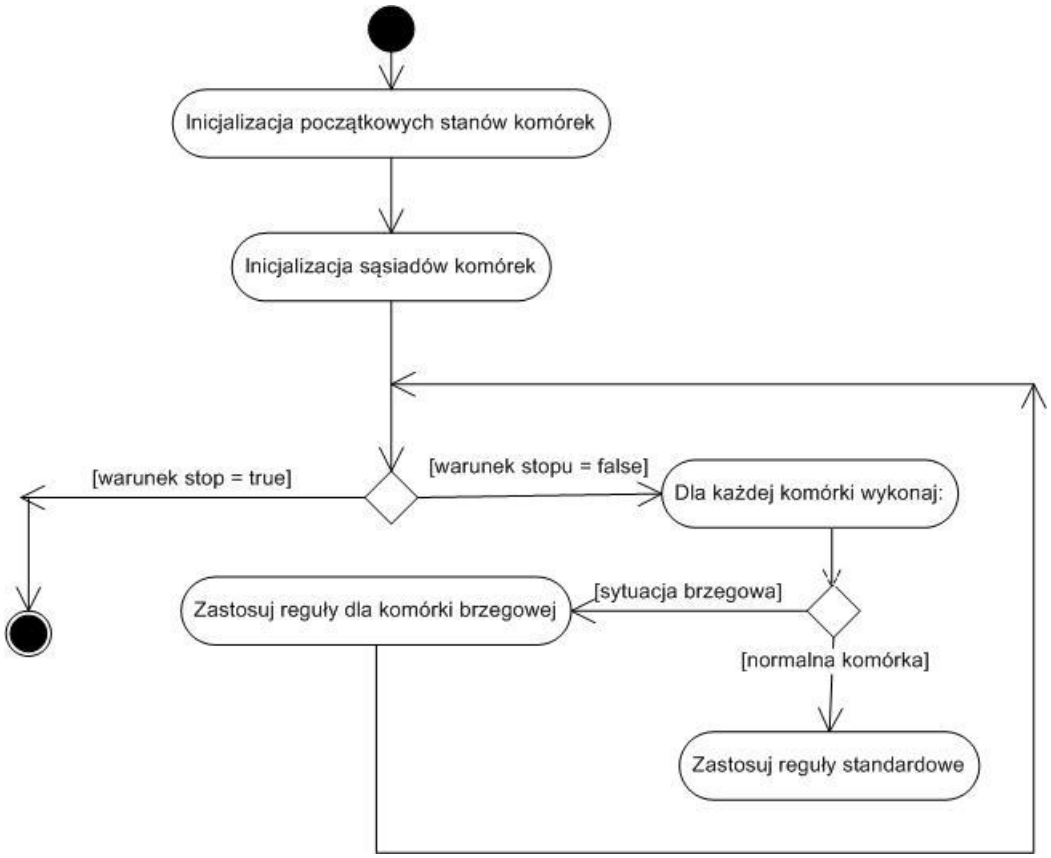
Według definicji *Ferbera* *Automat komórkowy jest dyskretnym, dynamicznych systemem, którego zachowanie jest całkowicie określone w warunkach lokalnych relacji.* Inną definicją, ukazującą automat komórkowy w matematycznym zapisie jest definicja *Weimara*, przedstawiająca automat jako czwórkę parametrów:

CellularAutomata=(L,S,N,f)(3.1) gdzie

- L - zbiór komórek tworzących automat
- S - zbiór stanów, które może przyjmować komórka
- N - zbiór sąsiadów danej komórki
- f - funkcja przejścia, która każdą komórkę ze zbioru L przeprowadza ze stanu $S_i$ w stan $S_(i+1)$. $S_t$ jest jednym ze stanów ze zbioru S przyjmowanym przez komórkę w i-tej jednostce czasu, natomiast $S_(i+1)$ jest stanem przyjmowanym przez komórkę w i+1-ej jednostce czasu na podstawie analizy stanów sąsiadów komórki (N).

Innymi słowy, automat komórkowy jest to model matematyczny opisujący siatkę komórek, ich stany oraz reguły przejść między kolejnymi stanami. Każda komórka z pewnej dyskretnej siatki komórek może przyjmować jeden z określonego zbioru stanów. W dyskretnych przedziałach czasowych następuje ewolucja komórki, na podstawie jej *stanu poprzedniego* oraz *stanu jej sąsiadów*. Ewolucję komórki określa funkcja przejścia. Odpowiedni dobór zbioru stanów oraz funkcji przejść kształtuje cechy automatu, jego dynamikę oraz szybkość. W wyniku ewolucji komórka przyjmuje kolejny stan ze zbioru S. Ze względu na wielowymiarowość automatów komórkowych, sąsiedztwa można definiować w różny sposób. W przypadku dwuwymiarowego automatu najprostszym sąsiedztwem, zwanym też sąsiedztwem von Neumana będzie zbiór czterech komórek N,S,E,W gdzie kolejne literki oznaczają kierunki zgodne z różą wiatrów. W dwuwymiarowych automacie jako sąsiedztwo można przyjmować także zbiór ośmiu komórek (sąsiedztwo *Moore'a*: cztery wspomniane powyżej kierunki wraz z kierunkami pośrednimi N, S, E, W, NW, NE, SE, SW. W trójwymiarowym automacie najprostszym sąsiedztwem jest zbiór wszystkich komórek połączonych ścianami z aktualnym sześcianem. Innym sąsiedztwem mogą być wszystkie sześciany połączone z aktualnie badanym zarówno ścianami jak i krawędziami. Ważnymi elementami automatu komórkowego jest stan początkowy automatu, czyli z góry określony stan każdej komórki w

chwili 0 (przed pierwszą iteracją algorytmu). W przypadku automatów komórkowych o ograniczonej siatce konieczne jest także zdefiniowanie warunków brzegowych. Warunki brzegowe określają reguły przejść dla komórek znajdujących się na brzegach automatu. W przypadku jednowymiarowej siatki, przykładem warunku brzegowego jest określenie sąsiadów ostatniego $n - tego$ elementu jako $n - 1$ oraz 1 - przedostatni oraz pierwszy element. Poniżej przedstawiono ogólny schemat działania algorytmu komórkowego:




Automaty komórkowe dzięki możliwościom zastąpienia bardzo skomplikowanych wzorów prostymi regułami są szeroko stosowane do modelowania procesów fizycznych i chemicznych np. symulacje pożarów lasów, budynków, rozprzestrzenianie lekarstw w organizmie ludzkim. Innym zastosowaniem jest symulacja zjawisk, które ze względu na swoją naturę, w wyniku braku znajomości dokładnych wzorów nie mogą być odzwierciedlone w sposób dokładny. Przykładem takiej symulacji jest na przykład ruch ludzi podczas ucieczki z ewakuowanego budynku.

## 3.2. Klasyfikacja automatów komórkowych

Od wprowadzenia pojęcia automatu komórkowego, które datuje się na lata 40-ste XX-go wieku powstały różne metody klasyfikacji automatów komórkowych. Jedną z najważniejszych jest podział ze względu na homogeniczność automatu. *Homogenicznym automatem komórkowym* nazywamy, automat który spełnia wszystkie postulaty homogeniczności. Należą do nich:

- jednakowy zbiór stanów dla każdej komórki
- jednakowy zbiór reguł dla każdej komórki
- stały obszar siatki automatu

- jednakowy schemat określający sąsiadów dla każdej komórki
- jednakowa metoda aktualizacji wszystkich komórek

Jeżeli automat nie spełnia, *któregokolwiek* z wyżej wymienionych warunków jest klasyfikowany jako *niehomogeniczny automat komórkowy*.

Najprostszym przykładem klasycznego, homogenicznego automatu komórkowego jest gra *Life*. W grze Life mamy zazwyczaj do czynienia z nieskończoną planszą Każda z komórek siatki może przyjmować dwa stany: jest żywa lub martwa. Każda z komórek posiada ośmiu sąsiadów - są to komórki przylegające krawędziami i rogami.W grze tej wszystkie komórki zmieniają swój stan, co pewien ustalony odstęp czasu. Nowy stan komórki jest obliczany wyłącznie na podstawie jej poprzedniego stanu oraz stanów sąsiadów. Metoda aktualizacji komórek oraz schemat określający nowy stan jest identyczny dla wszystkich komórek automatu. Mimo nazwy omawianej symulacji jedynym udziałem człowieka w tej grze jest ustawienie stanu początkowego komórek.

Innymi sposobami klasyfikacji automatów komórkowych jest podział ze względu na sąsiedztwo (sąsiedztwo Moore'a oraz von Neumana), ze względu na wielowymiarowość planszy (jedno-,dwu-,trójwymiarowa,...), a także ze względu na kształt pojedynczej komórki. W jednowymiarowym automacie komórką jest odcinek, w dwuwymiarowym najprostszym wariantem jest kwadrat, a w trójwymiarowej sześcian. Różnie między tymi automatami zostały częściowo omówione w rozdziale poprzednim.

# 4. Algorytm

Rozdział przedstawia propozycję algorytmu symulacji rozprzestrzeniania się ognia i dymu podczas pożaru. Przedstawiony poniżej model jest niehomogenicznym automatem komórkowym i jako taki spełnia postulat niehomogeniczności. W pierwszej części rozdziału przedstawiono wartości parametrów tworzących automat komórkowy. Kolejne podrozdziały zawierają szczegółowy opis kluczowych funkcji współtworzących funkcję przejścia.

## 4.1. Model automatu

Zgodnie ze wzorem 3.1 będącym istotą przytoczonej w rozdziale 3 definicji automatu komórkowego według Weimara jednym z kluczowych elementów podczas tworzenia automatu jest określenie siatki, czyli powierzchni automatu. W modelu symulacji pożaru w budynku ze względu na trójwymiarowość zjawiska oraz istotę jego rzeczywistego odtworzenia (możliwość wykorzystania wyników w celu opracowania modelu ewakuacji osób, badanie przyczyn katastrofy i drogi rozchodzenia ognia) najbardziej naturalnym typem automatu jest automat *trójwymiarowy*. Powierzchnię automatu stanowi sześcian składający się z również sześciennych komórek o wymiarach $0.5mx0.5mx0.5m$.Odpowiedni dobór wielkości komórek automatu ma kluczowy wpływ na jego działanie. Zbyt mała ilość komórek może doprowadzić do utraty dokładności algorytmu oraz ukazać zniekształcony obraz działania modelu. Zbyt duża ilość elementów powoduje spadek wydajności algorytmu, a w komputerowej realizacji algorytmu oznacza zwiększone zapotrzebowanie na pamięć i moc procesora. W omawianym algorytmie wielkość komórek została wybrana empirycznie. Wybrany na podstawie doświadczeń rozmiar komórki jest najlepszym kompromisem między czasem działania a dokładnością modelu. Rozmiar całkowitej powierzchni automatu jest wielkością zmienną, definiowaną przez użytkownika systemu. Pozwala to na przeprowadzanie symulacji budynków o zróżnicowanej wielkości dopasowując rozmiar automatu tak aby całość siatki stanowiła budynek oraz aby budynek był w całości objęty przez siatkę.

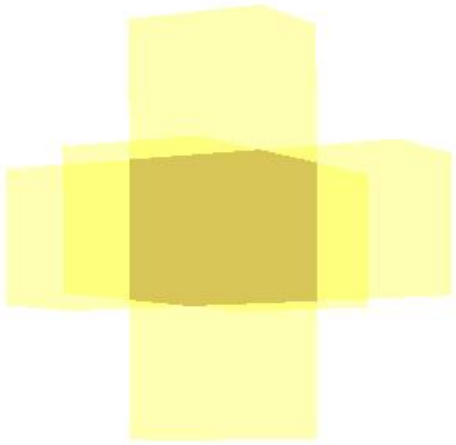
Typy komórek wchodzących w skład automatu można podzielić na dwie zasadnicze grupy:

- Ciała stałe
- Gazy

Model symulacji nie uwzględnia interakcji ognia z wodą lub innymi cieczami i nie przedstawia zjawisk fizycznych zachodzących podczas tych interakcji. W ciałach stałych funkcje przejścia odzwierciedlają zjawisko przewodnictwa cieplnego. W gazach będących płynami przewodnictwo zastąpione jest konwekcją. Wszystkie typy komórek poddane są zjawisku radiacji. Ponadto, komórki reprezentujące ciała stałe dzielimy ze względu na rodzaj materiału z jakiego są stworzone.

Każdy z materiałów posiada zestaw parametrów określających jego właściwości fizyczne:

- Ciepło właściwe - określa jak zmienia się temperatura ciała w zależności od ilości dostarczonego / oddanego ciepła
- Gęstość
- Współczynnik przewodnictwa ciepła - określa zdolność materiału do przewodnictwa cieplnego



Rysunek 4.1: Schemat sąsiedztwa

- Temperatura zapłonu - określa temperaturę charakterystyczną dla danego materiału, po której osiągnięciu dochodzi do produkcji palnych oparów
- Palność - określa procentową ilość oparów powstałych po osiągnięciu temperatury zapłonu. Determinuje szybkość spalania.

Wyżej wymienione parametry bezpośrednio wpływają na zachowanie funkcji przejścia, powodując zróżnicowane zachowanie komórek w zależności od typu materiału.

Poza różnymi typami komórek, o niehomogeniczności automatu świadczą różne definicje sąsiedztwa. Ze względu na fakt, że siatka automatu modeluje przestrzeń zamkniętą - budynek - konieczne jest zróżnicowanie sąsiedztwa w środku siatki oraz na jej brzegach. W zaproponowanym algorytmie wykorzystano zmienną liczbę sąsiadów w zależności od położenia komórki. Komórka znajdująca się w środku siatki posiada sześciu sąsiadów. Sąsiadami są komórki przylegające ścianami do aktualnie rozpatrywanej, co jest trójwymiarowym wariantem sąsiedztwa von Neumana. Rozkład sąsiadów dla komórki znajdującej się w centrum przestrzeni przedstawia rysunek 4.1. W przypadku gdy komórka znajduje się na skraju siatki liczba sąsiadów ulega zmniejszeniu. Do zbioru sąsiadów należą komórki, przylegające ścianami do rozpatrywanej oraz jednocześnie będące wewnątrz przestrzeni modelu. W skrajnym przypadku, gdy komórka znajduje się w narożniku liczba sąsiadów z sześciu spada do trzech. Stan komórki brzegowej jest obliczany, podobnie jak w przypadku komórki znajdującej się wewnątrz automatu na podstawie wszystkich jej sąsiadów. Zmniejszona ilość sąsiadów powoduje zmieniony rozkład ich wpływu na nowy stan bieżącej komórki. Waga znaczenia każdej z komórek wzrasta dwukrotnie. Innymi, alternatywnymi rozwiązaniami sytuacji brzegowych są :

1. Uznanie za sąsiada ostatniego elementu w danej płaszczyźnie, elementu pierwszego czyli znajdującego się na przeciwległym brzegu. Jest to tak zwane sąsiedztwo periodyczne. W przypadku symulacji pożaru taki tym sąsiedztwa nie odzwierciedla rzeczywistych interakcji między komórkami w pomieszczeniu. Komórka znajdująca się po drugiej stronie budynku nie wpływa bezpośrednio na stan aktualnie rozpatrywanej.
2. Zastosowanie warunków pochłaniających, czyli nadanie komórkom brzegowym z góry określonego, nie uwzględniającego sąsiedztwa stanu. Rozwiązanie to powoduje, że komórki brzegowe nie mogą być traktowane jako elementy budynku z rzeczywistymi właściwościami fizycznymi.

Każda z komórek automatu może przyjmować jeden z trzech stanów:

- Komórka niezapalona
- Komórka paląca się
- Dym

Poza wyżej wymienionymi trzema stanami, każda komórka posiada wartość swojej aktualnej temperatury, która również w sposób pośredni określa jej stan i ma znaczący wpływ na wynik funkcji przejścia.

# 5. Implementacja

Poniższy rozdział zawiera przegląd ciekawszych aspektów implementacyjnych projektu. Na wstępie omówiono użyte technologie oraz narzędzia wraz z uzasadnieniem wyboru. Następnie przedstawiono dokładny model aplikacji z uwzględnieniem wybranych metod implementacyjnych. W rozdziale tym przedstawiono także problemy, z którymi zetknęli się autorzy podczas realizacji projektu oraz sposoby ich rozwiązania.

## 5.1. Wybrane technologie i narzędzia

Przed przystąpieniem do implementacji końcowej wersji aplikacji powstawały prototypy testujące zarówno różne podejścia algorytmiczne opisane w rozdziale 4 jak i możliwe do wykorzystania narzędzia. W sekcji *Język implementacji* przedstawiono język wybrany do implementacji końcowej wersji programu, natomiast w sekcji *Języki prototypowania* przedstawiono technologie używane do tworzenia prototypów systemu. W kolejnym podrozdziale pokrótce zaprezentowano narzędzia pracy grupowej wykorzystywane przez autorów pracy. Na końcu wymieniono biblioteki i narzędzia, które okazały się pomocne podczas realizacji projektu Sparkle i które zostaną szczegółowo omówione w kolejnych rozdziałach.

### 5.1.1. Język implementacji

Do implementacji końcowej wersji aplikacji został wybrany język Java. Język ten został wybrany ze względu na przenośność aplikacji, obiektowość ułatwiającą projektowanie, dostęp do licznych bibliotek oraz mechanizmy pozwalające uchronić program przed wyciekami pamięci krytycznymi z punktu widzenia aplikacji czasu rzeczywistego. Innym, rozważanym językiem implementacji był język C++. Przed końcowym wyborem zostały przeprowadzone badania dotyczące wydajności rozważanych języków. Przyniosły one następujące rezultaty:

### 5.1.2. Języki prototypowania

Jednym z powstałych ptotypów był prosty automat komórkowy z regułami opisanymi w paragrafie [WSTAWIC ODNOSNIK DO PARAGRAGU]. Do implementacji tego modelu został korzystany język Common Lisp.

Common Lisp to nowoczesny dialekt Lispu - drugiej najstarszej rodziny używanych obecnie wysokopoziomowych języków programowania. Common Lisp jest językiem kompilowalnym, dynamicznie typowanym i umożliwia tworzenie na znacznie wyższym poziomie abstrakcji niż C++ czy Java. Samo programowanie w tym języku opiera się głównie o stopniową rozbudowę działającego obrazu programu poprzez REPL (ang. Read-Eval-Print Loop) - pętlę tworzącą swoisty interpreter, pozwalającą modyfikować, dokompilowywać i wymieniać fragmenty kodu w czasie działania samego programu. Te wszystkie cechy języka Common Lisp tworzą z niego idealne narzędzie do szybkiego prototypowania programów. Istotnie, omawiany prototyp automatu komórkowego powstał w przeciągo około dziesięciu godzin.

Drugim, powstałym podczas badań pierwowzorem był program napisany w języku Java. Celem tego modelu było przetestowanie języka Java oraz biblioteki Java3D pod kątem możliwości wykorzystania do implementacji symulacji czasu rzeczywistego w oparciu o uproszczony model automatu.

### 5.1.3. Narzędzia pracy grupowej

Ze względu na fakt, że omawiana aplikacja powstała jako wynik pracy dwuosobowego zespołu, bardzo przydatnym narzędziem okazał się system kontroli wersji. Jako system zarządzania wersjami został wybrany Git. Git jest darmowym, rozproszonym systemem, opartym na architekturze peer-to-peer. W przeciwieństwie do scentralizowanych odpowiedników (np. SVN) nie posiada jednego, centralnego repozytorium z którym członkowie zespołu synchronizują swoje zmiany ale całkowicie niezależne repozytoria, które można synchronizować w różnorodny sposób.

Git, będąc rozproszonym systemem kontroli wersji, posiada dwie istotne cechy kluczowe dla rozwoju niniejszej pracy. Po pierwsze, techniczna równorzędność wszystkich kopii repozytoriów pozwoliła na bezpieczny i niezależny rozwój projektu nawet w sytuacjach, gdy komputer odłączony był od sieci Internet. Możliwość niezależnej pracy na lokalnych repozytoriach skutecznie zachęca do rejestrowania w systemie kontroli wersji nawet drobnych zmian, gdyż synchronizację pomiędzy poszczególnymi developerami można odłożyć do stosownego momentu. Drugą ważną właściwością Gita jest łatwość, z jaką łączy on zmiany wprowadzone przez rózne osoby w tym samych plikach (ang. "merging"). Brak problemów z łączeniem różnych ścieżek rozwoju - problemów znanych z tradycyjnych, centralnych systemów kontroli wersji - pozwolił na szybki rozwój projektu.

### 5.1.4. Inne narzędzia

1. Implementacja prototypu w Common Lisp została oparta o edytor Emacs oraz biblioteki SDL i OpenGL.

2. Do implementacji projektu zostały wykorzystane środowiska programistyczne: Eclipse i NetBeans. Użycie systemu kontroli wersji do wymiany plików źródłowych aplikacji dało możliwość niezależniej pracy w różnych środowiskach.

3. Podczas realizacji Graficznego Interfejsu Użytkownika został wykorzystany program WindowBuilder Pro.

4. Wizualizacja symulacji została zaimplementowana z wykorzystaniem biblioteki Java3D.

5. Metody tworzące algorytmikę aplikacji zostały przetestowane z wykorzystniem biblioteki JUnit4.

6. Do badania pokrycia kodu testami jednostkowymi zostało wykorzystane narzędzie Emma.

## 5.2. Implementacja modelu aplikacji

Zgodnie z architekturą aplikacji omówioną w rozdziale **??** program został podzielony na

- Moduł wizualizacji danych
- Moduł realizujący logikę aplikacji
- Moduł kontrolera, odpowiadający za interakcje z użytkownikiem.

Poniżej przedstawiono najważniejsze implementacyjne aspekty realizacji poszczególnych modułów. Opis implementacji wzbogacony został o diagramy klas UML przedstawiające budowę poszczególnych pakietów, zwracające uwagę na powiązania między jednostkami oraz uwypuklające wybrane metody implementacyjne.

### 5.2.1. Moduł wizualizacji

Wizualizacja symulacji została zaimplementowana z wykorzystaniem Java3D API. Java3D to interfejs programowy (API) umożliwiający tworzenie trójwymiarowej grafiki. Jest to produkt firmy Sun Microsystems wykorzystujący w swoim działaniu renderery OpenGL i Direct3D w zależności od platformy. Główną zaletą determinująca wybór bilioteki Java3D jest umożliwienie wykorzystania technologii OpenGL i DirectX poprzez dostarczenie wysokopoziomowych struktur pozwalających zachować pełnię obiektowości języka Java. Zastosowana w bibliotece Java3D obiektowość ułatwia zarządzanie elementami sceny oraz umożliwia skupienie uwagi na logice aplikacji bez zagłębiania się w szczegóły renderingu. Inną, niezmiernie ważną zaletą biblioteki Java3D jest optymalizacja procesów renderingu.

#### Instalacja biblioteki Java3D

Instalacja biblioteki Java3D jest stosunkowo prosta i może przebiegać na dwa sposoby. Pierwszym sposobem jest ściącienie ze strony

https://java3d.dev.java.net/binary-builds.html

archiwum zip. Po jego rozpakowaniu należy, postępując zgodnie z instrukcją zawartą w pliku README-unzip.html, zmodyfikować zmienną CLASSPATH dodając ścieżki do wymienionych w instrukcji plików z rozszrzeniem .jar oraz zmodyfikować zmienną PATH dodając ścieżkę do katalogu libı386.

Alternatywą jest pobranie ze strony firmy Oracle

http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html

instalatora, który pod dwukrotnym kliknięciu przeprowadzi użytkownika przez proces instalacyjny. W wyniku instalacji, Java Runtime Environment System Library zostanie wzbogacone o pliki j3dcore.jar i j3dutils.jar.

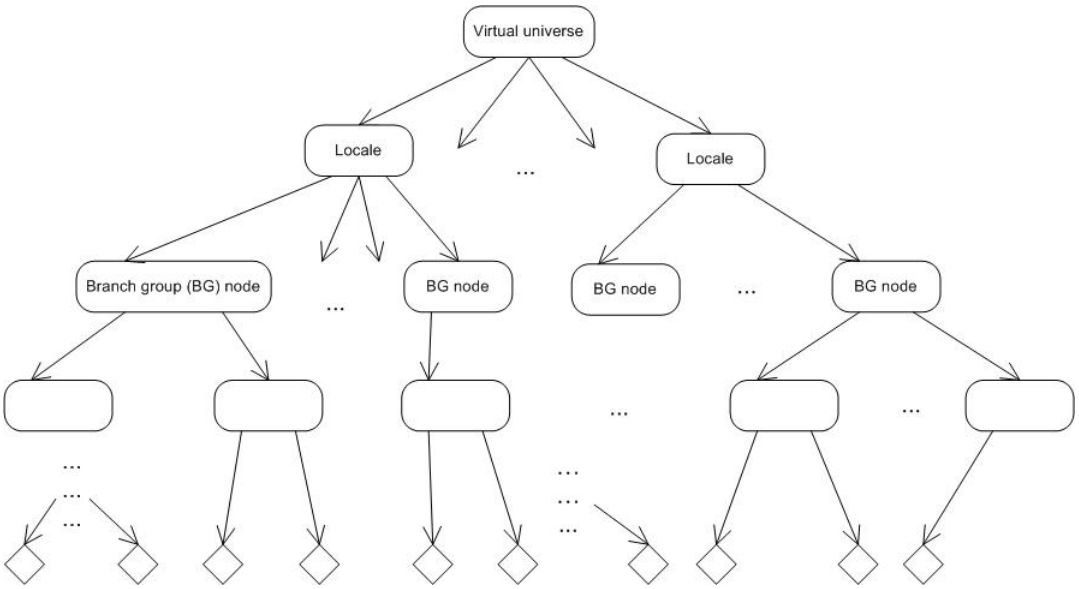
#### Budowa sceny z wykorzystaniem struktur Java3D API

Głównym elementem biblioteki Java3D jest graf sceny (*scene graph*) rerezentujący cały wirtualny wszechświat (*virtual universe*). Graf sceny jest acyklicznym grafem skierowanym. Użytkownik tworzy podgrafy reprezentujące pewne wycinki rzeczywistości, które następnie są dołączane jako dzieci do wirtualnego wrzechświata. Każdy z podgrafów jest renderowany osobno i niezależnie od pozostałych, co pozwala na zrównoleglenie obliczeń. Strukturę grafu sceny przedstawia diagram 5.1. Wśród jego wierzchołków możemy wyróżnić:

- Group nodes - wierzchołki reprezentujące grupę elementów. Mogą zawierać zero lub więcej dzieci. Grupowanie elementów umożliwia wykonywanie operacji dla całej grupy. Na diagramie 5.1 przedstawione są jako zaokrąglone kształty.
- Leaf nodes - liście reprezentują obiekt tworzący scenę (geometrię, oświtlenie, dźwięk). Nie mogą posiadać dzieci. Na diagramie 5.1 występują w postaci rombów.

Korzeniem całego grafu jest obiekt klasy *VirtualUniverse* lub dowolnej, dziedziczącej po niej klasie. Graf sceny może posiadać wiele wierzchołków, jednak w większości przypadków jeden jest wystarczający. Obiekt klasy *Locale* jest kontenerem przechowującym kolekcję podgrafów, których korzeniami są instancje klasy *BranchGroup*. Obiekt klasy *BranchGroup* powinien być postrzegany jako jednostka, która po stworzeniu jest kompilowana, a następnie dołączana do grafu reprezentującego wirtualny wszechświat. W dowolnym momencie może zostać odłączana i powtórnie dołączona w inne miejsce. Tworzenie obiektu 3D przy użyciu Java3D API przebiega następująco:

1. Tworzony jest obiekt. Ustalane są jego właściwości.
2. Stworzony obiekt dołączany jest do odpowiedniego grafu reprezentowanego przez obiekt BranchGroup.




Rysunek 5.1: Java3D - reprezentacja sceny

3. Powstałe grafy w ten sposób grafy są łączone w jedną strukturę reprezentującą wirtualny wszechświat.

Dobrą praktyką jest wspomniane powyżej kompilowanie stworzonych podgrafów przed ich połączeniem. Nie jest ono konieczne ale powoduje zamianę grafu do wewnętrznego formatu umożliwiającego jego optymalizację. Na uwagę zasługuje fakt, że domyślnie obiekty mogą być modyfikowane jedynie podczas tworzenia podgrafu. Po dołączeniu już stworzonego podgrafu do większej całości informacja o jego obiektach jest oddzielnie zapamiętywana, w sposób przyspieszający operacje dostępu do danych. Można to zmienić wywołując podczas tworzenia obiektu metodę *setCapability* z odpowiednim parametrem.

Przedstawiony powyżej graf sceny jest logiczną reprezentacją renderowanych obiektów. Do renderingu przygotowanej sceny konieczny jest obiekt klasy Viewer. Zawiera on wszelkie informacje potrzebne do stworzenia obrazu całej sceny. Z obiektem klasy VirtualUniverse połączony jest poprzez instancję klasy ViewingPlatform, która inicjalizuje obiekt klasy Viewer dla stworzoneg grafu. Obiekt klasy Viewer posiada referencje do obiektów klasy View, będących jego lokalnym odpowiednikiem. Obiekt klasy View zawiera wszystkie iformacje potrzebne do renderingu pojedynczego elementu sceny oraz referencję do instancji klasy Canvas3D, gdzie opisywany element ma być wyrenderowany. Jest on połączony z każdym liściem grafu sceny poprzez instancję klasy *ViewPlatform*. Klasa ViewPlatform odpowiada za inicjalizację obiektu View oraz pozycję, orientację i skalę opisywanego wierzchołka grafu sceny.

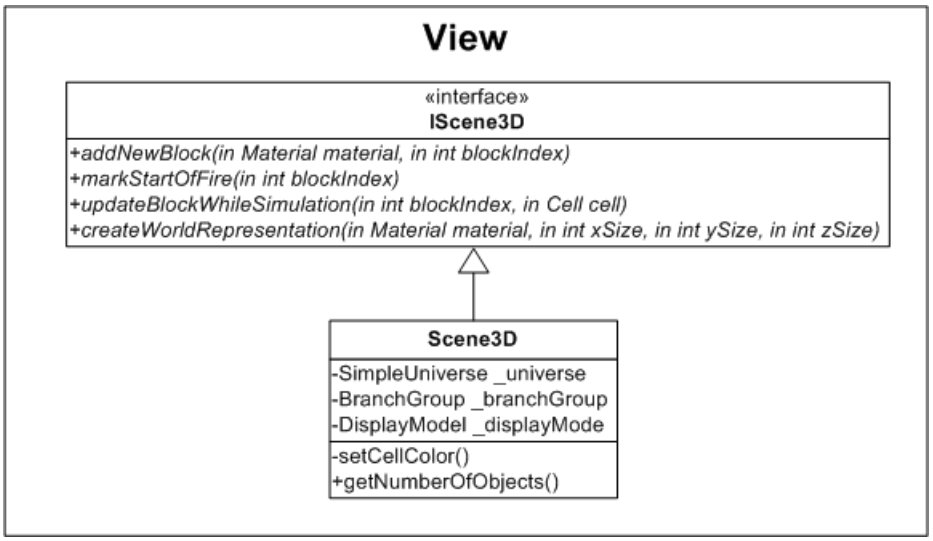
### Implementacja modułu wizualizacji

Moduł wizualizacji zaimplementowany w pakiecie View jest realizacją schematu przedsawionego na diagramie 5.2.

Klasa Scene3D, implementująca interfejs IScene3D, realizuje przedstawioną powyżej budowę sceny z wykorzystaniem interfejsu Java3D. Składa się z następujących atrybutów:

- _universe - Do budowy korzenia grafu sceny została wykorzystana klasa *SimpleUniverse*. SimpleUniverse pozwala na najprostszą formę inicjalizacji grafu sceny z wykorzystaniem domyślnie tworzonych obiektów klasy Viewer oraz *ViewingPlatform*.

- _branchGroup - Obiekt klasy BranchGroup jest podgrafem reprezentującym tworzony budynek. Jest on listą obiektów klasy *TransformGroup*, reprezentujących pojedyncze komórki automatu.




Rysunek 5.2: Moduł View

Zastosowanie klasy *TransformGroup* umożliwia przemieszczenia poszczególnych elementów budynku w trakcie działania aplikacji.

- _displayMode - DisplyMode odpowiada za tryb wyświetlania wyników symulacji. Może przyjmować jedną z dwóch wartości: TEMPERATURE - powodującą przejście symulacji w tryb wyświetlania rozkładu temperatur podczas pożaru lub REGULAR - przedstawiającą aktualny stan komórki.

Interfejs IScene3D dostarcza metody umożliwiające wizualizację symulacji. Poniżej przedstawiono ich krótką charakterystykę:

- *addNewBlock* - dodaje do widoku sceny nowy element knstrukcyjny poprzez zaaplikowanie atrybutów określających wygląd wskazego materiału do określonej przez *blockIndex* komórki. BlockIndex jest numerem odpowiadającego elementu z listy *_branchGroup*. Index jest wyliczany w klasie World, z której wywoływana jest omawiana metoda.

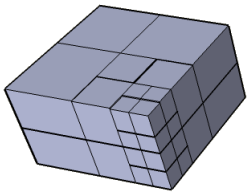
- *markStartOfFire* - pokazuje źródło pożaru poprzed pokolorowanie odpowiedniej komórki.

- *updateBlockWhileSimulation* - zmienia wygląd komórki określonej przez *blockIndex* zgodnie z wartościami jej temperatury i materiału oraz uwzględniając aktualny tryb wyświetlania symulacji.

- *createWorldRepresentation* - tworzy wizualizację automatu komórkowego. Podczątkowy cały automat składa się z komórek wypełnionych tlenem. Metoda *createWorldRepresentation* odpowiada za wypełnienie automatu o wskazanych rozmiarach wytypowanym materiałem (sugerowanym materiałem jest tlen). Tak przygotowany automat możliwy jest do modyfikacji poprzez dodawanie kolejnych elementów konstrukcyjnych lub w wyniku przeprowadzonej symulacji.

### 5.2.2. Logika aplikacji

Logika aplikacji zaimplementowana jest w obrębie pakietu *Model*. Jego budowę przedstawia diagram 5.4. Diagram klas poza wewnętrzną budową modułu obrazuje jego interakcje z klasami zewnętrznymi, pochodzącymi z innych modułów. W przypadku Modelu jest to jedynie klasa *Scene3D* z pakietu *View*. Główną klasą pakietu *Model* jest World. Zawiera ona strukturę danych reprezentującą autmat komórkowy oraz zestaw metod umożliwiających przeprowadzanie symulacji.



Rysunek 5.3: Podział sceny przy pomocy drzewa ósemkowego

**World**

Klasa World została zaimplementowana jako *Singleton*, ze względu na fakt jest unikalności w obrębie aplikacji.

Strukturą danych, która posłużyła do przechowywania automatu jest trójwymiarowa tablica. Innym kontenerem bardzo często wykorzystywanym do reprezentacji przedstawianego świata w symulacjach jest drzewo ósemkowe, którego użycie również zostało rozważone podczas realizacji projektu Sparkle. Poniżej prezentowane są krótkie rozważania na temat wymienionych struktur wraz z uzasdanieniem wyboru.

Drzewo ósemkowe to strukturą pozwalająca podzielić trójwymiarowy świat na mniejsze, regularne części. Jest ono tworzone zgodnie z następującymi regułami:

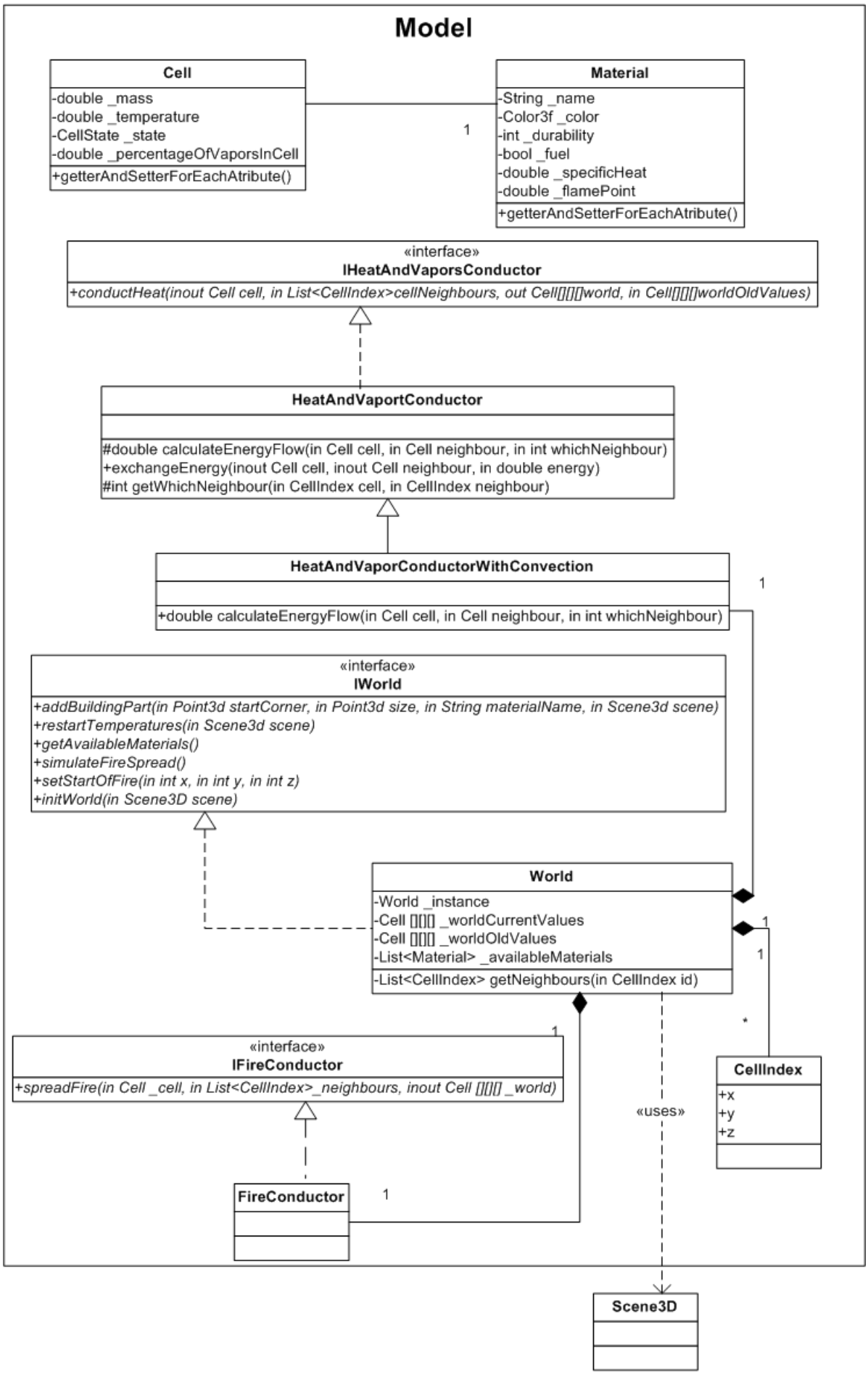
- Korzeniem drzewa jest sześcian, w którym zawarty jest modelowany świat.
- Każdy z wierzchołków reprezentuje sześcian będący częścią całego obiektu.
- Każdy z wierzchołków nie będący liściem posiada osiem dzieci, reprezentujących sześciany w nim zawarte zgodnie z rysunkiem 5.3.

Drzewo ósemkowe daje możliwość optymalizacji wykrozystywanej pamięci poprzez przechowywanie jedynie komórek, których stan różni się od stanu rodzica. Komórki identyczne z rodzicem są przez niego reprezentowane, a rodzic taki staje się liściem drzewa. Taki sposób przechowania danych jest szczególnie korzystny w przypadku dużej ilości komórek nie biorących udziału w symulacji. Jest też często wykorzystywany przy renderingu sceny składającej się w znacznej mierze z powietrza, które można zaniedbać.

W przypadku symulacji pożaru, próba adaptacji omawianej struktury okazała się nieefektywna. Główną przyczyną jest dynamika modelowanego systemu, w którym stan większości komórek zmienia się w krótkim czasie po uruchomieniu symulacji, a kolejne zmiany następują z dużą częstotliwością. Algorytmy propagacji ciepła powodują, że możliwość wyodrębnienia sześcianu którego części składowe miałyby stan identyczny z rodzicem należy do rzadkości. Powoduje to, że korzyści płynące ze stosowania wyżej omówionej optymalizacji w przypadku symulacji pożaru są znikome. Bardzo istotnym faktem jest także dodatkowe zużycie pamięci wynikające ze stosowania wskaźnikowej struktury danych. W przypadku gdy drzewo ósemkowe jest drzewem pełnym (każdy z wierzchołków nie będących liściem posiada osiem dzieci) ilość pamięci zużytej przez drzewiastą reprezentajcę przewyższa ilość wykorzystywaną w przypadku implementacji tablicowej.

Klasa World przechowuje macierz komórek w dwóch egzemplarzach. Podwójne przechowywanie wartości automatu umożliwia odczytywanie parametrów komórki z jednej tablicy, a ich uaktualnianie w drugiej. Taki zabieg nosi nazwę *podwójne buforowanie* czyli *double-buffering* i pozwala uniknąć kierunkowości symulacji. W przypadku nie zastosowania powyższej taktyki temperatura podczas symulacji pożaru rozchodzi się zawsze zgodnie z kierunkiem aktualizacji komórek. Instancja klasy World przechowuje także listę materiałów, które mogą być elementami konstrukcyjnymi budynku. Klasa World zawiera wewnętrzną klasę pomocnicza CellIndex, która przedstawia index komórki w postaci trzech współrzędnych (x,y,z).

Klasa World impleentuje interfejs IWorld, do którego metod należy:




Rysunek 5.4: Moduł Model

- *addBuildingPart* - do odpowiedzialności tej metody należy zmiana materiału komórek wchodzących w skład dodawanego bloku, a także wywołanie odpowiedniej metody klasy Scene3D dokonującej akualizacji widoku podczas edycji budynku. Zadaniem metody addBuildingPart jest zamiana indeksu komórki zapisanego w postaci trójki liczb (x,y,z) na numer odpowiadającej komórki w liście.

- *restartTemperatures* - metoda wywoływana podczas restartu symulacji. Nadaje wszystkim komórkom parametry domyślne.

- *getAvailableMaterials* - zwraca listę materiałów, z którym można konstruować budynek.

- *simulateFireSpread* - wykonuje jeden przebieg algorytmu aktualizacji automatu komórkowego, wywołując w tym celu metody klas HeatAndVaporConductorWithConvection i FireConducter. Metoda ta jest także odpowiedzialna za wywołanie aktualizacji widoku po każdym przebiegu pętli. Odpowiada za aktualizację macierzy automatu tak aby w każdej chwili macierz, z której czytane są dane była możliwie aktualna z zachowaniem spójności.

- *setStartOfFire* - ustala punkt startowy pożaru, zmieniając odpowiednio wartości wskazanej komórki.

- *initWorld* - inicjalizuje autmat wartościami domyślnymi. Domyślnym materiałem wypełniającym przestrzeń jest Tlen. Temperatura komórek na początku działania aplikacji ustawiana jest na $20^{\circ}C$. Metoda ta odpowiada także, za wywołanie funkcji tworzącej graficzną reprezentację modelu (*createdWorldRepresentation* z klasy *Scene3D*).

Ponadto klasa World zawiera metody pomocnicze, do których należy:

- *getNeighbours* - zwracająca listę indeksów sąsiadów badanej komórki

- *updateOldValues* - aktualizująca macierz wartości do odczytu

#### Cell i Material

To klasy, które zgodnie z opisem zawartym w rozdziale **??** zawierają zestaw prywatnych atrybutów określających odpowiednio stan komórki i właściwości materiału. Z każdym atrybutem związane są metody get i set, kontrolujące dostęp do zmiennych.

#### FireConductor

Klasa FireConductor odpowiada za rozprzestrzenianie ognia zgodnie z algorytmem opiany w rozdziale 4. Implementuje interfejs IFireConductor zawierający metodę *spreadFire*. Metoda ta jest odpowiedzialna za zmianę stanu komórki z naturalnego na palący i odwrotnie.

#### HeatAndVaporsConductor oraz HeatAndVaporConductorWithConvection

Do bardziej rozbudowanych klas należy HeatAndVaporsConductor oraz HeatAndVaporConductorWithConvection. Klasa HeatAndVaporsConductor implementuje interfejs IHeatAndVaporsConductor, definiując metodę *conductHeat*. Metoda ta wywoływana jest dla każdej komórki automatu, w każdym obiegu algorytmu. Odpowiada za obliczenie energii przepływającej między badaną komórką i jej wszystkimi sąsiadami oraz symulację wymiany ciepła, której wynikiem jest uaktualnienie temperatur komórek. Wszystkie obliczenia przeprowadzane są zgodnie z metodami odpisanymi w rozdziale *Algorytm*. W ich realizacji wykorzystywane są pomocnicze metody:

- *calculateEnergyFlow* - odpowiadająca za obliczenie ilości energii przepływającej między dwoma komórkami
  oraz

- *exchangeEnergy* - symulująca wymianę ciepła.

Klasa HeatAndVaporsConductor podczas wyliczania przepływającej energii uwzględnia jedynie zjawisko przewodnictwa cieplnego, zaniedbując konwekcję.

Symulacja konwekcji, jak wyjaśniono w rozdziale 4 opiera się na ułatwieniu przepływu ciepła do górnych sąsiadów komórki. Konwekcja jest więc szczególnym przypadkiem przewodnictwa, występującym w powietrzu. Zgodnie z powyższym, klasą realizującą konwekcję jest HeatAndVaporConductorWithConvection dziedzicząca po klasie HeatAndVaporsConductor. Przedefiniowana w klasie potomnej metoda *calculateEnergyFlow* wywołuje odpowiadającą funkcję klasy nadrzędnej, a następnie wprowadza do wyliczonej zgodnie z regułami przewodnictwa energii zmiany wywołane konwekcją. Zmiany te aplikowane są tylko w przypadku gdy badana komórka oraz komórka sąsiednia, z którą następuje wymiana ciepła są powietrzem. Metoda *exchangeEnergy* poza wymianą ciepła między sąsiadującymi komórkami implementuje także przekazywanie drogą konwekcji oparów koniecznych do zapłonu komórki.

### 5.2.3. Moduł obsługi zdarzeń

Głównym zadaniem modułu obsługi zdarzeń zwanym także *Controllerem* jest obsługa interakcji z użytkownikiem. Zgodnie z omówioną w rozdziale **??** moduł ten łączy w sobie elementy widoku, tworzące Graficzny Interfejs Użytkownika oraz metody obsługujące wywołane zdarzenia. Implementacja modułu zawarta jest w pakiecie *Controller*, a jej budowę przedstawia diagram 5.5. Adekwatnie do diagramu klas reprezentującego moduł *Model*, ilustracja 5.5 przedstawia powiązania między klasami modułu *Controller* oraz innymi modułami aplikacji.

#### WindowBuilder Pro - narzędzie do budowy GUI

Elementy tworząceg GUI zostały stworzone z wykorzystaniem narzędzia WindowBuilder Pro. Jest to darmowe narzędzie, które można zainstalować jako Plugin środowiska Eclipse. WindowBuilder Pro jest edytorem typu WYSIWYG (What You See Is What You Get), pozwalającym na tworzenie graficznych interfejsów użytkownika z wykorzystaniem bibliotek Swing, SWT czy GWT.

WindowBuilder poza możliwościa tworzenia w łatwy sposób interfejsu aplikacji, umożliwia edycję GUI stworzonego przy pomocy innego oprogramowania. Posiada wbudowany parser, który po otrzymaniu jako argument głównej klasy tworzącej interfejs, przeprowadza analizę kodu i jego wizualizację. Pozwala to na wykorzystanie edytora do poprawy wyglądu aplikacji jednocześnie zachowując przejrzystość kodu w wyniku ręcznej refaktoryzacji.

Innymi przydatnymi cechami wyróżniającymi edytow WindowBuilder Pro jest możliwość tworzenia hierarchii elementów oraz związana z nią możliwość zamiany typu elementu tworzącego GUI po jego osadzeniu w aplikacji. WindowBuilder umożliwia tworzenie własnych elementów wizualnych dziedzicących po standardowych klasach zawartych w bibliotekach Swing, SWT czy GWT oraz po typach stworzonych przez projektanta. Ta funkcjonalność pozwala na łatwe dostosowywanie komponentów do potrzeb aplikacji oraz dostarcza możliwość tworzenia modułu interfejsu łatwego do utrzymania i modyfkacji. Druga z wymienionych opcji została nazwana przez autorów programu Morph. Umożliwia ona zamianę obiektu dowolnej klasy na instancję klasy po niej dziedziczącej zgodnie z polimorfizmem oraz instancję klasy podobnej. Do klas podobnych zaliczane są typy posiadające wspólne cechy. W przypadku takiej zamiany zachowywane są właściwości wspólne dla obiektów obu klas.

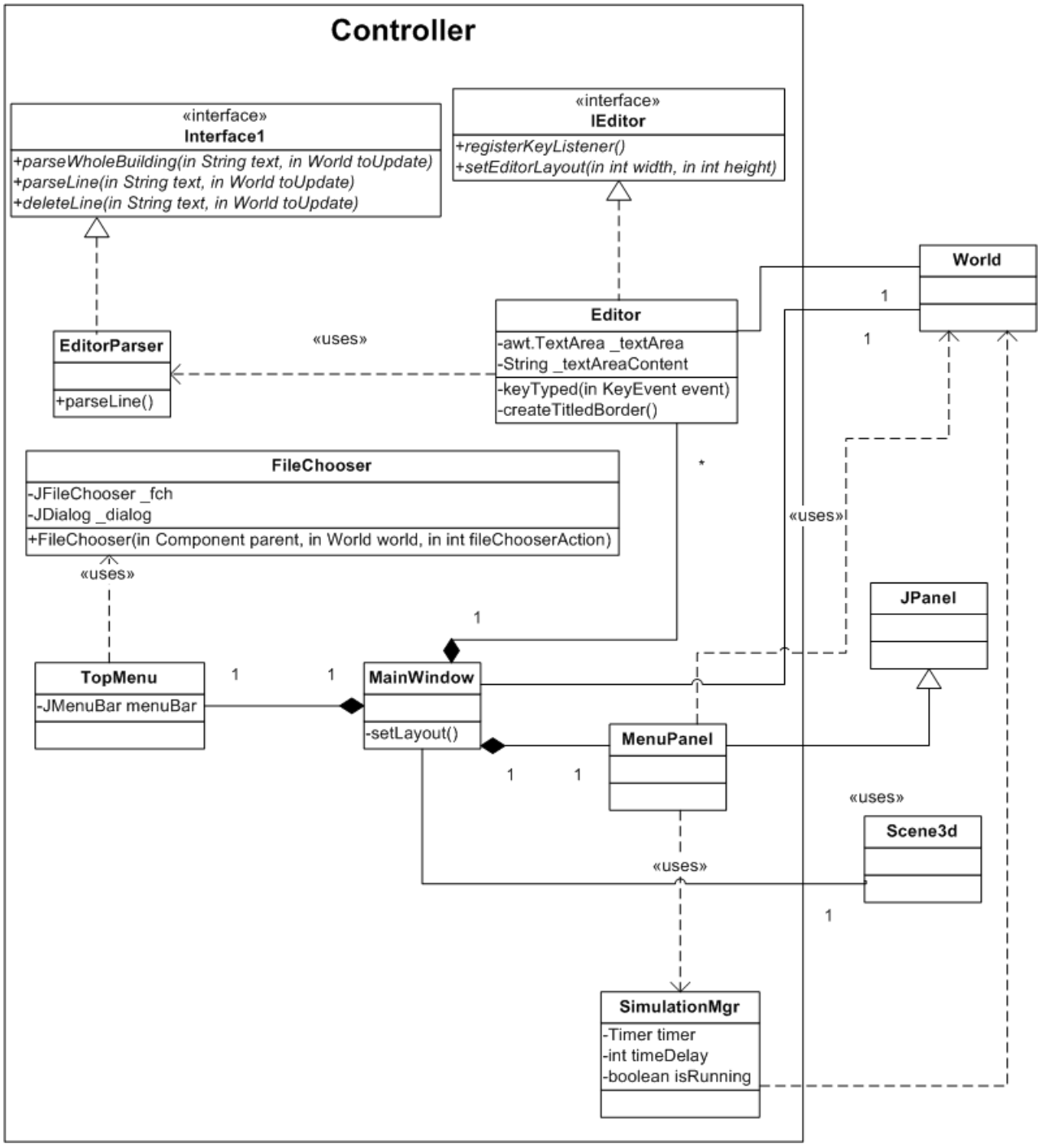
#### Budowa modułu

Poniżej została przestawiona implementacja poszczególnych klas wchodzących w skład modułu.

#### MainWindow

Jest to główna klasa modułu Controller i jednocześnia główna klasa aplikacji, zawierająca metodę main. W konsturktorze obiektu tworzone są elementy składowe głownego okna, do których należą obiekty klas:

- MenuPanel




Rysunek 5.5: Moduł Controller

- TopMenu
- Editor
- Scene3D

Klasa MainWindow zawiera także referencję do obiektu typu World, który jest aktualizowany w czasie symulacji. Konstruktor odpowiada także za rejestrację metod obsługujących zakończenie działania aplikacji. W prywatnej metodzie *setLayout*, która jest wywoływana podczas konstrukcji okna, ustalane jest rozmieszczenie jego elementów składowych. Zastosowanym sposobem rozmieszczania jest *BorderLayout*, który pozwala na elastyczne dopasowywanie rozmiaru komponentów do aktualnej wielkości okna.

**MenuPanel**

MenuPanel jest klasą, która pozwala na sterowanie symulacją oraz pozwala na dodawanie elementów konstrukcyjnych budynku. Zbudowana jest z dwóch obiektów klasy JPanel odpowiadających zakładkom Simmulation i AddingNewBlocks. Wszystkie elementy składowe paneli(pola tekstowe, etykiety, guziki) będące również atrybutami klasy MenuPanel zostały stworzone z wyłącznym wykorzystaniem biblioteki Swing. Do rozmieszczenia elementów w obrębie paneli został wykorzystany szablon GroupLayout. Klasa MenuPanel poza tworzącymi jej wygląd elementami posiada także referencję do obiektu typu SimulationMgr, który odpowiada za aktualizację symulacji oraz instancji klasy World, na której przeprowadzana jest symulacja. W wyniku naciśnięcia przez użytkownika jednego z guzików (Start, Stop, Pause, Continue, Restart) metody obsługujące te zdarzenia wywołują odpowiednie akcje klasy SimulationMgr aktywujące lub deaktywujące symulację. Metda obsługująca zdarzenie wykonane na guziku Start

**SimulationMgr**

SimulationMgr to klasa odpowiedzialna za sterowanie symulacją.

# 6. Graficzny interfejs użytkownika i instrukcja obsługi

Poniższy rozdział zawiera opis Graficznego Interfejsu Użytkownika oraz wskazówki i przykłady jak z niego korzystać w sposób efektywny. Interfejs użytkownika, przedstawiony na rysunku 6.1 W celu zapewnienia wygody i efektywności pracy został podzielony na następujące części:

1. Panel górny
2. Widok scenmy
3. Menu boczne
4. Edytor

## 6.1. Panel górny

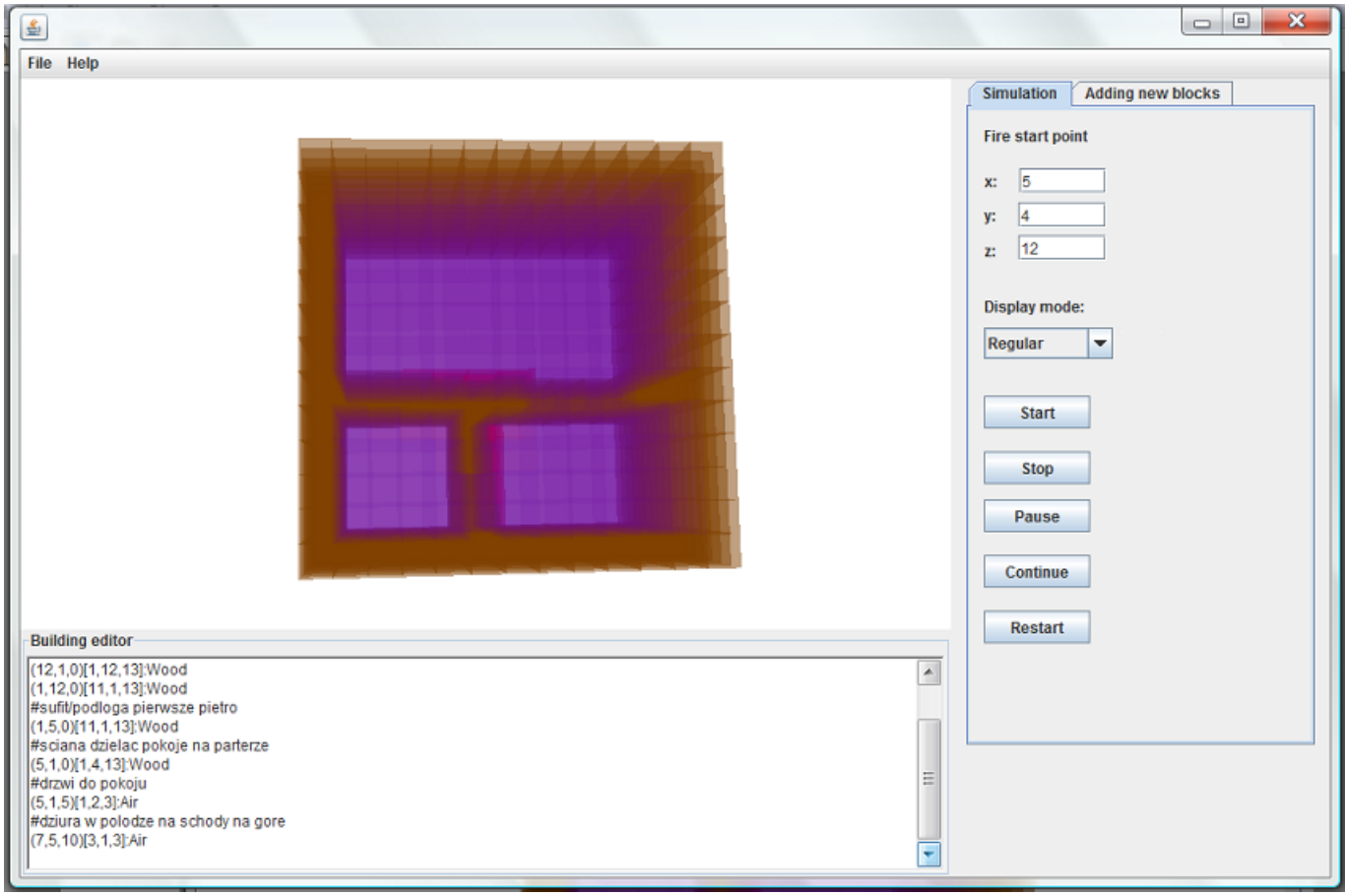
Panel górny zawiera elementy:

- File, składający się z pozycji:
  - Save cell states - zapis stanu komórek automatu do pliku. Automat zapisany jest w postaci zbioru macierzy dwuwymiarowych, z których każda przedstawia jeden poziom automatu począwszy od poziomu 0 zgodnie z osią Y.
  - Save temperatures - zapis rozkładu temperatur automatu. Format zapisu identyczny z przypadkiem stanu komórek.
  - Read building from file - możliwość wczytania budynku z pliku. Format zapisu budynku w pliku jest taki sam jak w edytorze. Po wczytaniu budynku z pliku, jego opis pojawia się w edytorze co pozwala na dalsze modyfikacje.
- Help, skonstruowany z opcji:
  - User guide - podstawowe informacje o użytkowaniu programu: jak korzystać z edytora oraz panelu bocznego.
  - About program - podstawowe informacje o programie.

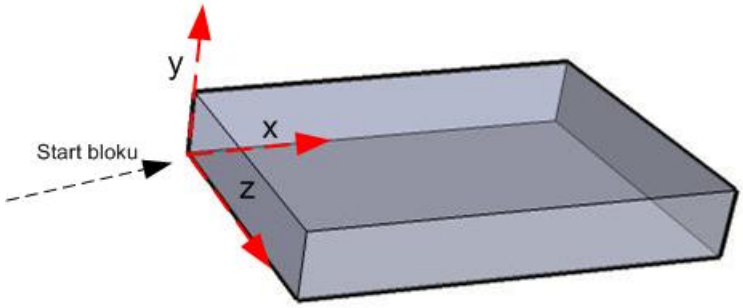
## 6.2. Scena

Główny elementem interfejsu jest scena, przedstawiająca wyniki symulacji. Aplikacja dostarcza możliwość manipulowania sceną w celu zróżnicowania widoku. Możliwe są do wykonania następujące akcje:

- Oddalanie i przybliżanie sceny za pomocą strzałek klawiatury. Strzałka w górę powoduje przybliżenie, zaś strzałka w dół oddalenie.
- Obrót elementów na scenie za pomocą strzałek w bok. Strzałka w prawo obraca scenę zgodnie z ruchem wskazówek zegara, zaś strzałka w lewo przeciwnie do ich ruchu.




Rysunek 6.1: Graficzny interfejs użytkownika




Rysunek 6.2: Orientacja bloku

## 6.3. Edytor

Elementem wykorzystywanym do edycji budynku jest umieszczony na dole interfejsu edytor. Edytor, jak zostało to opisane w rozdziale 5 jest polem tekstowym i jako takie umożliwia operacje zaznaczania, kopiowania, wklejania i usuwania tekstu za pomocą standardowych skrótów klawiaturowych. Tekts umieszczany w edytorze służy do konstrukcji budynku, w którym przeprowadzana jest symulacja pożaru i powinien być zgodny z następującym formatem:

1. Konieczne jest aby każdy element konstrukcyjny był zapisany w osobnej linii.
2. Pojedynczy, dodawany element jest ortogonalnym, jednolitym blokiem o następującej postaci:

   (Start_X,Start_Y,Start_Z)[Rozmiar_X,Rozmiar_Y,Rozmiar_Z]:Material

   gdzie:

   - (Start_X,Start_Y,Start_Z) - są to adekwatnie współrzędne (X,Y,Z) Lewego-Dolnego-Tylnego wierzchołka bloku, będącego początkiem bloku co przedstawia rysunek 6.2.
   - [Rozmiar_X,Rozmiar_Y,Rozmiar_Z] - określają rozmiar wprowadzanego bloku.
   - Komórki w automacie numerwane są od 0. Wszystkie współrzędne są całkowitymi liczbami nieujemnymi i oznaczają wielokrotność komórek w danej osi.
   - Materiał jest nazwą jednego z dostępnych materiałów. Dostępne materiały to:
     - Drewno
     - Metal
     - Cegła
     - Powietrze

     Blok zbudowany z materiału powietrze, może być wykorzystany do wstawienia dziury w ścianie lub podłodze odpowiadającej oknu lub przejściu na wyższe piętro budynku.
3. Linia rozpoczynająca się od znaku  jest traktowana jako komentarz i nie podlega analizie.

## 6.4. Panel boczny

Panel boczny składa się z dwóch zakładek:

1. Zakładka *Simulation* dostarcza opcje pozwalające kontrolować przebieg symulacji.

   - Trzy pola tekstowe oznaczone etykietami *x*,y i $z$ pozwalają podać współrzędne bloku, w którym rozpoczyna się pożar.
   - Pole wyboru trybu *Diplay mode* przełącza scenę w jeden z dwóch trybów:
     - Tryb regularny - domyślny tryb, w którym pokazany jest aktualny stan komórki. Komórka będą w stanie naturalnym zachowuje kolor i przeźroczystość materiału z którego została stworzona. Komórka paląca się ma kolor czerwony. Dym przedstawiony jest jako szary.
     - Tryb temperaturowy - przedstawia rozkład temperatur w budynku. Kolor niebiskie oznacza temperaturę z zakresu $[20 - 50°C]$. Temperatury powyżej $50°C$ przedstawione są w odcieniach czerwieni. Im intensywniejsza czerwień, tym większa temperatura.
   - Guziki *Start*, *Stop*, *Pause*, *Continue*, *Restart* pozwalają na kontrolowanie symulacji. Umożliwiają na przykład zatrzymanie pracy automatu w celu zapisu wyników pośrednich lub modyfikacji budynku, a następnie wznowienie działania.

2. Zakładka *Adding new blocks* jest alternatywą *Edytora*. Umożliwia edycję budynku i jest przydatna szczególnie dla osób nieobytych z tekstowymi narzędziami pracy. Omawiana zakładka składa się z:

   - Pola umowżliwiającego wybór materiału dla dodawanego bloku.
   - Pól tekstowych pozwalających wpisać współrzędne punktu startu dodawanego bloku oraz jego rozmiar.
   - Przycisku *Add*, którego kliknięcie powoduje zaakceptowanie wprowadzonych zmian i utworzenie bloku.

# Bibliografia