Politechnika Warszawska
Wydział Elektryczny

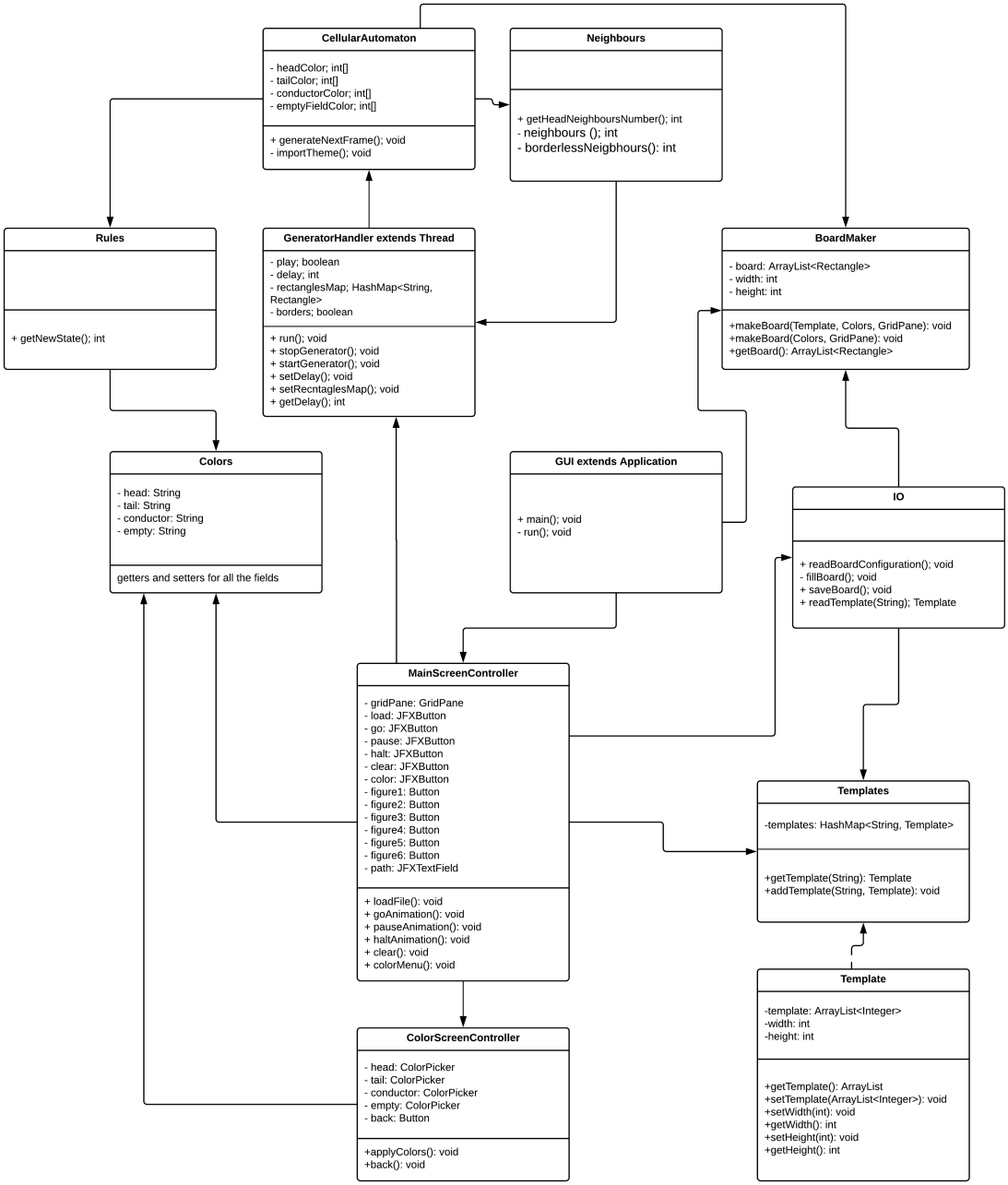
Specyfikacja Implementacyjna "WireWorld"

Autorzy:
Grzegorz Kopyt
Daniel Sporysz

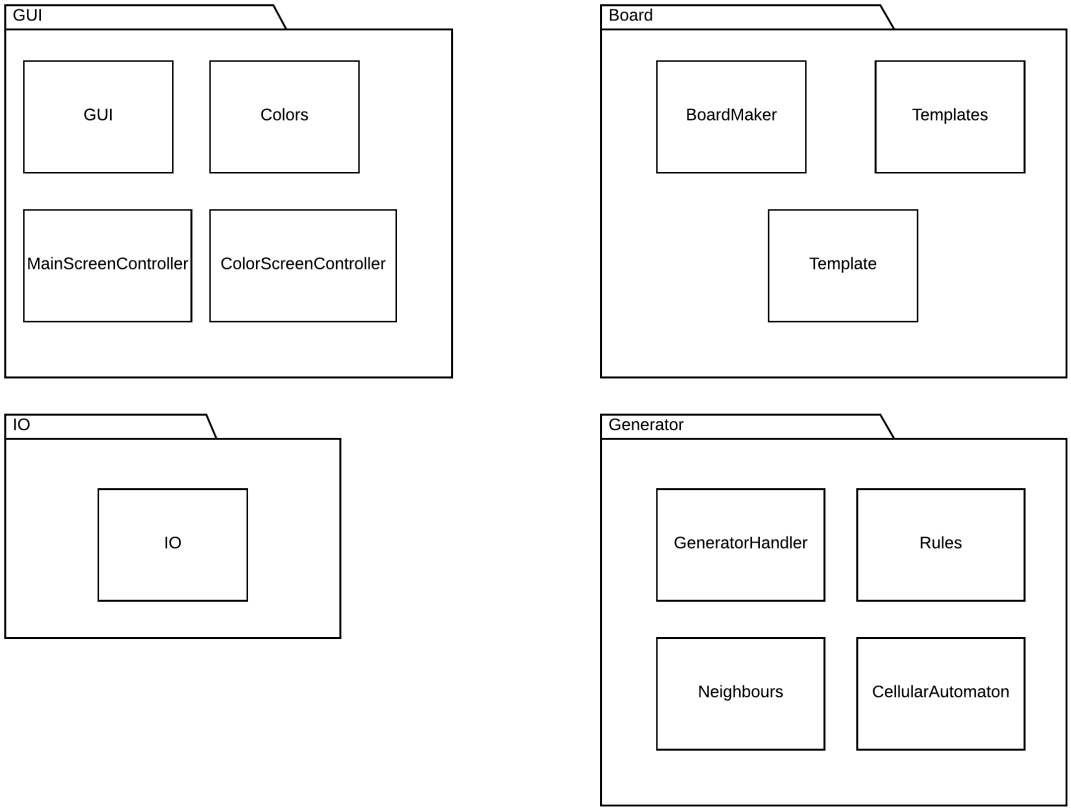
13 maja 2018

Spis treści

1 Diagram klas



2 Diagram pakietów



---

3 Pakiet GUI

Wykonany przy pomocy technologii javafx, zawiera dodatkową bibliotekę: jfoenix.

3.1 Pliki .fxml:

- MainScreen.fxml
- ColorScreen.fxml

3.2 GUI

Rozszerza klasę Application z javafx.

3.2.1 Pola

brak

3.2.2 Metody

- **main**

  Standardowo wywołuje metodę launch.

- **start**

  Wczytuje plik MainScreen.fxml, przygotowuje całą scenę i wyświetla ją w wymiarach (800, 600).

3.3 MainScreenController

Kontroler sceny MainScreen.fxml.

3.3.1 Pola

- GridPane *gridPane*
- JFXButton *load*
- JFXButton *go*
- JFXButton *pause*
- JFXButton *halt*
- JFXButton *clear*
- JFXButton *color*
- Button *figure1*
- Button *figure2*
- Button *figure3*
- Button *figure4*
- Button *figure5*
- Button *figure6*
- JFXTextField *path*

3.3.2 Metody

- void **loadFile()**

  Wywołuje metodę, która wczytuje plik tekstowy, którego ścieżkę podano w polu tekstowym.

- void **goAnimation()**

  Uruchamia animacje wywołując metodę z klasy Animation.

- void **pauseAnimation()**

  Pauzuje animacje metodą z klasy Animation.

- void **haltAnimation()**

  Powoduje powrót animacji do punktu początkowego.

- void **clear()**

  Zmienia kolor każdej komórki w tablicy na biały.

- void **colorMenu()**

  Wyświetla okno z pliku ColorMenu.fxml

3.4 ColorScreenController

Kontroler sceny ColorScreen.fxml.

3.4.1 Pola

- ColorPicker *head*
- ColorPicker *tail*
- ColorPicker *conductor*
- ColorPicker *empty*

3.4.2 Metody

- void **back()**

  Powoduje powrót do MainScreen.

- void **applyColors()**

  Pobiera z obiektów klasy ColorPicker informacje o kolorach i przekazuje je klasie Colors.

3.5 Colors

Kontroler sceny ColorScreen.fxml.

3.5.1 Pola

- String head
- String tail
- String conductor
- String empty

3.5.2 Metody

- String **getHead()**
- void **setHead()**
- String **getTail()**
- void **setTail()**
- String **getConductor()**
- void **setConductor()**
- String **getEmpty()**
- void **setEmpty()**

3.5.3 Konstruktor

Domyślne kolory to:

- head - żółty
- tail - czerwony
- conductor - czarny
- empty - biały

---

4 Pakiet IO

4.1 IO

Klasa służy do odczytywania z oraz zapisu do polików graficznych konfiguracji pól na planszy.

4.1.1 Pola

- brak

4.1.2 Metody

- public void **readBoardConfiguration(String path, HashMap<String, Rectangle> map)**

  Funkcja czyta plik graficzny o nazwie „path", piksel po pikselu oraz zapisuje konfigurację do macierzy liczb całkowitych - reprezentujących stany komórek.

  Przy odczycie konieczna jest konwercja wartości z formatu RGB na jednocyfrowy znak stanu. Należy postępować według wzoru:

  - RGB[230-255, 230-255, 230-255] -> 0
  - RGB[0-25, 0-25, 0-25] -> 1
  - RGB[230-250, 0-25, 0-25] -> 2
  - RGB[230-255, 230-255, 0-150] -> 3

  Przedziały oznaczają tolerancję wariacji koloru.

  Następnie wywołaj funkcje fillBoard.

- private void **fillBoard(int[][] matrix, HashMap<String, Rectangle> map**

  Metoda na podstawie otrzymanej macierzy liczb całkowitych, aktualizuje macierz „map", zmieniając kolory obiektów zgodnie ze wzorem:

  - 0 -> zmień kolor na biały
  - 1 -> czarny
  - 2 -> żółty
  - 3 -> czerwony

- private void **saveBoard(String name, HashMap<String, Rectangle> map**

  Funkcja zapisuje do pliku graficznego o nazwie „name" konfigurację planszy.

- public Template **readTemplate**

  Funkcja odczytuje z pliku graficznego konfigurację wzoru, przy odczycie konwertując wartość koloru pikseli z formatu RGB na jednocyfrową liczbę całkowitą:

  - 0 -> zmień kolor na biały
  - 1 -> czarny
  - 2 -> żółty
  - 3 -> czerwony

  A następnie zapisuje do obiektu Template i go zwraca.

5 Pakiet Board

5.1 BoardMaker

Odpowiada za stworzenie tablicy obiektów klasy Rectangle (*board*). Tablica ta będzie służyła jako obszar edytowany przez użytkownika, a także będzie na niej wyświetlana animacja.

5.1.1 Pola

- ArrayList<Rectangle> *board*
- int *width*
- int *height*

5.1.2 Metody

- void **makeBoard(Template, Colors, GridPane)**

  Metoda tworzy tablicę *board* obiektów klasy Rectangle na podstawie wzoru podanego w Template o 30 kwadratach w rzędzie i 30 kwadratach w kolumnie. Wymiary obiektów Rectangle wynoszą (20, 20). Wszystkie obiekty dodane zostają do kolekcji *board*. Obiektom zostaje nadane ID jako kolejne liczby naturalne całkowite zaczynając od 1. Kolor wypełnienia obiektów nadawany jest zgodnie z zawartością Colors, obramowanie - czarne. Tablica tworzona jest poprzez dodanie obiektów do GridPane. Docelowy GridPane znajduje się w klasie MainScreenController.

- void **makeBoard(Colors, GridPane)**

  Metoda tworzy tablicę *board* obiektów klasy Rectangle o 30 kwadratach w rzędzie i 30 kwadratach w kolumnie. Wymiary obiektów Rectangle wynoszą (20, 20). Wszystkie obiekty dodane zostają do kolekcji *board*. Obiektom zostaje nadane ID jako kolejne liczby naturalne całkowite zaczynając od 1. Kolor wypełnienia obiektów nadawany jest zgodnie z zawartością Colors, obramowanie - czarne. Tablica tworzona jest poprzez dodanie obiektów do GridPane. Docelowy GridPane znajduje się w klasie MainScreenController.

- ArrayList<Rectangle> **getBoard()**

5.2 Template

Klasa reprezentująca wzór obiektu do wstawiania na tablice *board*. Wzór przechowywany jest w tablicy *template* jako ciąg liczb 0(pusty), 1(przewodnik), 2(ogon), 3(głowa). Przy tworzeniu wzorów należy uwzględnić to, że rozmiar tablicy wynosi 30 kwadratów na 30 kwadratów.

5.2.1 Pola

- ArrayList<Integer> *template*
- int *width*
- int *height*

5.2.2 Metody

- Template **getTemplate()**
- void **setTemplate(ArrayList<Integer>)**
- void **setWidth(int)**
- int **getWidth()**
- void **setHeight(int)**
- int **getHeight()**

5.3 Templates

Przeznaczeniem klasy jest przechowywanie obiektów klasy template w kolekcji.

5.3.1 Pola

- HashMap<String, Template> *templates*

5.3.2 Metody

- Template **getTemplate(String)**

  Metoda otrzymawszy klucz klasy *String* zwraca obiekt klasy Template z kolekcji *templates*.

- void **addTemplate(String, Template)**

  Metoda dodaje do kolekcji *templates* obiekt klasy Template i nadaje mu klucz podany jako zmienna klasy String.

---

6 Pakiet Generation

6.1 GenerationsHandler

Obiekt ten, jako rozszerzenie klasy „Thread", zapewnia wątek na którym wykonywana będzie generacja kolejnych plansz. Klasa ma za zadanie obserwować zmiany jakie użytkownik wprowadza poprzez interfejs graficzny i po odpowiednim skonfigurowaniu, cyklicznie uruchamiać, bądź czekać na uruchomienie generatora.

6.1.1 Pola

- boolean *play*
- boolean *borders*
- int *delay*
- CellularAutomation *generator*

6.1.2 Metody

- void **run**

  W nieskończonej pętli wywołuje metodę „generateNextFrame”, poprzednio sprawdzając wartość pola „play”, oznaczającego, czy generacja ma być wykonywana. Konstrukcja pętli umożliwia wstrzymywanie i wznawianie pracy generatora. Kolejne uruchomienia generatora uruchamiane są w odstępach czasowych określonych w zmiennej „delay” wyrażonej w milisekundach.

- public void **startGenerator**

  Zmienia wartość pola „play” na „true”.

- public void **stopGenerator**

  Zmienia wartość pola „play” na „false”.

- public void **setDelay(int value**

  Przypisuje polu „delay” wartość „value”.

- int **getDelay**

  Zwraca wartość pola „delay”.

6.2 CellularAutomaton

Klasa pełni rolę generatora kolejnych plansz.

6.2.1 Pola

- int *boardWidth*
- int *boardHight*
- int[][] *tmp*
- HashMap<String, javafx.scene.shape.Rectangle> *map*
- int[] *headColor*
- int[] *tailColor*
- int[] *conductorColor*
- int[] *emptyFieldColor*
- Rules *rules*
- Neighbours *neighbours*

6.2.2 Metody

- public void **generateNextFrame(boolean borders)**

  Na początku należy zaktualizować konfigurację kolorów za pomocą metody „importTheme”.

  Następnie dla każdej komórki w mapie „map”, należy wywołać funkcje „getHeadNeighboursNumber” oraz „getNewState”, a wynik zapisać pod odpowiedniej indeksem w tablicy „tmp”.

  Po przeanalizowaniu całej planszy i zapisaniu nowych stanów do macierzy „tmp”, należy wyświetlić zmiany na ekranie, aktualizując kolory kwadratów w macierzy „map”, zgodnie z konfiguracja z „tmp”. Konwersja liczby całkowitej na kolor przebiega w następujący sposób:

- 0 -> emptyFieldColor
- 1 -> conductorColor
- 2 -> tailColor
- 3 -> headColor

6.3 Rules

Klasa determinuje stan w który komórka przechodzi w następnej generacji, uwzględniając jej aktualny stan oraz stan komórek sąsiednich.

6.3.1 Pola

- brak

6.3.2 Metody

- public int **getNewState(int state, int headsNumber)**

  Zależnie od otrzymanych argumentów funkcja zwraca:

  - state = 0, zwróć 0
  - state = 3, zwróć 2
  - state = 2, zwróć 1
  - state = 1 i headsNumber != 1 != 2, zwróć 1
  - state = 1 i headNumber = 1 lub 2, zwróć 3

6.4 Neighbours

Obiekt zajmuje się analizowaniem stanów komórek. Jego zadaniem jest zwrócenie informacji o ilości komórek, które są „głowami", dookoła konkretnej komórki. Sposób analizy można konfigurować.

6.4.1 Pola

- int *boardWidth*
- int *boardHight*
- HashMap<String, javafx.scene.shape.Rectangle> *rectangleMap*

6.4.2 Metody

- public int **getHeadNeighboursNumber(boolean borders, int x, int y)**

  Funkcja pełni role przekaźnika. Jeśli zmienna „borders" ma wartość true, wywołaj metodę „neighbours", a w przeciwnym wypadku „borderlessNeigbhours".

- private int **neighbours(int x, int y,)**

  Funkcja sprawdza ile jest komórek w stanie HEAD(czerwony) dookoła komórki o pytanym indeksie. Indeksy spoza macierzy są ignorowane - pola poza macierzą mają stan różny od HEAD.

- private int **borderlessNeigbhours(int x, int y,)**

  Funkcja sprawdza ile jest komórek w stanie HEAD(czerwony) dookoła komórki o pytanym indeksie. Indeksy spoza macierzy są zawijane, zamieniane na przeciwne.

7 Przepływ Sterowania

Przykładowe uruchomienie programu:

- start programu.
- start wątku generatora; GenerationsHandler.start()
- przygotowanie okna interfejsu graficznego; GUI.
- tworzenie tablicy komórek; BoardMaker.makeBoard()
- określenie nazwy pliku i polecenie wczytania konfiguracji planszy z pliku
  - obsłużenie czynności przez MainScreenController
- wywołanie funkcji czytającej z pliku; IO.readBoardConfiguration()
- aktualizacja planszy; IO.fillBoard
- użytkownik wciska przycisk start; MainScreenController
- generatorHandler.startGenerator()
- wątek GenerationsHandler rozpoczyna pracę i cyklicznie wywołuje CellularAutomation.generateNextFrame()
- aktualizacja motywu kolorów; CellularAutomation.importTheme()
- określenie liczby sąsiadów w stanie HEAD; Neighbours.getHeadNeighboursNumber()
- określenie następnego stanu; Rules.getNewState()
- Użytkownik wciska przycisk stop; MainScreenController
- GeneratorHandler.stopGenerator()
- wątek GenerationsHandler wstrzymuje pracę
- użytkownik zleca zapis planszy do pliku graficznego; MainScreenController
- zapis do pliku; IO.saveBoard()
- użytkownik kończy pracę programu

---

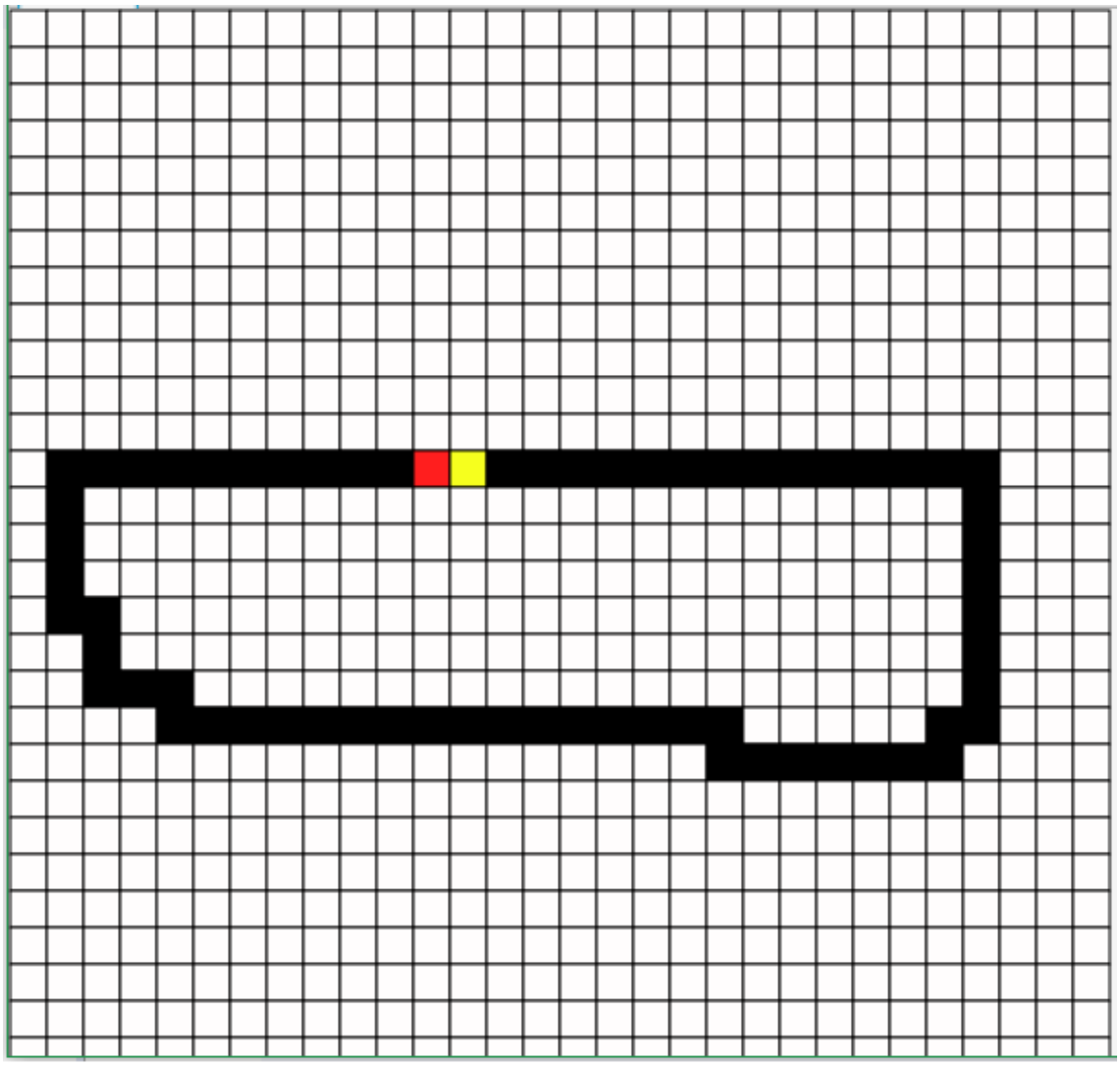
8 Testy klas i GUI

8.1 GUI

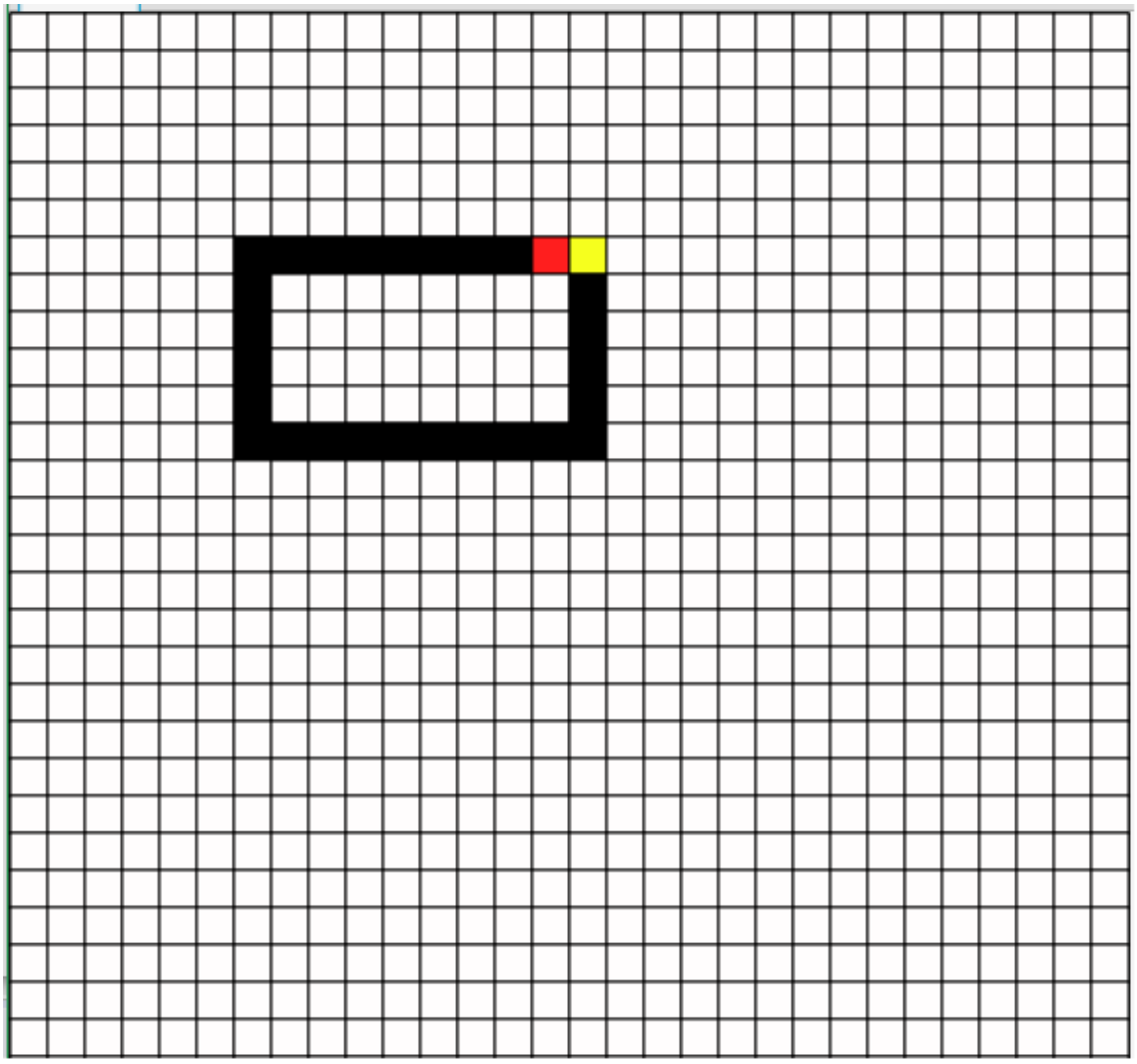
Scenariusze

1. Należy wpisać w pole tekstowe poprawną ścieżkę pliku (TestLoad.txt) i kliknąć w przycisk Load.
2. Należy wpisać w pole tekstowe niepoprawną ścieżkę pliku i kliknąć w przycisk Load.
3. Należy wpisać w pole 101 znaków.
4. Należy wpisać w pole tekstowe *„...!fte./tetat/assfs/////* ".
5. Należy najechać myszką na planszę edycji.
6. Należy czterokrotnie kliknąć na dowolne pole planszy.

7. Należy przejechać po planszy ze wciśniętym lewym przyciskiem myszy.
8. Należy wyklikać na planszy taki wzór:

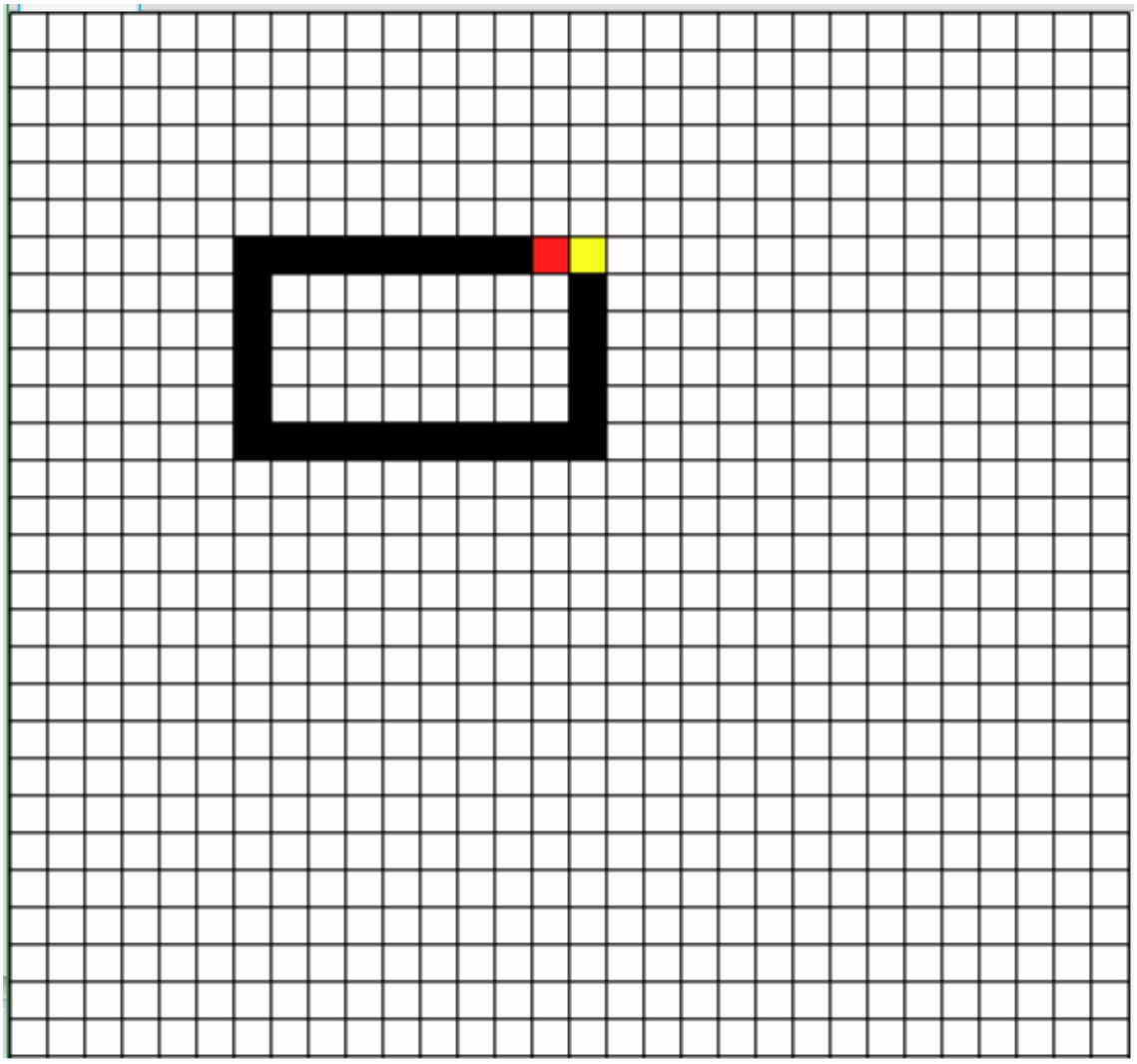


9. Po wykonaniu poprzedniego testu należy wcisnąć przycisk GO.
10. Po wykonaniu poprzedniego testu należy wcisnąć przycisk Pause.
11. Po wykonaniu poprzedniego testu należy wcisnąć przycisk Halt!.
12. Należy kliknąć w przycisk Clear.
13. Kiedy na planszy są same białe komórki należy wcisnąć każdy z przycisków GO, Pause, Halt!
14. Dla każdego przycisku figure, należy: kliknąć na niego, następnie kliknąć na planszy w komórkę o współrzędnych (15,15) i komórkę (30,15)
15. Należy kliknąć w przycisk Color.
16. Należy ustawić plansze w sposób przedstawiony na obrazku poniżej.W oknie ColorMenu ustawić cztery różne kolory, następnie kliknąć przycisk back i kliknąć przycisk GO.

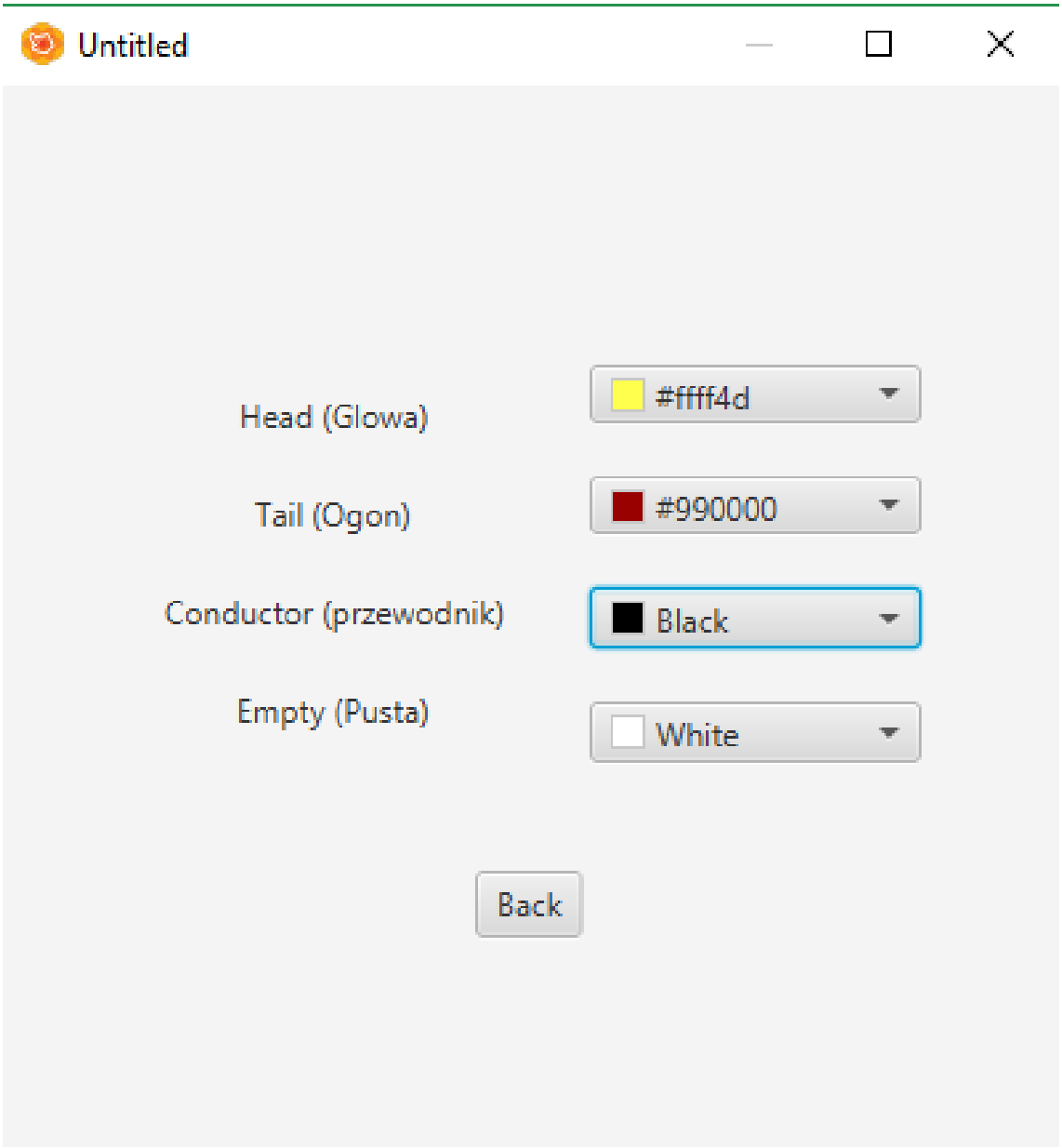


Kryteria oceny poprawnej pracy

1. Jeśli na planszy pojawi się taki układ pól jak poniżej, test jest zaliczony.



2. Program powinien wyświetlić w polu tekstowym czerwony komunikat ”Can’t open the file.".
3. Program powinien wyświetlić w polu tekstowym czerwony komunikat ”Can’t open the file.".
4. Program powinien pozwolić na wpisanie tylko 100 znaków.
5. Kolory nie powinny się zmienić.
6. Kolory powinny zmienić się cztery razy (czarny–>czerwony->żółty->biały).
7. Kolory powinny się zmienić o jeden.
8. Jeśli udało się to test jest zdany.
9. Kolory powinny się zmieniać zgodnie z zasadami wire world.
10. Animacja powinna się zatrzymać.
11. Animacja powinna wrócić do początkowego układu sprzed kliknięcia przycisku GO.
12. Wszystkie komórki powinny być białe.
13. Kolory na planszy nie powinny się zmieniać.
14. Elementy powinny zostać wstawione na plansze, z wyjątkiem przypadku, w którym zahaczają o krawędź ekranu.
15. Powinno pojawić się ColorMenu:



16. Kolory na planszy powinny zostać zmienione zgodnie z ustawieniami, a animacja odbyć się zgodnie z zasadami wire world.

8.2 IO

8.2.1 IO

Scenariusze

1. Odczyt z pliku w formacie .png
2. Odczyt z pliku w formacie innym niż .png
3. Próba odczytu z pliku do którego brak praw odczytu
4. Zapis planszy do pliku .png
5. Próba zapisu planszy do pliku bez praw do zapisywania na dysku

Kryteria oceny poprawnej pracy

1. Poprawny odczyt konfiguracji z pliku i aktualizacja komórek na ekranie
2. Obsłużenie wyjątków z brakiem praw do zapisu na dysk
3. Obsłużenie wyjątków z brakiem praw do czytania z dysku

4. Podanie ścieżki do pliku, który nie istnieje, bądź zawierającego niepoprawną konfigurację, skutkuje przerwaniem pracy i niezaktualizowaniem planszy.

8.3 Board

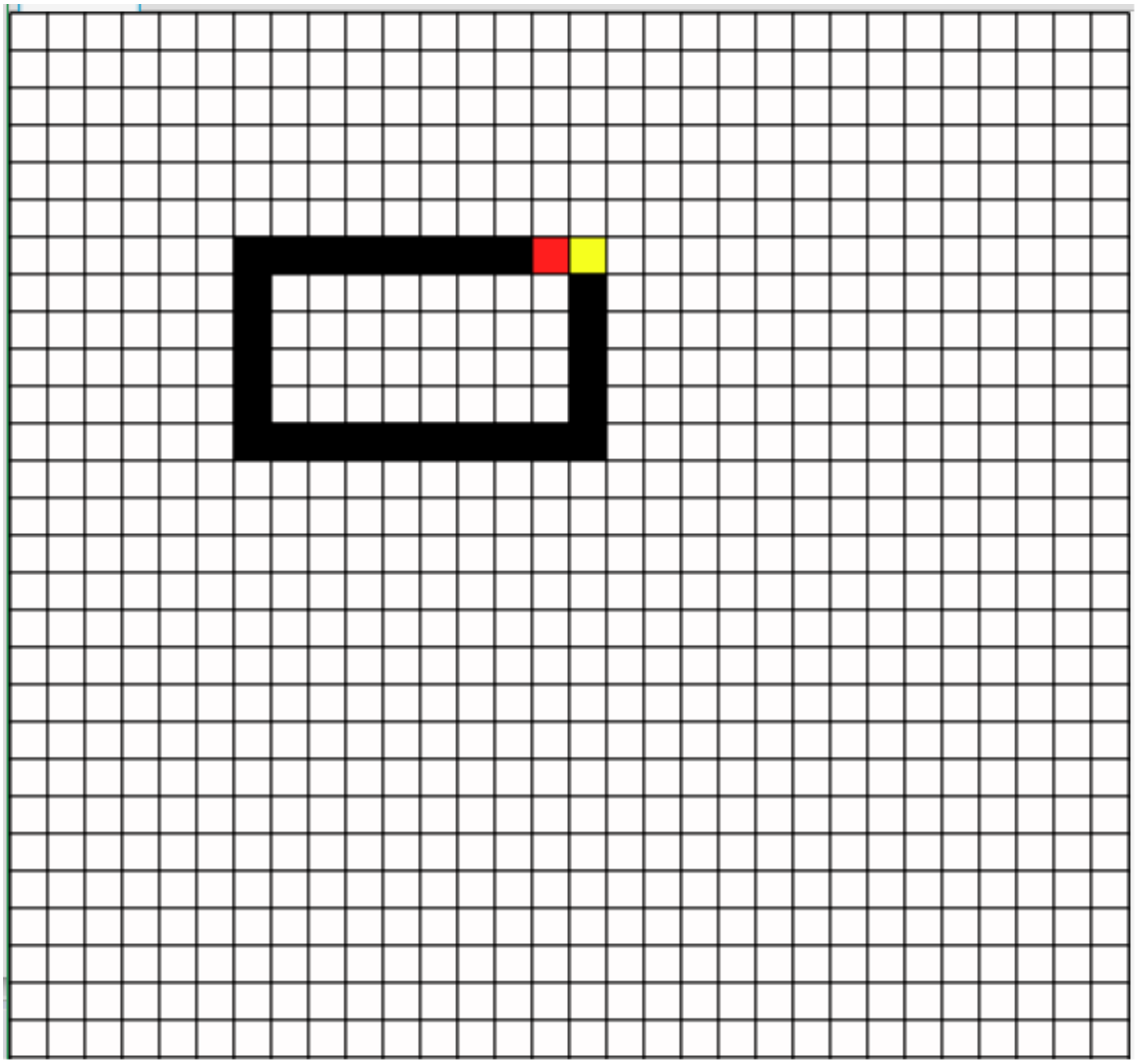
8.3.1 BoardMaker, Template

Scenariusze

1. Stworzyć obiekt Template jako wartości pól wymiarów podać (30, 30), a listę obiektów klasy Integer taką jak w pliku TestLoad.txt. Wypisać macierz na konsole.
2. Wywołać metode makeBoard w wariancie bez argumentu Template. Podając domyślny GridPane i obiekt Colors z domyślnymi kolorami.
3. Wywołać metode makeBoard w wariancie z argumentem Template takim jak w teście pierwszym. Podając domyślny GridPane i obiekt Colors z domyślnymi kolorami.

Kryteria oceny poprawnej pracy

1. Jeśli macierz jest taka sam jak w pliku TestLoad.txt, test jest zdany.
2. W głównym menu interfejsu graficznego powinna pojawić się tablica białych kwadratów 30 na 30.
3. W głównym menu interfejsu graficznego powinna pojawić się taka tablica:



8.3.2 Templates

Scenariusze

1. Stworzyć puste obiekty Template, których pola x wypełnić kolejno wartościami 1, 2, 3. Dodać te obiekty do Tempaltes za pomocą metody *addTempalte* nadając im klucze 1, 2 ,3. Pobrać z Templates obiekty Template po kolejno 1, 2, 3 nadanych kluczach i wypisać wartości ich pól x na konsole.

Kryteria oceny poprawnej pracy

1. Na konsoli powinien pojawić się komunikat: „1 2 3"

8.4 Generation

8.4.1 GenerationsHandler

Scenariusze

1. pauzowanie i startowanie pracy generatora
2. zmiana parametru „delay”

Kryteria oceny poprawnej pracy

1. Wątek reaguje na zmiany w konfiguracji programu

8.4.2 CellularAutomation

Scenariusze

1. analiza równobocznej planszy w dwóch trybach analizy
2. analiza nierównobocznej(poziomo) planszy w dwóch trybach analizy
3. analiza nierównobocznej(pionowo) planszy w dwóch trybach analizy

Kryteria oceny poprawnej pracy

1. Metoda przeanalizowała wszystkie pola HashMap'y map oraz za pomocą innych funkcji , wypełniła macierz „tmp”. Kończąc pracę funkcja poprawnie zaktualizowała stan obiektów w „map”.

8.4.3 Neighbours

Scenariusze

1. wywołanie funkcji dla poprawnych argumentów
2. wywołanie dla indeksów spoza macierzy
3. praca na tablicy zawierającej nierozpoznawane oznaczenia stanów

Kryteria oceny poprawnej pracy

1. funkcja zwraca poprawną ilość sąsiadów w stanie HEAD, dookoła komórki o podanych indeksach
2. stany oznaczone nierozpoznawanym symbolami traktowane są jako pole w stanie pustym(0).

8.4.4 Rules

Scenariusze

1. wywołania z poprawnymi argumentami
2. wywołanie dla nieznanego stanu
3. wywołanie dla ujemnej liczby sąsiadów

Kryteria oceny poprawnej pracy

1. funkcja zwraca wartości zgodne co do reguły określonej w opisie funkcji.
2. niepoprawny znak stanu lub liczba sąsiadów skutkuje zwróceniem 0.