Opis modułów i wykorzystywanych wzorców projektowych projektu Labirynt

Polina Nesterova
01192875@pw.edu.pl

Martyna Kochalska
01187553@pw.edu.pl

26 maja 2024

1 Podział na moduły

Program składa się kolejno z modułów:

1. `MainFrame` - Obsługa GUI, przycisków.
2. `Render` - Generowanie wizualnej reprezentacji labiryntu, ścieżki, wyjść i wejść.
3. `Maze` - Przechowywanie informacji dotyczących labiryntu, pośrednik w komunikacji pomiędzy MainFramem i Loaderem, Solverem.
4. `Loader` - Załadowywanie pliku tekstowego i binarnego z pliku.
5. `FileValidator` - Sprawdzenie poprawności załadowanych plików binarnych i tekstowych.
6. `Solver` - Znalezienie najkrótszej ścieżki w labiryncie przy pomocy implementacji algorytmu Djikstra.
7. `Downloader` - Zapis ścieżki do pliku, przepisanie labiryntu na wymaganą formę dla pliku binarnego.
8. `MessageUtils, Coordinates` - Moduły pomocnicze dostępne dla każdego modułu, MessageUtils umożliwia wyświetlenie wiadomości, Coordinates służy jako struct trzymający w sobie koordynaty x i y.

2 Opis działania modułów

- `Moduł 1 - MainFrame`

  Główne funkcjonalności:

1. Wyświetlenie panelu z interfejsem
2. Oczekiwanie na input w postaci naciśnięcia przycisku i komunikacja z pozostałymi klasami.
3. Współpracuje z klasami: Maze, Render, MessageUtils

- `Moduł 2 - Render`

  Główne funkcjonalności:

  1. Wyświetlanie labiryntu, ścieżki
  2. Aktualizacja kolorów wyświetlanego labiryntu
  3. Zapisywanie wygenerowanego labiryntu jako png.
  4. Współpracuje z klasami: MainFrame(wywoływany), Coordinates

  Metody publiczne:

  – public void saveMazeAsImage(String filepath) throws IOException - umożliwia pobranie obrazu mapy labiryntu do lokalizacji podanej w argumencie.

  – public void saveSolvedAsImage(String filepath) throws IOException - umożliwia pobranie obrazu mapy labiryntu razem z rozwiązaniem do lokalizacji podanej w argumencie.

  – public void setExit/Entance(Coordinates exit/entrance) - setter do wejścia wyjścia w klasie.

- `Moduł 3 - Maze`

  Główne funkcjonalności:

  1. Przechowywuje dane dotyczące labiryntu: ścieżkę, wejście, wyjście, mapę.
  2. Pośredniczy w komunikacji między klasami
  3. Współpracuje z klasami: MainFrame(wywoływany), Solver, Loader, Downloader, Coordinates, MessageUtils

  Metody publiczne:

  – boolean load(String filepath) – Wywoływane przez MainFrame, wywołuje metodę loadMaze z Loadera, przypisuje zmiennym maze, entrance, exit odpowiednie wartości, przy błędzie zwraca false, przy poprawnym załadowaniu - true.

  – void solve(String filepath) - Wywołuje Solver.solve, przypisuje wartości ścieżce path.

- void download(String filepath) - wywołuje odpowiednio funkcję SavePathToBin() lub SavePath w zależności od pliku wejściowego.

- public void removePath() - usuwa ścieżkę po wywołaniu.

- public void restorePath() - przywraca ścieżkę.

- Dodatkowo: gettery i settery do: Maze, path, entrance, exit.

- `Moduł 4 - Loader`

  Główne funkcjonalności:

  1. Rozpoznanie rodzaju pliku wejściowego
  2. Wywołanie klasy FileValidator
  3. Wczytanie pliku tekstowego i zwrócenie go w odpowiedniej formie.
  4. Wczytanie pliku binarnego i zwrócenie go w odpowiedniej formie.
  5. Współpracuje z klasami: Maze(wywoływany), FileValidator, Coordinates.

  Metody publiczne:

  - public char[][] loadMaze(String filepath) - metoda otrzymująca ścieżkę do pliku wejściowego, sprawdzająca typ pliku, wywołująca klasę sprawdzającą jego poprawność, wywoływująca funkcje załadowywujące i korygujące formę pliku, zwracająca mazeMatrix[][].

  - public static boolean isBinary(String filepath) - metoda sprawdzająca typ pliku (binarny/tekstowy)

- `Moduł 5 - FileValidator`

  Główne funkcjonalności:

  1. Odczyt danych zapisanych w pliku wejściowym binarnym i tekstowym, sprawdzenie ich poprawności.
  2. W razie błędu wywołuje informację przy pomocy MessageUtils.
  3. Współpracuje z klasami: Loader(wywoływany), MessageUtils, Coordinates.

  Metody publiczne:

  - public static void ErrorMessage(String message) - metoda wywołująca popup z ikoną błędu wyświetlający wiadomość podaną w argumencie.

- public static void SuccessMessage(String message) - metoda wywołująca popup z ikoną informacji wyświetlający wiadomość podaną w argumencie.

- `Moduł 6 - Solver`

  Główne funkcjonalności:

  1. Przeskanowanie labiryntu w poszukiwaniu ścieżek
  2. Znalezienie najkrótszej ścieżki i zwrócenie jej.
  3. Współpracuje z klasami: Maze(wywoływany), MessageUtils, Coordinates.

  Metody publiczne:

  - public List<Coordinates> solve(char[][] maze, Coordinates entrance, Coordinates exit) - wywołuje metody rozwiązujące labirynt, zwraca ścieżkę.

- `Moduł 7 - Downloader`

  Główne funkcjonalności:

  1. Konwertuje ścieżkę na format pliku binarnego (literka N/S/W/E + iloś kroków zapisana bitowo w 1 bajcie) i zapisuje ją do niego.
  2. Konwertuje ścieżkę na format TURN RIGHT/LEFT, FOWARD [liczba] i zapisuje wynik do pliku.
  3. Współpracuje z klasami: Maze(wywoływany), Coordinates, MessageUtils.

  Metody publiczne:

  - public void savePathtoTxt(List<Coordinates> Path, String filename) - wywołuje funkcję konwertującą na odpowiedni format i zapisuje do pliku o ścieżce filename.

  - public void savePathtoBin(List<Coordinates> Pathm) - wywołuje funkcję konwertującą na format i zapisuje do pliku wejściowego.

- `Moduł 8 - Coordinates`

  Główne funkcjonalności:

  1. Klasa przechowywująca koordynaty x, y

- `Moduł 9 - MessageUtils`

  Główne funkcjonalności:

1. Umożliwia innym modułom wyświetlenie informacji o stanie operacji po akcji na interfejsie graficzym.

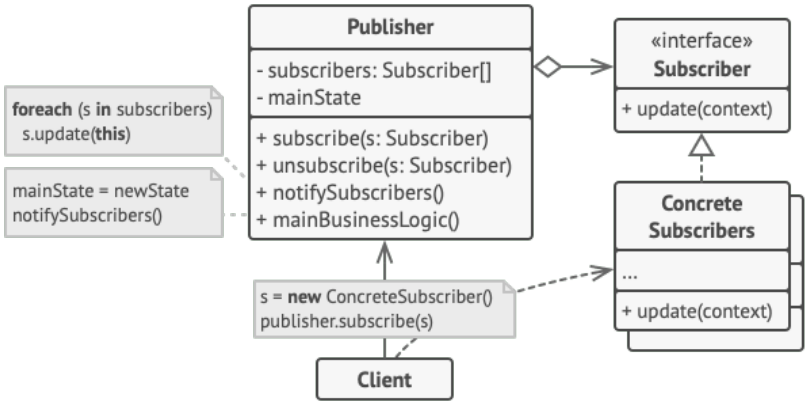
Metody publiczne:

- public void savePathtoTxt(List<Coordinates> Path, String filename) - wywołuje funkcję konwertującą na odpowiedni format i zapisuje do pliku o ścieżce filename.

- public void savePathtoBin(List<Coordinates> Pathm) - wywołuje funkcję konwertującą na format i zapisuje do pliku wejściowego.

3 Opis wykorzystywanych wzorców projektowych

- `Obserwator`

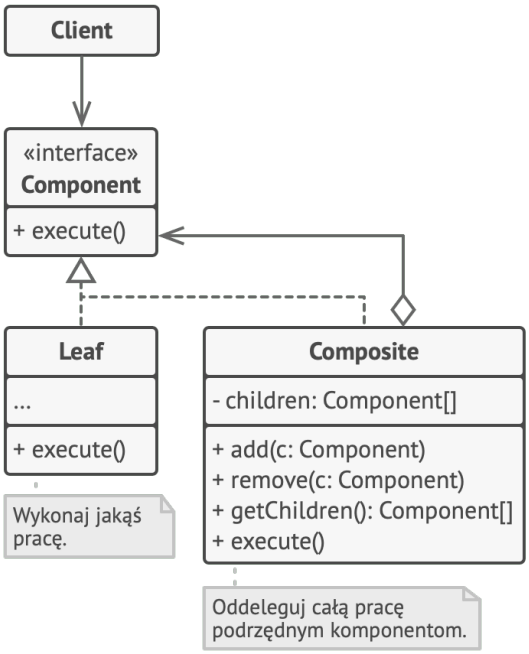


  - Wykorzystywany w java.util.EventListener.

  - Obserwator to czynnościowy (behawioralny) wzorzec projektowy pozwalający zdefiniować mechanizm subskrypcji w celu powiadamiania wielu obiektów o zdarzeniach dziejących się w obserwowanym obiekcie.
    Pozwala każdemu obiektowi implementującemu interfejs subskrypcji otrzymywać powiadomienia o zdarzeniach w obiektach publikujących. Można dodać mechanizm subskrypcji do swoich przycisków, pozwalając klientom na podłączenie do przycisków ich kodu za pośrednictwem własnych klas subskrybentów.
  - W projekcie wzorzec ten jest wykorzystywany przy okazji użycia biblioteki Swing. Do każdego utworzonego przycisku przypisywana jest lista akcji, które czekają na jego naciśnięcie by zacząć się wykonywać.

```
SolveMaze.addActionListener(e -> {
    maze.solve();
    render.setMaze(maze.getMaze());
    applyColors();
    ScrollPane.setViewportView(render);
    render.revalidate();
    render.repaint();
    downloadSolved.setVisible(true);
});
```

- `Kompozyt`



  - Wykorzystywany w java.awt.Container#add(Component)
  - Kompozyt to strukturalny wzorzec projektowy pozwalający komponować obiekty w struktury drzewiaste, a następnie traktować te struktury jakby były osobnymi obiektami. Określa on dwa podstawowe typy elementów współdzielących jednakowy interfejs: proste liście oraz złożone kontenery. Kontener może być złożony zarówno z liści, jak i z innych kontenerów. Pozwala to skonstruować zagnieżdżoną, rekurencyjną strukturę obiektów przypominającą drzewo.
  - W projekcie labiryntu wzorzec ten wykorzystywany jest przy okazji tworzenia interfejsu przy pomocy biblioteki Swing. Komponentami w tym przypadku są kontenery, a liściami poszczególne elementy jak np. guziki.

```
<colspec value="1111:150px:noGrow"/>
<colspec value="left:12dlu:noGrow"/>
<constraints>
  <xy x="20" y="20" width="744" height="939"/>
</constraints>
<properties/>
<border type="none"/>
<children>
  <component id="f06e8" class="javax.swing.JLabel" binding="ProgrammeName">
    <constraints>
      <grid row="2" column="2" row-span="1" col-span="8" vsize-policy="0" hs
      <forms/>
    </constraints>
    <properties>
      <text value="MazeSolver"/>
    </properties>
  </component>
  <component id="addab" class="javax.swing.JLabel" binding="ColorInfo">
    <constraints>
      <grid row="10" column="2" row-span="1" col-span="8" vsize-policy="0" h
      <forms/>
    </constraints>
```

Fragment wykorzystania wzorca composite w projekcie. W jednym elemencie mogą znajdować się różne inne, lub podobne elementy.