API (Application Programming Interface) – zbiór reguł ściśle opisujący, w jaki sposób programy lub podprogramy komunikują się ze sobą; specyfikacja wytycznych, jak powinna przebiegać interakcja między komponentami programowymi; celem jest dostarczenie odpowiednich specyfikacji podprogramów, struktur danych, klas obiektów i wymaganych protokołów komunikacyjnych

Gra Saper

Jakie mamy moduły:
- **MainActivity.java** – strona docelowa z grą; na razie jedyna, więc również strona startowa. Później stroną startową będzie ta z logowaniem, która będzie przekierowywać do strony z wyborem poziomu trudności, która będzie przekierowywać do strony z grą
- **activity_main.xml** – tekstowy opis interfejsu użytkownika. Interfejs użytkownika tworzony graficznie, activity_main.xml generowany na podstawie zmian designu
- element GameEngine: klasa Game w pliku **Game.java**
- element GameEngine: klasa abstrakcyjna Field w pliku **Field.java**
- element GameEngine: klasa EmptyField w pliku **EmptyField.java**
- element GameEngine: klasa Bomb w pliku **BombField.java**

activity_main.xml
- guzik restart (id: “restart”, onClick: restart)
- guzik setFlag (id: “setFlag”, onClick: setFlag)
- pole tekstowe z feedbackiem (id: “feedback”)
- wewnętrzna warstwa z guzikami symbolizującymi pola planszy (id: numer pola, onClick: handleClick)

Klasa MainActivity.java

Arybuty:
- guzik restart
- guzik setFlag
- pole tekstowe z feedbackiem. Wyświetlane na końcu gry
- lista z guzikami (do wykorzystania przy kończeniu gry)
- int ilość wierszy, int ilość kolumn, int ilość bomb
- TableLayout mainTL (warstwa wewnątrz domyślnej warstwy MainActivity, służy do reprezentowania planszy)
- Game currentGame (obiekt klasy Game, która jest głównym silnikiem)

Metody:

- **void onCreate(Bundle savedInstanceState)**

Znajduje przez id atrybuty klasy MainActivity. Tworzy obiekt klasy Game, przekazując liczbę wierszy, kolumn i bomb do postawienia. Generuje planszę (tabelę guzików), dodając guziki do listy. Ustawia tekst w polu z feedbackiem i z flagą.

- **void onClick(View v)**

Znajduje id przypisane do v. Wywołuje metodę int handleClick(int id) klasy Game, która zwraca w wyniku liczbę całkowitą z zakresu [-3, 8]. Ustawia odpowiedni tekst na View v lub wywołuję metodę void gameEnds(boolean win) klasy MainActivity przy okazji wejścia na bombę. Jeśli nie weszliśmy na bombę, wywołuje metodę boolean isWin() klasy Game i w razie wyniku true wywołuje void gameEnds(true).

- **void restart(View v)**

Ustawia widok każdego z guzików i pola tekstowego na początkowy. Ustawia klikalność guzików. Generuje nowy obiekt klasy Game.

- **void setFlag(View v)**

Wywołuję metodę bool isFlagSet() klasy Game, żeby wiedzieć, w jaki sposób zmienić wygląd guzika. Zmienia wygląd guzika. Wywołuje metodę void changeFlag() klasy Game.

- **void gameEnds()**

Przechodzi po liście guzików. Wywołuje metodę boolean showIfBomb(int id). Zmienia klikalność każdego z guzików i wyświetla feedback.

Klasa Game.java

Atrybuty:

- int ilość wierszy, ilość kolumn, ilość bomb
- lista obiektów klas dziedziczących po nadklasie Field
- boolean flag, ustawione początkowo na false
- lista numerów pól, którym będziemy przypisywali bomby
- int ilość pól do odkrycia

Metody:

- **Game(int rows, int columns, int bombs)**

Konstruktor. Przypisuje odpowiednie wartości atrybutom: ilość wierszy, ilość kolumn, ilość bomb, ilość pól do odkrycia (rows * columns – bombs). Losuje numery pól, którym będziemy przypisywali bomby (bombs razy, z zakresu [0, rows *

columns). Jeśli wylosowany numer już znajduje się w liście, losuje jeszcze raz i nie zmienia licznika odbytych losowań. Potem dla i z zakresu [0, rows * columns) sprawdza, czy i znajduje się wśród wylosowanych wcześniej numerów, po czym wrzuca do listy pól nowoutworzony obiekt klasy EmptyField lub BombField. Następnie przechodzi po liście pól i sprawdza, ile pól wokół to bomby (korzystając z funkcji boolean isBomb() klasy Field lub z funkcji showIfBomb(id)) i na tej podstawie ustawia atrybuty obiektów podklas klasy Field (int bombsAround).

- **int handleClick(int id)**
  Zwraca wynik otrzymany po wywołaniu funkcji int handleClick(flag) na odpowiednim obiekcie podklasy klasy Field, znalezionym dzięki id w liście będącej atrybutem klasy Game.

- **boolean isWin()**
  Zwraca true, jeśli ilość pól do odkrycia to 0. False w przeciwnym przypadku.

- **boolean isFlagSet()**
  Zwraca wartość logiczną zmiennej flag.

- **void changeFlag()**
  Zmienia wartość logiczną zmiennej flag.

- **bool showIfBomb(int id)**
  Zwraca wynik funkcji isBomb wykonanej na odpowiednim obiekcie podklasy klasy Field.

Klasa Field.java

Atrybuty:
- int bombsAround
- boolean flagSet (początkowo false)

Metody:

- **bool isBomb() (abstrakcyjna)**
- **int handleClick(boolean flagActive) (nie jest abstrakcyjna)**
  Jeśli **this.flagSet** i **flagActive**, to ustawia flagę i zwraca -3.
  Jeśli **!** **this.flagSet** i **flagActive**, to ustawia flagę i zwraca -2.
  Jeśli **this.flagSet** i **!** **flagActive**, to nic nie robi.
  Jeśli **!** **this.flagSet** i **!** **flagActive**, to zwraca wynik funkcji showNumberOrBomb().
- **int showNumberOrBomb() (abstrakcyjna)**
- **void setBombsAround(int number) (nie jest abstrakcyjna, setter)**

Klasa EmptyField.java

Atrybuty:

Odziedziczone.

Metody:

- **bool isBomb()**

  Zwraca false.

- **int showNumberOrBomb()**

  Zwraca wartość atrybutu bombsAround.

Klasa BombField.java

Atrybuty:

Odziedziczone.

Metody:

- **bool isBomb()**

  Zwraca true.

- **int showNumberOrBomb()**

  Zwraca -1.