Wisielec

Raport z testów.

16.06.2018

Spis treści:

ad 1. Wprowadzenie, terminologia, metodyka

Testowanie oprogramowania jest czynnością przeprowadzaną w czasie procesu developmentu określonego produktu w celu dostarczenia informacji o obecnej jakości oprogramowania względem założonych scenariuszy użytkowania.

Testowania pozwala na zobrazowanie w sposób obiektywny I niezależny (zgodny z normami) obecnego stanu poprawności kodu.

Skrupulatnie przeprowadzone testy pomagają we wczesnym wykryciu defektów oprogramowania, oraz ich wyeliminowaniu. Ważnym elementem na tym etapie jest dostarczenie informacji o powadze odkrytego błędu, oraz sposobie jego reprodukcji w czasie użytkowania oprogramowania.

Testy systemowe I akceptacyjne pozwalają na podjęcie decyzji czy produkt jest gotowy do użytku.

Do głównych celów wykonywanych przez zespół testów należało dostarczenie informacji o:
- poprawnym zachowaniu aplikacji dla wszelkich rodzajów danych wejściowych,
- wykonywaniu założonych funkcji w akceptowalnym czasie,
- stabilności oprogramowania,
- zgodności oprogramowania z założeniami projektowymi,
- możliwści uruchomienia oprogramowania na różnych platformach testowych.

W naszym przypadku platformami testowymi były systemy operacyjne z rodziny Microsoft Windows 10, oraz dystrybucja Ubuntu systemu Linux.

Do testowania jednostkowego tworzonego produktu wykorzystano framework **jUnit** w wersji **4.12**.

ad 2. Kalendarium testowe

TESTY JEDNOSTKOWE

Data przeprowadzenia testów: 05.05.2018r. - 17.06.2018r
Rezultat: Oprogramowania pomyślnie przechodzi testy jednostkowe.
Dodatkowe informacje: Tworzenie testów jednostkowych polegało na stworzeniu unit testów dla klas: CsvDao, HangManEntity, RankEntity, WordEntity oraz Game w wyniku czego utworzono następujący zbiór klas: **CsvDaoTest, HangManEnitityTest, RankEntityTest, WordEntityTest, GameTest.**

W czasie testów jednostkowych wykorzystano metodę czarnej skrzynki.

TESTY SYSTEMOWE

Data przeprowadzenia testów: 13.06.2018r – 17.06.2018r
Rezultat: Oprogramowania pomyślnie przechodzi testy systemowe.
Dodatkowe informacje: Testowanie systemowe przeprowadzono względem wszelkich scenariuszy użytkowania (wskazane w dokumencie “Scenariusze użytkowania” dołączonym do wyżej opisywanego projektu), zarówno na systemach Linux (Ubuntu 16.04) jak i Windows (10 i 8.1).

TESTY AKCEPTACYJNE

Data przeprowadzenia testów: 17.06.2018r – 18.06.2018r
Rezultat: Oprogramowania pomyślnie przechodzi testy akceptacyjne.

ad 3. Wybrane przypadki, testy jednostkowe

Zaimplementowane testy jednostkowe można znaleźć w folderze “SRC/src/test/java” w katalogu głównym projektu.

Przed każdą metodą testową wywoływana jest metoda setUp, pozwalająca na stworzenie nowej instancji testowanego obiektu.

Metody prywatne nie były testowane.

Do zautomatyzowania procesu testowania oprogramowania posłużono się narzędziem wspierającym integrację ciągłą Travis CI, pozwala ono na zbudowanie, oraz automatyczne przetestowanie (względem testów jednostkowych) kodu znajdującego się w repozytorium Github.

CsvDaoTest

Wersja ostateczna w commicie: **e7ce005a4d164c87dfc3177631ff956a0d6f5fdc**
12.06.2018r.

Wersja pierwotna w commicie: **210df9e3a955d589c9440c49659e94650a91ec17**

18.06.2018r.

Powyższa klasa reprezentuje zestaw testów służących do analizy poprawności implementacji komunikacji z bazą danych w aplikacji.

Lista zmiennych prywatnych:

```
private CsvDao testDao;
```

Lista metod:

setUp → metoda pomocnicza, zestawienie połączenia z zamockowaną bazą danych, korzysta z dekoratora @Before, spodziewany wynik: instancja klasy.

```
public void setUp() {
    String path = "sample.csv";
    Category category = Category.SPORT;
    LevelDifficulty level = LevelDifficulty.EASY;
    testDao = new CsvDao(path, category, level);
}
```

readRecordsTest → test poprawności wykonania odczytu z bazy danych, spodziewany wynik: odczyt dokładnie 90 rekordów.

```
public void readRecordsTest() {
    List<List<String>> records = testDao.readRecords();

    Assert.assertEquals(90, records.size());
}
```

canAddTest → test poprawności możliwości dodania wpisu do bazy danych, spodziewany rezultat: prawda,

```
    public void canAddTest() {
        String wordToAdd = "testword";
        String successMessage = "Haslo dodane do bazy";

        try {
            Assert.assertEquals(successMessage, testDao.canAdd(wordToAdd, wordToAdd));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }
```

canAddNotMatchingTest → test poprawności możliwości dodania niepoprawnego hasła do bazy, spodziewany rezultat: prawda (nieudana próba dodania)

```
public void canAddNotMatchingTest() {
    String wordToAdd = "testword";
    String wordToAddConfirmed = "wordtest";
    String failureMessage = "Hasla nie zgadzaja sie";

    try {
        Assert.assertEquals(failureMessage, testDao.canAdd(wordToAdd, wordToAddConfirmed));
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (URISyntaxException e) {
        e.printStackTrace();
    }
}
```

canAddTooLongTest → test poprawności niemożności dodania zbyt długiego tekstu do bazy danych, spodziewany rezultat: prawda (hasło zbyt długie, komunikat o błędzie)

```
    public void canAddTooLongTest() {
        String wordToAdd = "testwordtestwordtestwordtestword";
        String failureMessage = "Nieprawidlowa dlugosc hasla";

        try {
            Assert.assertEquals(failureMessage, testDao.canAdd(wordToAdd, wordToAdd));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }
```

canAddTooShortTest → j/w przy czym testowo możliwość dodania zbyt krótkiego hasła, spodziewany rezultat: prawda (hasło zbyt krótkie, komunikat o błędzie)

```
public void canAddTooShortTest() {
    String wordToAdd = "a";
    String failureMessage = "Nieprawidlowa dlugosc hasla";

    try {
        Assert.assertEquals(failureMessage, testDao.canAdd(wordToAdd, wordToAdd));
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (URISyntaxException e) {
        e.printStackTrace();
    }
}
```

canAddNoLettersTest → test możliwości dodania do bazy danych hasła nie składającego się z liter, spodziewany rezultat: prawda (hasło nieprawidłowe, komunikat o błędzie)

```
public void canAddNoLettersTest() {
    String wordToAdd = "12345";
    String failureMessage = "Haslo musi skladac sie z liter";

    try {
        Assert.assertEquals(failureMessage, testDao.canAdd(wordToAdd, wordToAdd));
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (URISyntaxException e) {
        e.printStackTrace();
    }
}
```

HangManEntityTest

Wersja pierwotna w commicie:210df9e3a955d589c9440c49659e94650a91ec17
18.05.2018r

Wersja ostateczna w commicie: j/w

Klasa testująca metody opisujące aktualny stan wisielca.

Lista metod:

setUp → metoda wstępna, przygotowuje instancję wisielca, korzysta z dekoratora @Before

```
public void setUp() throws Exception {
    testEntity = new HangManEntity();
}
```

getHearthsTest → test poprawności zwracanej liczby pozostałych punktów życia wisielca, spodziewana wartość: prawda (10 żyć w nowo utworzonej instancji)

```
public void getHearthsTest() {
    int expectedLives = 10;

    int resultLives = testEntity.getHearths();

    Assert.assertEquals(expectedLives, resultLives);
}
```

decrementLivesTest → test poprawności metody zmniejszającej liczbę żyć o jedno, spodziewany wynik: prawda (poprawna dekrementacja liczby żyć)

```
public void decrementLivesTest() {
    int expectedLives = 9;

    testEntity.decrementLives();
    int resultLives = testEntity.getHearths();

    Assert.assertEquals(expectedLives, resultLives);
}
```

isAliveTest → test poprawności metody zwracającej informację czy wisielec jest wciąż żywy (posiada dodatnią liczbę żyć), spodziewany wynik: prawda (nowa instancja wisielca jest żywa)

```
public void isAliveTest() {
    boolean isAlive = testEntity.isAlive();

    Assert.assertTrue(isAlive);
}
```

isDeadTest → test poprawności metody zwracającej informację czy wisielec jest martwy, spodziewany rezultat: prawda (dziesięciokrotne zmniejszenie liczby żyć podstawowej instancji klasy wisielca prowadzi do jego uśmiercenia, licznik żyć jest równy 0)

```
public void isDeadTest() {
    for (int i = 0; i < 10; i++) {
        testEntity.decrementLives();
    }

    boolean isAlive = testEntity.isAlive();

    Assert.assertFalse(isAlive);
}
```

RankEntityTest

Wersja pierwotna w commicie: ea1a5397e1f7891189ff46854142b6ad46636966
31.05.2018r.

Wersja ostateczna w commicie: 3de2333632c0465787111c1fcf308b512189525f,
2.05.2018r.

Klasa testująca metody operacji na wbudowanym w grze rankingu.

Lista metod:

setUp → metoda pomocnicza, tworzy nową instancję rankingu

```
public void setUp() {
    testEntity = new RankEntity();
}
```

addToRankTest → test sprawdzający poprawność dodawania encji punkt:słowo do instancji rankingowej, spodziewany rezultat: prawda (dodana encja jest zgodna z predefiniowaną encją testową ‘testRecord’)

```
public void addToRankTest() {
    testEntity.addToRank(testPoints, testWord);
    Assert.assertEquals(testEntity.getScoreAtSpecificIndex(0), testRecord);
}
```

serializeTest → test pozwalający stwierdzić poprawność procesu serializacji, sprawdza poprawność zapisu pliku na dysk po etapie serializacji, spodziewany rezultat: prawda (plik istnieje)

```
public void serializeTest() {
    RankEntity.serialize(testEntity);
    File file = new File(RankEntity.class.getResource("/rank.ser").getFile());
    Assert.assertTrue(file.exists());
}
```

deserializeTest → test pozwalający stwierdzić poprawność odczytanych danych (zdeserializowanych) z tymi które uprzednio zostały zapisane, spodziewany rezultat: prawda (obydwie encje są jednakowe co do przetrzymywanych danych)

```
public void deserializeTest() {
    RankEntity deserializedEntity = RankEntity.deserialize();

    for(int i = 0; i < testEntity.getNumberOfRecords(); ++i) {
        Assert.assertEquals(testEntity.getScoreAtSpecificIndex(i),
                deserializedEntity.getScoreAtSpecificIndex(i));
    }
}
```

WordEntityTest

Wersja pierwotna w commicie: 210df9e3a955d589c9440c49659e94650a91ec17
18.05.2018r.

Wersja ostateczna w commicie: j/w

Powyższa klasa reprezentuje zestaw testów służących do analizy poprawności klasy zgadywanego słowa w programie.

Lista zmiennych prywatnych:

```
private WordEntity testEntity;
private final String testWord = "test";
private String testUnknownWord = "____";
```

Lista metod:

setUp → klasa pomocnicza, zwracająca nową instancję reprezentującą słowo ‘testWord’

```
public void setUp() {
    testEntity = new WordEntity(testWord);
}
```

revealLetterTest → metoda pozwalająca przetestować poprawność działania metody odkrywającej litery w odgadywanym słowie, spodziewany rezultat: prawda (odkryto poprawnie obydwie litery)

```
public void revealLetterTest() {
    String revealedLetter = "t__t";

    testEntity.revealAllDuplicatesIfWordToGuessContainsLetter('t');
    char resultWord[] = testEntity.getUnknownWord();

    Assert.assertArrayEquals(revealedLetter.toCharArray(), resultWord);
}
```

getWordToGuessTest → metoda testująca poprawność zwracanego słowo do zgadnięcia, spodziewany rezultat: prawda (słowa identyczne w nowo przetrzymywanej instancji)

```
public void getWordToGuessTest() {
    String resultWord = testEntity.getWordToGuess();

    Assert.assertEquals(testWord, resultWord);
}
```

getUnknownWordTest → test pozwalający na stwierdzenie

poprawności zwracanego obecnego stanu odgadywanego słowa, spodziewany rezultat: prawda (brak odkrytych liter)

```
public void getUnknownWordTest() {
    char resultWord[] = testEntity.getUnknownWord();

    Assert.assertArrayEquals(testUnknownWord.toCharArray(), resultWord);
}
```

addToAlreadySelectedLetterTest → metoda pozwalająca stwierdzić poprawność wyboru obecnie zgadywanej (wybranej) litery, spodziewany rezultat: prawda (litery są identyczne)

```
public void getUnknownWordTest() {
    char resultWord[] = testEntity.getUnknownWord();

    Assert.assertArrayEquals(testUnknownWord.toCharArray(), resultWord);
}
```

alreadySelectedLetterTest → test pozwalający stwierdzić poprawność wyboru obecnie wybranej litery, spodziewany rezultat: prawda (dla wyboru litery ‘e’), fałsz (dla wyboru litery ‘t’, która nie została wybrana)

```
public void alreadySelectedLetterTest() {
    testEntity.addToAlreadySelectedLetter('e');

    Assert.assertTrue(testEntity.alreadySelectedLetter('e'));
    Assert.assertFalse(testEntity.alreadySelectedLetter('t'));
}
```

notAllLettersRevealedTest → test pozwalający stwierdzić czy nie wszystkie litery w wyrazie zostały już odkryte, spodziewany rezultat: fałsz (nie odkryto wszystkich liter po wprowadzeniu litery ‘t’)

```
public void notAllLettersRevealedTest() {
    testEntity.revealAllDuplicatesIfWordToGuessContainsLetter('t');

    Assert.assertFalse(testEntity.allLettersRevealed());
}
```

allLettersRevealedTest → test pozwalający sprawdzić poprawność metody zwracającej informację czy wszystkie litery w słowie zostały odkryte, spodziewany rezultat: prawda (odkryto całe słowo po wprowadzeniu liter ‘t’, ‘e’, ‘s’)

```
public void allLettersRevealedTest() {
    testEntity.revealAllDuplicatesIfWordToGuessContainsLetter('t');
    testEntity.revealAllDuplicatesIfWordToGuessContainsLetter('e');
    testEntity.revealAllDuplicatesIfWordToGuessContainsLetter('s');

    Assert.assertTrue(testEntity.allLettersRevealed());
}
```

wordToGuessContainsLetterTest → test pozwalający stwierdzić czy dane słowo zawiera podana literę, spodziewana wartość: prawda (słowo testowe zawiera literę ‘t’)

```
public void wordToGuessContainsLetterTest() {
    Assert.assertTrue(testEntity.doesTheWordToGuessContainsLetter('t'));
}
```

wordToGuessNotContainsLetterTest → j/w dla podanej litery która nie istnieje w wyrazie zwraca wartość false, spodziewana wartość: false (litera ‘a’ nie jest obecna w wyrazie testowym)

```
public void wordToGuessNotContainsLetterTest() {
    Assert.assertFalse(testEntity.doesTheWordToGuessContainsLetter('a'));
}
```

GameTest

Wersja ostateczna w commicie: aa402d0155d84202862b2b57d12abd505549f3cd
13.06.2018r

Wersja pierwotna w commicie: e7ce005a4d164c87dfc3177631ff956a0d6f5fdc
12.06.2018r

Klasa reprezentująca zestaw testów dla głównej instancji gry.

Lista zmiennych prywatnych:

```
private Game testGame;
```

Lista metod:

setUp → metoda pomocnicza, tworzy instancję gry, z predefiniowanymi opcjami.

```
public void setUp() throws Exception {
    testGame = new Game(LevelDifficulty.EASY, Category.ALL, GameTypes.NEW_GAME, false);
}
```

checkIfInGameTest → test pozwalający stwierdzić poprawność zwracanej wartości informującej, czy aktualnie znajdujemy się w grze, spodziewany rezultat: prawda (po wystartowaniu, znajdujemy się w grze)

```
public void checkIfInGameTest() {
    testGame.start();
    Assert.assertTrue(testGame.checkIfInGame());
}
```

ad. 4 Scenariusz użytkowania, testy systemowe

SCENARIUSZ	WYNIK TESTU
1. Ranking	Sukces
Brak wyników do wyświetlenia	Sukces
Zamknięcie aplikacji w trakcie wyświetlania rankingu przez przycisk „X” / awaria sprzętowa	Sukces
Historia zmian:	
Usunięcie błędu usuwania duplikatów	commit: 5010114c48ac104f6a5cfaa60889b3788a908f72
Wersja testowana	commit: d3f6cb7fbc857f840cb066792cb5ebc6ee80262f
2. Wizualizacja liczby żyć	Sukces
Wybór litery występującej w haśle	Sukces
Odgadnięcie hasła poprzez poprawne wybranie ostatniej nieznanej litery/wpisanie poprawnego hasła do odpowiedniego pola	Sukces
Wykorzystanie ostatniej szansy/ wpisanie niepoprawnego hasła i zatwierdzenie go przyciskiem	Sukces
Zamknięcie aplikacji w trakcie gry, z zapisaniem stanu gry	Sukces
Koniec czasu przed odgadnięciem hasła (w trybie gry na czas)	Sukces
Historia zmian:	
Wersja testowana	commit: cce7ab600d427ca046fb95eb66a4eebe8e129fbf
3. Serializacja	Sukces
Tryb gry na czas	Sukces
Próba wczytania nieistniejącego zapisu gry	Sukces
Historia zmian:	

refaktoryzacja kodu, błędy serializacji	commit: 006a645a32cb31be7b6e59078c1eb83b1ad76224
Wersja testowana:	commit: aa402d0155d84202862b2b57d12abd505549f3cd
4. Tryb nocny	Sukces
Zamknięcie aplikacji w trybie nocnym	Sukces
Ponowna zmiana trybu nocnego	Sukces
Historia zmian:	
Zaktualizowana wersja trybu nocnego:	commit: 2b7e92223bc4999539177a7bfca89ee2a3aa1f09
Wersja testowana:	commit: a77b07abafba19d5d3d5e96833a4fe1d6549b3c8
5. Prosta animacja (“pajacyk”)	Sukces
Zamknięcie aplikacji z zapisem gry w takcie trwania animacji	Sukces
Ponowne naciśnięcie na wisielca w trakcie animacji	Sukces
Wybór litery w trakcie trwania animacji/ Próba odgadnięcia hasła	Sukces
Historia zmian:	
Zaktualizowana wersja trybu nocnego:	commit: 2b7e92223bc4999539177a7bfca89ee2a3aa1f09
Wersja testowana:	commit: a77b07abafba19d5d3d5e96833a4fe1d6549b3c8
6. Dźwięki w grze podczas rozgrywki	Sukces
Wybór litery, która nie występuje w haśle	Sukces
Wykorzystanie ostatniej szansy podczas gry/ podanie niepoprawnej odpowiedzi podczas odgadywania hasła	Sukces
Odgadnięcie hasła poprzez, podanie brakującej litery/ wpisanie go do odpowiedniego pola	Sukces
Koniec czasu przed odgadnięciem hasła (w trybie gry na czas)	Sukces
Historia zmian:	

Wersja testowana:	commit: cce7ab600d427ca046fb95eb66a4eebe8e129fbf
7. Poziomy trudności	Sukces
Zamknięcie aplikacji w momencie wybierania trybu gry przez przycisk „X”/awarię sprzętową	Sukces
Zamknięcie aplikacji w momencie wybierania poziomu trudności przez przycisk „X”/awarię sprzętową	Sukces
Zamknięcie aplikacji w momencie wybierania kategorii przez przycisk „X”/awarię sprzętową	Sukces
8. Dwa tryby gry	Sukces
Zamknięcie aplikacji w momencie wybierania trybu gry przez przycisk „X”/awarię sprzętową	Sukces
Zamknięcie aplikacji w momencie wybierania poziomu trudności przez przycisk „X”/awarię sprzętową	Sukces
Zamknięcie aplikacji w momencie wybierania kategorii przez przycisk „X”/awarię sprzętową	Sukces
Wersja testowana:	commit: 61a999a76d3980d744f6a7058fb28e18c732702a
9. Możliwość dodawania własnych słów do bazy danych	**Wykryty błąd:** • po zatwierdzeniu hasła gra przenosiła użytkownika do menu głównego; **rozwiązany:** po zatwierdzeniu gracz pozostaje w ekranie dodawania hasła i może wprowadzić kolejne słowo do bazy;
Zamknięcie aplikacji w momencie wybierania kategorii przez przycisk „X”/awarię sprzętową	Sukces
Zamknięcie aplikacji po wpisaniu nowego hasła przez przycisk „X”/awarię sprzętową (Hasło nie zostaje dodane do bazy danych)	Sukces
Zamknięcie aplikacji, po potwierdzeniu wpisanego hasła w polu tekstowym, przez przycisk „X”/awarię sprzętową (Hasło nie	Sukces

zostaje dodane do bazy danych)	
Powrót do menu po wpisaniu hasła w pole tekstowe (Hasło nie zostaje dodane do bazy danych)	Sukces
Wpisane hasło jest nieprawidłowe (nie spełnia co najmniej jednego z wymaganych warunków)	**Wykryte błędy:** ● warunek mówiący, że słowa nie mogą zaczynać ani kończyć się spacjami, był ignorowany; **rozwiązany:** program automatycznie obcina białe znaki; ● warunek mówiący, że słowa nie mogą zawierać dwóch lub więcej spacji obok siebie, był ignorowany; **rozwiązany:** program automatycznie zamienia wielokrotne spacje na pojedynczą; ● po pojawieniu się komunikatu o błędzie w haśle i wyłączeniu go, gra cofała użytkownika do menu głównego; **rozwiązany:** po zamknięciu komunikatu gracz pozostaje na ekranie dodawania hasła;
Wersja testowana:	commit: 251be7d119b84ca5bc250e01f38b4ee8a25492ec
10. Kategorie haseł	Sukces
Zamknięcie aplikacji w momencie wybierania trybu gry przez przycisk „X”/awarię sprzętową	Sukces
Zamknięcie aplikacji w momencie wybierania poziomu trudności przez przycisk „X”/awarię sprzętową	Sukces
Zamknięcie aplikacji w momencie wybierania kategorii przez przycisk „X”/awarię sprzętową	Sukces
Wersja testowana:	commit: 251be7d119b84ca5bc250e01f38b4ee8a25492ec

5. Podsumowanie testów, wnioski

W czasie testów aplikacji stwierdzono zgodność otrzymanego produktu z założeniami projektowymi (względem podstawowych scenariuszy użytkowania). Testy systemowe pozwoliły zobrazować w jaki sposób aplikacja będzie użytkowana w środowisku produkcyjnym.

Wyżej opisywana aplikacja przeszła pomyślnie etap testowania, zarówno względem testów jednostkowych, jak systemowych I akceptacyjnych.

Kod aplikacji (ze względu na odkryte błędy) musiał być wielokrotnie refaktorowany celem usunięcia odkrytych uchybień.

Przeprowadzenie procesu testowania pozwoliło zespołowi na realne doświadczenie problemów związanych z inżynierią oprogramowania, oraz zobrazowaniu komunikacji w zespole (na linii programista – tester).