

Plan Testów

Projekt: Bomberman

1. Wprowadzenie	4
1.1 Cele	4
1.2 Odniesienia	4
1.3 Przegląd projektu i funkcjonalności	4
1.4 Przegląd testów	4
1.4.1 Organizacja	4
1.4.2 Harmonogram	4
1.4.3 Podział pracy	5
1.4.4 Narzędzia i metody	5
1.4.5 Forma dokumentacji	6
2. Szczegóły	7
2.1 Proces testowania	7
2.1.1 Podział na testy jednostkowe, integracyjne i systemowe	7
2.1.2 Testy jednostkowe	7
2.1.2.1 Wygenerowane przypadki testowe	7
2.1.2.2 Wyniki testów	9
2.1.2.3 Raporty incydentów testowych	10
2.1.3 Testy systemowe	10
2.1.3.1 Podział testów	10
2.1.3.2 Wygenerowane przypadki testowe	10
2.1.3.2.1 Testy funkcjonalne	10
2.1.3.2.2 Testy wydajnościowe	10
2.1.3.2.4 Testy dokumentacyjne	11
2.1.3.3 Wyniki testów	12
2.1.3.4 Raporty incydentów testowych	12
3. Podsumowanie	16
3.1 Komentarz autorski	16
4.1 Słowniczek	17

1. Wprowadzenie

1.1 Cele

1. Tworzenie szybkich, niezależnych, powtarzalnych, samo kontrolujących się testów
2. Lokalizacja błędów
3. Wczesne zapobieganie błędnemu działaniu

1.2 Odniesienia

W poniższym dokumencie znajdują się odniesienia do elementów dokumentacji projektu:

1. IEEE 830
2. Funkcjonalności
3. Dodatkowe funkcjonalności
4. Diagramy UML:
 - a. OtrzymanieObrazen
 - b. PodstawienieBomb
 - c. Poruszanie
 - d. WybuchBomby

1.3 Przegląd projektu i funkcjonalności

Bombberman jest grą zręcznościową, przygotowaną do użytku na komputery osobiste.

Pozwala ona na grę od 1 do 4 graczy naraz, lokalnie. Funkcjonalności zostały spisane w [1], zaś wymagania co do nich w [2] oraz [3] (funkcjonalności dodatkowe).

1.4 Przegląd testów

1.4.1 Organizacja

Podczas procesu testowania dokonać testów jednostkowych, integracyjnych, systemowych (w tej kolejności). Wszystkie poprzednio przygotowane testy powtarzać po refaktoryzacji kodu

Podczas każdego z trzech głównych etapów testowania uwagi należy kierować bezpośrednio do programistów (testy jednostkowe, testy systemowe) lub kierownika projektu (niezgodność z dokumentacją). W przypadkach spornych należy odnieść się do dokumentacji projektu oraz zastosować się do poleceń kierownika projektu. Po naprawieniu błędów przez programistów, testy należy przeprowadzić na nowo (co może oznaczać nowe testy jednostkowe i integracyjne dla metod utworzonych dla brakujących funkcjonalności).

1.4.2 Harmonogram

20.04.19 Ustalenie harmonogramu pracy, podziału obowiązków

25.04.19 - 01.05.19 Wstępne przygotowanie planu testów, narzędzi testowania

01.05.19 Przygotowanie narzędzi pracy

01.05.19 - 07.05.19 Przygotowanie testów jednostkowych oraz testowanie kodu źródłowego przy ich pomocy.

07.05.19 - 12.05.19 Ponowne przeprowadzanie testów jednostkowych po wprowadzeniu poprawek (opcjonalne).
12.05.19 - 19.05.19 Przygotowanie testów integracyjnych oraz testowanie kodu źródłowego przy ich pomocy
19.05.19 - 25.05.19 Ponowne przeprowadzanie testów integracyjnych po wprowadzeniu poprawek, przygotowanie dodatkowych testów jednostkowych (opcjonalne).
25.05.19 - 05.06.19 Przeprowadzanie testów systemowych
05.06.19 - 10.06.19 Ponowne przeprowadzenie testów systemowych po wprowadzeniu poprawek i dokonaniu refaktoryzacji. Przygotowanie dodatkowych testów jednostkowych, integracyjnych oraz ich przeprowadzenie (opcjonalne).
10.06.19 - 15.06.19 Tworzenie ostatecznej dokumentacji testów

Punkty oznaczone jako opcjonalne są zależne od tego, czy zostanie dokonana refaktoryzacja kodu.

EDIT:

Harmonogram ostateczny:

05.05.19 Przygotowanie narzędzi pracy
05.05.19 - 15.05.19 Przygotowanie testów jednostkowych oraz testowanie kodu źródłowego przy ich pomocy.
15.05.19 - 20.05.19 Ponowne przeprowadzanie testów jednostkowych po wprowadzeniu poprawek.
20.05.19 - 27.05.19 Przeprowadzanie testów systemowych
27.05.19 - 03.06.19 Ponowne przeprowadzenie testów systemowych po wprowadzeniu poprawek i refaktoryzacji. Przygotowanie dodatkowych testów jednostkowych oraz ich przeprowadzenie.
01.05.19 - 07.05.19 Tworzenie ostatecznej dokumentacji testów

1.4.3 Podział pracy

Yuliia Buchko - tester jednostkowy, integracyjny

Dominik Wołek - tester systemowy, zebranie i opracowanie dokumentacji.

EDIT: Ostatecznie testy integracyjne nie są wykonywane

1.4.4 Narzędzia i metody

Do testów jednostkowych zastosować narzędzie Godot Unit Testing (GUT). Jest to biblioteka opublikowana na licencji wolnego oprogramowania (MIT license). Narzędzie pozwala na uruchomienie zarówno wszystkich testów, jak i testów z jednego pliku (w tym przypadku: testów dotyczących jednej klasy). Decyzja o wyborze tej biblioteki została podjęta z powodu braku konkurencyjnych bibliotek / frameworków.

Do testów integracyjnych zastosować metodę top-down.

Do testów wydajnościowych zastosować narzędzia platformy Steam.

EDIT:

Z narzędzi dostarczonych przez GUT zostały użyte m.in.:

- Klasy testujące (TestKlasaTestowana)
- Asercje

- assert_eq / assert_ne
- Asercje dotyczące plików: assert_file_exists, assert_file_does_not_exists, assert_file_empty
- Asercje dotyczące wartości logicznych: assert_true, assert_false
- Oznaczenie testu jako nieukończonego (pending)

Testy integracyjne nie są wykonywane.

Testy wydajnościowe mają jedynie jedną wartość mierzalną: FPS.

W celach sprawdzenia zgodności funkcjonalności dodatkowych z dokumentacją użyto programu Peek do nagrania wybuchu bomby do pliku .webm a następnie zmierzenia czasu efektów. Program Peek jest udostępniony na licencji wolnego oprogramowania (GNU General Public Licence).

W trakcie trwania testów przeciążeniowych zastosowano technikę sztucznego ograniczenia zasobów obliczeniowych poprzez uruchamianie wielu programów działających w tle (np. Przeglądarek internetowych).

Testy systemowe przeprowadzono je na:

PC1 (procesor i5-3450, karta graficzna GTX 770, 8GB RAM, Windows 10 64 bit);

Laptop1 (Lenovo Ideapad 510-15IKB, Ubuntu 19 64 bit)

Laptop2 (Lenovo Thinkpad t520i5-2540m, Intel HD 2000, Windows 10 64 bit). Wybór tego laptopa jest nieprzypadkowy: obsługuje jedynie OpenGL 2, a nie 3).

1.4.5 Forma dokumentacji

Przypadki testowe testów jednostkowych należy przedstawić w następującej formie:

1. Klasa testowana
 - a. metoda_testowana(argumenty, metody, testowanej) - komentarz do argumentów lub metody
 - i. przypadek testowy. Oczekiwane zachowanie (np. zmiana wartości pól), Oczekiwana wartość zwracana

Raporty incydentów należy opisać w następującej formie:

Nr. Zgłoszenia	Numer
Podsumowanie	Skrót opisu błędu; tytuł
Odkryty przez	Imię i nazwisko
Priorytet	Wysoki / niski / średni
Status	FIXED / NOT FIXED (RESOLVED / NOT RESOLVED w przypadku błędów dokumentacyjnych)
Kategoria	Jednostkowe / Integracyjne / Funkcjonalne / Dokumentacyjne / Wydajnościowe
Zgłoszony do	Imię i nazwisko
Opis zgłoszenia	Krótki opis Opcjonalne:

	Zrzut ekranu ilustrujący incydent Nazwa komputera
--	--

2. Szczegóły

2.1 Proces testowania

2.1.1 Podział na testy jednostkowe, integracyjne i systemowe

Metody publiczne, nieodwołujące się do obiektów innych klas mają być testowane testami jednostkowymi.

Należy wykonać testy integracyjne, zwłaszcza do metod niemożliwych do przetestowania testami jednostkowymi.

Wszelkie funkcjonalności gotowego projektu mają zostać sprawdzone przy pomocy testów systemowe funkcjonalnych. Ponadto należy przeprowadzić testy wydajnościowe, dokumentacji, opcjonalnie testy przeciążeniowe.

EDIT: Testy integracyjne nie są wykonywane

2.1.2 Testy jednostkowe

Łącznie zostało przygotowanych 137 testów jednostkowych. Pokrywają one metody, które

- A. nie są metodami prywatnymi (z dwoma wyjątkami)
- B. mogą być testowane przy pomocy testów jednostkowych, a nie integracyjnych.

Podjęto się jednego testu metod prywatnych, z powodów:

- A. jest ona kluczowe dla działania programu
- B. specyfika języka GDScript oznacza, że metody prywatne są de facto publiczne (każdy obiekt i funkcja może je wywołać), co ułatwia pisanie testów do takiej funkcji

2.1.2.1 Wygenerowane przypadki testowe

Poniżej zostały przedstawione przypadki testowe.

1. Board
 - a. place_bomb(pos, player) - pos := Vector2(x,y)
 - i. player := nowa instancja, x, y > 0 Oczekiwane zachowanie: placed_by == player
 - ii. player := nowa instancja, x > y = 0 Oczekiwane zachowanie: placed_by == player
 - iii. player := nowa instancja, y > x = 0 Oczekiwane zachowanie: placed_by == player
 - iv. player := nowa instancja, x = y = 0 Oczekiwane zachowanie: placed_by == player
 - v. player := nowa instancja, x, y < 0 Oczekiwane zachowanie: placed_by == player

- vi. player := nowa instancja, $x < y = 0$ Oczekiwane zachowanie:
placed_by == player
- vii. player := nowa instancja, $y < x = 0$ Oczekiwane zachowanie:
placed_by == player
- b. step(number)
 - i. step := 0 Oczekiwana wartość zwracana: Vector2(-64, 0)
 - ii. step := 1 Oczekiwana wartość zwracana: Vector2(0, 64)
 - iii. step := 2 Oczekiwana wartość zwracana: Vector2(64, 0)
 - iv. step := 3 Oczekiwana wartość zwracana: Vector2(0, -64)
- c. get_color()
 - i. Oczekiwana wartość zwracana: tablica kolorów (Color(0,0,0,1), Color(0.8, 0.36, 0.36, 1), Color(0.69, 0.88, 0.9, 1), Color(0.6, 0.98, 0.6, 1), Color(1, 1, 0, 1), Color(0.55, 0.55, 0.55, 1), Color(1, 0.41, 0.71, 1))

2. Player

- a. set_nickname(nickname)
 - i. nickname - poprawny nick ("nick123"); Oczekiwane przypisanie nickname do name
 - ii. Nickname zawiera niedozwolony znak '/', oczekiwane zachowanie: NIE ustawienie name na podany nickname
- b. speed_up():
 - i. Wywołanie procedury i razy, ($1 \leq i \leq 10$). Oczekiwane zachowanie: zwiększenie szybkości gracza o $i * 70$
- c. increase_dmg():
 - i. Wywołanie procedury i razy, ($1 \leq i \leq 10$). Oczekiwane zachowanie: zwiększenie bomb_dmg o i
- d. add_bomb():
 - i. Wywołanie procedury i razy, ($1 \leq i \leq 10$). Oczekiwane zachowanie: zwiększenie can_plant o i
- e. is_immortal()
 - i. Nowo utworzona instancja Player. Oczekiwana wartość: true.
- f. _check_color() - funkcja prywatna odwołująca się do pól color, modulate
 - i. Przypisanie do color koloru niebędącego czarnym (wartości: cyan, blue, white, itd.) Oczekiwane zachowanie: modulate == color
 - ii. Przypisanie do color koloru czarnego (0, 0, 0, 1) Oczekiwane zachowanie: modulate != color

3. Bot

- a. possible_plant(position)
 - i. Liczba posiadanych bomb (can_plant) równa 0. Oczekiwany wartość zwracana: false
- b. opposite_direction(direction)
 - i. direction := "left" Oczekiwana wartość zwracana: "right"
 - ii. direction := "right" Oczekiwana wartość zwracana: "left"
 - iii. direction := "up" Oczekiwana wartość zwracana: "down"
 - iv. direction := "down" Oczekiwana wartość zwracana: "up"

4. Highscore

- a. `get_list()`
 - i. Plik `high.score` istnieje i zawiera 10 wyników (10 par (`nick1`, `value1`) .. (`nick10`, `value10`) gdzie nicki - poprawny nick, `value` - nieujemna liczba). Oczekiwana wartość zwracana: lista z parami `nick`, `value` odpowiadająca tej z pliku
 - ii. Plik `high.score` jest pusty. Oczekiwana wartość zwracana: pusta lista
 - iii. Plik `high.score` nie istnieje. Oczekiwana wartość zwracana: pusta lista
- b. `try_to_add(nickname, result)`:
 - i. `nickname` jest poprawnym nickiem, plik `high.score` istnieje, zawiera mniej niż 10 wyników. Oczekiwana wartość zwracana: `true`.
Oczekiwane zachowanie: po wykonaniu `try_to_add`, plik zawiera wynik (`nickname`, `result`).
 - ii. Plik `high.score` nie istnieje, poprawny nick. Oczekiwane zachowanie: utworzenie nowego pliku `high.score`. Oczekiwana wartość zwracana: `true`
 - iii. Plik `high.score` istnieje, zawiera 10 wyników. `Result < najmniejszego result w high.score`. Oczekiwane zachowanie: po wykonaniu `try_to_add`, plik NIE zawiera wyniku (`nickname`, `result`).
- c. `reset()`
 - i. Plik `high.score` istnieje. Oczekiwane zachowanie: po wykonaniu się procedury otrzymujemy pusty plik
 - ii. Plik `high.score` nie istnieje. Oczekiwane zachowanie: po wykonaniu się procedury otrzymujemy pusty plik

5. Panel

- a. `name_correct(name)`
 - i. `name` := wartość poprawna. Oczekiwana wartość zwracana: `true`
 - ii. `name` := wartość niepoprawna. Oczekiwana wartość zwracana: `false`

6. Scorepair

- a. `sort(a, b)`
 - i. `a.score > a.score > 0` Wartość oczekiwana: `false`
 - ii. `b.score > a.score > 0` Wartość oczekiwana: `true`
 - iii. `a.score = b.score > 0` Wartość oczekiwana: `true`
 - iv. `0 > b.score > a.score` Wartość oczekiwana: `false`
 - v. `0 > a.score > b.score` Wartość oczekiwana: `true`
 - vi. `a.score > 0 > a.score` Wartość oczekiwana: `false`
 - vii. `a.score > a.score > 0` Wartość oczekiwana: `true`

2.1.2.3 Wyniki testów

Z 106 początkowo napisanych testów odnotowano 1 incydent testowy. Po naprawie awarii oraz każdej kolejnej aktualizacji testy były powtarzane. Po pierwszych testach systemowych zostało utworzone kolejne 21 testów jednostkowych. Nie odnotowano więcej incydentów.

2.1.2.3 Raporty incydentów testowych

Nr. Zgłoszenia	1
Podsumowanie	Awaria: funkcji <code>get_list()</code> zwraca zawsze pustą listę
Odkryty przez	Yuliia Buchko
Priorytet	Średni
Status	FIXED
Kategoria	Jednostkowe
Zgłoszony do	Filip Sowa, Rafał Brożek
Opis zgłoszenia	Przypadek testowy: Plik <code>high.score</code> istnieje i zawiera więcej niż jeden poprawnych wpisów (<code>nick</code> , <code>score</code>). Podczas uruchomienia sceny GUT asercja <code>assert_ne(t, [])</code> (gdzie <code>t := highscore.get_list()</code>) przerywa test.

2.1.3 Testy systemowe

2.1.3.1 Podział testów

Przygotowano testy funkcjonalne, wydajnościowe, dokumentacji oraz jeden test przeciążeniowy. Testy funkcjonalne oraz dokumentacji oparto na wymaganiach i scenariuszach użycia zawartych w [2][3] oraz diagramach UML w [4]. Testy wydajnościowe oparto na wymaganiu 9 opisanych w [2].

2.1.3.2 Wygenerowane przypadki testowe

2.1.3.2.1 Testy funkcjonalne

Wygenerowane przypadki testowe dotyczące funkcjonalności projektu zostały w całości oparte na scenariuszach zawartych w [2] oraz diagramach UML [4]. Jako że finalna wersja testowa zawiera również dodatkowe funkcjonalności, zostały przetestowane scenariusze zawarte w [3]

2.1.3.2.2 Testy wydajnościowe

Testy wydajnościowe oparto na [2] a dokładnie na wymaganiu 4 oraz wymaganiu 9.

Przypadek testowy 1.

Tester sprawdza liczbę klatek na sekundę na komputerze pozwalającym na działanie gry w 60 FPS. Test przeprowadzany jest podczas rozgrywki. Test przeprowadzany jest na wszystkich komputerach. Oczekiwana ilość FPS: 60

Przypadek testowy 2.

Tester sprawdza liczbę klatek na sekundę na komputerze pozwalającym na działanie gry w 60 FPS. Test przeprowadzany jest podczas przeglądania highscore, zmiany mapy, zmiany ustawień sterowania, zmiany ustawień dźwięku, zmiany nicków oraz kolorów gracza i botów. Test przeprowadzany jest na wszystkich komputerach. Oczekiwana ilość FPS: 60.

2.1.3.2.3 Test przeciążeniowy

Przypadek testowy 1.

Tester wymusza ograniczenie zasobów komputera (pamięć RAM <1 GB, zużycie procesora > 85%) poprzez uruchomienie innych programów w tle. Test przeprowadzany jest na wszystkich komputerach. Test przeprowadzany jest zarówno podczas rozgrywki, jak i w menu. W trakcie trwania testu tester wielokrotnie przełącza się między oknami otwartych programów. Oczekiwane zachowanie: brak crashu.

2.1.3.2.4 Testy dokumentacyjne

Testy dokumentacyjne oparto na głównej specyfikacji projektu [1] oraz wymaganiach zawartych w [2].

Przypadek testowy 1.

Tester sprawdza zgodność okna powitalnego z dokumentacją: ilość przycisków(4), napisy na nich się znajdujące.

Przypadek testowy 2.

Tester sprawdza zgodność tabeli highscore z dokumentacją: ilość przycisków(2), napisy na nich się znajdujące, zawartość tabeli, ilość wpisów w tabeli.

Przypadek testowy 3.

Tester sprawdza zgodność okna ustawień z dokumentacją: ilość przycisków(3), napisy na nich się znajdujące.

Przypadek testowy 4.

Tester sprawdza zgodność okna ustawień dźwięku z dokumentacją: ilość przycisków(3), napisy, suwaki(2), skalę suwaków.

EDIT: w ostatecznej wersji projektu suwaki po przyciskami Music i Sound nie mają skali

Przypadek testowy 5.

Tester sprawdza zgodność okna ustawień sterowania z dokumentacją: ilość przycisków(21), napisy na nich się znajdujące, napisy znajdujące się nad i obok napisów (tworzących pewnego rodzaju tabelę), kształt i rozmieszczenie przycisków służących do sterowania.

Przypadek testowy 6.

Tester sprawdza zgodność okna opcji rozgrywki (wybór mapy) z dokumentacją: ilość przycisków(4), napisy na nich się znajdujące, podgląd mapy, liczba map do wyboru.

Przypadek testowy 7.

Tester sprawdza zgodność okna opcji rozgrywki (ustawienia graczy) z dokumentacją: ilość przycisków(11), napisy na nich się znajdujące, pola z możliwością modyfikacji (4) pola, rozmieszczenie przycisków, pole na wiadomość o błędzie.

Przypadek testowy 8.

Tester sprawdza zgodność stałych czasowych dotyczących zmniejszania się mapy (120 + 5 sekund, 30 + 5 sekund na każde kolejne) (wymaganie 5).

EDIT: w ostatecznej wersji projektu czas ten wynosi 45 + 5 sekund między każdym zmniejszeniem.

Przypadek testowy 9.

Tester sprawdza zgodność stałych dotyczących prawdopodobieństwa pojawienia się power-up na planszy (10%).

EDIT: w ostatecznej wersji projektu prawdopodobieństwo wynosi 50%

Przypadek testowy 10.

Tester sprawdza zgodność stałych czasowych z związanymi z efektami cząsteczkowymi (0.3 sekund).

Przypadek testowy 10.

Tester sprawdza zgodność stałych czasowych z związanymi z efektami świetlnymi (0.3 sekund).

2.1.3.3 Wyniki testów

Podczas przeprowadzania testów funkcjonalnych odnotowano 4 incydenty, niektóre powodujące nieprawidłowe przerwanie działania programu.


Podczas przeprowadzania testów dokumentacyjnych odnotowano 3 niezgodności. Zostały one rozwiązane przez kierownika projektu, po dyskusji z programistą.

Testy wydajnościowe przeprowadzone na Laptop1, Laptop2 i PC1 wykazały, że projekt spełnia wymaganie 9. Gra utrzymuje stałe 60 FPS zarówno w menu gry, jak i podczas rozgrywki. Odnotowano krótkie spadki (<0.5 sekundy) do 58/59 FPS w przypadku wybuchu kilku bomb naraz. Odnotowano "klatkowanie", do 30 FPS włącznie, w przypadku ograniczenia zasobów. Nie odnotowano różnicy w wydajności gry w zależności od systemu operacyjnego. Podczas testów przeciążeniowych nie odnotowano crashy. Nie odnotowano różnicy w częstotliwości klatek w zależności od mapy na jakiej prowadzona jest rozgrywka, liczby botów, liczby graczy lub wybranych kolorów i nicków.

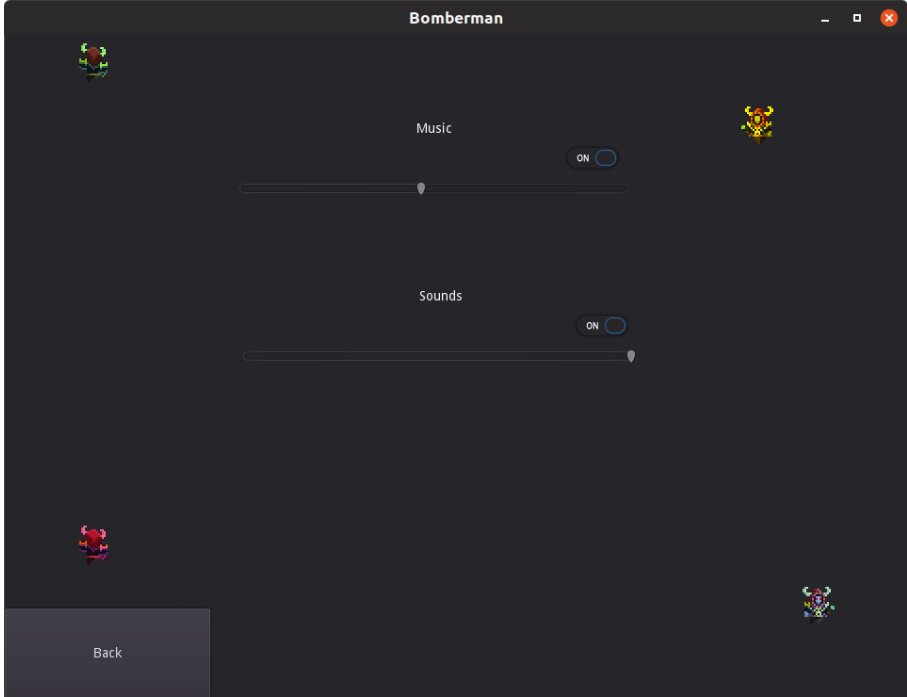
Po ostatecznym powtórzeniu testów funkcjonalnych, wydajnościowych, dokumentacyjnych nie odnotowano incydentów (oprócz awarii opisanej w raporcie nr 5.)

2.1.3.4 Raporty incydentów testowych

Nr. Zgłoszenia	2
Podsumowanie	Defekt: Brak powrotu do głównego menu po skończonej grze
Odkryty przez	Dominik Wołek
Priorytet	Wysoki
Status	FIXED
Kategoria	Funkcjonalne
Zgłoszony do	Filip Sowa
Opis zgłoszenia	Po śmierci wszystkich graczy i botów gra wyświetla komunikat o

	<p>zwycięstwie gracza z najwyższą ilością punktów. Nie następuje powrót do głównego menu. Co 45 + 5 sekund zmniejsza się mapa, co doprowadza ostatecznie do wypełnienia całej mapy blokami niezniszczalnymi.</p> <p>Komputer: Laptop1</p> <p>Nicki graczy i botów: abc123 / abe123 / abd123 / bot3</p> <p>Mapa: Wood</p> 
--	--

Nr. Zgłoszenia	3
Podsumowanie	Wyjątek: wczytywanie mapy
Odkryty przez	Dominik Wołek
Priorytet	Wysoki
Status	FIXED
Kategoria	Funkcjonalne
Zgłoszony do	Filip Sowa
Opis zgłoszenia	<p>Po wybraniu dowolnej mapy (testowano wszystkie), ustawieniu 3 botów i jednego gracza wciśnięty zostaje przycisk NEXT. Powoduje to zakończenie działania programu i uniemożliwienie dalszego działania.</p> <p>Nicki graczy i botów: abc123 / abe123 / abd123 / bot3</p> <p>Komputer: PC1</p>

Nr. Zgłoszenia	4
Podsumowanie	Defekt: wczytywania mapy Wood
Odkryty przez	Dominik Wołek
Priorytet	Średni
Status	FIXED
Kategoria	Funkcjonalne
Zgłoszony do	Filip Sowa
Opis zgłoszenia	<p>Po wybraniu mapy Wood (drewniane pudła), ustawieniu 2 graczy zostaje wciśnięty przycisk NEXT. Tło mapy nie zostaje wczytane. Nie pojawiają się żadne bloki.</p> <p>Nicki graczy i botów: player1 / player2 / player3 / player4</p> <p>Komputer: Laptop1</p> 

Nr. Zgłoszenia	5
Podsumowanie	Awaria: przypisywanie klawisza prawy alt
Odkryty przez	Dominik Wołek
Priorytet	Średni
Status	NOT FIXED

Kategoria	Funkcjonalne
Zgłoszony do	Rafał Brożek
Opis zgłoszenia	Po wejściu w menu OPTIONS -> CONTROLS na dowolnym z przycisków następuje próba przypisania klawisza prawy alt jako klawisza sterowania. Klawisz ten nie jest przypisywany, do czynności nadal jest przypisany poprzedni klawisz. Komputer: Laptop1, PC1

EDIT: Awaria nierozwiązywalny. Godot nie pozwala na rozpoznawanie naciśnięcia klawisza prawy alt, nie jest zatem błędem programisty

Nr. Zgłoszenia	6
Podsumowanie	Niezgodność: Zły czas pomiędzy zmniejszeniami się mapy
Odkryty przez	Dominik Wołek
Priorytet	Niski
Status	RESOLVED
Kategoria	Dokumentacyjne
Zgłoszony do	Kacper Schnetzer
Opis zgłoszenia	Odstęp czasowy pomiędzy kolejnymi zmniejszeniami się mapy wynosi 45 sekund + 5 sekund migania krawędzi. Dokumentacja zakłada 120 sekund + 5 sekund migania na pierwsze zmniejszenie, 30 sekund + 5 sekund na każde kolejne. Wymaganie 5, Scenariusz 5.1

Nr. Zgłoszenia	7
Podsumowanie	Niezgodność: menu sound
Odkryty przez	Dominik Wołek
Priorytet	Średni
Status	RESOLVED
Kategoria	Dokumentacyjne
Zgłoszony do	Kacper Schnetzer
Opis zgłoszenia	Po przejściu z menu głównego do OPTIONS -> SOUND program wyświetla menu dźwięku. Stwierdzone rozbieżności z dokumentacją: Skala nie ma naniesionych wartości granicznych 0 / 100

	(niezgodność). Po naciśnięciu przycisku ON (zmiana na OFF), a następnie naciśnięciu na skalę następuje przełączenie przycisku na ON i włączenie dźwięku (niezgodność). Wymaganie 7, Scenariusz 7.2
--	---

Nr. Zgłoszenia	8
Podsumowanie	Niezgodność: zły współczynnik power-up / zniszczony blok
Odkryty przez	Dominik Wołek
Priorytet	Niski
Status	RESOLVED
Kategoria	Dokumentacyjne
Zgłoszony do	Kacper Schnetzer
Opis zgłoszenia	Po zniszczeniu obiektów zaobserwowano, że w około 40% - 50% przypadków pojawia się power-up. Dokumentacja zakłada 10% szans. Wymaganie 12

3. Podsumowanie

3.1 Komentarz autorski

Testy zostały przygotowane i przeprowadzone z należytą dokładnością. Małą ilość incydentów odnotowanych na poziomie testowania jednostkowego można tłumaczyć wysoką jakością kodu pojedynczych metod. Wszystkie odnotowane incydenty były natychmiastowo zgłaszane do programisty lub kierownika projektu (testy dokumentacyjne). Projekt spełnia wymagania stawiane mu poprzez dokumentację i jest z nią zgodny.

4.1 Słowniczek

Godot - silnik gry, na którego bazie został utworzony projekt

GUT - Godot Unik Testing

GDScript - wysokopoziomowy język skryptowy używany w Godocie

Nickname - pseudonim dobierany przez gracza (graczy)

Poprawny nickname - nickname spełniający wyrażenie regularne `[a-zA-Z0-9]{4,20}`

FPS - frames per second - klatki na sekundę

PC1 - komputer do testów wydajnościowych

Laptop1 - komputer do testów wydajnościowych

Laptop2 - komputer do testów wydajnościowych

Scena - jeden z podstawowych typów obiektów w Godot. Grupa wierzchołków