DOKUMENTACJA TESTÓW

1. Cel dokumentu

Celem niniejszego dokumentu jest szczegółowe opisanie procesu testowania gry. Dokument ten ma na celu przedstawienie planu testów, przypadków testowych, środowiska testowego, wyników testów oraz wniosków końcowych. Testy zostały przeprowadzone w celu zapewnienia, że oprogramowanie działa zgodnie z wymaganiami i specyfikacjami.

Testowanie obejmowało następujące rodzaje testów:

1.1. Testy jednostkowe

Testy jednostkowe zostały przeprowadzone w celu sprawdzenia poprawności działania poszczególnych funkcji i metod oprogramowania. Testy jednostkowe zostały napisane przy użyciu zewnętrznej biblioteki CUnit.

1.2. Testy funkcjonalne

Testy funkcjonalne zostały przeprowadzone ręcznie i miały na celu sprawdzenie, czy oprogramowanie spełnia swoje wymagani. Testy te będą bardziej szczegółowo opisane w dalszych częściach dokumentu.

1.3. Testy sprzętowe i środowiskowe

Testy sprzętowe miały na celu sprawdzenie, czy oprogramowanie działa poprawnie na różnych konfiguracjach sprzętowych. Testy te obejmowały testowanie na wirtualnych maszynach z różną ilością pamięci RAM, a także na różnych systemach operacyjnych. Celem było upewnienie się, że oprogramowanie działa poprawnie w różnych

środowiskach sprzętowych. Testy wyszły prawidłowe, gra ma niskie wymagania sprzętowe.

1.4. Testy wydajnościowe

Testy wydajnościowe zostały przeprowadzone aby ocenić wydajność gry przy najwyższych ustawieniach graficznych na wirtualnych maszynach z mniejszą ilością pamięci RAM. Testy wyszły prawidłowe gra jest wysoce wydajna nawet przy małej ilości pamięci RAM.

2. Testy jednostkowe

2.1 Opis

Opis testów jednostkowych

Testy jednostkowe mają na celu weryfikację poprawności działania poszczególnych funkcji w kodzie. Zostały przeprowadzone w celu upewnienia się, że każda jednostka działa zgodnie z oczekiwaniami i spełnia założenia funkcjonalne. W ramach tych testów skoncentrowano się na następujących aspektach:

1. Poprawność działania funkcji CreateNewSave:
    - Sprawdzenie, czy funkcja poprawnie tworzy nowe zapisy gry przy różnych kombinacjach nazw zapisu i nazw postaci.
    - Weryfikacja, czy funkcja obsługuje przypadki brzegowe, takie jak puste ciągi, znaki specjalne, bardzo długie nazwy oraz znaki spoza ASCII.
    - Testowanie poprawności walidacji nazw zapisu i postaci, aby zapewnić, że przyjmowane są tylko prawidłowe znaki (litery, cyfry, spacje).

2. Testowanie funkcji związanych z elementami gry (GUI):
    - UpdateAchievementElem: Sprawdzenie, czy funkcja poprawnie aktualizuje elementy związane z osiągnięciami, w tym liczbę wymagań, nazwy, opisy i status.
    - CalculateButtonPosition: Weryfikacja, czy pozycje przycisków są poprawnie obliczane dla różnych konfiguracji (środkowy, lewy górny, prawy górny, lewy dolny, prawy dolny).
    - InitializeChooseAction, AttackPlayer, UpdateChooseAction: Testowanie poprawności inicjalizacji, atakowania i aktualizowania elementów wyboru akcji w grze.

- InitializeEquipementBox, UpdateEquipementBox: Sprawdzenie, czy skrzynka z ekwipunkiem jest poprawnie inicjalizowana i aktualizowana.

3. Funkcje związane z postaciami i przedmiotami:
   - InitializeFighterLabel, FreeFighterLabel, DrawFighterLabel: Testowanie inicjalizacji, zwalniania zasobów i rysowania etykiet postaci.
   - countDurability: Sprawdzenie poprawności obliczania trwałości elementów zbroi postaci.

4. Funkcje związane z interfejsem użytkownika:
   - CalculateInputBoxPosition, DrawInputBox, InternalUpdateInputBox: Weryfikacja poprawności obliczania pozycji, rysowania i aktualizowania elementów wejściowych (input box).
   - CalculateItemBoxPosition, DrawItemBox: Sprawdzenie, czy pozycje i rysowanie skrzynek na przedmioty są poprawnie obsługiwane.
   - nCalculateSlideBoxPosition, nInternalUpdateSlideBox: Testowanie poprawności obliczania pozycji i aktualizowania slajdów z liczbami.
   - CalculateSlideBoxPosition, InternalUpdateSlideBox: Weryfikacja poprawności obsługi slajdów z tekstem.

5. Funkcje związane z danymi zapisu:
   - initializeSaveData, cmpSaveData: Testowanie inicjalizacji danych zapisu oraz porównywania tych danych.

Wszystkie testy jednostkowe zostały napisane zgodnie ze standardami IEEE 610 i mają na celu zapewnienie wysokiej jakości kodu oraz jego niezawodności. Testy zostały przeprowadzone przy użyciu biblioteki CUnit, co umożliwiło szczegółową weryfikację poszczególnych funkcji i ich poprawnego działania.

2.2 Przypadki testowe

Nazwa przypadku testowego: test_UpdateAchievementElem

- **Cel testu:** Sprawdzenie, czy funkcja UpdateAchievementElem prawidłowo aktualizuje element osiągnięcia.
- **Warunki początkowe:** Element osiągnięcia z ustawionymi podstawowymi parametrami.
- **Kroki testowe:**
    1. Utwórz strukturę achievementElem i Achievement z wymaganymi wartościami.
    2. Wywołaj funkcję UpdateAchievementElem.
    3. Sprawdź wartości poszczególnych pól struktury achievementElem.
- **Oczekiwane wyniki:** Liczba wymagań, nazwa, opis oraz status są prawidłowo zaktualizowane.

Nazwa przypadku testowego: test_UpdateAchievementElem_Status0

- **Cel testu:** Sprawdzenie działania funkcji UpdateAchievementElem dla osiągnięcia z statusem 0.
- **Warunki początkowe:** Element osiągnięcia z ustawionym statusem 0.
- **Kroki testowe:**
    1. Utwórz strukturę achievementElem i Achievement z ustawionym statusem 0.
    2. Wywołaj funkcję UpdateAchievementElem.
    3. Sprawdź wartości poszczególnych pól struktury achievementElem.
- **Oczekiwane wyniki:** Liczba wymagań, nazwa, opis oraz status są prawidłowo zaktualizowane.

Nazwa przypadku testowego: test_UpdateAchievementElem_MultipleRequirements

- **Cel testu:** Sprawdzenie funkcji UpdateAchievementElem dla osiągnięcia z wieloma wymaganiami.
- **Warunki początkowe:** Element osiągnięcia z trzema wymaganiami.
- **Kroki testowe:**
    1. Utwórz strukturę achievementElem i Achievement z trzema wymaganiami.
    2. Wywołaj funkcję UpdateAchievementElem.
    3. Sprawdź wartości poszczególnych pól struktury achievementElem.
- **Oczekiwane wyniki:** Wszystkie wymagania są prawidłowo zaktualizowane.

Nazwa przypadku testowego: test_UpdateAchievementElem_NoRequirements

- **Cel testu:** Sprawdzenie funkcji UpdateAchievementElem dla osiągnięcia bez wymagań.
- **Warunki początkowe:** Element osiągnięcia bez wymagań.
- **Kroki testowe:**
    1. Utwórz strukturę achievementElem i Achievement bez wymagań.
    2. Wywołaj funkcję UpdateAchievementElem.
    3. Sprawdź wartości poszczególnych pól struktury achievementElem.
- **Oczekiwane wyniki:** Brak wymagań jest prawidłowo zaktualizowany.

Nazwa przypadku testowego: test_CalculateButtonPosition_Center

- **Cel testu:** Sprawdzenie funkcji CalculateButtonPosition dla przycisku wycentrowanego.
- **Warunki początkowe:** Przycisk z ustawionymi współrzędnymi x i y oraz wyśrodkowaniem.
- **Kroki testowe:**
    1. Utwórz strukturę button z parametrami wyśrodkowania.
    2. Wywołaj funkcję CalculateButtonPosition.
    3. Sprawdź współrzędne przycisku.
- Oczekiwane wyniki: Współrzędne przycisku są prawidłowo obliczone dla wyśrodkowania.

Nazwa przypadku testowego: test_CalculateButtonPosition_TopLeft

- **Cel testu:** Sprawdzenie funkcji CalculateButtonPosition dla przycisku w lewym górnym rogu.
- **Warunki początkowe:** Przycisk z ustawionymi współrzędnymi x i y oraz pozycją w lewym górnym rogu.
- **Kroki testowe:**
    1. Utwórz strukturę button z parametrami lewego górnego rogu.
    2. Wywołaj funkcję CalculateButtonPosition.
    3. Sprawdź współrzędne przycisku.
- **Oczekiwane wyniki:** Współrzędne przycisku są prawidłowo obliczone dla lewego górnego rogu.

Nazwa przypadku testowego: test_CalculateButtonPosition_TopRight

- **Cel testu:** Sprawdzenie funkcji CalculateButtonPosition dla przycisku w prawym górnym rogu.
- **Warunki początkowe:** Przycisk z ustawionymi współrzędnymi x i y oraz pozycją w prawym górnym rogu.
- **Kroki testowe:**
    1. Utwórz strukturę button z parametrami prawego górnego rogu.
    2. Wywołaj funkcję CalculateButtonPosition.
    3. Sprawdź współrzędne przycisku.
- **Oczekiwane wyniki:** Współrzędne przycisku są prawidłowo obliczone dla prawego górnego rogu.

Nazwa przypadku testowego: test_CalculateButtonPosition_BottomLeft

- **Cel testu:** Sprawdzenie funkcji CalculateButtonPosition dla przycisku w lewym dolnym rogu.
- **Warunki początkowe:** Przycisk z ustawionymi współrzędnymi x i y oraz pozycją w lewym dolnym rogu.
- **Kroki testowe:**
    1. Utwórz strukturę button z parametrami lewego dolnego rogu.
    2. Wywołaj funkcję CalculateButtonPosition.
    3. Sprawdź współrzędne przycisku.
- **Oczekiwane wyniki:** Współrzędne przycisku są prawidłowo obliczone dla lewego dolnego rogu.

Nazwa przypadku testowego: test_CalculateButtonPosition_BottomRight

- **Cel testu:** Sprawdzenie funkcji CalculateButtonPosition dla przycisku w prawym dolnym rogu.
- **Warunki początkowe:** Przycisk z ustawionymi współrzędnymi x i y oraz pozycją w prawym dolnym rogu.
- **Kroki testowe:**
    1. Utwórz strukturę button z parametrami prawego dolnego rogu.
    2. Wywołaj funkcję CalculateButtonPosition.
    3. Sprawdź współrzędne przycisku.
- **Oczekiwane wyniki:** Współrzędne przycisku są prawidłowo obliczone dla prawego dolnego rogu.

Nazwa przypadku testowego: test_initializeChooseAction

- **Cel testu**: Sprawdzenie inicjalizacji struktury chooseAction.
- **Warunki początkowe:** Pusty obiekt chooseAction.

- **Kroki testowe:**
    1. Utwórz strukturę chooseAction.
    2. Wywołaj funkcję initializeChooseAction.
    3. Sprawdź, czy pola struktury są prawidłowo zainicjalizowane.
- **Oczekiwane wyniki:** Struktura chooseAction jest poprawnie zainicjalizowana.

Nazwa przypadku testowego: test_AttackPlayer

- **Cel testu:** Sprawdzenie funkcji AttackPlayer.
- **Warunki początkowe:** Struktura chooseAction z dwoma fighterLabel.
- **Kroki testowe:**
    1. Utwórz struktury chooseAction, animateAttack i fighterLabel.
    2. Wywołaj funkcję AttackPlayer.
    3. Sprawdź, czy pola struktury animateAttack są poprawnie zaktualizowane.
- **Oczekiwane wyniki:** Struktura animateAttack jest prawidłowo zaktualizowana.

Nazwa przypadku testowego: test_UpdateChooseAction

- **Cel testu:** Sprawdzenie funkcji UpdateChooseAction.
- **Warunki początkowe:** Struktura chooseAction z wierszami celle.
- **Kroki testowe:**
    1. Utwórz strukturę chooseAction z wierszami celle.
    2. Wywołaj funkcję UpdateChooseAction.
    3. Sprawdź wynik funkcji.
- **Oczekiwane wyniki:** Funkcja UpdateChooseAction zwraca poprawny wynik.

Nazwa przypadku testowego: test_InitializeEquipementBox

- **Cel testu:** Sprawdzenie, czy funkcja InitializeEquipementBox prawidłowo inicjalizuje strukturę equipementBox.
- **Warunki początkowe:** Struktura equipementBox z nieprawidłowymi wartościami początkowymi.
- **Kroki testowe:**
    1. Utwórz strukturę equipementBox.
    2. Wywołaj funkcję InitializeEquipementBox.
    3. Sprawdź wartości poszczególnych pól struktury equipementBox.
- **Oczekiwane wyniki:** Wszystkie pola są prawidłowo zainicjalizowane, a pamięć jest poprawnie zaalokowana.

Nazwa przypadku testowego: test_UpdateEquipementBox

- **Cel testu:** Sprawdzenie, czy funkcja UpdateEquipementBox prawidłowo aktualizuje stan ekwipunku.
- **Warunki początkowe:** Struktura equipementBox z zainicjalizowanymi wartościami.
- **Kroki testowe:**
    1. Utwórz strukturę equipementBox i zainicjalizuj ją za pomocą InitializeEquipementBox.
    2. Utwórz strukturę playInfo.
    3. Wywołaj funkcję UpdateEquipementBox.
    4. Sprawdź wartości zwrócone przez funkcję.
- **Oczekiwane wyniki:** Funkcja zwraca wartość false, a pola struktury equipementBox są prawidłowo zaktualizowane.

Nazwa przypadku testowego: test_CountDurability

- **Cel testu:** Sprawdzenie, czy funkcja countDurability prawidłowo oblicza trwałość.
- **Warunki początkowe:** Tablica durability i armorPart z odpowiednimi wartościami.
- **Kroki testowe:**
    1. Utwórz i zainicjalizuj tablice durability i armorPart.
    2. Wywołaj funkcję countDurability.
    3. Sprawdź wynik zwrócony przez funkcję.
- **Oczekiwane wyniki:** Funkcja zwraca sumę wartości trwałości, co powinno wynosić 45.

Nazwa przypadku testowego: test_InitializeFighterLabel

- **Cel testu:** Sprawdzenie, czy funkcja InitializeFighterLabel prawidłowo inicjalizuje strukturę fighterLabel.
- **Warunki początkowe:** Struktura fighterLabel z nieprawidłowymi wartościami początkowymi.
- **Kroki testowe:**
    1. Utwórz strukturę fighterLabel.
    2. Wywołaj funkcję MockInitializeFighterLabel.
    3. Sprawdź wartości poszczególnych pól struktury fighterLabel.
- **Oczekiwane wyniki:** Wszystkie pola są prawidłowo zainicjalizowane.

Nazwa przypadku testowego: test_FreeFighterLabel

- **Cel testu:** Sprawdzenie, czy funkcja FreeFighterLabel prawidłowo zwalnia pamięć.
- **Warunki początkowe:** Struktura fighterLabel z zaalokowaną pamięcią.
- **Kroki testowe:**
    1. Utwórz strukturę fighterLabel i zaalokuj pamięć dla render.
    2. Wywołaj funkcję FreeFighterLabel.
    3. Sprawdź, czy pamięć została prawidłowo zwolniona.
- **Oczekiwane wyniki:** Funkcja prawidłowo zwalnia pamięć, a program nie zgłasza żadnych błędów.

Nazwa przypadku testowego: test_DrawFighterLabel

- **Cel testu:** Sprawdzenie, czy funkcja DrawFighterLabel prawidłowo rysuje etykietę wojownika.
- **Warunki początkowe:** Struktura fighterLabel z odpowiednimi wartościami.
- **Kroki testowe:**
    1. Utwórz strukturę fighterLabel i zainicjalizuj wartości leftCorner i render.
    2. Wywołaj funkcję DrawFighterLabel.
    3. Sprawdź, czy funkcja rysuje etykietę bez błędów.
- **Oczekiwane wyniki:** Funkcja prawidłowo rysuje etykietę wojownika.

Nazwa przypadku testowego: test_CalculateInputBoxPosition

- **Cel testu:** Sprawdzenie, czy funkcja CalculateInputBoxPosition prawidłowo oblicza pozycję elementu inputBox.
- **Warunki początkowe:** Struktura inputBox z zainicjalizowanymi wartościami początkowymi oraz przykładowym tekstem.
- **Kroki testowe:**
    1. Utwórz strukturę inputBox i zainicjalizuj jej wartości.
    2. Wywołaj funkcję CalculateInputBoxPosition.

3. Sprawdź wartości pól currentLength i lengthArrayLength.
- **Oczekiwane wyniki:** Wartości pól currentLength i lengthArrayLength są równe 0.

Nazwa przypadku testowego: test_DrawInputBox

- **Cel testu:** Sprawdzenie, czy funkcja DrawInputBox prawidłowo rysuje element inputBox.
- **Warunki początkowe:** Struktura inputBox z zainicjalizowanymi wartościami początkowymi oraz przykładowym tekstem.
- **Kroki testowe:**
    1. Utwórz strukturę inputBox i zainicjalizuj jej wartości.
    2. Wywołaj funkcję DrawInputBox.
    3. Sprawdź, czy funkcja rysuje element bez błędów.
- **Oczekiwane wyniki:** Funkcja rysuje element inputBox bez błędów.

Nazwa przypadku testowego: test_InternalUpdateInputBox

- **Cel testu:** Sprawdzenie, czy funkcja InternalUpdateInputBox prawidłowo aktualizuje stan elementu inputBox.
- **Warunki początkowe:** Struktura inputBox z zainicjalizowanymi wartościami początkowymi.
- **Kroki testowe:**
    1. Utwórz strukturę inputBox i zainicjalizuj jej wartości.
    2. Wywołaj funkcję InternalUpdateInputBox.
    3. Sprawdź, czy funkcja aktualizuje stan elementu inputBox bez błędów.
- **Oczekiwane wyniki:** Funkcja aktualizuje stan elementu inputBox bez błędów.

Nazwa przypadku testowego: test_CalculateItemBoxPosition

- **Cel testu:** Sprawdzenie, czy funkcja CalculateItemBoxPosition prawidłowo oblicza pozycję elementu itemBox.
- **Warunki początkowe:** Struktura itemBox z zainicjalizowanymi wartościami początkowymi.
- **Kroki testowe:**
    1. Utwórz strukturę itemBox i zainicjalizuj jej wartości.
    2. Wywołaj funkcję CalculateItemBoxPosition.
    3. Sprawdź wartości pól isActive, boxRectangle.x, boxRectangle.y, boxRectangle.width, boxRectangle.height.
- **Oczekiwane wyniki:** Wartości pól isActive, boxRectangle.x, boxRectangle.y, boxRectangle.width i boxRectangle.height są prawidłowo obliczone.

Nazwa przypadku testowego: test_DrawItemBox

- **Cel testu:** Sprawdzenie, czy funkcja DrawItemBox prawidłowo rysuje element itemBox.
- **Warunki początkowe:** Struktura itemBox z zainicjalizowanymi wartościami początkowymi.
- **Kroki testowe:**
    1. Utwórz strukturę itemBox i zainicjalizuj jej wartości.
    2. Wywołaj funkcję DrawItemBox.
    3. Sprawdź, czy funkcja rysuje element bez błędów.
- **Oczekiwane wyniki:** Funkcja rysuje element itemBox bez błędów.

Nazwa przypadku testowego: test_nCalculateSlideBoxPosition

- **Cel testu:** Sprawdzenie, czy funkcja nCalculateSlideBoxPosition prawidłowo oblicza pozycję elementu nSlideBox.
- **Warunki początkowe:** Struktura nSlideBox z zainicjalizowanymi wartościami początkowymi.
- **Kroki testowe:**
    1. Utwórz strukturę nSlideBox i zainicjalizuj jej wartości.
    2. Wywołaj funkcję nCalculateSlideBoxPosition.
    3. Sprawdź wartości pól rect[0].width, rect[0].height, rect[1].width, rect[2].width.
- **Oczekiwane wyniki:** Wartości pól są prawidłowo obliczone: rect[0].width = 220, rect[0].height = 40, rect[1].width = 40, rect[2].width = 40.

Nazwa przypadku testowego: test_nInternalUpdateSlideBox

- **Cel testu:** Sprawdzenie, czy funkcja nInternalUpdateSlideBox prawidłowo aktualizuje stan elementu nSlideBox.
- **Warunki początkowe:** Struktura nSlideBox z zainicjalizowanymi wartościami początkowymi.
- **Kroki testowe:**
    1. Utwórz strukturę nSlideBox i zainicjalizuj jej wartości.
    2. Wywołaj funkcję nCalculateSlideBoxPosition.
    3. Zasymuluj kliknięcie lewego przycisku myszy.
    4. Sprawdź wartość pola currentOption.
    5. Zasymuluj kliknięcie prawego przycisku myszy.
    6. Sprawdź wartość pola currentOption.
- **Oczekiwane wyniki:** Po kliknięciu lewego przycisku myszy currentOption = 3, po kliknięciu prawego przycisku myszy currentOption = 2.

Nazwa przypadku testowego: test_initializeSaveData

- **Cel testu:** Sprawdzenie, czy funkcja initializeSaveData prawidłowo inicjalizuje strukturę saveData.
- **Warunki początkowe:** Brak, funkcja inicjalizuje strukturę saveData.
- **Kroki testowe:**
    1. Utwórz strukturę saveData.
    2. Wywołaj funkcję initializeSaveDataForTest z odpowiednią ścieżką do zapisu.
    3. Sprawdź wartości pól struktury saveData.
- **Oczekiwane wyniki:** Wartości pól są prawidłowo zainicjalizowane: text = "save\game1", year = 2024, month = 6, day = 8, hour = 10, minute = 30, second = 45.

Nazwa przypadku testowego: test_cmpSaveData

- **Cel testu:** Sprawdzenie, czy funkcja cmpSaveData prawidłowo porównuje dwie struktury saveData.
- **Warunki początkowe:** Dwie struktury saveData do porównania.
- **Kroki testowe:**
    1. Utwórz dwie struktury saveData z różnymi wartościami pól.
    2. Wywołaj funkcję cmpSaveData dla różnych par struktur.
    3. Sprawdź wyniki porównania.
- **Oczekiwane wyniki:** Wyniki porównania są prawidłowe: odpowiednie wartości zwracane przez funkcję cmpSaveData.

Nazwa przypadku testowego: test_CalculateSlideBoxPosition

- **Cel testu:** Sprawdzenie, czy funkcja CalculateSlideBoxPosition prawidłowo oblicza pozycję elementu slideBox.
- **Warunki początkowe:** Struktura slideBox z zainicjalizowanymi wartościami początkowymi.
- **Kroki testowe:**
    1. Utwórz strukturę slideBox i zainicjalizuj jej wartości.
    2. Wywołaj funkcję CalculateSlideBoxPosition.
    3. Sprawdź wartości pól rect[0].width, rect[0].height, rect[1].width, rect[2].width.
- **Oczekiwane wyniki:** Wartości pól są prawidłowo obliczone: rect[0].width = 220, rect[0].height = 40, rect[1].width = 40, rect[2].width = 40.

Nazwa przypadku testowego: test_InternalUpdateSlideBox

- **Cel testu:** Sprawdzenie, czy funkcja InternalUpdateSlideBox prawidłowo aktualizuje stan elementu slideBox.
- **Warunki początkowe:** Struktura slideBox z zainicjalizowanymi wartościami początkowymi.
- **Kroki testowe:**
    1. Utwórz strukturę slideBox i zainicjalizuj jej wartości.
    2. Wywołaj funkcję CalculateSlideBoxPosition.
    3. Zasymuluj kliknięcie lewego klawisza klawiatury.
    4. Sprawdź wartość pola currentOption.
    5. Zasymuluj kliknięcie prawego klawisza klawiatury.
    6. Sprawdź wartość pola currentOption.
    7. Zasymuluj kliknięcie lewego przycisku myszy.
    8. Sprawdź wartość pola currentOption.
    9. Zasymuluj kliknięcie prawego przycisku myszy.
    10. Sprawdź wartość pola currentOption.
- **Oczekiwane wyniki:** Po kliknięciu lewego klawisza currentOption = 1, po kliknięciu prawego klawisza currentOption = 2, po kliknięciu lewego przycisku myszy currentOption = 1, po kliknięciu prawego przycisku myszy currentOption = 2.

Testy jednostkowe CreateNewSave

Kroki testowe (ogólne dla wszystkich testów CreateNewSave)

1. Zainicjalizowanie tablicy bodyParts wartościami domyślnymi (wszystkie elementy ustawione na 0).
2. Wywołanie funkcji CreateNewSave z różnymi kombinacjami nazw zapisu (saveName) i nazw postaci (characterName).
3. Sprawdzenie wyniku funkcji za pomocą CU_ASSERT.

Zmieniane parametry testowe

- **Nazwa zapisu (saveName):** Może zawierać różne znaki, długości i formaty.
- **Nazwa postaci (characterName):** Może zawierać różne znaki, długości i formaty.

Szczegółowe opisy grup przypadków testowych

1. **Grupa przypadków testowych:** Podstawowe wejścia
    - **Cel testów:** Sprawdzenie podstawowego działania funkcji.
    - **Warunki początkowe:** Brak specjalnych warunków.
    - **Przypadki testowe:**
        - saveName = "testSave", characterName = "testCharacter".
        - Oczekiwane wyniki: Prawidłowe dane.

2. **Grupa przypadków testowych:** Istniejąca nazwa zapisu
    - **Cel testów:** Sprawdzenie działania funkcji dla już istniejącego zapisu.
    - **Warunki początkowe:** Katalog saves\\existingSave istnieje.
    - **Przypadki testowe:**
        - saveName = "existingSave", characterName = "testCharacter".
        - Oczekiwane wyniki: Nieprawidłowe Dane

3. **Grupa przypadków testowych:** Puste ciągi znaków
    - **Cel testów:** Sprawdzenie działania funkcji dla pustych nazw zapisu i postaci.
    - **Warunki początkowe:** Brak.
    - **Przypadki testowe:**
        - saveName = "", characterName = "testCharacter".
        - saveName = "testSave1", characterName = "".
        - Oczekiwane wyniki: Nieprawidłowe Dane

4. **Grupa przypadków testowych:** Nazwy zawierające spacje
    - **Cel testów:** Sprawdzenie działania funkcji dla nazw zawierających spacje.
    - **Warunki początkowe:** Brak.
    - **Przypadki testowe:**
        - saveName = "test Save", characterName = "testCharacter".
        - saveName = "testSave2", characterName = "test Character".
        - Oczekiwane wyniki: Prawidłowe dane.

5. **Grupa przypadków testowych:** Nazwy zawierające nieprawidłowe znaki
    - **Cel testów:** Sprawdzenie działania funkcji dla nazw zawierających nieprawidłowe znaki.
    - **Warunki początkowe:** Brak.
    - **Przypadki testowe:**
        - saveName = "test*Save?", characterName = "testCharacter".
        - saveName = "testSave3", characterName = "test*Character?".
        - Oczekiwane wyniki: Nieprawidłowe Dane

6. **Grupa przypadków testowych:** Nazwy zawierające znaki spoza ASCII
    - **Cel testów:** Sprawdzenie działania funkcji dla nazw zawierających znaki spoza ASCII.
    - **Warunki początkowe:** Brak.

- **Przypadki testowe:**
  - saveName = "testŚave", characterName = "testCharacter".
  - saveName = "testSave4", characterName = "testĆharacter".
  - Oczekiwane wyniki: Nieprawidłowe Dane

7. **Grupa przypadków testowych:** Bardzo długie nazwy
   - **Cel testów:** Sprawdzenie działania funkcji dla bardzo długich nazw zapisu i postaci.
   - **Warunki początkowe:** Brak.
   - **Przypadki testowe:**
     - saveName = "thisIsAVeryLongSaveNameThatExceedsNormalLimits", characterName = "testCharacter".
     - saveName = "testSave5", characterName = "thisIsAVeryLongCharacterNameThatExceedsNormalLimits".
     - Oczekiwane wyniki: Prawidłowe dane.

8. **Grupa przypadków testowych:** Nazwy zawierające wyłącznie cyfry
   - **Cel testów:** Sprawdzenie działania funkcji dla nazw zawierających wyłącznie cyfry.
   - **Warunki początkowe:** Brak.
   - **Przypadki testowe:**
     - saveName = "123456", characterName = "testCharacter".
     - saveName = "testSave6", characterName = "123456".
     - Oczekiwane wyniki: Prawidłowe dane.

9. **Grupa przypadków testowych:** Puste nazwy zapisu i postaci
   - **Cel testów:** Sprawdzenie działania funkcji dla pustych nazw zapisu i postaci.
   - **Warunki początkowe:** Brak.
   - **Przypadki testowe:**
     - saveName = "", characterName = "".
     - Oczekiwane wyniki: Nieprawidłowe Dane

10. **Grupa przypadków testowych:** Nazwy zawierające znaki specjalne
    - **Cel testów:** Sprawdzenie działania funkcji dla nazw zawierających znaki specjalne.
    - **Warunki początkowe:** Brak.
    - **Przypadki testowe:**
      - saveName = "test@Save#", characterName = "testCharacter".
      - saveName = "testSave7", characterName = "test@Character#".
      - Oczekiwane wyniki: Nieprawidłowe Dane

11. **Grupa przypadków testowych:** Nazwy zawierające kombinację liter, cyfr i znaków specjalnych
    - **Cel testów:** Sprawdzenie działania funkcji dla nazw zawierających kombinację liter, cyfr i znaków specjalnych.
    - **Warunki początkowe:** Brak.

- **Przypadki testowe:**
  - saveName = "test123@Save#", characterName = "testCharacter".
  - saveName = "testSave8", characterName = "test123@Character#".
  - Oczekiwane wyniki: Nieprawidłowe Dane

12. **Grupa przypadków testowych:** Losowe znaki w nazwie zapisu i postaci
    - **Cel testów:** Sprawdzenie działania funkcji dla nazw zawierających losowe znaki.
    - **Warunki początkowe:** Brak.
    - **Przypadki testowe:**
      - saveName = "][/-/&*E@", characterName = "testCharacter".
      - saveName = "testSave9", characterName = ";][/ ; = -/ &*E@".
      - Oczekiwane wyniki: Nieprawidłowe Dane

13. **Grupa przypadków testowych:** Jednoznakowe nazwy
    - **Cel testów:** Sprawdzenie działania funkcji dla jednoznakowych nazw zapisu i postaci.
    - **Warunki początkowe:** Brak.
    - **Przypadki testowe:**
      - saveName = "a", characterName = "testCharacter".
      - saveName = "testSave10", characterName = "b".
      - Oczekiwane wyniki: Prawidłowe dane.

14. **Grupa przypadków testowych:** Nazwy zawierające znak nowej linii
    - **Cel testów:** Sprawdzenie działania funkcji dla nazw zawierających znak nowej linii.
    - **Warunki początkowe:** Brak.
    - **Przypadki testowe:**
      - saveName = "test\nSave", characterName = "testCharacter".
      - saveName = "testSave11", characterName = "test\nCharacter".
      - Oczekiwane wyniki: Nieprawidłowe Dane

15. **Grupa przypadków testowych:** Nazwy zawierające znak TAB
    - **Cel testów:** Sprawdzenie działania funkcji dla nazw zawierających znak TAB.
    - **Warunki początkowe:** Brak.
    - **Przypadki testowe:**
      - saveName = "test\tSave", characterName = "testCharacter".
      - saveName = "testSave12", characterName = "test\tCharacter".
      - Oczekiwane wyniki: Nieprawidłowe Dane

16. **Grupa przypadków testowych:** Nazwy zawierające spacje na początku
    - **Cel testów:** Sprawdzenie działania funkcji dla nazw zawierających spacje na początku.
    - **Warunki początkowe:** Brak.
    - **Przypadki testowe:**
      - saveName = " testSave", characterName = "testCharacter".

- saveName = "testSave13", characterName = " testCharacter".
- Oczekiwane wyniki: Prawidłowe dane.

17. **Grupa przypadków testowych:** Nazwy zawierające wyłącznie spacje
    - **Cel testów:** Sprawdzenie działania funkcji dla nazw zawierających wyłącznie spacje.
    - **Warunki początkowe:** Brak.
    - **Przypadki testowe:**
        - saveName = " ", characterName = "testCharacter".
        - saveName = "testSave15", characterName = " ".
        - Oczekiwane wyniki: Nieprawidłowe Dane

2.3 Wnioski

Przeprowadzone testy jednostkowe miały kluczowe znaczenie w procesie weryfikacji poprawności kodu oraz w identyfikacji i naprawie błędów. Dzięki szczegółowej analizie i testowaniu poszczególnych funkcji udało się zapewnić, że każda jednostka kodu działa zgodnie z oczekiwaniami i spełnia założenia funkcjonalne.

Podczas testowania funkcji CreateNewSave, która odpowiada za tworzenie nowych zapisów gry, wykryto istotny błąd związany z obsługą niepoprawnych znaków w nazwie zapisu. Ten błąd został zidentyfikowany dzięki kilku przypadkom testowym:

Błąd w tworzeniu nowego zapisu z nieprawidłowymi znakami w nazwie zapisu:

- Test: test_CreateNewSave_invalid_characters_in_save_name
- Test: test_CreateNewSave_gibberish_save_name
- Test: test_CreateNewSave_newline_in_save_name
- Test: test_CreateNewSave_tab_in_save_name

Wszystkie te testy wykazały, że funkcja CreateNewSave niepoprawnie obsługuje nazwy zapisów zawierające nieprawidłowe znaki, takie jak znaki specjalne (*, ?, @), znaki nowej linii (\n) oraz tabulację (\t).

Dzięki testom jednostkowym udało się zlokalizować problem i wprowadzić odpowiednie poprawki. Dodano logikę sprawdzającą poprawność nazwy zapisu, która teraz weryfikuje, czy nazwa zapisu zawiera tylko dozwolone znaki (litery, cyfry i spacje). W przypadku wykrycia nieprawidłowego znaku, użytkownik otrzymuje odpowiedni komunikat ostrzegający o niepoprawnym wejściu.

3. Testy funkcjonalne

Testy funkcjonalne zostały przeprowadzone manualnie, aby zweryfikować poprawność działania kluczowych funkcji aplikacji. Poniżej przedstawiono zakres testów funkcjonalnych, które zostały przeprowadzone:

3.1 Menu startowe:

Tworzenie nowego zapisu:

- Przetestowano tworzenie nowej gry na różnych poziomach trudności (łatwy, średni, trudny).
- Sprawdzono działanie kreatora postaci, w tym wybór różnych kombinacji części ciała (głowa, tułów, ręce, nogi).
- Upewniono się, że każda kombinacja części ciała jest poprawnie zapisywana i wyświetlana w grze.

Wczytywanie gry:

Wczytywanie istniejących zapisów gry:

- Przetestowano wczytywanie zapisanych gier.
- Upewniono się, że zapisy są sortowane względem daty, co ułatwia odnalezienie najnowszych zapisów.
- Sprawdzono, czy wczytanie zapisów przywraca stan gry dokładnie sprzed momentu zapisu.

Ustawienia gry:

Wybór ustawień:

- Przetestowano wybór liczby klatek na sekundę (FPS), wymiarów okna, rozdzielczości tekstur i trybu pełnoekranowego.
- Zauważono, że zmiany ustawień są widoczne po restarcie gry.
- Zauważono błąd, że przycisk "zresetuj ustawienia" nie działa prawidłowo – nic nie zmienia.
- Upewniono się, że brak zaakceptowania zmian powoduje brak zapisu wprowadzonych ustawień, co jest poprawnym zachowaniem.
- Przetestowano różne kombinacje ustawień graficznych – wszystkie działają prawidłowo.

Samouczek:

Testowanie samouczka:

- Przetestowano działanie samouczka, który nie jest jeszcze ukończony, ale działa prawidłowo w dostępnych sekcjach.

Wyjście z gry:

Różne sposoby wychodzenia z gry:

- Przetestowano różne sposoby wychodzenia z gry do pulpitu, w tym korzystanie z opcji w menu oraz skrótów klawiaturowych.

3.2 Rozgrywka:

Menu pauzy:

Działanie menu pauzy w trakcie rozgrywki:

- Przetestowano działanie menu pauzy, w tym przeglądanie osiągnięć, wznawianie rozgrywki, wyjście do menu głównego i wyjście do pulpitu.
- Sprawdzono, że osiągnięcia działają prawidłowo, ale nie są jeszcze ukończone.

Poruszanie się postacią:

Sterowanie postacią:

- Przetestowano poruszanie się postacią za pomocą klawiszy WASD, myszki oraz spacji.
- Zidentyfikowano błąd związany z niechcianym obracaniem się postaci.

Ekwipunek:

Zarządzanie ekwipunkiem:

- Przetestowano wyposażanie przedmiotów, które działa prawidłowo.
- Przetestowano zdejmowanie i niszczenie przedmiotów. Zauważono, że po zniszczeniu wyposażonego przedmiotu, nie znika on z ciała postaci, dopóki nie zrestartuje się ekwipunku.
- Upewniono się, że przedmioty można wyposażyć tylko w odpowiednie części ciała, co jest poprawnym działaniem.

- Powracanie do rozgrywki po zarządzaniu ekwipunkiem działa prawidłowo.

Zapisywanie rozgrywki:

Zapisywanie i wczytywanie gry:

- Przetestowano zapisywanie rozgrywki, w tym nadpisywanie istniejących zapisów oraz tworzenie nowych.
- Upewniono się, że zapisy są poprawnie wczytywane i przywracają stan gry sprzed momentu zapisu.

Interakcje ze sprzedawcą:

Zakup i sprzedaż przedmiotów:

- Przetestowano interakcje ze sprzedawcą, w tym zakup i sprzedaż przedmiotów.
- Sprawdzono widok ekwipunku i obsługę go w trakcie interakcji ze sprzedawcą, co działa prawidłowo.

Przeciwnicy:

Zachowanie przeciwników:

- Przeciwnicy zmierzają w kierunku gracza z zamiarem wejścia w tryb walki, co jest prawidłowe.

Walka:

System walki:

- Przetestowano możliwość wykonania akcji, zmiany przedmiotu, ucieczki lub poddania się.
- Ucieczka i poddanie się nie są jeszcze zaimplementowane – tymczasowo są równoznaczne z porażką.
- Tymczasowo nie jest możliwe wybranie przedmiotów.
- Możliwe jest wykonanie akcji uderzenia, która działa prawidłowo.
- Po wykonaniu ruchu przez gracza, ruch wykonuje przeciwnik.
- Pokonywanie lub bycie pokonanym działa prawidłowo.
- Po wygranej przeciwnik znika, a gracz kontynuuje eksplorację świata. W przypadku porażki, postać gracza umiera, a gracz może spróbować ponownie rozegrać potyczkę, wczytać ostatni zapis, wrócić do menu lub wyjść do pulpitu.

Interakcje z otoczeniem i przeciwnikami:

Interakcje w grze:

- Przetestowano poruszanie się w otoczeniu przeciwników, w tym skakanie i próby wejścia w interakcję z kilkoma przeciwnikami naraz. Interakcja jest możliwa tylko z jednym przeciwnikiem naraz, co jest poprawnym działaniem.

3.3 Wnioski:

Przeprowadzone testy funkcjonalne pozwoliły na wykrycie kilku małych błędów, które mogłyby wpływać na jakość i stabilność gry. Podczas testów interakcji z otoczeniem stwierdzono, że postać prawidłowo wchodzi w interakcje z podłogą i ścianami, co uniemożliwia jej przechodzenie przez te elementy. To potwierdziło poprawność działania fizyki gry w tym zakresie.

Testy kreatora postaci i ekwipunku wykazały drobny problem związany z niszczeniem wyekwipowanego przedmiotu. Po jego zniszczeniu przedmiot nie znikał z ciała postaci, dopóki ekwipunek nie został zrestartowany. Także zauważono nieprawidłowe działanie przycisku *esc* po naciśnięciu którego gra zostaję wyłączona, co może negatywnie wpłynąć na doznania z rozgrywki.

Wnioski z testów funkcjonalnych wskazują na konieczność dalszego doskonalenia niektórych aspektów gry oraz potwierdzają stabilność wielu kluczowych funkcji. Testy te były kluczowe dla zidentyfikowania i zrozumienia potencjalnych problemów, które mogą wpływać na doświadczenia graczy.