

Maciej Bogusławski, Hubert Kaczyński

PROI 24L

05-06.2024

Dokumentacja projektowa

Spis treści

Sprawozdanie z prac	3
Wstęp	4
Informacje techniczne	4
Instalacja i uruchomienie	4
Opis środowiska	5
Źródła	5
Wycieki pamięci	6
Opis plików	7
Podział zadań	11
Możliwości rozwoju projektu	11
Poradnik	12
Wstęp	13
Spis zadań	13
Początek	14
Wieś niesie	14
Robin Smród	16
Dyweryfikacja portfolio inwestycyjnego	20
Redystrybucja mienia	22
Księżniczka i Żaba	23
Śliska sprawa	25
Koniec	25

Część 1.

Sprawozdanie z prac

Wstęp

Projekt stanowi implementację prostej gry RPG, w której gracz wciela się w rolę postaci mającej za zadanie uratować królestwo Dobrogrodu. W tym celu bohater musi zgładzić bandę rzezimieszków nurtujących miasto i odnaleźć księżniczkę.

W ramach projektu zaimplementowaliśmy szereg funkcjonalności znanych z gier RPG i umożliwiających sprawdzenie się w zakresie programowania obiektowego. Za główny cel postawiliśmy sobie zapewnienie uniwersalności kodu poprzez rozbudowaną hierarchię klas, co znaczco ułatwia ewentualne dodawanie nowych misji i funkcjonalności w przyszłości. Projekt oparty jest o bibliotekę multimedialną Simple and Fast Multimedia Library (SFML).

Informacje techniczne

Wymagane miejsce na dysku: 171 MiB

Wymagana minimalna rozdzielcość ekranu: 1920x1080px

Rekomendowana minimalna częstotliwość odświeżania monitora: 60Hz

Instalacja i uruchomienie

W celu poprawnego uruchomienia gry w środowisku WSL należy zainstalować bibliotekę multimedialną Simple and Fast Multimedia Library (SFML). W tym celu rekomendujemy wykonanie następującej komendy w terminalu:

```
sudo apt-get install libsfml-dev
```

Po instalacji biblioteki należy skompilować program. Reguły kompilacji zawarte zostały w katalogu głównym w pliku makefile. Dzięki temu w celu kompilacji programu wystarczy jedynie wykonać następującą komendę w terminalu:

```
make
```

Z uwagi na dużą liczbę plików źródłowych kompilacja potrwać może kilkadziesiąt sekund. Po poprawnej komplilacji gra powinna być gotowa do uruchomienia z pomocą komendy:

```
./main
```

Opis środowiska

Implementacja programistyczna i testowanie projektu przeprowadzone zostały, wykorzystując język C++ 17 oraz bibliotekę Simple and Fast Multimedia Library (SFML) w wersji 2.5.1.

Źródła

W trakcie projektowania prac graficznych na rzecz gry korzystaliśmy z pomocy narzędzi oraz assetów udostępnianych na następujących stronach:

- <https://www.photopea.com/> - narzędzie graficzne
- <https://kenney.nl/> - pliki graficzne
- <https://craftpix.net/> - pliki graficzne
- <https://www.bing.com/images/create> - generowanie obrazów
- <https://openai.com/index/dall-e-3/> - generowanie obrazów
- <https://fonts.google.com/> - czcionki
- <https://befonts.com/> - czcionki
- <https://pixabay.com/> - efekty dźwiękowe i muzyka
- <https://www.looperman.com/> - efekty dźwiękowe i muzyka

Wycieki pamięci

W trakcie implementacji projektu zaobserwowałyśmy, iż popularne narzędzie Valgrind, służące do wykrywania wycieków pamięci, reportuje olbrzymią liczbę (sięgającą kilkuset tysięcy) problemów z wyciekami pamięci. Starając się rozwiązać ten problem, zauważylismy jednak, iż nawet najprostszy program korzystający z biblioteki SFML, jedynie otwierający i zamkujący okno gry, reportuje podobną (w większości przypadków nawet wyższą) liczbę błędów (osiągającą w granicy 332 800). Próbując zgłębić ten problem, natknęliśmy się na liczne informacje sugerujące, iż jest to powszechny problem i nie świadczy o niepoprawnej implementacji programistycznej.

```
==14202==  
==14202== HEAP SUMMARY:  
==14202==     in use at exit: 416,785 bytes in 2,892 blocks  
==14202==   total heap usage: 135,775 allocs, 132,883 frees, 670,246,031 bytes allocated  
==14202==  
==14202== LEAK SUMMARY:  
==14202==   definitely lost: 66,672 bytes in 13 blocks  
==14202==   indirectly lost: 352 bytes in 2 blocks  
==14202==   possibly lost: 0 bytes in 0 blocks  
==14202==   still reachable: 349,761 bytes in 2,877 blocks  
==14202==   suppressed: 0 bytes in 0 blocks  
==14202== Rerun with --leak-check=full to see details of leaked memory  
==14202==  
==14202== Use --track-origins=yes to see where uninitialised values come from  
==14202== For lists of detected and suppressed errors, rerun with: -s  
==14202== ERROR SUMMARY: 332732 errors from 1000 contexts (suppressed: 0 from 0)
```

Rys. 1. Wyniki analizy wycieków pamięci w finalnej wersji gry z pomocą programu Valgrind (332 732 błędy)

```
Setting vertical sync not supported  
==20912==  
==20912== HEAP SUMMARY:  
==20912==     in use at exit: 349,588 bytes in 2,683 blocks  
==20912==   total heap usage: 136,759 allocs, 134,076 frees, 23,931,323 bytes allocated  
==20912==  
==20912== LEAK SUMMARY:  
==20912==   definitely lost: 66,352 bytes in 5 blocks  
==20912==   indirectly lost: 352 bytes in 2 blocks  
==20912==   possibly lost: 0 bytes in 0 blocks  
==20912==   still reachable: 282,884 bytes in 2,676 blocks  
==20912==   suppressed: 0 bytes in 0 blocks  
==20912== Rerun with --leak-check=full to see details of leaked memory  
==20912==  
==20912== Use --track-origins=yes to see where uninitialised values come from  
==20912== For lists of detected and suppressed errors, rerun with: -s  
==20912== ERROR SUMMARY: 332911 errors from 1000 contexts (suppressed: 0 from 0)
```

Rys. 2. Wyniki analizy wycieków pamięci w prostym, jednoplikowym programie korzystającym z biblioteki SFML, jedynie uruchamiającym i zamkującym okno gry z pomocą programu Valgrind (332 911 błędów)

Pliki źródłowe programu z rys. 2 dla porównania umieściliśmy [tutaj](#).

Opis plików

W pliku **main.cpp** znajduje się główna pętla gry.

Plik **makefile** zawiera reguły programu make, umożliwiające poprawną komplikację programu.

W folderze **scenes** znajdują się szablony poszczególnych scen gry.

W pliku **scene.h** znajduje się abstrakcyjna klasa sceny, z której dziedziczą poszczególne szablony scen.

W plikach **cutscenescene.h** i **cutscenescene.cpp** znajduje się implementacja klasy CutsceneScene stanowiącej szablon wstawki filmowej.

W plikach **endmenu.h** i **endmenu.cpp** znajduje się implementacja klasy EndMenuScene stanowiącej menu końcowe gry.

W plikach **fightscene.h** i **fightscene.cpp** znajduje się implementacja klasy FightScene stanowiącej szablon sceny walki.

W plikach **inventoryscene.h** i **inventoryscene.cpp** znajduje się implementacja klasy InventoryScene stanowiącej scenę inwentarza gracza.

W plikach **mapexplore.h** i **mapexplore.cpp** znajduje się implementacja klasy MapExplore stanowiącej szablon głównej sceny gry, w której użytkownik porusza się po mapie, zbiera przedmioty, wchodzi w interakcje z elementami otoczenia itp.

W plikach **menuscene.h** i **menuscene.cpp** znajduje się implementacja klasy MenuScene stanowiącej główne menu gry.

W plikach **questscene.h** i **questscene.cpp** znajduje się implementacja klasy QuestScene stanowiącej scenę prezentującą zadania gracza.

W plikach **scenemanager.h** i **scenemanager.cpp** znajduje się implementacja klasy SceneManager, odpowiadającej za poprawne zmienianie scen i kontrolowanie sceny bieżącej.

Plik **scenemanager.h** zawiera również enum SceneID, stanowiący listę unikalnych identyfikatorów sceny.

W folderze **levels** znajdują się kolejne poziomy gry, dziedziczące z szablonów scen znajdujących się w folderze scenes.

W plikach **level1.h** i **level1.cpp** znajduje się implementacja klasy Level1, reprezentującej pierwszą mapę gry (w mieście Dobrogrod).

W plikach **level2.h** i **level2.cpp** znajduje się implementacja klasy Level2, reprezentującej drugą mapę gry (w lesie opanowanym przez rozbójników na wschód od Dobrogrodu).

W plikach **level3.h** i **level3.cpp** znajduje się implementacja klasy Level3, reprezentującej trzecią mapę gry (w zagajniku na północny wschód od Dobrogrodu).

W plikach **combat1.h** i **combat1.cpp** znajduje się implementacja klasy Combat1, reprezentującej pierwszą i jedyną bitwę w grze.

W folderze **cutscenes** znajdują się konkretne wstawki filmowe.

W pliku **cutsceneframe.h** znajduje się implementacja klasy CutsceneFrame, stanowiącej pojedyncze ujęcie we wstawce filmowej. Klasa ta składa się ze ścieżki do pliku stanowiącego obraz będący tłem ujęcia i długość ujęcia wyrażona w sekundach.

W plikach **cutscene1_1.h**, **cutscene1_1.cpp**, **cutscene1_2.h**, **cutscene1_2.cpp**, **cutscene2_death.h**, **cutscene2_death.cpp**, **cutscene3_1.h** i **cutscene3_1.cpp** znajduje się implementacja konkretnych wstawek filmowych, uruchamianych w zależności od sytuacji gracza w grze.

W folderze **mechanics** znajdują się implementacje kluczowych funkcjonalności gry, które są przechowywane przez cały czas jej trwania.

W plikach **camera.h** i **camera.cpp** znajduje się implementacja klasy Camera stanowiącej rozszerzenie istniejącego komponentu sf::View, umożliwiające lepsze reagowanie na zmiany położenia gracza - m.in. blokowanie wyjścia kamery poza ekran gry.

W plikach **mapexplore_ui.h** i **mapexplore_ui.cpp** znajduje się implementacja klasy MapExploreUI stanowiącej półprzezroczystą nakładkę na ekran gry ze statystykami gracza i skrótami klawiszowymi.

W plikach **playerstats.h** i **playerstats.cpp** znajduje się implementacja kluczowej klasy PlayersStats przechowującej statystyki gracza - jego zdrowie, siłę, elementy w inwentarzu, podniesione przedmioty, zadania, pokonanych wrogów i pozycje na poszczególnych poziomach.

W plikach **staticcamera.h** i **staticcamera.cpp** znajduje się implementacja klasy StaticCamera stanowiącej rozszerzenie istniejącego komponentu sf::View.

W folderze **sceneelems** znajdują się implementacje poszczególnych elementów gry, w zależności od ich funkcjonalności.

W plikach **blockade.h** i **blockade.cpp** znajduje się implementacja klasy Blockade reprezentującej przezroczysty element blokujący poruszanie się gracza (nakładany np. na budynki na mapie).

W plikach **combatnpc.h** i **combatnpc.cpp** znajduje się implementacja klasy CombatNPC reprezentującej wroga w scenach walki.

W plikach **interactiveelement_adversary.h** i **interactiveelement_adversary.cpp** znajduje się implementacja klasy AdversaryNPCElement reprezentującej wroga w scenach eksploracji mapy (tacy wrogowie nie atakują na mapie, lecz zbliżenie się do nich gracza przenosi użytkownika do sceny walki, gdzie za wroga odpowiada już obiekt CombatNPC).

W plikach **interactiveelement_pickup.h** i **interactiveelement_pickup.cpp** znajduje się implementacja klasy PickupElement reprezentującej przedmiot położony na mapie, który można podnieść.

W plikach **interactiveelement.h** i **interactiveelement.cpp** znajduje się implementacja klasy InteractiveElement reprezentującej element interaktywny na mapie, który wykrywa zbliżanie się gracza i interakcję z elementem poprzez kliknięcie przycisku E przez użytkownika.

W pliku **objectids.h** znajduje się enum GameObjectID zawierający unikalne identyfikatory przedmiotów, które mogą zostać zebrane lub wrogów, którzy mogą zostać pokonani.

Identyfikatory te po zakończonej interakcji przez gracza (np. zebraniu przedmiotu) zapisywane są w obiekcie klasy PlayerStats.

W plikach **player.h** i **player.cpp** znajduje się implementacja klasy Player stanowiącej wizualną reprezentację gracza na mapie i w walce.

W plikach **sceneryelement.h** i **sceneryelement.cpp** znajduje się implementacja klasy SceneryElement stanowiącej element sceny. Z klasy tej dziedziczą klasy InteractiveElement i StaticElement.

W plikach **staticelement.h** i **staticelement.cpp** znajduje się implementacja klasy StaticElement stanowiącej statyczny element sceny, który nie reaguje na zachowanie gracza - np. drzewo czy tło sceny.

W folderze **inventory** znajdują się implementacje klas pozwalających na wyświetlanie zebranych przedmiotów w inwentarzu gracza.

W plikach **inventoryitem.h** i **inventoryitem.cpp** znajduje się implementacja klasy InventoryItem, stanowiącej pole w inwentarzu (łączącej klasę danego przedmiotu i ilość zebranych przedmiotów tego typu).

W pliku **pickupobject.h** znajduje się implementacja klasy PickupObject stanowiącej szablon kategorii danego przedmiotu oraz enum PickupCategory stanowiący listę wszystkich kategorii obiektów, które mogą wyświetlić się w inwentarzu (np. sakw z monetami, kryształów itp.).

W plikach **pickupdescr.h**, i **fooddescr.h** znajdują się implementacje klas reprezentujących kategorie konkretnych przedmiotów dziedziczące od klasy PickupObject, które mogą pojawić się w inwentarzu gracza wraz z ich nazwami, opisami, identyfikatorami i ścieżkami do plików reprezentujących ich ikony w inwentarzu.

W folderze **quests** znajdują się implementacje konkretnych zadań gracza wraz z ich nazwami, identyfikatorami, opisami i automatycznym sprawdzaniem wymagań.

W plikach **quest.h** i **quest.cpp** znajduje się implementacja klasy Quest, stanowiącej zadanie, które ma wykonać gracz (m.in. z informacją o zakończeniu zadania, wyświetleniu zadania graczowi w formie powiadomienia itp.).

W plikach **crystalquest.h**, **crystalquest.cpp**, **givebackmoneyquest.h**, **givebackmoneyquest.cpp**, **kissquest.h**, **kissquest.cpp**, **moneybagsquest.h**, **moneybagsquest.cpp**, **princessquest.h**, **princessquest.cpp**, **robinstinkquest.h**, **robinstinkquest.cpp**, **towncrierquest.h** i **towncrierquest.cpp** znajdują się konkretne zadania, które gracz otrzymuje do wykonania na przestrzeni gry.

W folderze **notifications** znajdują się implementacje powiadomień, które gracz otrzymuje na przestrzeni gry.

W plikach **notification.h** i **notification.cpp** znajduje się implementacja klasy Notification, stanowiącej szablon powiadomienia wraz z implementacją animacji pojawiania się i znikania powiadomienia.

W pliku **hintnotification.h** znajduje się implementacja klasy HintNotification, stanowiącej powiadomienie udzielające graczowi wskazówki. Funkcjonalność ta nie została ostatecznie

zaimplementowana w grze, jednak stanowi dobrą podstawę do dodania takiej funkcjonalności przy dalszym rozwoju projektu.

W pliku **questnotification.h** znajduje się implementacja klasy QuestNotification, stanowiącej powiadomienie informujące gracza o otrzymaniu bądź ukończeniu zadania.

W folderze **assets** znajdują się pliki niebędące kodem źródłowym, ale kluczowe dla poprawnego działania gry.

W folderze **audio** znajdują się pliki dźwiękowe w formacie wav (m.in. muzyka, odgłos kroków, dźwięk ze wstawek filmowych itp.).

W folderze **fonts** znajdują się czcionki w formacie ttf wykorzystywane w grze.

W folderze **img** znajdują się pliki graficzne stanowiące tła do gry, tekstury postaci, ikony w inwentarzu, przedmioty do zebrania, elementy interfejsu oraz ikony gry.

Podział zadań

Na każdym etapie projektowania rozwiązania, implementacji programistycznej, testowania, a także planowania i tworzenia prac graficznych, montażowych oraz pisania kwestii dialogowych zadania wykonywaliśmy wspólnie (komunikując się zdalnie) lub dzieliliśmy się zadaniami i wykonywaliśmy je równolegle, później wspólnie składając nasze rozwiązania w jedną całość.

Możliwości rozwoju projektu

Projekt rozwijaliśmy z myślą o jego uniwersalności, mając na celu ułatwienie dodawania nowych mechanik, misji i map w przyszłości. Wśród pomysłów, które uważamy za ciekawe i możliwe do realizacji w ramach dalszego rozwoju projektu, uważamy:

1. Dodanie postaci NPC, chodzących po wyznaczonych trasach i wypowiadających krótkie kwestie mijając gracza.
2. Mechanika interakcji z przedmiotami w ekwipunku.
3. Możliwość zapisu stanu gry.
4. Rozwinięcie fabuły i dodanie większej ilości scen oraz zadań.

Część 2.

Poradnik

Wstęp

Gra rozgrywa się w miasteczku Dobrogród - sercu królestwa, niegdyś słynącego ze spokoju i bezpieczeństwa. Niestety, czasy świetności miasta dawno przeminęły, a na domiar złego Dobrogród napastować zaczęli ostatnio rozbójnicy brutalnej bandy Robina Smróda, a księżniczka Dobrawa zginęła. Gracz ma za zadanie rozwiązać problemy miasta i przywrócić świetność miasta.

Niniejszy poradnik ma za zadanie ułatwić przejście gry, tłumacząc poszczególne misje i ukazując sposób na ich ukończenie.

Spis zadań

Na przestrzeni gry gracz otrzymuje następujące zadania, konieczne do ukończenia gry:

- Wieś niesie - odnajdź wieszczę w mieście.
- Robin Smród - zgładź Robina Smróda i jego towarzyszy.
- Dywersyfikacja portfolio inwestycyjnego - zbierz 5 sakw z monetami.
- Redystrybucja mienia - zwróć skradzione pieniądze wieszczowi.
- Księżniczka i Żaba - znajdź księżniczkę.
- Śliska sprawa - pocałuj żabę.

Dodatkowo, gra oferuje również zadanie opcjonalne:

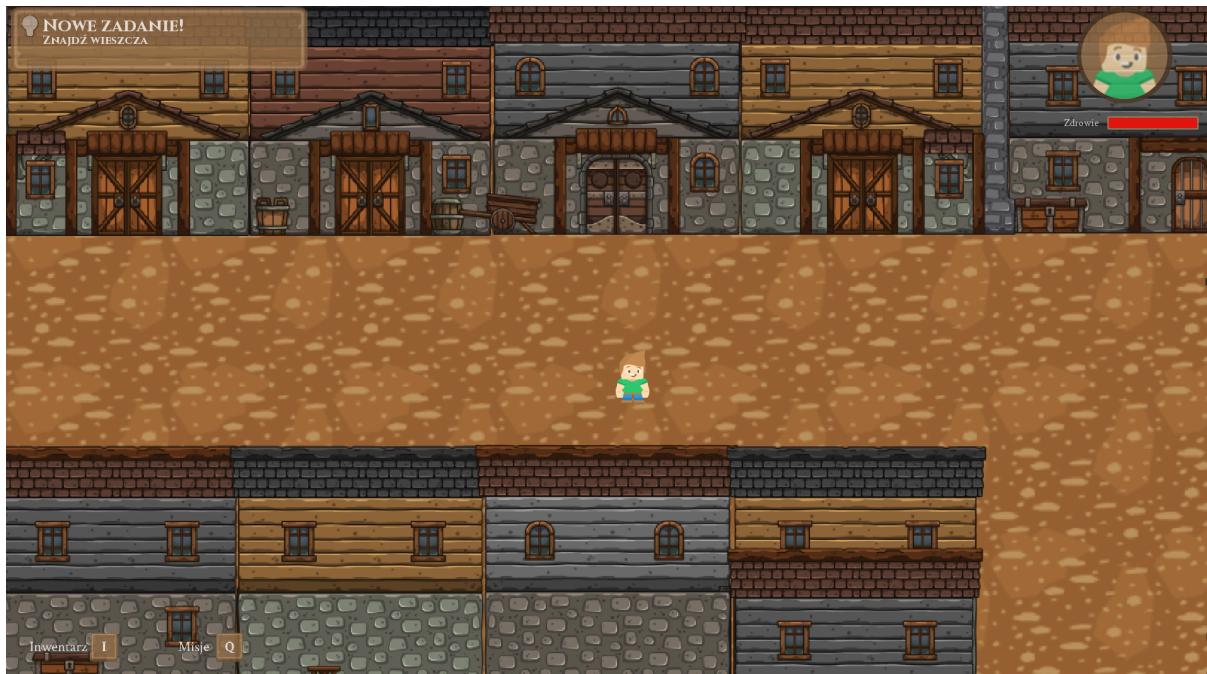
- Zlap je wszystkie - zbierz 10 kryształów.

Początek



Gra wita gracza ekranem tytułowym. Po kliknięciu przycisku “Nowa gra” użytkownik rozpoczyna nową grę.

Wieść niesie



Po wejściu do gry gracz otrzymuje powiadomienie o nowym zadaniu, polegającym na znalezieniu wieszczka. Gracz porusza się po mapie z pomocą klawiszy WASD. Sprawdzenie ekwipunku możliwe jest z pomocą klawisza I, a misji - z pomocą klawisza Q.



Po zbliżeniu się do wieszcza otrzymujemy informację o ukończeniu zadania. Kliknięcie E przenosi gracza do wstawki filmowej, w której dostaje nowe zadanie - ma on pokonać Robina Smróda i jego towarzyszy. Wstawkę filmową można pominąć klawiszem Spacji.



Robin Smród

Po otrzymaniu zadania od wieszcza gracz może udać się na południowy wschód mapy i przejść przez bramę (bez otrzymania zadania od wieszcza przejście przez bramę byłoby zablokowane).



Przejście przez bramę możliwe jest poprzez kliknięcie klawisza E.



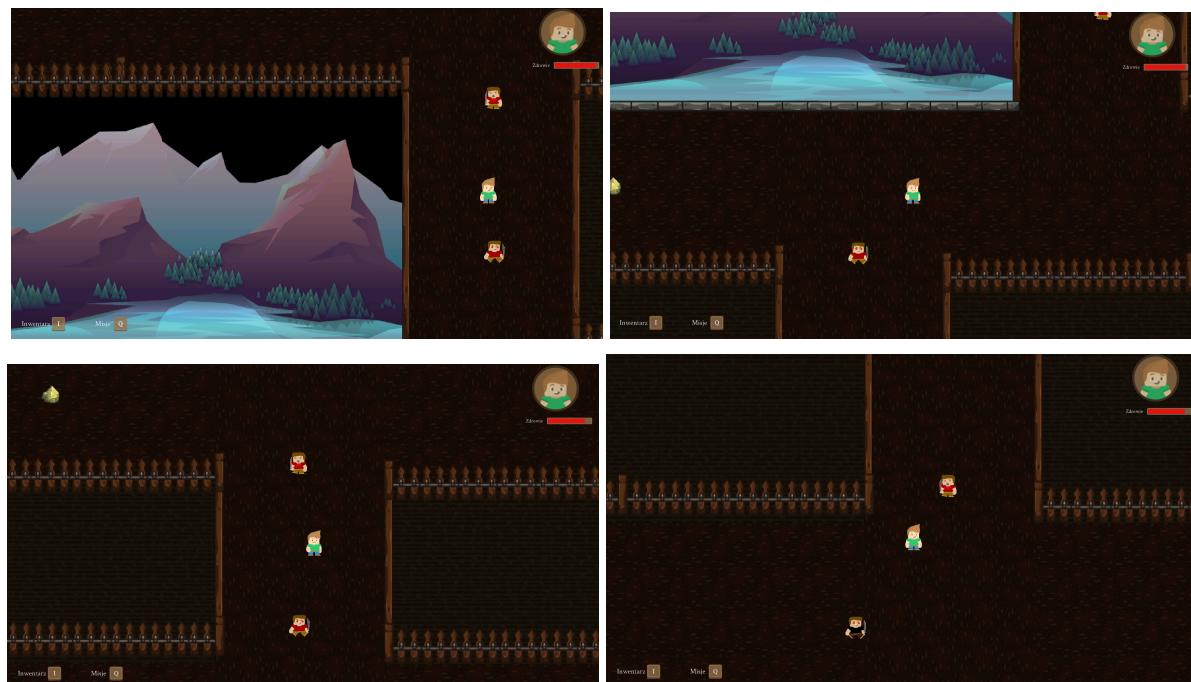
Gracz trafia do lasu na wschód od Dobrogrodu. Idąc w prawo, a następnie w dół, napotyka on kolejnych rozbójników bandy Robina Smróda.



Zbliżenie się do rozbójnika uruchamia walkę automatycznie.



W trakcie walki przeciwnika zaatakować można z pomocą klawisza C. Zaatakowana postać odsuwa się na pewien dystans przez impakt uderzenia. Podczas walki należy zwracać uwagę na swój pasek zdrowia w prawym górnym rogu. Śmierć gracza prowadzi do wstawki filmowej, lecz użytkownik nie musi rozpoczynać gry od początku - gra przywróci gracza do momentu, w którym wszedł on do lasu (zapisując przeciwników, których ten już pokonał), i przywróci graczu 100% zdrowia.



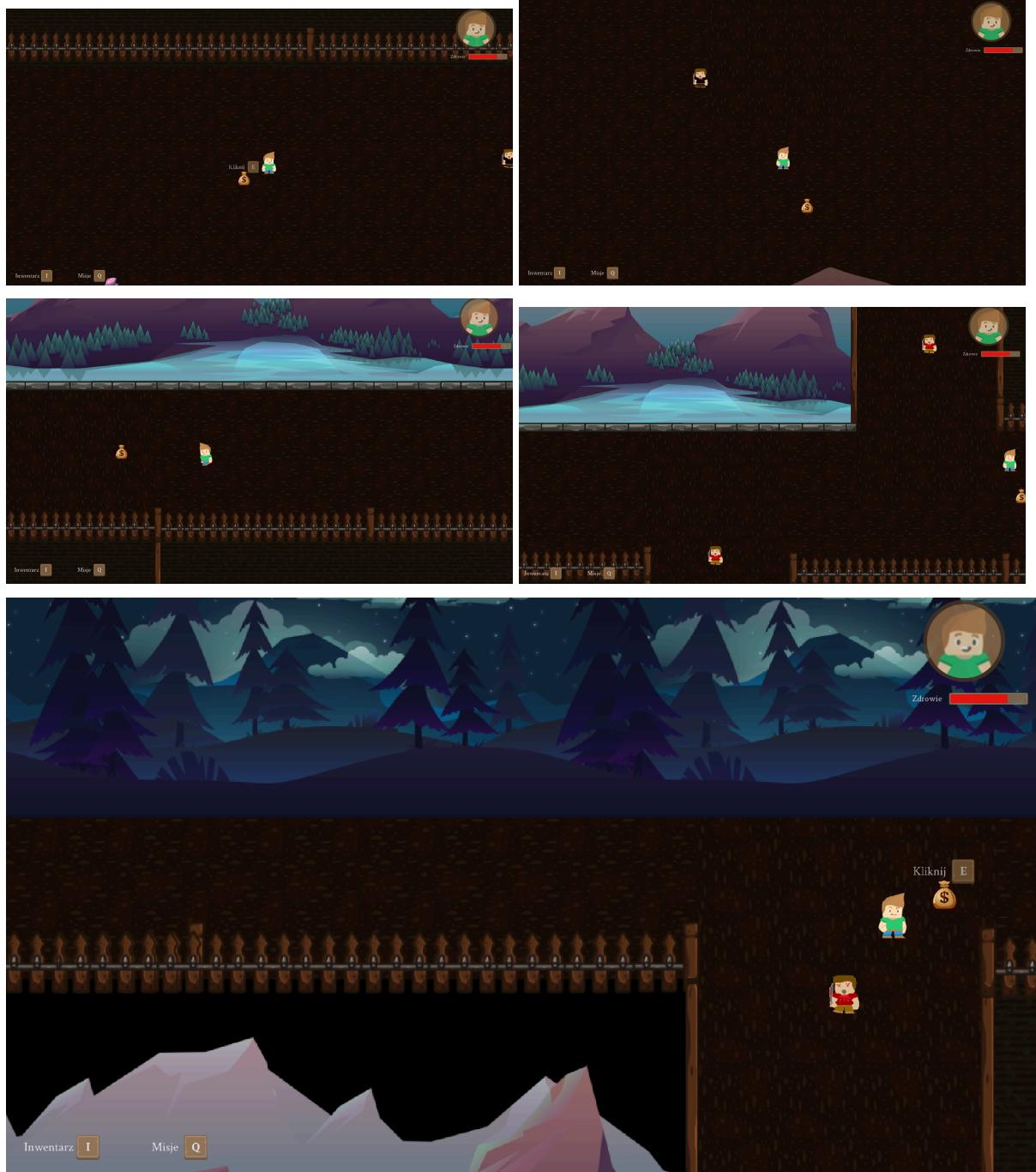
Gracz pokonać musi czterech rozbójników bandy Robin Smród, a następnie samego przywódcę bandy. Robin Smród odznacza się większym zdrowiem i większą siłą ataku, stanowiąc większe wyzwanie dla gracza.



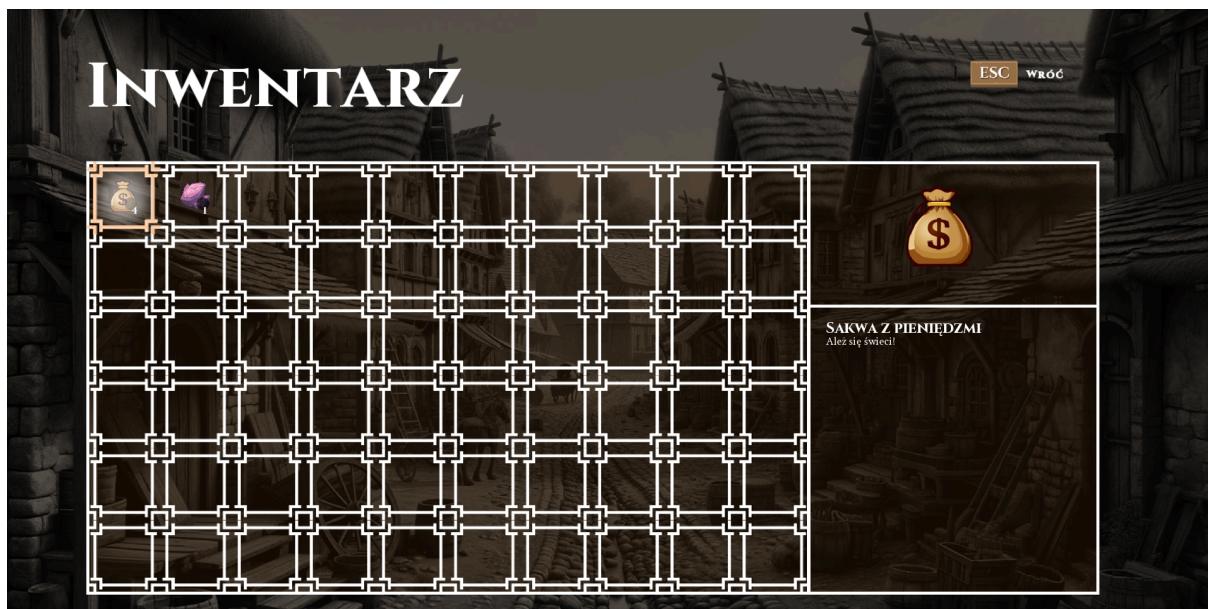
Po pokonaniu Robina zadanie zostaje ukończone, a gracz otrzymuje nowe zadanie - Dywersyfikacja portfolio inwesytycyjnego.

Dyweryfikacja portfolio inwestycyjnego

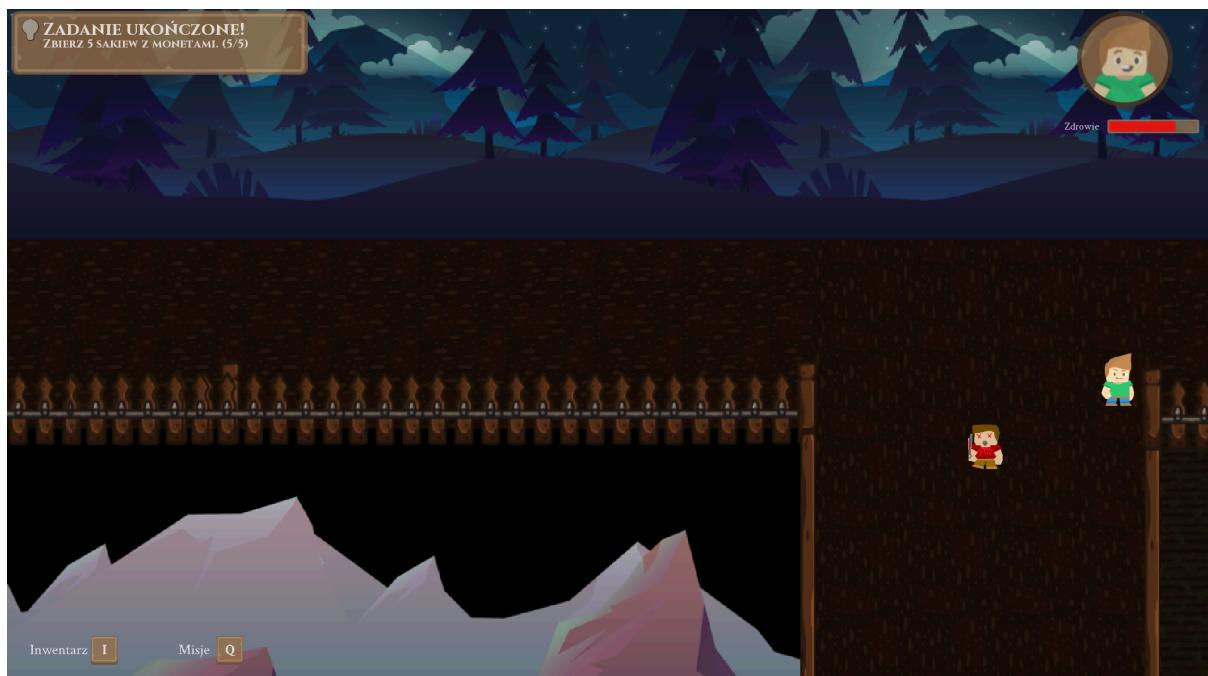
Po otrzymaniu zadania gracz znaleźć musi na mapie pięć sakw z monetami, które ukradła banda Robina Smróda. Wszystkie pięć sakw znajduje się w lesie, w którym gracz pokonał rozbójników. Sakwy zebrać można poprzez podejście do nich i kliknięcie przycisku E.



Liczbę zebranych dotychczas sakw można sprawdzić w każdym momencie, klikając klawisz I w celu otworzenia inwentarza.



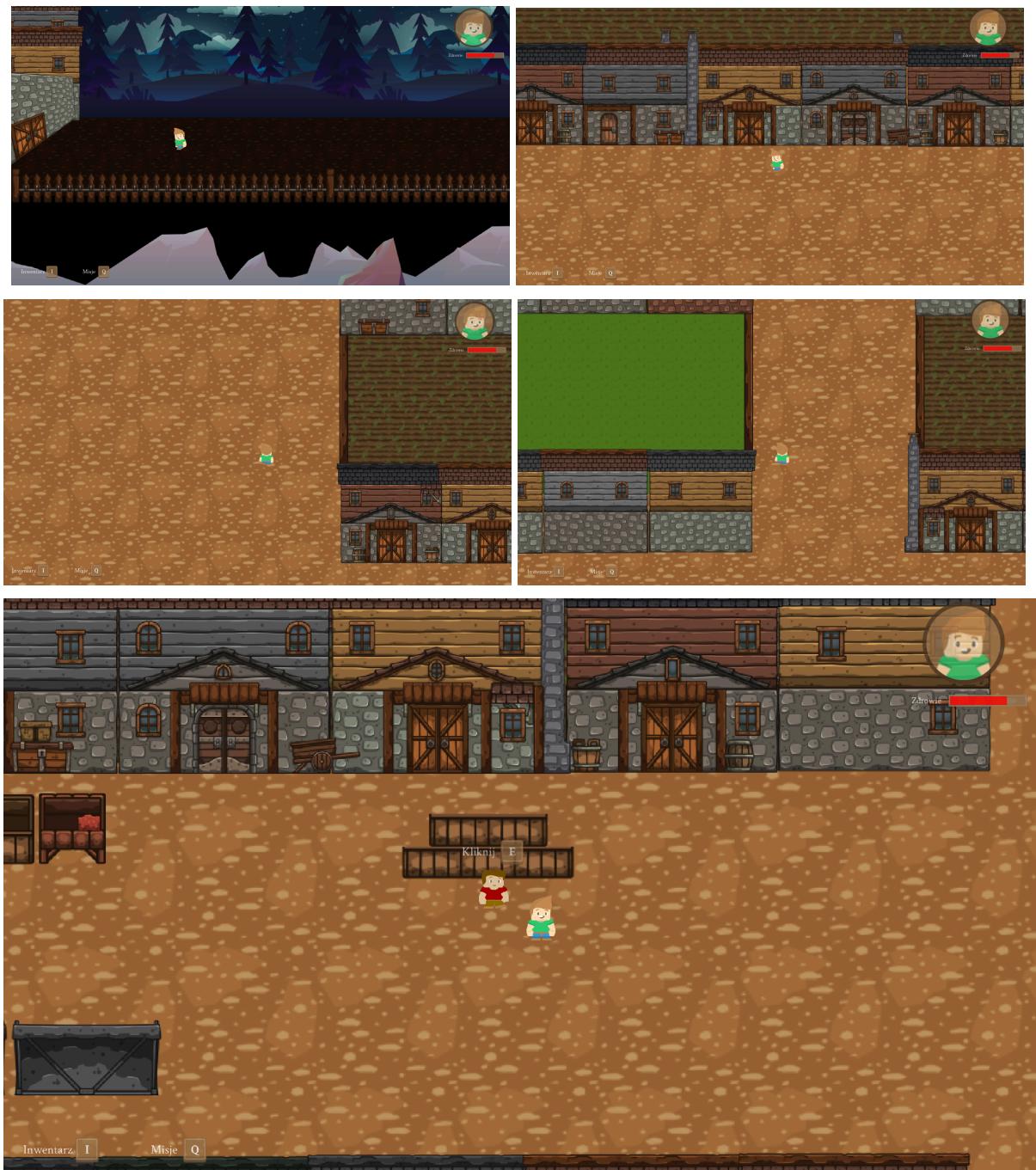
Inwentarz zamknąć można z pomocą klawisza ESC.



Po zebraniu wszystkich pięciu sakw zadanie zostaje ukończone i gracz otrzymuje kolejne zadanie.

Redystrybucja mienia

Po otrzymaniu zadania gracz musi wrócić do wieszcza z zadania “Wieś niesie”. Użytkownik powinien więc wrócić do Dobrogrodu i powrócić na plac główny.



Po zbliżeniu się do wieszcza i kliknięciu klawisza E rozpoczyna się nowa wstawa filmowa, w której wieszcz dziękuje graczowi za zebranie sakw i daje mu nowe zadanie.



Wstawkę filmową można pominąć klawiszem Spacji. Po wstawce filmowej zadanie zostaje ukończone, a gracz otrzymuje nowe zadanie - znalezienie księżniczki.

Księżniczka i Żaba



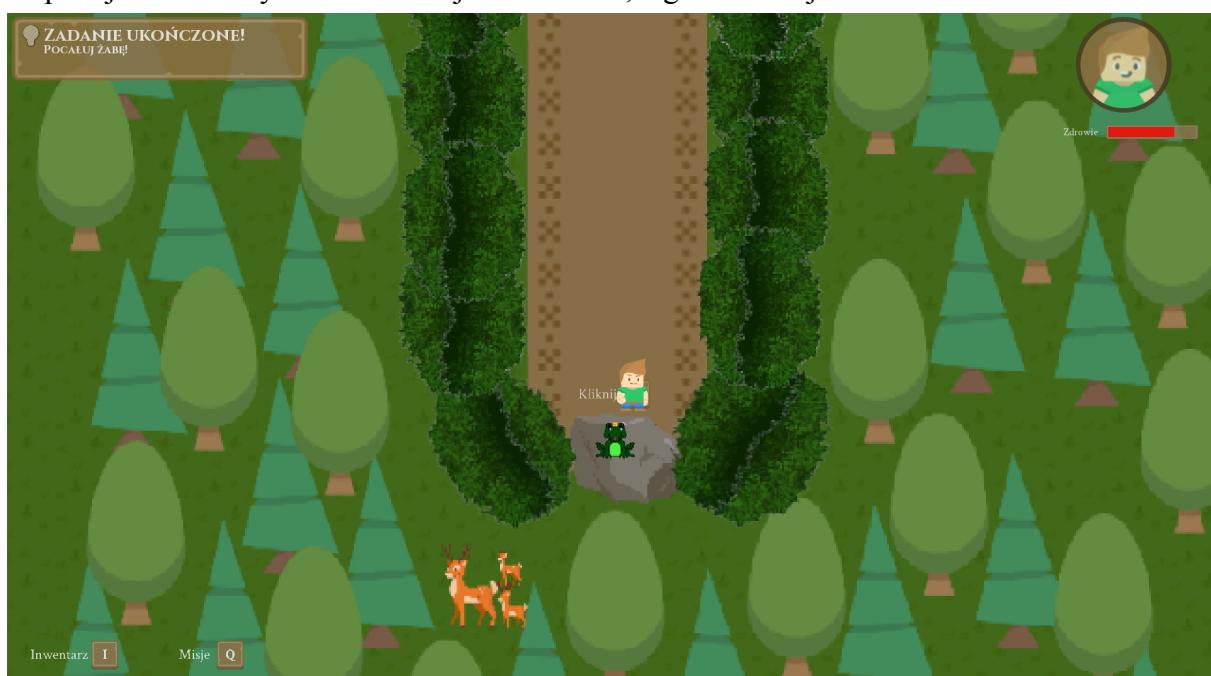
Gracz ma za zadanie znaleźć księżniczkę, która znajduje się w zagajniku na północny wschód od Dobrogrodu. W tym celu gracz powinien iść do bramy sadu w prawym górnym rogu mapy.



Po dotarciu do bramy sadu i kliknięciu klawisza E gracz przeniesiony zostaje do zagajnika, w którym znajduje się żaba w koronie.



Po podejściu do żaby zadanie zostaje ukończone, a gracz dostaje ostatnie zadanie.



Śliska sprawa

Po kliknięciu E przy żabie w zagajniku uruchomiona zostaje wstawka filmowa, w której księżniczka dziękuje bohaterowi za pomoc. W ten sposób gra dobiera końca.



Koniec

Po ukończeniu gry gracz może wrócić do menu głównego, klikając "Wróć do menu".



Ponowne uruchomienie gry z pomocą przycisku "Nowa gra" włącza grę ze zresetowanym postępem gracza.