Maciej Bogusławski, Hubert Kaczyński
PROI 24L
05-06.2024

Project documentation

Contents

Part 1.

Work report

Entry

The project is an implementation of a simple RPG game in which the player takes on the role of a character whose task is to save the kingdom of Dobrogród. To do this, the hero must kill a gang of thugs troubling the city and find the princess.
As part of the project, we implemented a number of functionalities known from RPG games, enabling practice in object-oriented programming. Our main goal was to ensure the universality of the code through an extensive class hierarchy, which significantly facilitates the possible addition of new missions and functionalities in the future. The project is based on the Simple and Fast Multimedia Library (SFML).

Technical information

Required disk space: 171 MiB
Required minimum screen resolution: 1920x1080px
Recommended minimum monitor refresh rate: 60Hz

Installation and commissioning

In order to run the game correctly in the WSL environment, you must install the Simple and Fast Multimedia Library (SFML). To do this, we recommend running the following command in the terminal:

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

After installing the library, you need to compile the program. The compilation rules are contained in the main directory in the makefile. Thanks to this, to compile the program, you only need to execute the following command in the terminal:

```
make
```

Due to the large number of source files, compilation may take several dozen seconds. After correct compilation, the game should be ready to run with the command:

```
./main
```

Description of the environment

Programming implementation and testing of the project were carried out using the C++ 17 language and the Simple and Fast Multimedia Library (SFML) version 2.5.1.

Sources

When designing graphic works for the game, we used tools and assets available on the following websites:

- https://www.photopea.com/ - graphic tool
- https://kenney.nl/ - graphic files
- https://craftpix.net/ - graphic files
- https://www.bing.com/images/create - image generation
- https://openai.com/index/dall-e-3/ - image generation
- https://fonts.google.com/ - fonts
- https://befonts.com/ - fonts
- https://pixabay.com/ - sound effects and music
- https://www.looperman.com/ - sound effects and music

Memory leaks

During the implementation of the project, we observed that the popular tool Valgrind for detecting memory leaks reports a huge number (reaching several hundred thousand) of memory leak problems. While trying to solve this problem, we noticed that even the simplest program using the SFML library, opening and closing the game window, reports a similar (in most cases even higher) number of errors (around 332,800). While trying to investigate this problem, we came across numerous information suggesting that this is a common problem and does not indicate incorrect programming implementation.

```
==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)
```

Fig. 1. Results of memory leak analysis in the final version of the game using Valgrind (332,732 errors)

```
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)
```

Fig. 2. Results of memory leak analysis in a simple, single-file program using the SFML library, only launching and closing the game window using Valgrind (332,911 errors)

We have included the source files of the program in Fig. 2 for comparison here.

File description

The file **main.cpp** contains the main game loop.
File **makefile** contains make program rules that enable correct compilation of the program.

The folder **scenes** contains templates for individual game scenes.
In the file **scene.h** there is an abstract scene class from which individual scene templates inherit.
In files **cutscenescene.h** i **cutscenescene.cpp** there is an implementation of the CutsceneScene class, which is a template for a movie insert.
In files **endmenu.h** i **endmenu.cpp** there is an implementation of the EndMenuScene class, which is the end menu of the game.
In files **fightscene.h** i **fightscene.cpp** there is an implementation of the FightScene class which is a fight scene template.
In files **inventoryscene.h** i **inventoryscene.cpp** there is an implementation of the InventoryScene class, which is the player's inventory scene.
In files **mapexplore.h** i **mapexplore.cpp** there is an implementation of the MapExplore class, which is a template for the main game scene in which the user moves around the map, collects items, interacts with elements of the environment, etc.
In files **menuscene.h** i **menuscene.cpp** there is an implementation of the MenuScene class, which is the main menu of the game.
In files **questscene.h** i **questscene.cpp** there is an implementation of the QuestScene class, which is a scene presenting the player's tasks.
In files **scenemanager.h** i **scenemanager.cpp** there is an implementation of the SceneManager class, which is responsible for correctly changing scenes and controlling the current scene. File **scenemanager.h** also contains the SceneID enum, which is a list of unique scene identifiers.

In the folder **levels** there are subsequent levels of the game, inheriting from scene templates located in the scenes folder.
In files **level1.h** i **level1.cpp** there is an implementation of the Level1 class, representing the first map of the game (in the city of Dobrogród).
In files **level2.h** i **level2.cpp** there is an implementation of the Level2 class, representing the second map of the game (in the forest overrun by bandits east of Dobrograd).
In files **level3.h** i **level3.cpp** there is an implementation of the Level3 class, representing the third map of the game (in the grove northeast of Dobrograd).
In files **combat1.h** i **combat1.cpp** there is an implementation of the Combat1 class, representing the first and only battle in the game.

In the folder **cutscenes** there are specific film inserts.

In the file **cutsceneframe.h** there is an implementation of the CutsceneFrame class, which is a single shot in a movie insert. This class consists of the path to the file constituting the image in the background of the shot and the length of the shot expressed in seconds.
In files **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** there is the implementation of specific cinematic inserts, triggered depending on the player's situation in the game.

In the folder **mechanics** there are implementations of key game functionalities that are stored throughout its duration.
In files **camera.h** i **camera.cpp** there is an implementation of the Camera class, which is an extension of the existing sf::View component, enabling better response to changes in the player's position - including: blocking the camera from leaving the game screen.
In files **mapexplore_ui.h** i **mapexplore_ui.cpp** there is an implementation of the MapExploreUI class, which is a semi-transparent overlay on the game screen with player statistics and keyboard shortcuts.
In files **playerstats.h** i **playerstats.cpp** there is an implementation of the key PlayersStats class storing the player's statistics - his health, strength, items in the inventory, items picked up, tasks, defeated enemies and positions at individual levels.
In files **staticcamera.h** i **staticcamera.cpp** there is an implementation of the StaticCamera class, which is an extension of the existing sf::View component.

In the folder **sceneelems** there are implementations of individual game elements, depending on their functionality.
In files **blockade.h** i **blockade.cpp** there is an implementation of the Blockade class representing a transparent element blocking the player's movement (applied, for example, to buildings on the map).
In files **combatnpc.h** i **combatnpc.cpp** there is an implementation of the CombatNPC class that represents the enemy in combat scenes.
In files **interactiveelement_adversary.h** i **interactiveelement_adversary.cpp** there is an implementation of the AdversaryNPCElement class representing the enemy in map exploration scenes (such enemies do not attack on the map, but when the player approaches them, the user moves to the combat scene, where the CombatNPC object is responsible for the enemy).
In files **interactiveelement_pickup.h** i **interactiveelement_pickup.cpp** there is an implementation of the PickupElement class representing an item located on the map that can be picked up.
In files **interactiveelement.h** i **interactiveelement.cpp** there is an implementation of the InteractiveElement class representing an interactive element on the map that detects the approach of a player and interaction with the element by the user clicking the E button.
In the file **objectids.h** there is a GameObjectID enum containing the unique IDs of items that can be collected or enemies that can be defeated. After the player completes the interaction (e.g. collecting an item), these identifiers are saved in the PlayerStats class object.

In files **player.h** i **player.cpp** there is an implementation of the Player class, which is a visual representation of the player on the map and in combat.
In files **sceneryelement.h** i **sceneryelement.cpp** there is an implementation of the SceneryElement class which is a scene element. The InteractiveElement and StaticElement classes inherit from this class.
In files **staticelement.h** i **staticelement.cpp** there is an implementation of the StaticElement class, which is a static element of the scene that does not respond to the player's behavior - e.g. a tree or scene background.

In the folder **inventory** there are implementations of classes that allow displaying collected items in the player's inventory.
In files **inventoryitem.h** i **inventoryitem.cpp** there is an implementation of the InventoryItem class, which is a field in the inventory (combining the class of a given item and the number of collected items of this type).
In the file **pickupobject.h** there is an implementation of the PickupObject class, which is a template for the category of a given item, and the PickupCategory enum, which is a list of all object categories that can be displayed in the inventory (e.g. bags with coins, crystals, etc.).
In files **pickupdescr.h**, i **fooddescr.h** there are implementations of classes representing categories of specific items inheriting from the PickupObject class, which may appear in the player's inventory along with their names, descriptions, identifiers and paths to files representing their icons in the inventory.

In the folder **quests** there are implementations of specific player tasks with their names, identifiers, descriptions and automatic checking of requirements.
In files **quest.h** i **quest.cpp** there is an implementation of the Quest class, which is a task to be performed by the player (including information about the completion of the task, displaying the task to the player in the form of a notification, etc.).
In files **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** there are specific tasks that the player receives to complete throughout the game.

In the folder **notifications** there are implementations of notifications that the player receives throughout the game.
In files **notification.h** i **notification.cpp** there is an implementation of the Notification class, which is a notification template along with the implementation of the animation of the notification appearing and disappearing.
In the file **hintnotification.h** there is an implementation of the HintNotification class, which is a notification providing the player with advice. This functionality was not finally implemented in the game, but it is a good basis for adding such functionality in further development of the project.
In the file **questnotification.h** there is an implementation of the QuestNotification class, which is a notification informing the player about receiving or completing a task.

In the folder **assets** there are files that are not source code, but are crucial for the correct operation of the game.
In the folder **audio** there are sound files in wav format (including music, footsteps, sound from movie inserts, etc.).
In the folder **fonts** there are fonts in TTF format used in the game.
In the folder **img** there are graphic files that constitute game backgrounds, character textures, inventory icons, items to collect, interface elements and game icons.

Distribution of tasks

At each stage of solution design, programming implementation, testing, as well as planning and creating graphic and assembly works and writing dialogue lines, we performed the tasks together (communicating remotely) or we shared tasks and performed them in parallel, then putting our solutions together into one whole.

Project development possibilities

We developed the project with its universality in mind, with the goal of making it easier to add new mechanics, missions and maps in the future. Among the ideas that we consider interesting and possible to implement as part of the further development of the project, we consider:

1. Adding NPCs walking along designated routes and exclaiming short phrases while passing the player.
2. Mechanics of interacting with items in the inventory.
3. Possibility to save the game state.
4. Expanding the plot and adding more scenes and tasks.

Part 2.

Guide

Entry

The game takes place in the town of Dobrogród - the heart of the kingdom, once famous for peace and security. Unfortunately, the city's glory days are long gone, and to make matters worse, Dobrogród has recently been attacked by robbers from Robin Smród's brutal gang, and Princess Dobrawa has gone missing. The player's task is to solve the city's problems and restore the city's glory.

This guide is intended to make the game easier by explaining individual missions and showing how to complete them.

List of tasks

Throughout the game, the player receives the following tasks necessary to complete the game:

- Hear ye! - find the town crier in the city.
- Robin Stink - Kill Robin Stink and his companions.
- Diversification of investment portfolio - collect 5 bags of coins.
- Redistribution of property - return stolen money to the bard.
- The Princess and the Frog - find the princess.
- Slippery affair - kiss the frog.

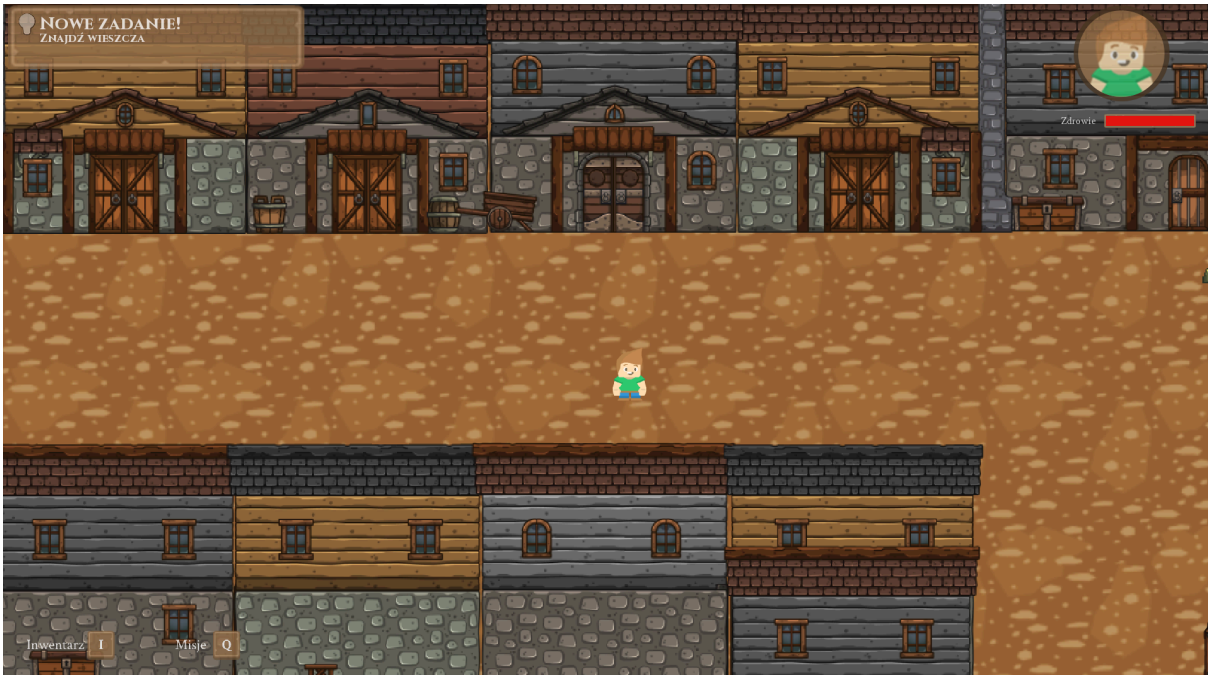
Additionally, the game also offers an optional task:

- Catch ‘em all - collect 10 crystals.

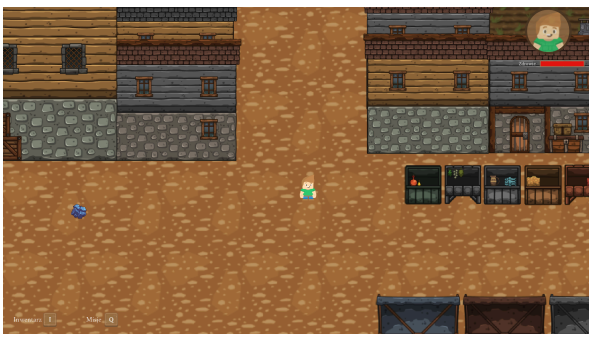
Beginning

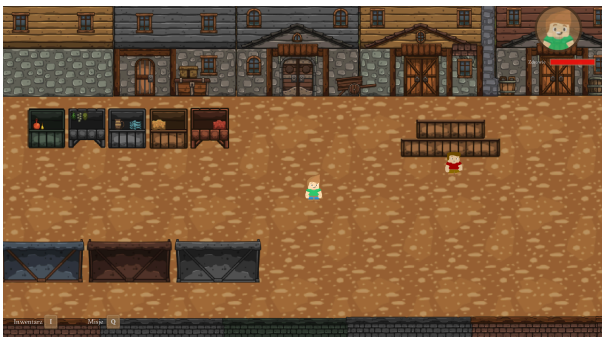
The game welcomes the player with a title screen. After clicking the “New Game” button, the user starts a new game.

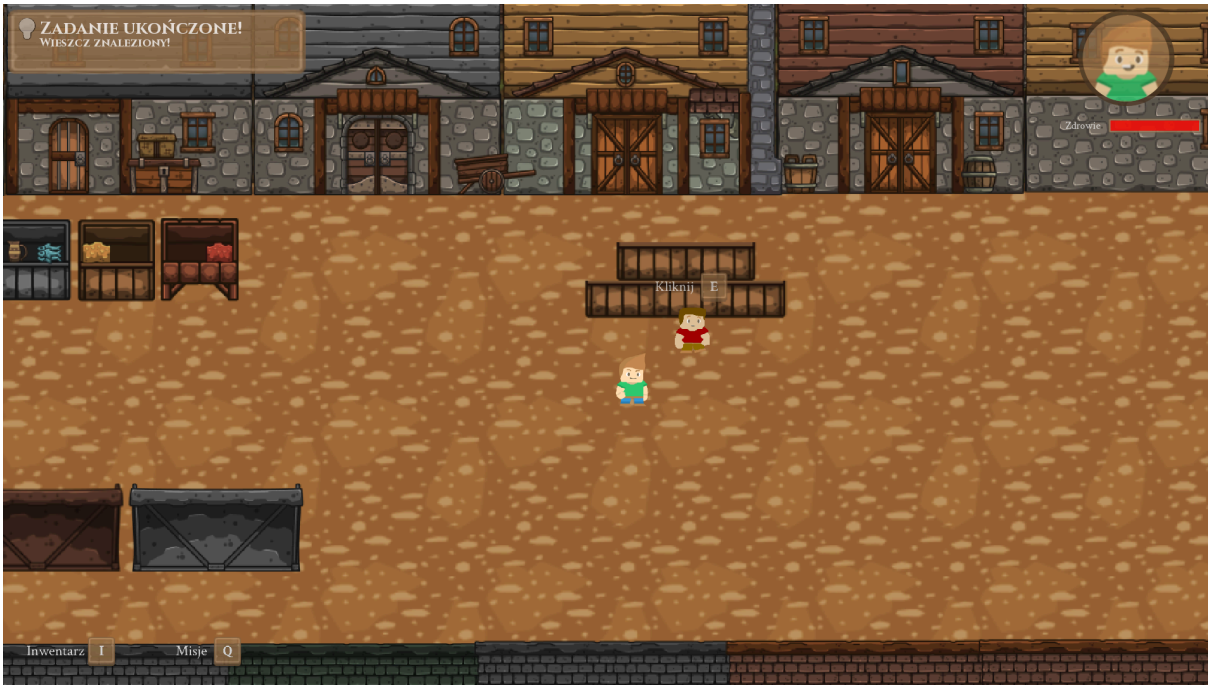
Hear ye!



After entering the game, the player receives a notification about a new task, which involves finding a town crier. The player moves around the map using the WASD keys. You can check your equipment by pressing the I key, and missions by pressing the Q key.



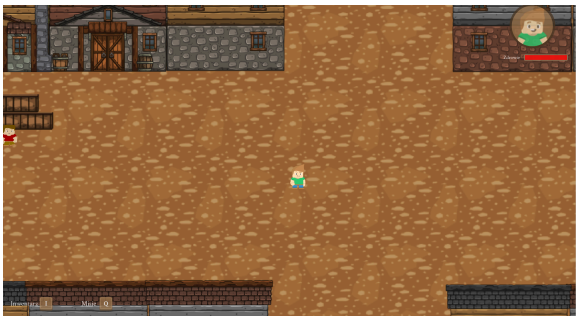


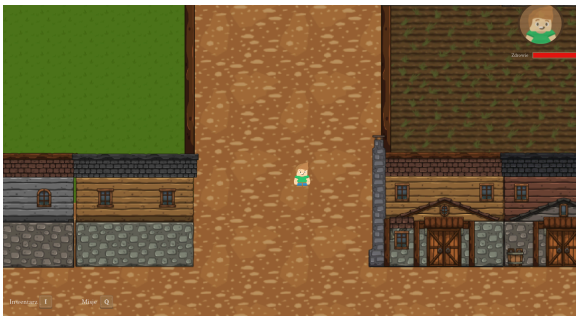


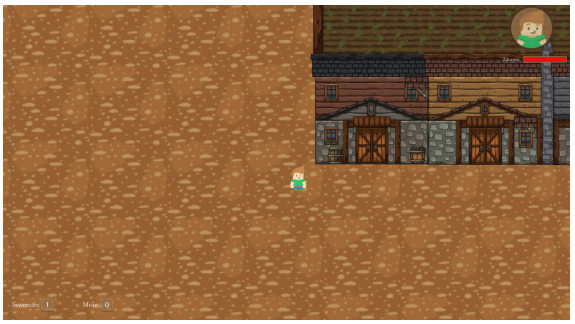
After approaching the town crier, we receive information that the task has been completed. Clicking E takes the player to a cutscene in which he receives a new task - he has to defeat Robin Stink and his companions. The cutscene can be skipped by pressing the Spacebar.

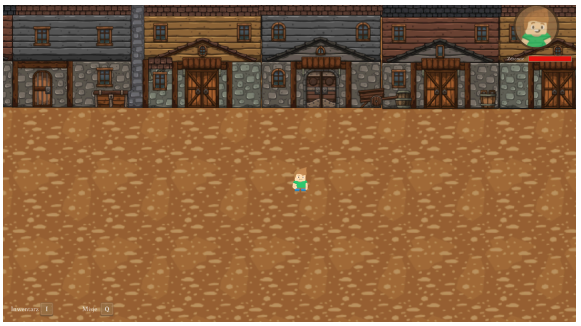
Robin Stink

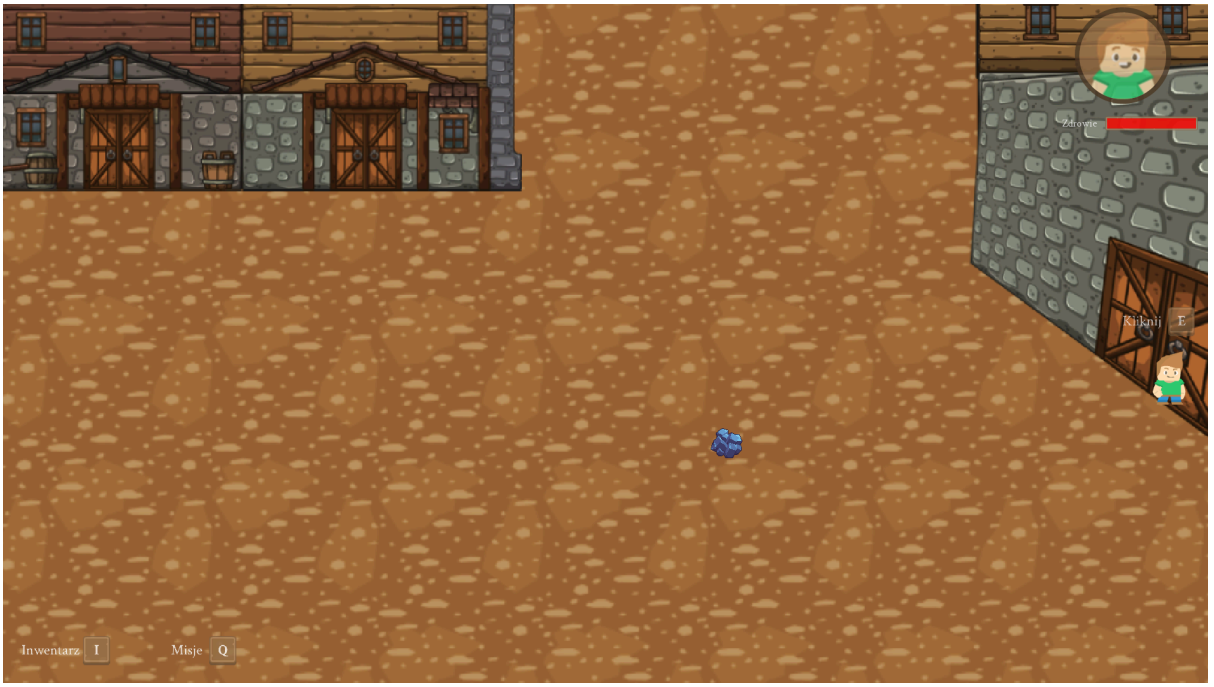
After receiving the quest from the town crier, the player can go to the southeast of the map and go through the gate (without receiving the quest from the bard, the passage through the gate would be blocked).











It is possible to pass through the gate by clicking the E key.

The player goes to the forest east of Dobrogród. Going right and then down, he encounters more robbers of Robin Stink's gang.

Approaching a robber triggers a fight automatically.

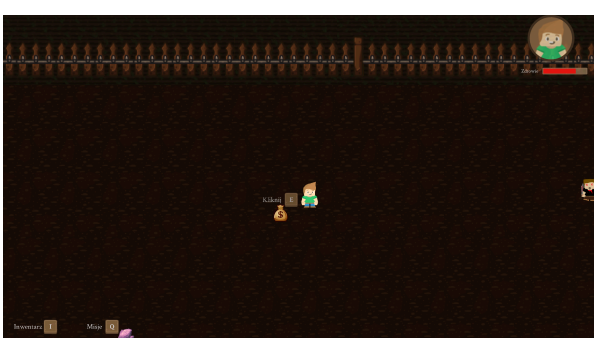
During combat, you can attack an opponent by pressing the C key. The attacked character moves away some distance due to the impact of the attack. During combat, pay attention to your health bar in the upper right corner. The player's death leads to a cutscene, but the user does not have to start the game from the beginning - the game will return the player to the moment when he entered the forest (saving the enemies he has already defeated) and restore the player to 100% health.

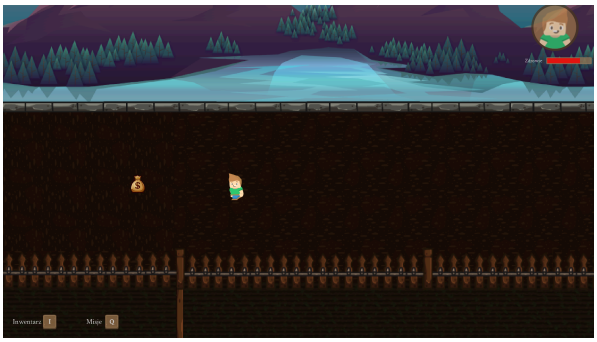
The player must defeat four robbers of Robin Stink's gang, and then the leader of the gang himself. Robin Stink has greater health and attack power, posing a greater challenge for the player.

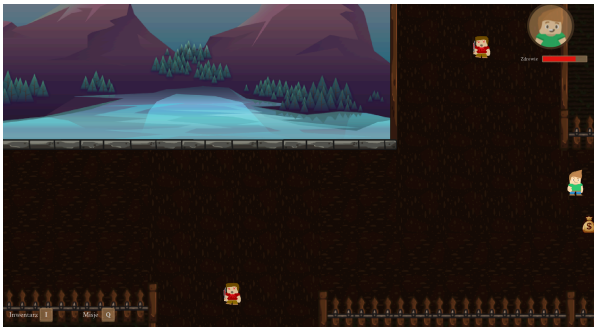
After defeating Robin, the quest remains finished, and the player receives a new task - Diversification of investment portfolio.

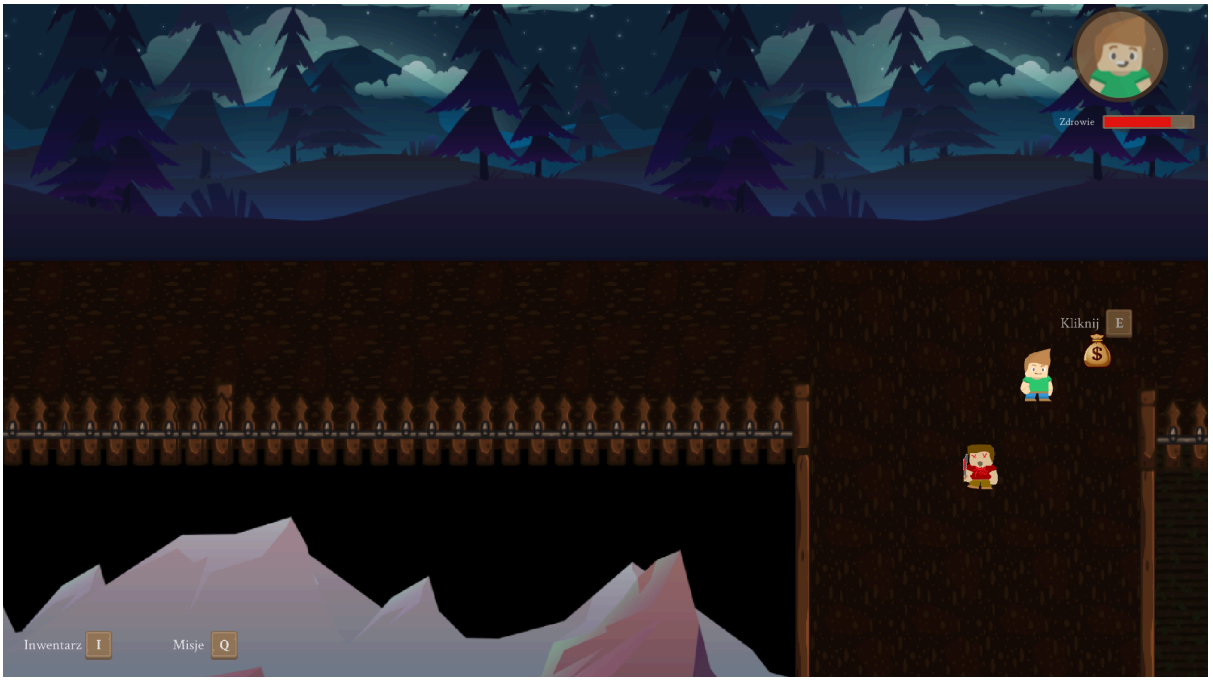
Diversification of investment portfolio

After receiving the task, the player must find five bags of coins on the map that were stolen by Robin Stink's gang. All five bags are located in the forest where the player defeated the bandits. Money bags can be collected by approaching them and clicking the E button.

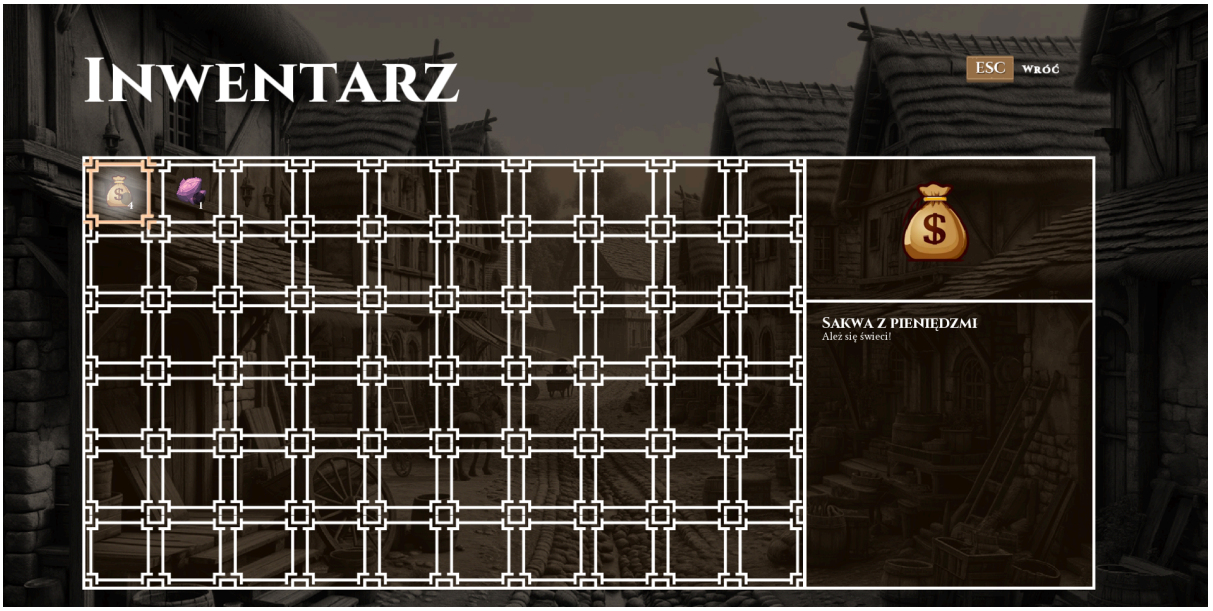








You can check the number of bags collected so far at any time by clicking the I key to open the inventory.



The inventory can be closed using the ESC key.

After collecting all five bags, the task is completed and the player receives the next task.

Redistribution of property

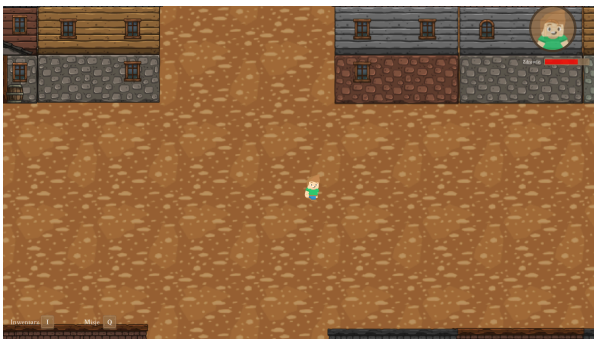
After receiving the task, the player must return to the bard from the task "Hear ye!". The user should therefore return to Dobrogród and return to the main square.

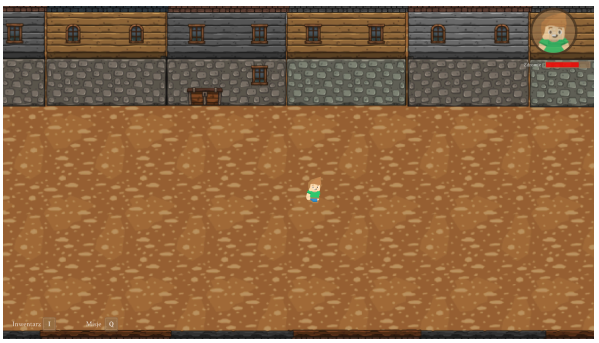
After approaching the bard and pressing the E key, a new cutscene begins in which the town crier thanks the player for collecting the bags and gives him a new task.

The movie insert can be skipped by pressing the Spacebar. After the cutscene, the task is finished, and the player receives a new task - The princess and the Frog.

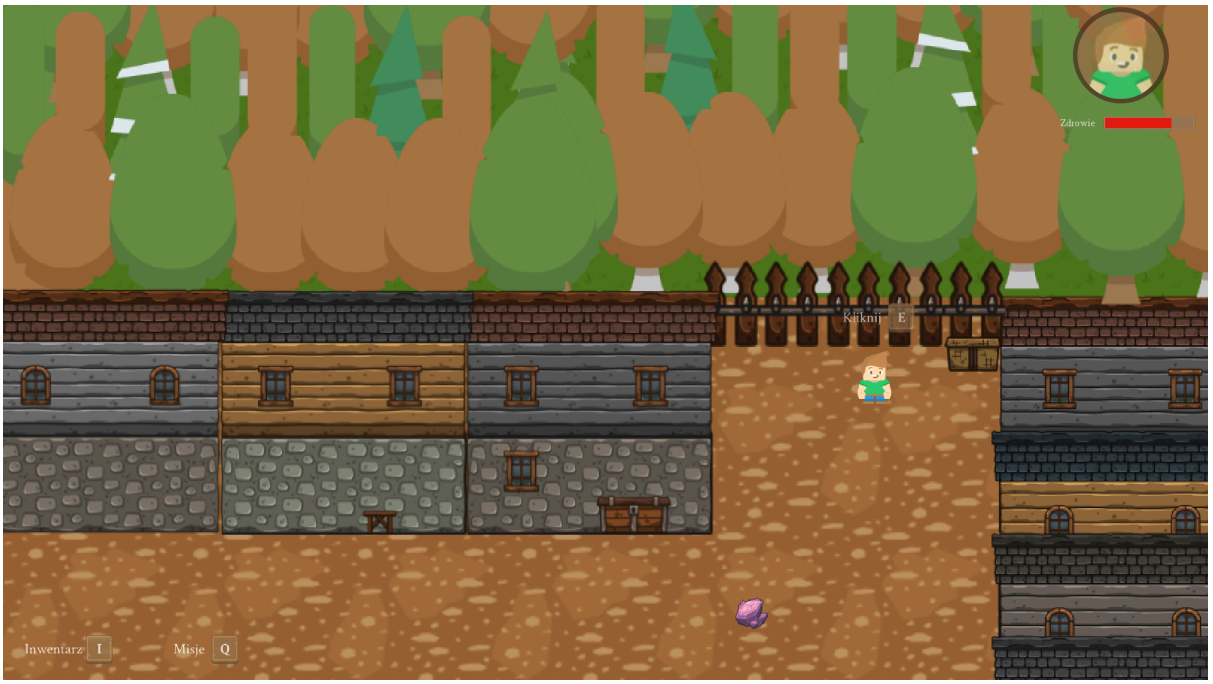
The princess and the Frog

The player's task is to find the princess, who is located in a grove northeast of Dobrogród. To do this, the player should go to the gate in the upper right corner of the map.





Once you reach the gate and click the E key, the player is transported to the grove where you will see a crowned frog.



After approaching the frog, the task is finished, and the player gets the last task.

Slippery affair

After clicking E next to the frog in the grove, a cutscene will start in which the princess thanks the hero for his help. This is how the game ends.

End

After completing the game, the player can return to the main menu by clicking “Return to Menu”.

Restarting the game using the "New Game" button launches the game without previous progress.