Chorzów, 7.12.20

Semester: II
Group: II

Computer Programming Laboratory

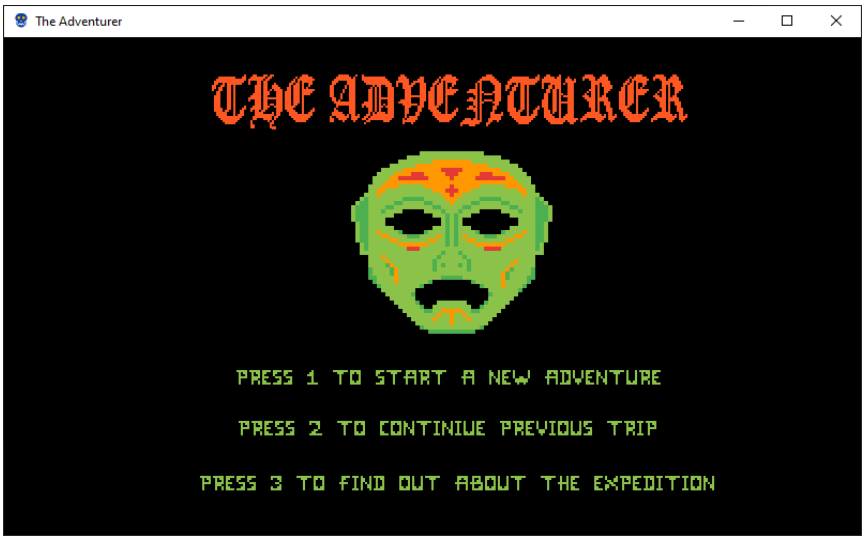
Author: Dominik Zagórnik
E-mail: domizag369@student.polsl.pl

1. Introduction

The Adventurer – arcade game inspired by “Adventure” (Atari 2600) and “Rogue” (DOS). The main goal is to collect all keys, open gates, and get to the exit. The entire level with several rooms is displayed on the screen in the form of a top view. In these rooms, monsters move with different kinds of behaviors. The player has to be careful because it has only 3 lives. The movement is based on the grid. The game contains autosave. The level is loaded from the file. The graphic layer is created using the SFML library.

2. External specification

In the menu, we navigate by 1, 2, and 3 keys.

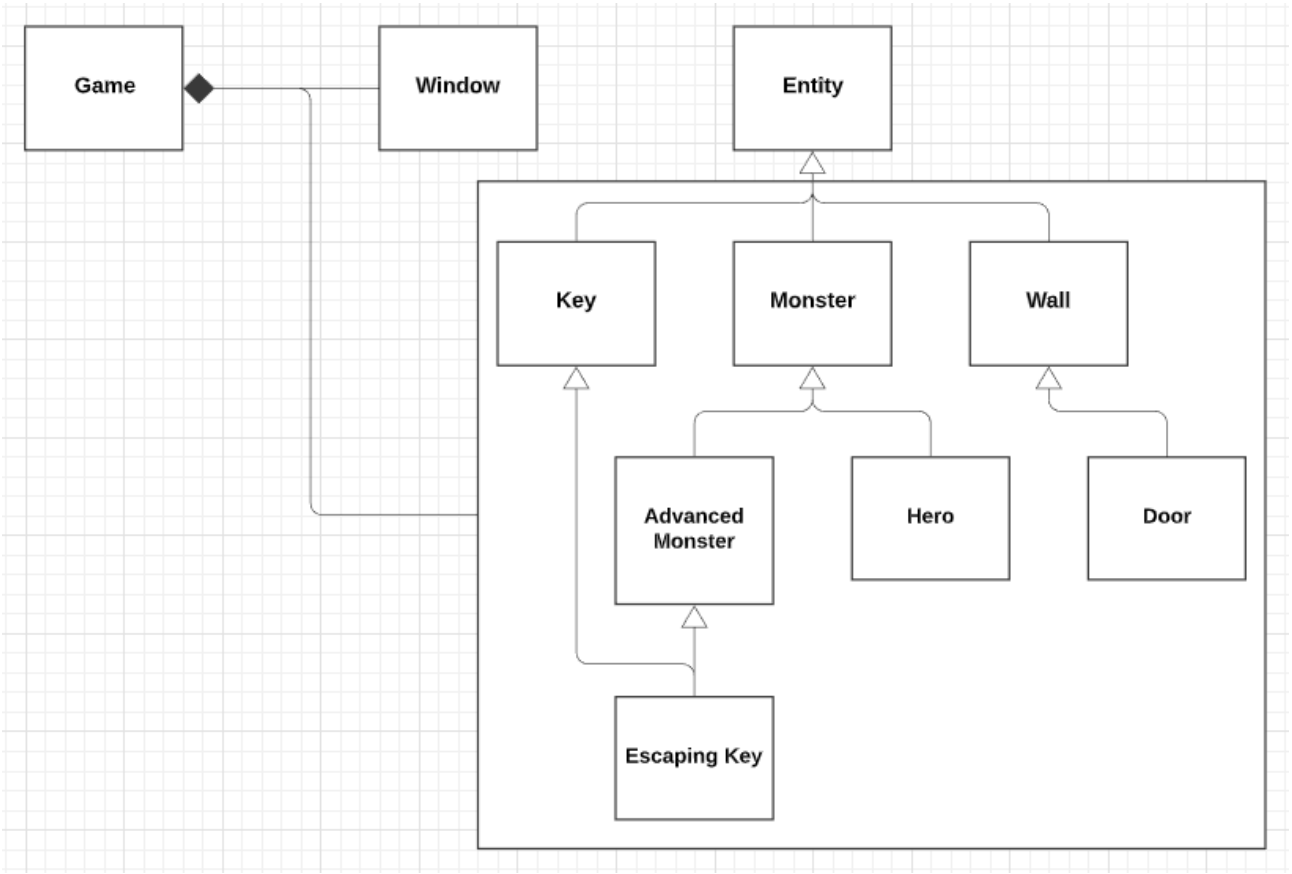


In the game, we move our hero by pressing arrow keys or W, A, S, and D keys. The Esc key opens the menu.

The game is saved whenever we open the menu.

3. Internal specification

The UML class diagram



C++ features used in the program:

1. Inheritance
   Entity is an abstract base class for hybrid inheritance described by the UML

2. Multiple inheritance
   Class *Escaping Key* inherits from classes *Key* and *Advanced Monster*

3. Polymorphism:
   Wall::Collision(Monster &l_monster) is overloaded by
   Wall::Collision(AdvancedMonster &l_advanced).
   AdvancedMonster::Search(Hero& l_hero) is overloaded by
   EscapingKey::Search(Hero &l_hero).
   Parameterized constructors overload several default constructors.

4. RTTI
   Door::Collision(Monster &l_monster) downcast pointer of type *Monster* to *Hero type.*

5. Streams
   Game uses fstream class to load from and save in a file.

6. STL template
   Game uses vectors.

Classes description:

1. Entity
   Abstract base class for game objects rendered on screen. It contains getters to the object position and bounds, setter to position, and render method.

2. Monster
   Describes enemy, which moves randomly and can hit the player.

3. Wall
   Checks collisions with moving entities and don't let them pass through walls objects.

4. Key
   Collectible that disappears after being collected by the player and affects its keys counter.

5. Hero
   Contains information about the player: a number of lives and keys, spawn point position, and flag to pause the game for a few seconds after player death. Checks if the player was hit or is dead.

6. Door
   Behave the like wall, but the player can open it if a collision with the player is detected and the player has the key.

7. Advanced Monster
   Monster, which starts chasing the player if it is in the range. Stores coordinates of last player position in range and additional direction if monster hit in the wall.

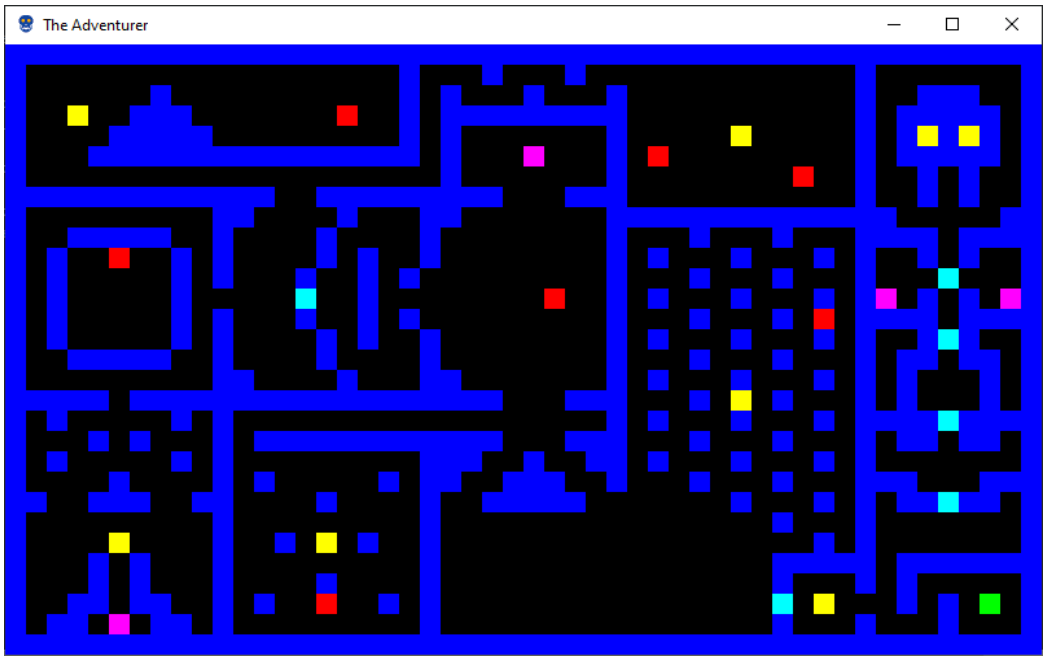
8. Escaping Key
   Key, which moves opposite the advanced monster – tries to escape before the player.

9. Window
   Class the describes SFML render window object and interactions with it. Displays rendered objects.

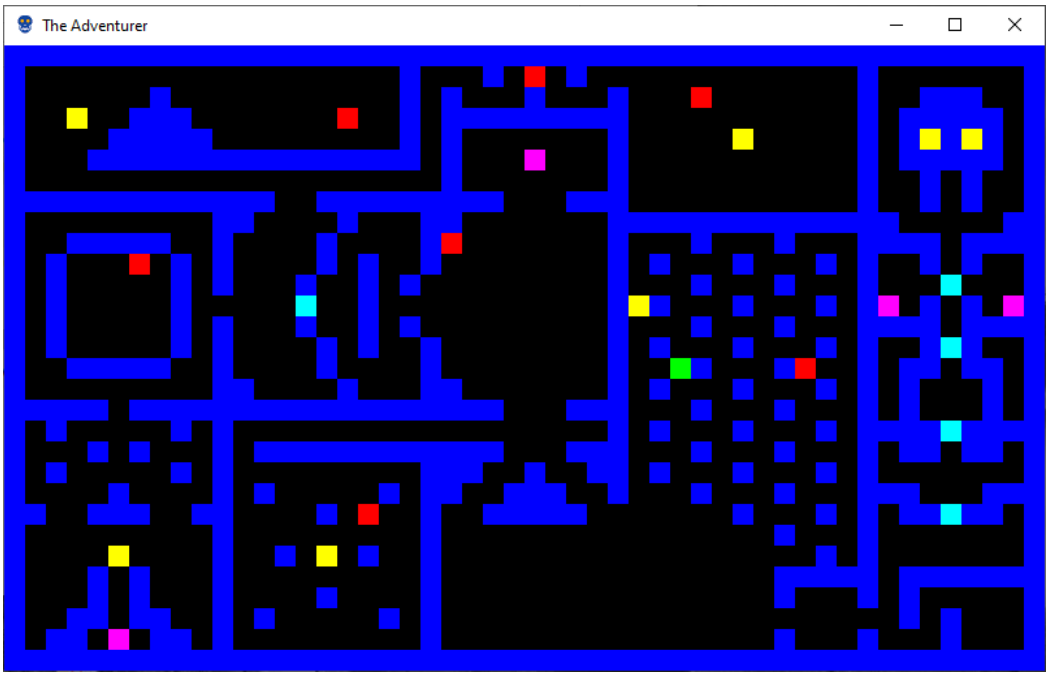
10. Game
    The main class that uses classes mentioned above as members. Handles input to specify a player direction, set the time of all program iterations and framerate, renders objects, saves, and loads progress.

4. Tests

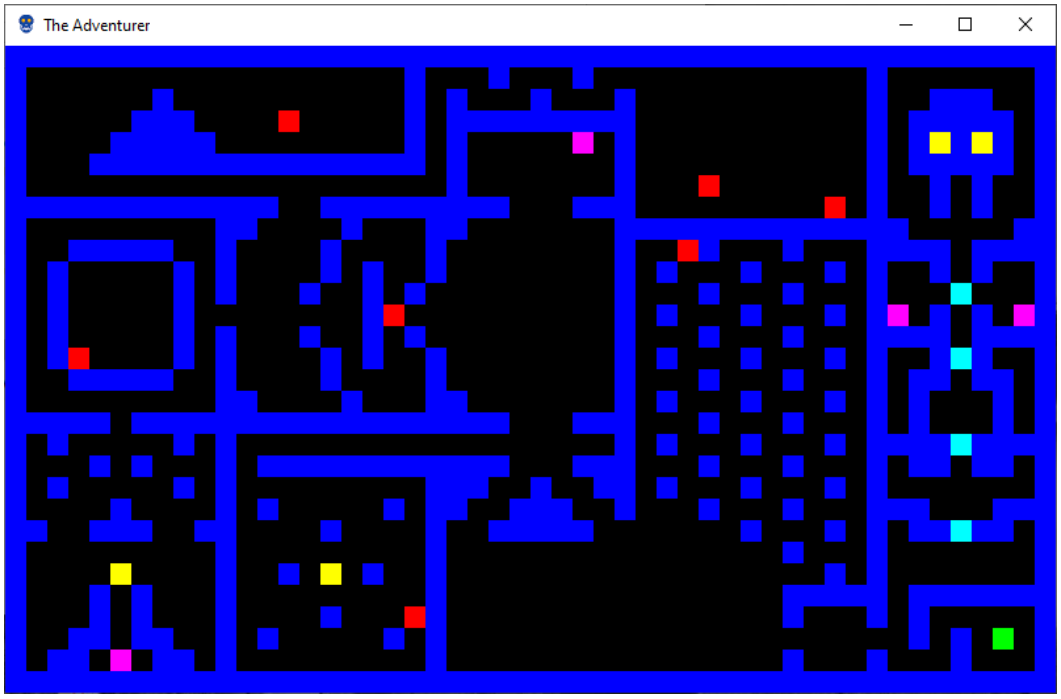
After pressing 1 the game immediately starts.



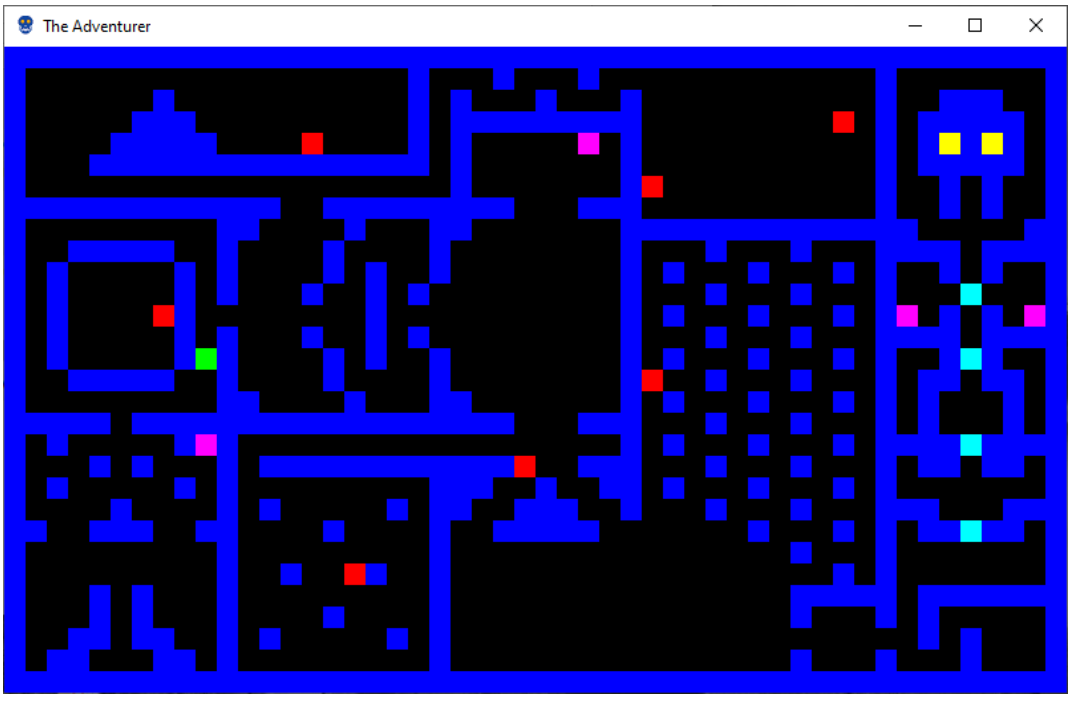
The interaction with the key and door works correctly and also escaping key is already trying to escape.



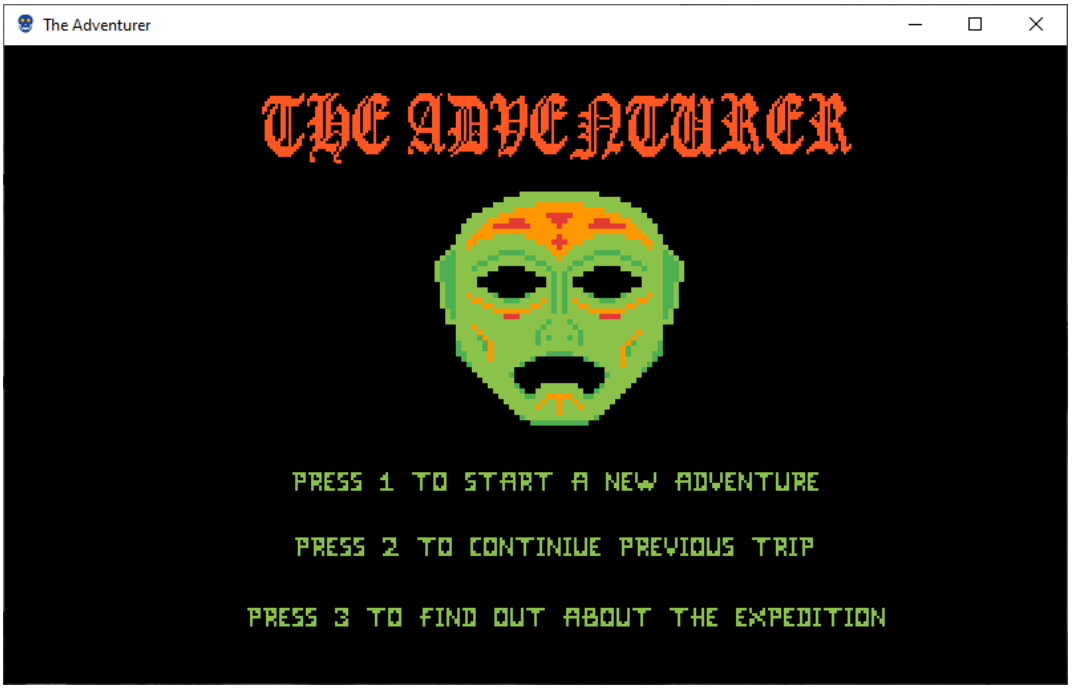
The hero was hit by a monster so there is 2 seconds break.



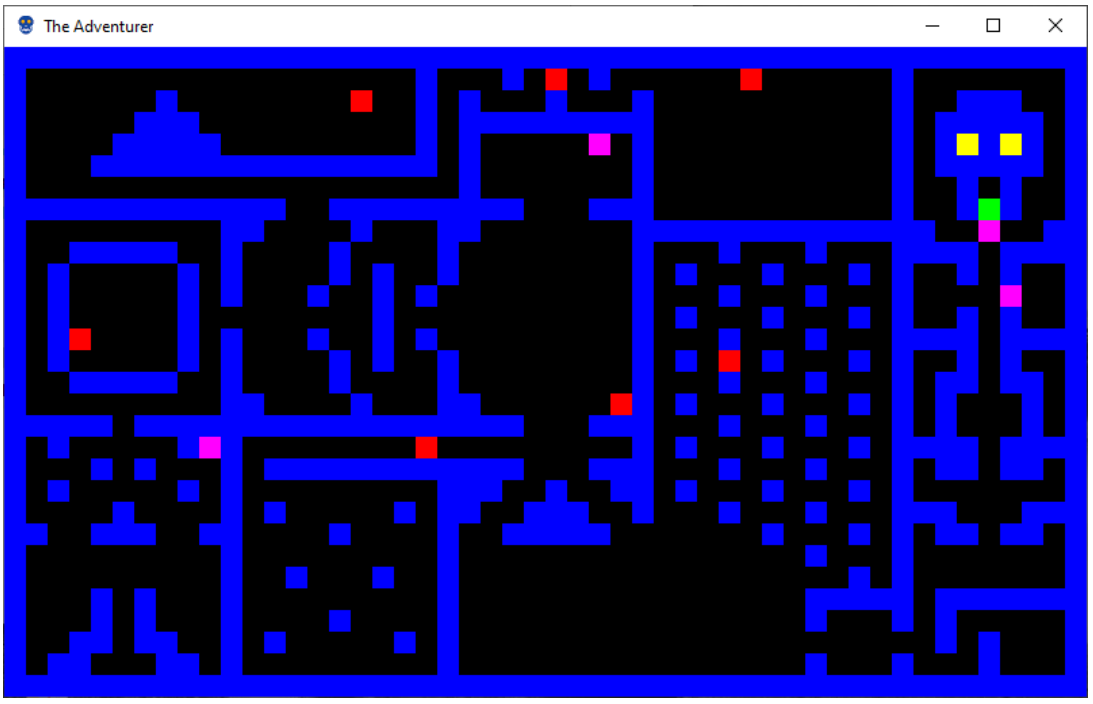
The advanced monster tried to catch the hero.



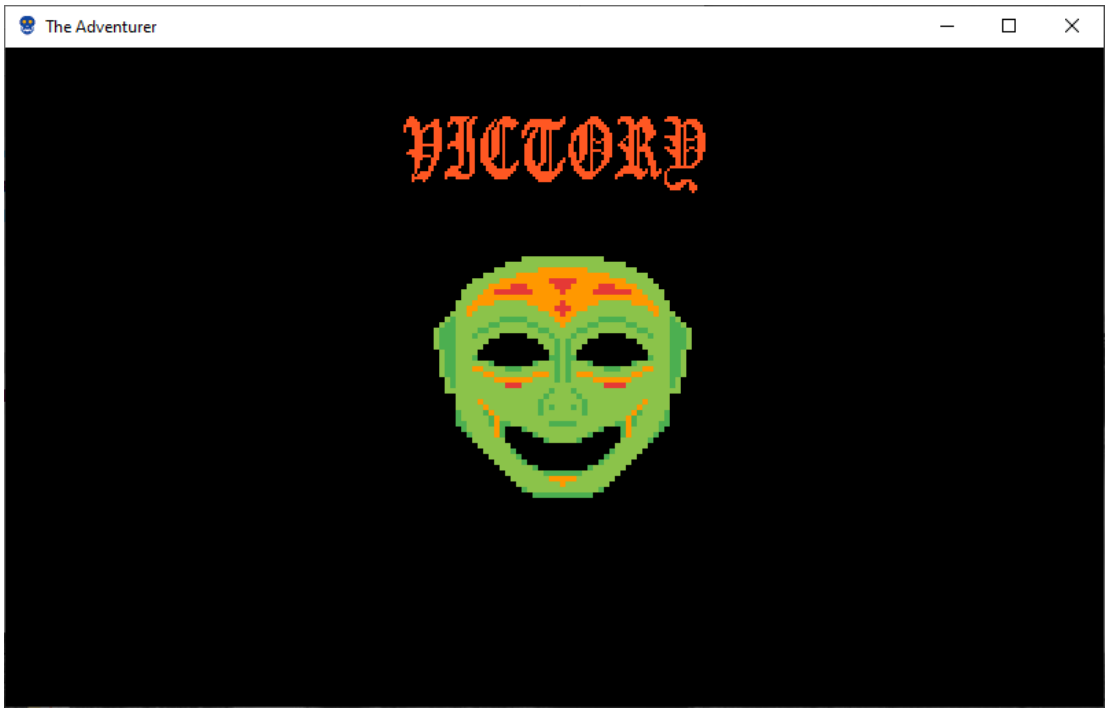
Now we save the game in a safe place by opening the menu



By pressing 2, we back to the game and run to the exit.

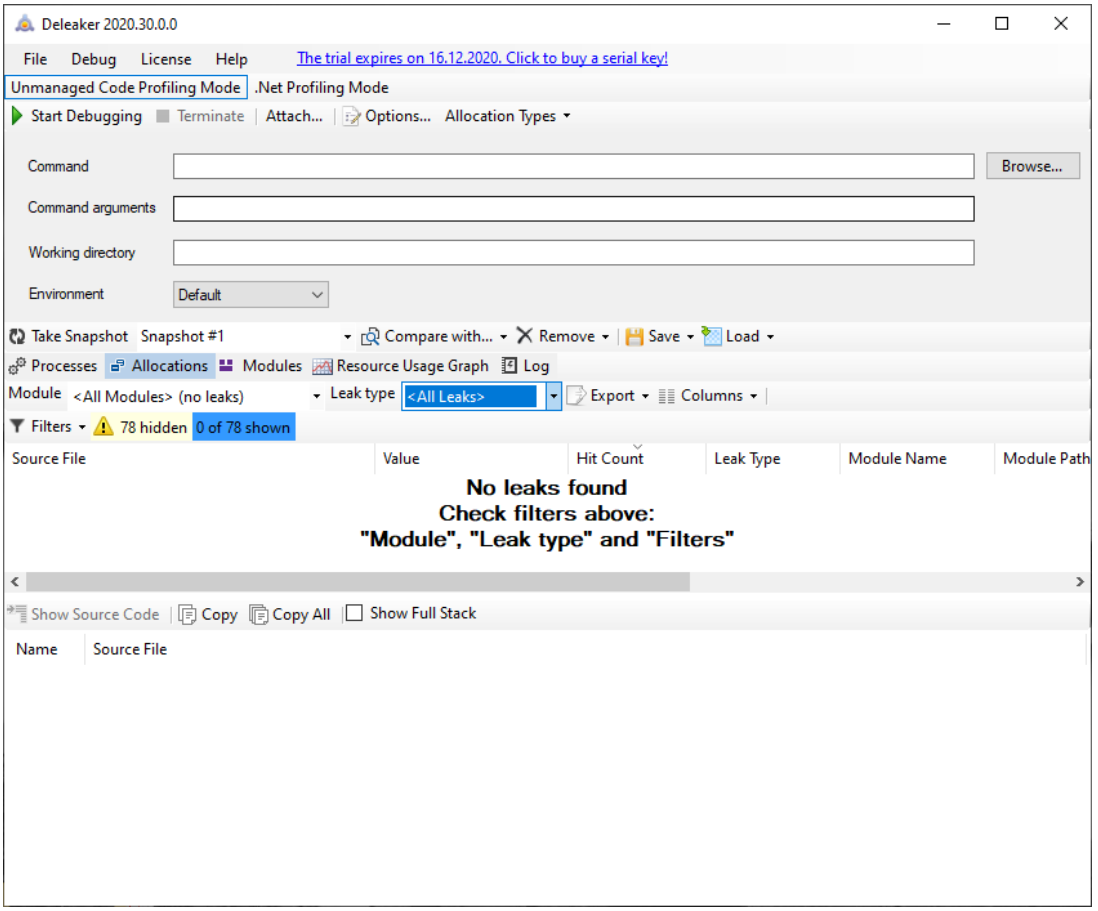


The victory screen.



But if the magenta monster catches the hero we will see another screen.

Looking for memory leaks:



5. Summary

The game works correctly. The initial assumptions gave me a lot of freedom in implementing the features of C++.