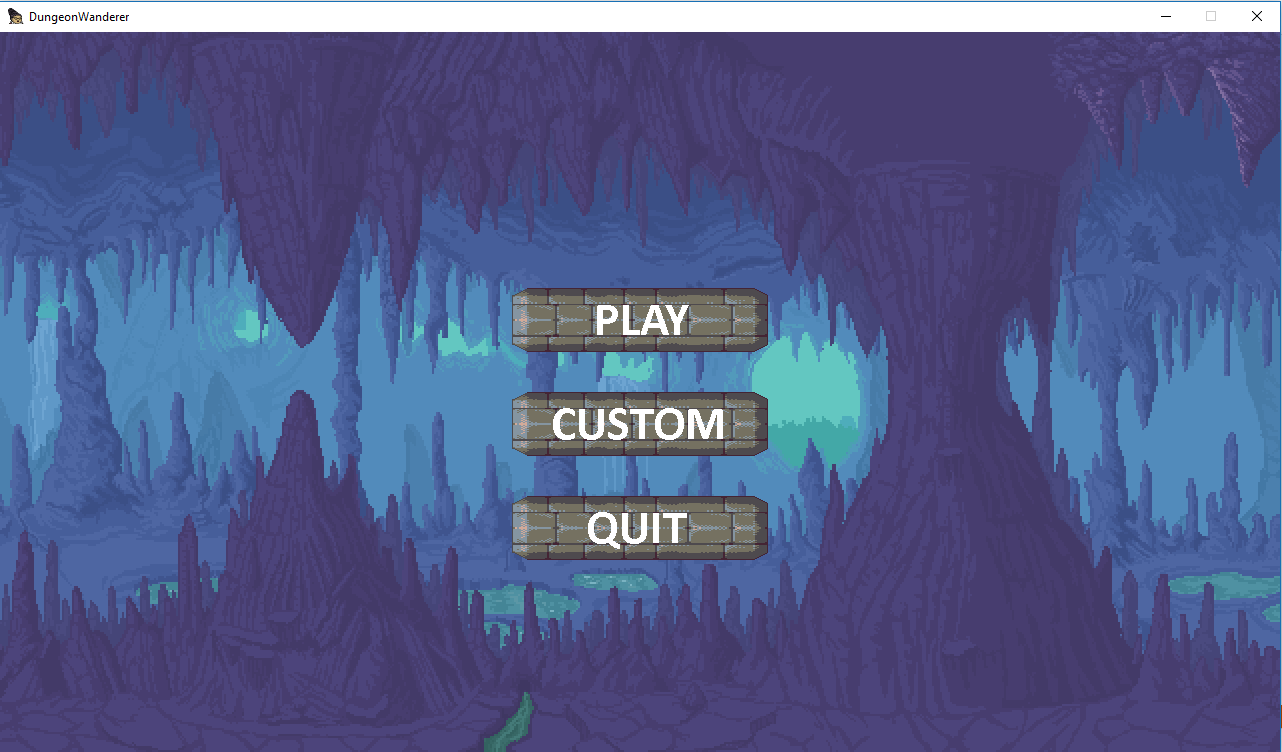
**Explanatory note for the Dungeon Wanderer team project**

**Repo URL:** <https://github.com/Tetawex/dungeon_wanderer>

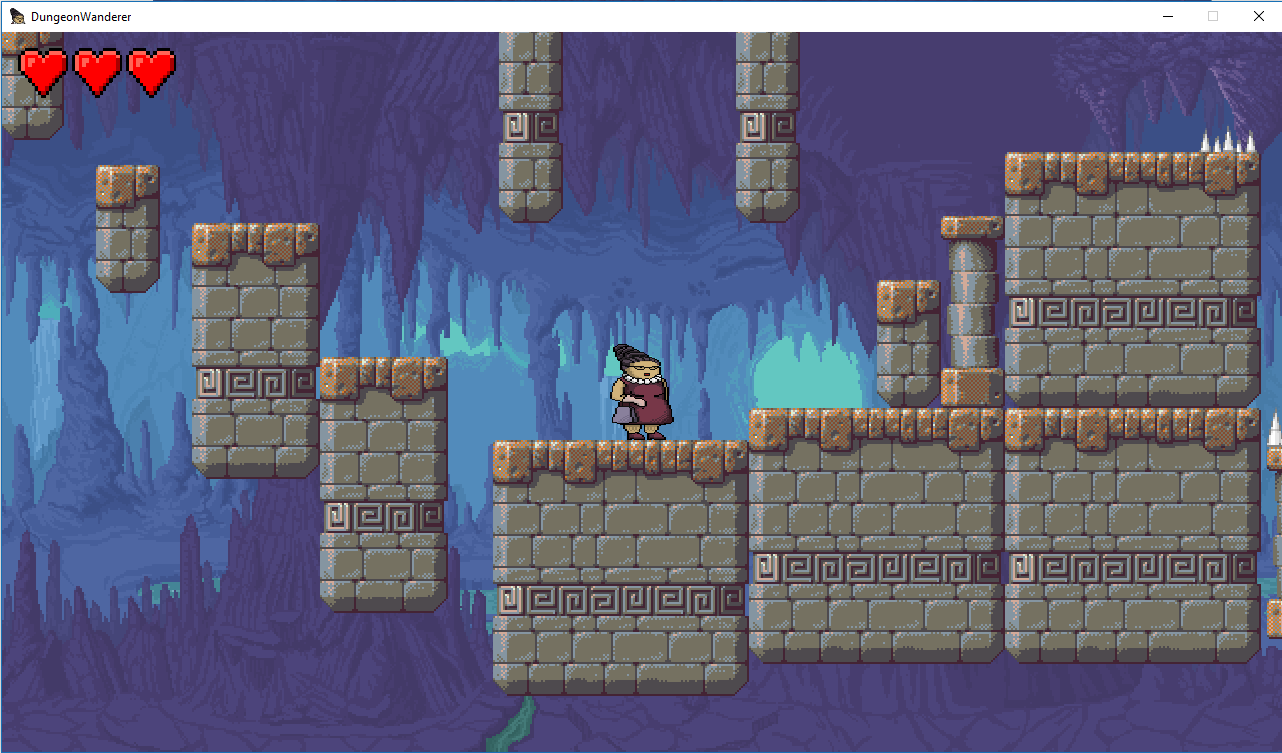
**The program is a game about a lone wanderer hunting for treasure in an ancient dungeon**. There are two levels(there could be more, but the team didn’t really have enough time to do more levels), loaded from json files. Player has three lives which are reset on level progression. The goal is to reach a golden trophy on every level. While trying to do so, player might fall down or touch spikes. Both actions lead to loss of one life and immediate teleportation to the level start. When life count is zero, the player proceeds to the “You lost”screen. Other way, if player succeeds to collect all golden trophies, he then proceeds to the “You won”screen. Both lead to the main menu on mouse click. There is an option to load user- created level, named custom.json and located at the .exe’s folder.

**Some screenshots:**

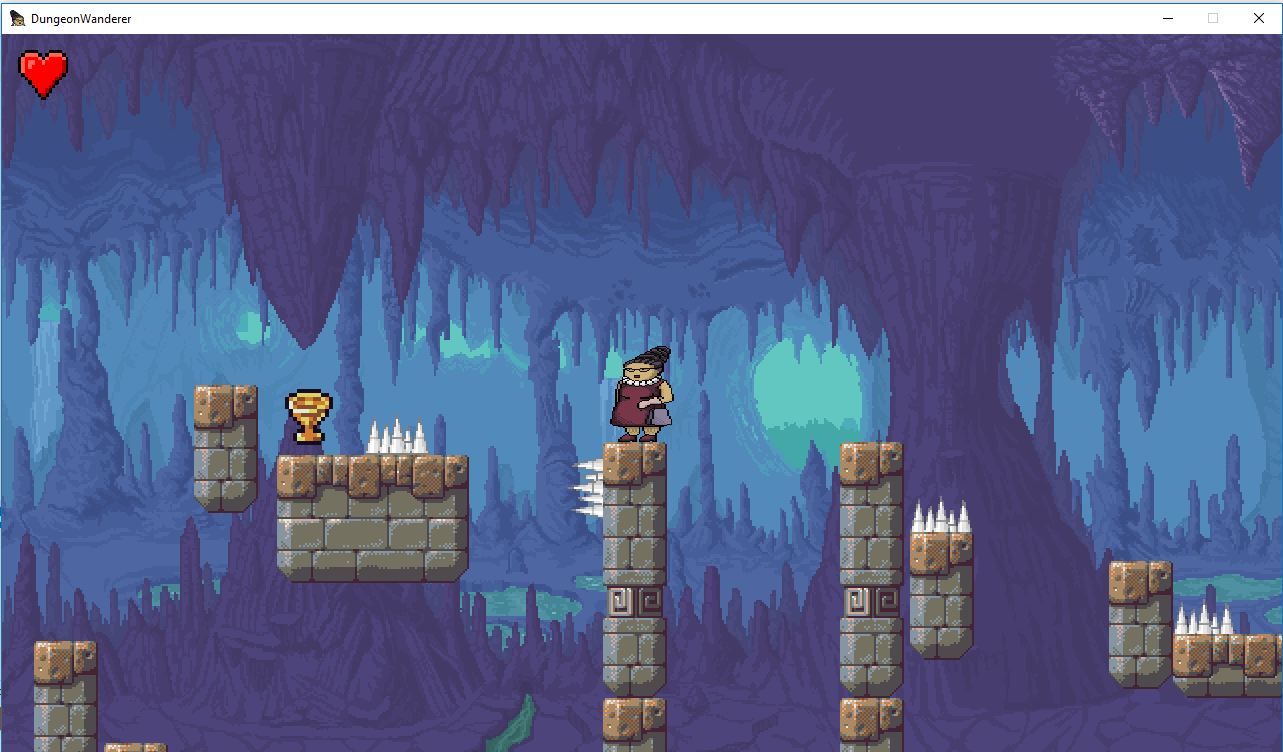
Main menu



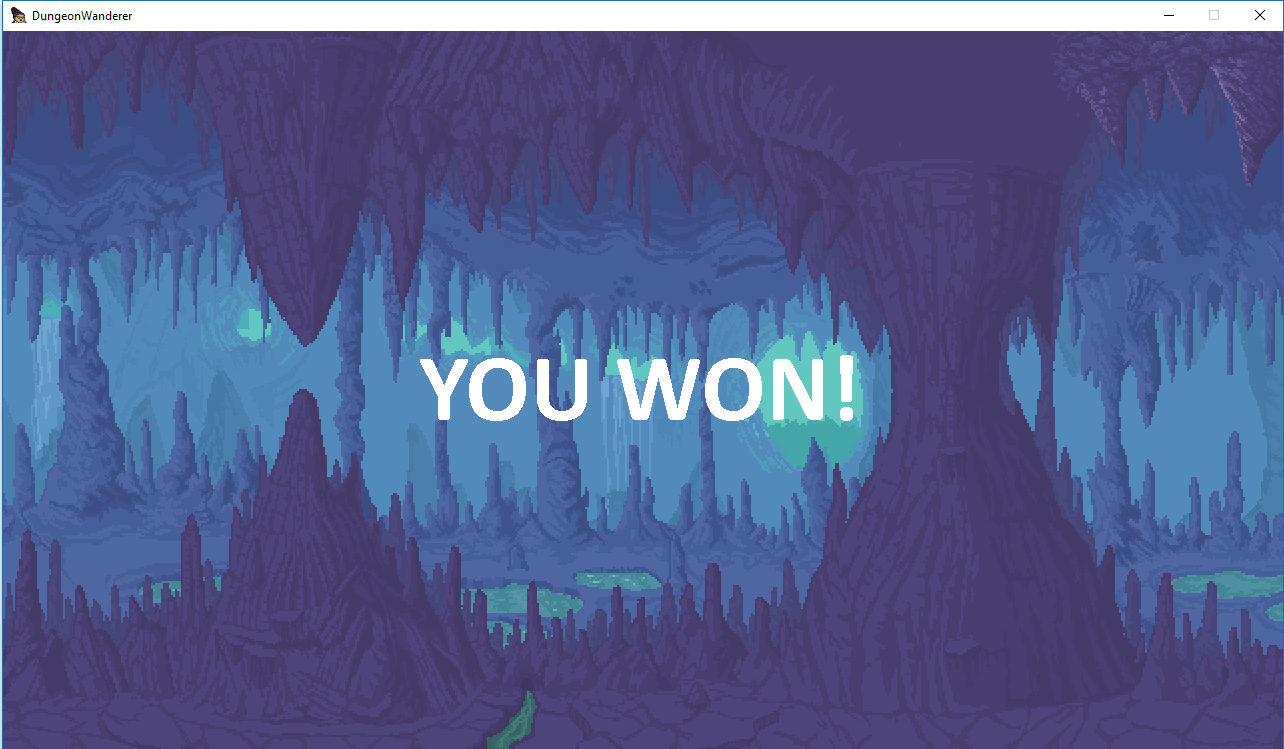
Player at the level start



Player is about to reach the trophy!



Game over – player won!



**Libraries and frameworks used:**

* **Artemis** – ECS library
* **Farseer Physics** – box2D-like physics(not a straight port though)
* **Newtonsoft JSON** – library for some JSON parsing
* **MonoGame** – framework aimed on creation of cross-platform games, makes game development much simpler when compared to WPF/WindowsForms

**The ECS – Entity Component System pattern** is used in code, the pattern helps to implement data-oriented design. **How it works:** there is an **entity world** with a bunch of entities. Every entity has a list of **components** attached to it. Components are just some **bags of data** with no logic whatsoever. There are also **systems** iterating through entities with specified set of components every tick or through preset timespan. Such design makes **collaborative development** way easier because parts of code with different functions are separated, as there are no heavy **“god objects”** called player, enemy, etc. like ones created with traditional game architecture approach. Another benefit is **extensibility** – adding new features doesn’t require massive changes in code.

**Project structure(namespaces):**

**Components:**

Classes with no logic whatsoever – simply data bags to be processed by systems. Attached to entities.

**Systems:**

Classes which process entities with required set of components every tick or every timespan.

**Templates:**

Template classes are basically entity presets. They help to reduce the amount of boilerplate code.

**Core:**

Classes like Screen and its extenders, which encapsulate logic for different screens of the application, DWGame – class extending MonoGame’s core class Game, GameStateManager which controls screen transitions, AssetManager – class responsible of assets delivery, ensures that assets are loaded only once, some utility classes like ShapeFactory.

**JSON:**

Utility classes created for loading levels from JSON files. LevelModel is a class that represents a structure of json file. Level loader – static class with the only self-explanatory method – LoadLevel.