Achraf Yandouzi 20211812

Meat Boy Report

Advanced Programming

The main focus of this project was not the functionality and features of the game I implemented, but rather to prove that I have a good understanding of the theory that we've discussed during this course. In this game I tried to apply all the new design patterns and other coding conventions that I've learned.

Design Patterns

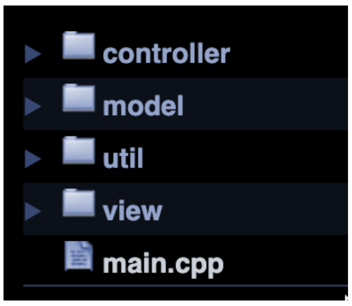
The most significant convention that I applied is that of the design patterns. Design patterns are a great way to accelerate the development process of a large project. Rather than reinventing the wheel, I was challenged to recognize these patterns and apply them where appropriate.

MVC

The first, and arguably most important, design pattern is the so-called Model-View-Controller (MVC) pattern. This pattern forced me to separate the game's data model, user interface and control logic into three distinct components. The *Model* component, containing classes for the game's entities, such as the player and the walls. The *View* component, containing classes for rendering these entities on the screen. And finally the *Controller* component, containing the game's control logic and managing the interaction between *Model* and *View*.

Although this might require a bit more work initially, once it's set up properly, it makes it much easier to make changes to one component without interfering with another. I could for example decide to represent the game using a different library, other than SFML, without having to change the code in the *Model* component.

This design pattern can also be found in my folder structure, where I made sure to put each file in the right component folder.



Observer

Next up we have the Observer pattern. This was implemented using an Observer and a Subject class. The Observer class defines an interface for observing the state of the player, and the Subject class allows Observers to register and be notified of changes in an entity's state, e.g. a change in position.

Abstract Factory

The Abstract Factory pattern was used to provide an interface on how the entities in the game should be created. The AbstractFactory class contains a pure virtual method for creating an entity, and the ConcreteFactory class overrides this method and actually creates the different entities using EntityView.
This makes it so the World does not need to use any of the View related logic.

Singleton

The Singleton pattern was used for the Stopwatch class. This ensures that there’s only one instance of the Stopwatch at all times in the program. On top of that, this makes the Stopwatch easily accessible to all the classes that will need this helper class.

State

The State pattern was implemented by the StateManager. This class has a Struct Level, which contains a state variable. It is this state variable that tells us in which state the game currently is, which could either be the main menu, or a certain level.

Other Coding Conventions

Other than the design patterns, namespaces were used to divide the modular sections of the code. For example, all the Classes in the Model folder are also in a Model namespace.

Also, the entire project does not use a single hard pointer, only smart pointers were used to ensure no memory leaks are present.

And finally, the project makes great use of polymorphism where applicable, which is visualized with the Doxygen documentation.

There’s obviously some other coding conventions (like making good use of keywords, using exception handling, etc.) which can all be found in the project.