Controller and its Design Patterns

In this document we analyse the Controller of this project and the used design patterns for its realization.

Additional hypothesis used by Controller

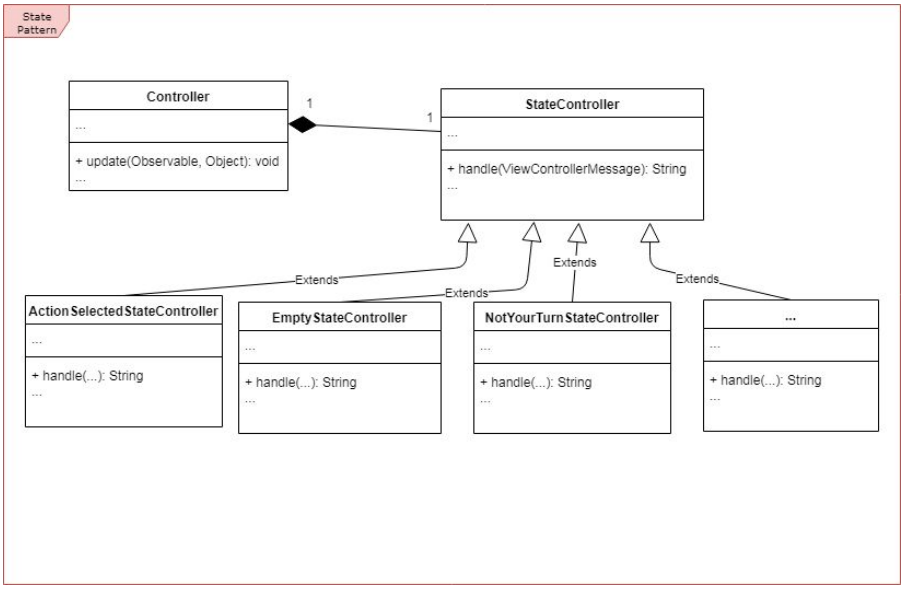
- If the players are disconnected and their characters are killed, they automatically respawn in a random room.

Concept of the Controller

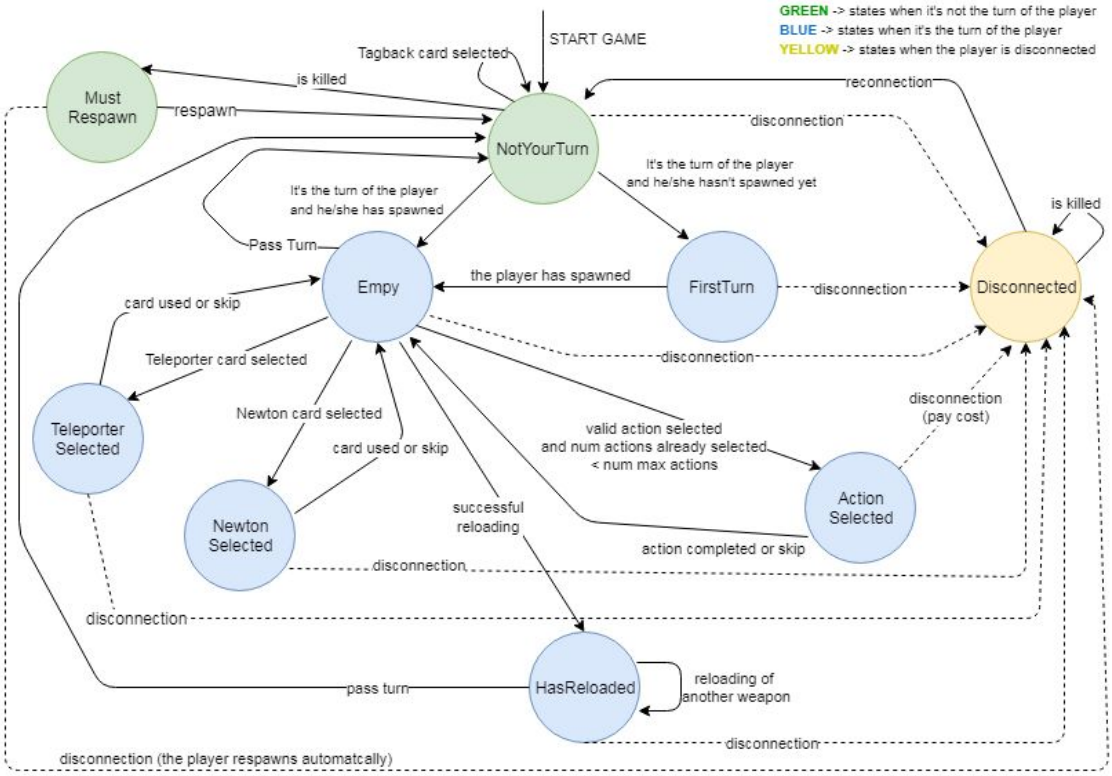
Following the Model-View-Controller pattern, we decided to limit as far as possible the controls on the inputs in the View. When an element of the View is selected by the player, a message is generated. For example, if we select a player target, a PlayerMessage will be created. If we select a cell, a CellMessage will be created. These messages arrive to the Controller through the Observer pattern. The Controller will decide afterwards if the messages are valid or not. If I have selected a Firemode that needs a player target and the user selects a cell, the message will be discarded by the Controller. This permits to create a far more simple View, leaving all the logic of the game in the Controller.
Furthermore, the Controller can write simple string messages in the View, to be showed to the user. These string messages are used to make requests to the user and, in some cases, to explain why the inputs are not valid. The action “Move” requires a cell where to move the player. So, the Controller will write in the View “Please, select a cell.”. If the player selects a too distant cell, the Controller will write “This cell is too distant. Please, select a cell.”.
Finally, we decided to create a Controller for each Player. In this way, the state of the Controller represents the state of the Player and the behaviour can change accordingly.
This behaviour requires a use of State and Strategy Patterns, as it will be showed.

Design Patterns

State Pattern: State of the Controller



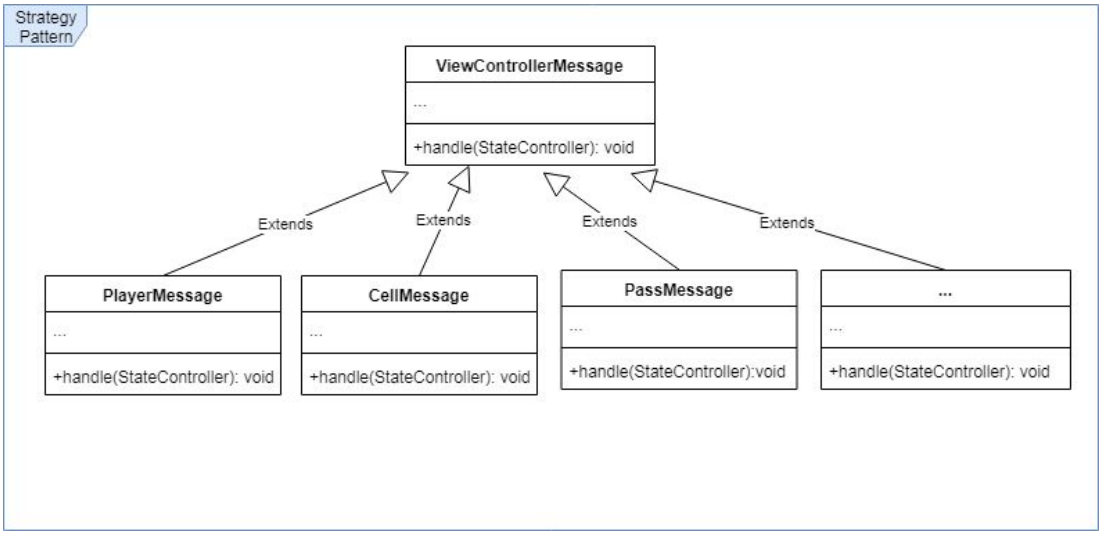
When the update method of the Controller is called, the message is forwarded to the StateController object, which represents the state of the controller. The following schema shows the Finite-State machine of the Controller



Only the most relevant events are showed for simplicity.

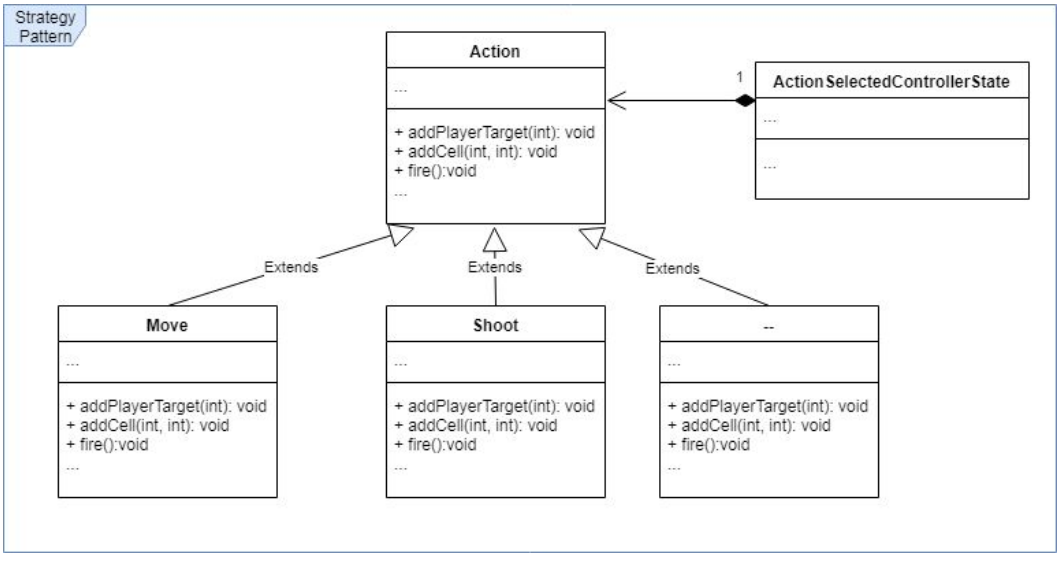
Strategy Pattern: Messages

Each ViewControllerMessage must call a specific method of the Controller to be handled. PlayerMessage must call handlePlayer(int) and CellMessage must call handleCell(int, int), for example. We used the Strategy Pattern to implement this. The StateController calls the method ViewController.handle(StateController). In this method, the ViewController itself will call the right method of the StateController, depending on its dynamic type.



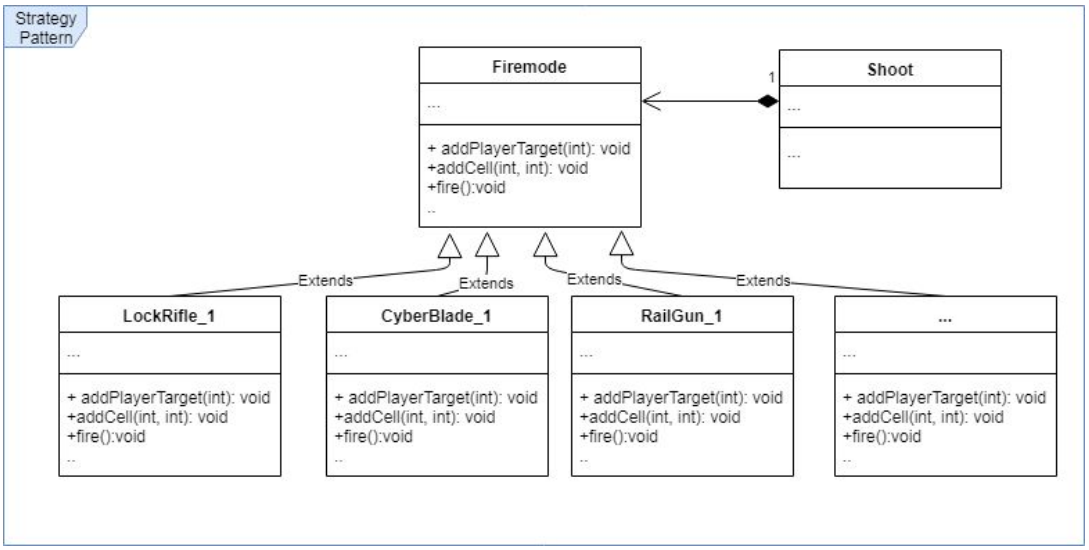
Strategy Pattern: Actions

When the state of the controller is ActionSelectedControllerState, the player has selected an Action. The behaviour of the Controller must change according to the dynamic type of the Action. We used the Strategy pattern.



Strategy Pattern: Firemodes

A Shoot action changes its behaviour according to the Firemode selected by the Player, so the Strategy pattern is used.



Example of Sequence Diagram

We add a Sequence Diagram to show an example of interaction with the controller that uses all the Pattern explained before. In this example, it is showed the interaction of a valid PlayerMessage requested by the Firemode LockRifle_1.