Marvel Legendary

Almost Done

Revision: 1.0.0

Development Team Agreement Practices

Contents

[Coding Style 3](#_Toc495868243)

[Project folder structure 3](#_Toc495868244)

[Minimum requirements to run the game 3](#_Toc495868245)

[Main Game Loop 3](#_Toc495868246)

## Coding Style

The team should use the *“High Integrity C++ Coding Standard”* rules, as defined in: <http://www.codingstandard.com/section/index/>.

Future changes in code style are possible, as long as it brings some benefit to the team.

## Project folder structure

* Assets (original files, should not be included on repository)
* Docs
* Source
* Temp (should not be included on repository)
* Test
* Game
* Lib

## Minimum requirements to run the game

This information is part of the project’s scope and is described in the Game Design Document.

## Main Game Loop

The Game class has an instance of the Board Class, which is responsible by the control of the game flow, control of the turns, as well as the conditions of victory or defeat. This class is also responsible for dealing with events triggered by the player, such as buying a card, playing a card, attacking, etc.

In this architecture, the main loop takes place after the initialization of resources and after the player chooses to start a new game in the main menu.

Main Loop Pseudo-code:

While not game over  
 start new turn

while player does not end the turn

intercepts player events

update game

render scene

test victory or defeat conditions

**Intercepts player events**: Each event held by the player triggers a "broadcast event" that is either intercepted by the board or by another class responsible for handling that event. That class then, makes the necessary changes in the game.

**Update game:** The Board class updates the STATUS of the game, executing its own update method, or the update method of other classes, since in this architecture, the Board class has instance of class Player, that owns an instance of Class Deck, that has instances of the Card class and so on, creating a hierarchical chain of update methods calls, from each class that is necessary, according to the triggered event.

**Render scene:** Similar to what happens with the update method calls, each class that intercepts a particular event has its own method for rendering and updating the scene, based on the received event.