

Bilkent University

Department of Computer Engineering

|  |  |
| --- | --- |
| |  | | --- | |  | |

**CS 319 - Object Oriented Software**

**Engineering Project**

Color Shooter: The Spectrum Adventurer

Final Report Second Iteration

Group 3-F

Hasan Selim Yağcı

İrem Ural

Erdem Adaçal

Alper Mehmet Özdemir

Course Instructor: Bora Güngören

Final Report

Dec 16, 2017

**Table Of Contents**

1. **Introduction………………………………………………………...2**
2. **Description………………………………………………………….2**
3. **How To Play………………………………………………………...3**
4. **Implementation Process…………………………………………....4  
   4.1. Challenges ……………………………………………………..4  
   4.2. Changes After The Design……………………………………6  
   4.3. Status of Implementation…..…………………………………7**
5. **General API Document…………………………………………....7**
6. **Conclusion………………………………………………………….7**
7. **Appendix…………………………………………………………....8**

### **1. Introduction** The purpose of the final report is to document the implementation process and the final work including a description of the project. In the description, the system will be explained briefly parallel to the Analysis and Design Report. Since the system that is projected on the document is a game, a ‘how to play’ manual is given in the final report, which will basically include user interface and related descriptions. In the implementation process part, challenges that are faced and handled, changes from the Design Report of the project will be explained. **2. Description** The projected system is a game called Color Shooter: Spectrum adventurer, which is a 2D platform-shooter kind with a unique jumping and color system. It is user friendly in terms of learning the game play but also becomes challenging in later levels. There are basically two modes of the game, easy and hard. The user tries to navigate through enemies and obstacles to reach the goal in a crystal themed world. The user uses his/her gaming skills to find the correct route and follow it to reach the goal in each ‘level’ as well as he/she uses reaction skills/speed to overcome the numbers of different kind ‘enemies’ and ‘obstacles’ without dying. Reaching the goal in every level will unlock the next level allowing the user to play a more challenging level. The system keeps track of how many times the user have ‘died’ and tell users how many lives it took for ‘the player’ to finish the game in the end.

**3. How To Play**

Our game is developed using java language, hence the player needs a Java environment to play the game. User can reach the game from our github repository:

https://github.com/erdemadacal/CS319\_Group3F

When the game is developed fully the .jar extension will be available. Players can download it and play the game.

Following are the very instructions for a user to play the game:

The user uses the keyboard to play the game and uses the mouse to select options in the main menu. After the necessary selection regarding new game button on the frame, user can start the game. In the game, user can move left and right with the corresponding arrow keys, make shoot with the ‘spacebar’ and ‘change colors’ with the key ‘Z’ (blue), key ‘X’ (red) and key ‘C’ (green). The user can pause the game and bring up the ‘pause menu’ with the key, ‘P’. The user can select an option from the pause menu with left mouse button. The user can resume the game by clicking the resume button on the pause menu. The end product stick to the properties defined in the Analysis and Design Reports.

The color system in the game is one of the features that makes the game challenging. User need to adapt the player’s color accordingly to the blocks and enemies color. When user want to kill an enemy with a bullet, player’s color need to be changed to that color by user. Besides, in order to pass through the blocks or sometimes not to fall below of the blocks, user need to change the player’s color as well.

In the hard mode of the game, there is the ‘automatic jump’ system instead of jumping intentionally. When player moves to the ledges of the platform, it automatically jumps.It could be a strategy for the user as well as being challenging. These two special features of the game are considered while level design is being created.

**4. Implementation Process  
  
 4.1. Challenges**

As the design of the system is modeled before coding, sometimes it is hard to implement according to modeled system. There are some missing parts that is not predetermined, hence it creates huge gaps between classes or subsystems. To implement the system in the same manner, as the developers, we had to think in detailed and investigate how to achieve our goals. We tried to design our system by using design patterns to decrement coupling between the subsystems and used Facade, Factory, Strategy design patterns and MVC design architecture. Although the Facade classes were identified before in the Design Report, how we should use these classes are not. Outer classes in other subsystems should access the Facade class to access/modify any data in the subsystem. That’s why they were harder to implement as these classes should have several methods to allow these connections. Strategy pattern allowed us to separate view and controller subsystems and allows decoupling. Lastly, Factory pattern encapsulates the creation of objects and it helps us to it for future improvements and modifications. Hence it provides extendibility in the project.

One of the most important problems that occurred was where to draw all of the model objects according to the level design. Whether it should draw it in the Model, however it is not Model’s job. It should store the necessary data for the program and allow necessary methods to do changes on them. Hence as a last decision, the draw methods should be located in the Model, however through the Controller, the GameManager class call these draw methods in the render method of the GameManager, though the View will initialize the draw in the MainMenuView class.

Secondly, according to the first Design Report there is one ActionListener implemented in the MainMenuView as a private class. Because of the fact that it is declared as private class it could not passed to the other panels, while instantiating these panels. Hence the buttons declared in each class is accessed through setters and added Actionlisteners to these buttons. Besides, during the design we decided on java’s swing and awt library and designed the view according to its components. However these libraries complicates the implementation, with other libraries the view can be implemented more easily, but we stick to our decision and implement it in this way. Hence it is important to learn different libraries, the services that they provide beforehand.

GameManager class is the most important class in the controller and it accesses Model subsystem and gives necessary information to the View subsystem. Hence it was hard to code it according to the second and first Design Report, as there are several missing parts that are not considered during design. Additionally, there are several accesses in this class, it was a bit harder to access these objects without violating the patterns. In the controller subsystem, Handler more ınteracts with the model, while Game Manager interacts with the MainMenuView. Hence it is important for the game play that they should be written in an effective way. Firstly while writing this class the functionalities are considered, however as the program gains more modularity, the implementation is restructured and it fitted better to the Object Oriented Design. Therefore, there are additional setters and getters defined in some classes.

As a last challenge though in the first final report we had hard times to draw the image to the view. However, as a final product we accomplished it and added all the features to the game.

**4.2. Changes After The Design**

There are several changes according to the second design report, they can be considered as small, as they did not change the subsystem decomposition, design pattern or the layering of the subsystems. Hence the system can be easily understanded from the design report, as the same design patterns are used during the implementation as well.

As a small change, we added some boolean properties to the model classes, to avoid handlers interaction to the model objects, this allowed us to separate these two subsystems. For instance, in the Player class the following properties are added, such as health and numberOfDeaths. Then the getter and setter methods of these properties are added in the model classes. As the GameMap has all the objects created during the game in a linked list, by using the properties of the model, new properties are set and passed to the handler. As Handler has acces to the GameMap class. Then inside the Handler the properties of the GameMap are used. This allows us to prevent the access of Model classes to the Controller class, Handler. Though we considered in this way in the Design report we have to define some new properties inside these classes. As additional methods, we defined saveGame and loadGame methods in the GameManager. These are added in order to save the game when player exits the game, the level, the mode and the number of deaths of the player are saved. Moreover, the loadGame method is used to load the game, it reads the level.txt file and loads the properties specified before. Besides, the Texture class is initialized in the GameMap class, therefore it is only accessed in the Model class and decoupled from the Controller. We defined some additional methods in the GameManagerInterface, to use the Strategy pattern. These must be defined in the controller to use the view. In the SoundManager class, there are some changes according to the Design Report. We added several musics, therefore several soundManager objects are created inside the SoundManager class statically, therefore this allows us to access it everywhere. However, we prefered to use it in the GameManager class only, hence we defined some new methods. Then the methods of SoundManager are accessed in the MainMenuView class by calling the methods of the GameManager. Lastly, we removed two of the power ups, to ease the gameplay, the increased rate of fire powerup would be meaningless because the player shoots upon releasing the spacebar and the full spectrum mode would decrease the playability of the game due to collisions with colored blocks.

**4.3. Status of Implementation**

The project is finished in the second iteration! There is no to do’s left.

**5. General API Documentation**

General API documentation for now can be found in the github page. The main design pattern used in Color Shooter: Spectrum Adventurer is the Model-View-Controller desgin pattern. Additionally, facade design pattern is used to reduce coupling between the classes in different subsystems.

**6. Conclusion**

As a team we learned how to develop a game and how to design it in an object oriented way. Although in some ways it enhances the implementation, sometimes it creates difficulties to implement. We learned that forward engineering requires many knowledge such as the system, the environment that is used and the design patterns which will be used during implementation. It is hard to figure out the design of the system beforehand without any knowledge about these. However it cannot be denied that well defined analysis and design reports, eases the developers job extremely. While designing and implementing the project we learned the principles of object oriented software engineering and become familiar with the real world problems.

**7. Appendix**

User Interfaces from the GamePlay



