

Bilkent University

Department of Computer Engineering

**CS 319 - Object Oriented Software**

**Engineering Project**

Color Shooter: The Spectrum Adventurer

Final Report First Iteration

Group 3-F

Hasan Selim Yağcı

İrem Ural

Erdem Adaçal

Alper Mehmet Özdemir

Course Instructor: Bora Güngören

Final Report

Nov 4, 2017

**Table Of Contents**

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

### **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, and non implemented parts of the system for the first iteration 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. 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.

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 keys, ‘Enter’ or ‘P’. The user can select an option from the pause menu with left mouse button. The user can resume the game by either pressing ‘P’ key again or clicking the resume button on the pause menu. The user can restart the current level by pressing the key ‘R’. **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 use Facade design and MVC design pattern. Although the Facade classes were identified before in the Design Report, how we should use these classes are it. 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.

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 ıts 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 Game Manager class call these draw methods in the draw method of the Game Manager, though the View will initialize the draw in the MainMenuView class. Secondly, according to the 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 it complicates the implementation, with other classes 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 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, CollisionManager is significant for our game as it has a dynamic structure. It 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.

As a last challenge we could not manage to draw the game on the screen, though the game is ready to draw first level and the player to the view. Although the interaction of the view and controller is accomplished, draw to the screen cannot be managed. We have hard times to decide where we should have a runnable class.

**4.2. Changes After The Design**

There are several changes according to the first 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, it is unnecessary to add a class named as EnumColor class instead of that it is added as an attribute to the GameObject class, which is the parent of several model classes. In the View subsystem, according to design report ScreenManager is used to get frame, game panel and graphics, however, now the view will use the game manager and will draw the game objects inside the GamePanel which is a JPanel and declared in the MainMenuView class. Hence ScreenManager class will not be implemented. It is an improvement as well, the design now fits perfectly to the opaque layering. Additonally, to allow transitions between the panels, CardLayout is used. It was not specified in the report before. In the Collision Manager, new attributes are added, as well as new methods. For instance the collision of objects differ hence it divided into several parts and called in handleAllCollisions method. For now, the collision collision detection for player and enemy is implemented. calculateCorners method is used to detect collisions with using rectangle around the game object. However this class needs to be developed furthermore. For now the GameStateManager class will not receive game information from the game information class (we have removed it from the implementation for now) as there is only one level in the game. Update methods are separated for players, enemies, and power ups. For game state manager frame width and height attributes are unnecessary and they are removed. We couldn’t completely implement the game manager. For now we only made the game with only the normal difficulty We removed the screen manager class from our implementation and made changes to the actionlisteners of the view subsystem. The current attributes present in the classes of the model subsystem were insufficient for handing collisions and deciding where the entity would move. To overcome this problem we introduced new variables to classes in the model subsytem (e.g. dx and dy for movement direction, xtemp and ytemp for handling collision and deciding the entities next position values).

**4.3. Status of Implementation**

**To do parts for the second iteration:**

Sound Manager (musics and sound effects)

Input Manager

Power ups

Keyboard listener

Color system of the game, it is declared in the system, however collision manager does not detects the color now.

Types of enemies, their classes are not implemented.

Easy and hard options of the game is not implemented yet. For now one level is designed. Several levels will be implemented. For hard gameplay option a jump phobia system will be developed, however it is not considered for first iteration.

Other classes are implemented. However the gameplay cannot be accomplished yet. For now the menu dynamics is accomplished, and a player can be seen in the first level, however key interactions are not accomplished.

**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.