IMPLEMENTATION OF SOKOBAN GAME

Iraz ŞANLI

Ghada Alsayed

1. INTRODUCTION

The purpose of the project was to produce and develop a framework with an application connected to it. For our work, the framework was designed to be patternless for the best possible dynamic functionality. The requirements were clear for implementation to Sokoban. The application should put the necessary data in the different data structures to verify that the framework had something to work with. The application design was set to a MVC pattern. When the framework is implemented with the application, the framework will adapt to the given application pattern. The programmer does not need to use all the parts of the framework. The framework will adapt its own pattern to the applications given pattern. The framework handles image reading, keyboard events to an JFrame. During the implementation there were several problems encountered such as how to handle the player movements. It was resolved for this game but, for the framework, the process is still being progressed. We hope the framework together with the other applications works properly and it will be simple to continue developing the framework.

This report was written according to our plans that will be completed.

2. FRAMEWORK

a. Framework Requirements

Unlike the application requirement all the nonlogical and not game specified parts of the code should be in the framework to prepare a relevant draft without ay modifications of the framework when any programmer started coding.

b. Framework Design

The design of the framework is patternless. A framework is for use usually packaged in a way that makes creating an application much easier. When the framework is patternless it allows the application programmer to choose what pattern design to have on the application. This will then be heritaged to the framework so the framework can be a part of many design patterns for a given application. If the framework would have had a pattern the application would have been forced to follow the same pattern. Therefore the framework will now follow the same pattern as the application.

On this perspective, the user interface and business logic must be separate from each other. In other words, during the design and development of screens in front of the users, analysis should be made with the user's eye and should be developed by designing the most useful (user friendly) screen. In the back, business logic must be observed and the user must be connected to this logic from their interfaces.

This framework consists of three abstract classes, Model, View and Controller.

The abstract framework class Model is the heart of the framework. The Model class is supposed to help the application programmer to develop the core of the game.

The abstract framework class View extends JPanel and enables the drawing and painting of images.

The abstract framework class Controller implements the interfaces KeyListener. This part of the framework tells the application programmer the end user actions without having to implement a listener for the events.

3. APPLICATION

a. Application Requirements

In order to have a fully functioning logical puzzle application the programmer need to implement the given methods inside the framework class Model. All the logic and the specific things about the game need to be in the application code, e.g. which images or animations the framework should paint, what action should be performed after a button is pressed and so on. The painting itself is made in the framework but the application programmer need to tell the framework what image to paint at what coordinates in the application code. The same goes for the audio. The application programmer need to tell the framework what song to play. All the data structures that hold e.g. object coordinates and object values are created and initialized in the application code.

b. Application Design

The application is designed according to the MVC pattern which works in a pretty simple way but can be hard to implement. By designing the application in a MVC pattern we increase the efficiency of the code but this only applies for larger applications which games usually are. The MVC pattern consists of three classes, Actor, Game and GameScreen.

Before the definitions, we should add that we could not achieve our idea about Controller completely. In our framework we had some problems that are about providing access to methods. Therefore, we had to present our game in a different way from this idea. In our Sokoban, GameScreen class includes main method which calls the Game class that has TAdapter within it. The main idea was that the Main method will be located in a separate .java file that is called Run and this class will call the GameScreen class which is as a intermediator for the other classes.

The GameScreen extends Game and contains all methods associated with end user actions. It manipulates the values inside Model through View. The Controller have an inner class called TAdapter which verifies that the real time reading of the end user input is read all the time. It extends the abstract framework class Controller which provides the functionality of reading events.

The Game extends Actor which enables Game to read all new coordinates and data correctly. The Game contains the GUI that the end user sees. Inside Game there is an inner class called Map which extends the abstract framework class View. Map initiates the call to View to confirm the repainting of objects for every move done.

The Actor extends the abstract framework class Model. It contains all the data structures needed for the given application. The class verify that the input from the user read from

GameScreen is updated within the Game class. Actor is the main point of the MVC pattern. All of this verify the data within the project is updated with the real time information that the application needs.

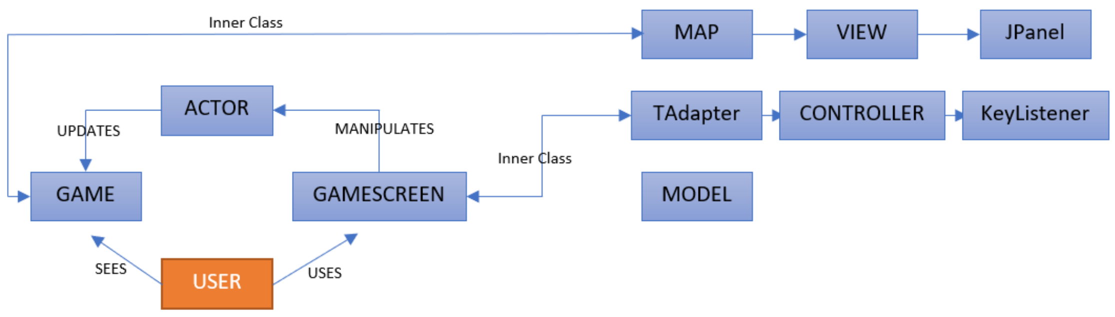
4. GAME DEBUGGING

The game-play testing was made in the same fashion as the code debugging. In these scenarios the actual GUI[6] part was involved. One example of where the console debugging[2] was useful was in the player movement methods.

5. INTERESTING PARTS

The reading of image files happens in an efficient way. The method readImages reads a given number of images. The image names are stored in the arraylist getImageName. They are then read as images into another arraylist called picList, where they can be used.

The figure shows the connection.



To be able to work a lot of information has been read and variety works have been reviewed from different sources.

6. RESULTS

The result of the framework together with the application came out as unexpected. Some parts could have been improved.

The planned framework is easy to modify, add new classes and functionality because of the approach of having it patternless.