The Dodge Game design and modifiability

Author : Boitumelo Mahlong
Prepared for : Joshua Lewis
Date : 30 April 2014

Abstract

The paper discusses the design and implementation of The Dodge Game explaining its flexibility to modification. The objective of the expected implementation is to allow a player to dodge rain drops by moving to the left or right controlled by the keyboard. The paper critiques and discusses design and modifiability of The Dodge Game in Java. The paper will explain and discuss key objects and classes that collaborate to provide the game functionality. Modifiability of the game is discussed through an evaluation of specific future change scenarios and how the game implements the changes. The Dodge Game will meet new requirements by making minimal existing code changes, in some case, and adding new code for new functionality.

1. Introduction

The Dodge Game (the game) where a player is to move left or right dodging rain drops is implemented. The paper discusses and critiques the game design and modifiability. The game implementation is extensible and modifiable without changing a lot of or with minimal changes to the existing code but by adding new code. The paper explains the game design, it further explains responsibility of each class and how it contributes to the game functionality.

2. Game Design

The design uses the separation of concerns principle. Classes a build in a modular fashion, keeping responsible for a single role. The game follows a simple strategy of using a controller class, ***DodgeController.java (the controller)***, that coordinates game classes and objects to provide the game functionality. The controller's role is to run the game.

The controller uses ***DodgeGUI.java*** to create and manage the game user interface(UI). The UI uses the inner class ***KeyEventManager.java*** to listen to keyboard events which interprets events and processes them accordingly.

AbstractSprite.java provides default functionality for all game characters extended by ***DodgePlayer.java*** to enable game player character and ***FallingObject.java*** enabling the rain functionality. The game state is held by th class ***DodgeConfig.java***

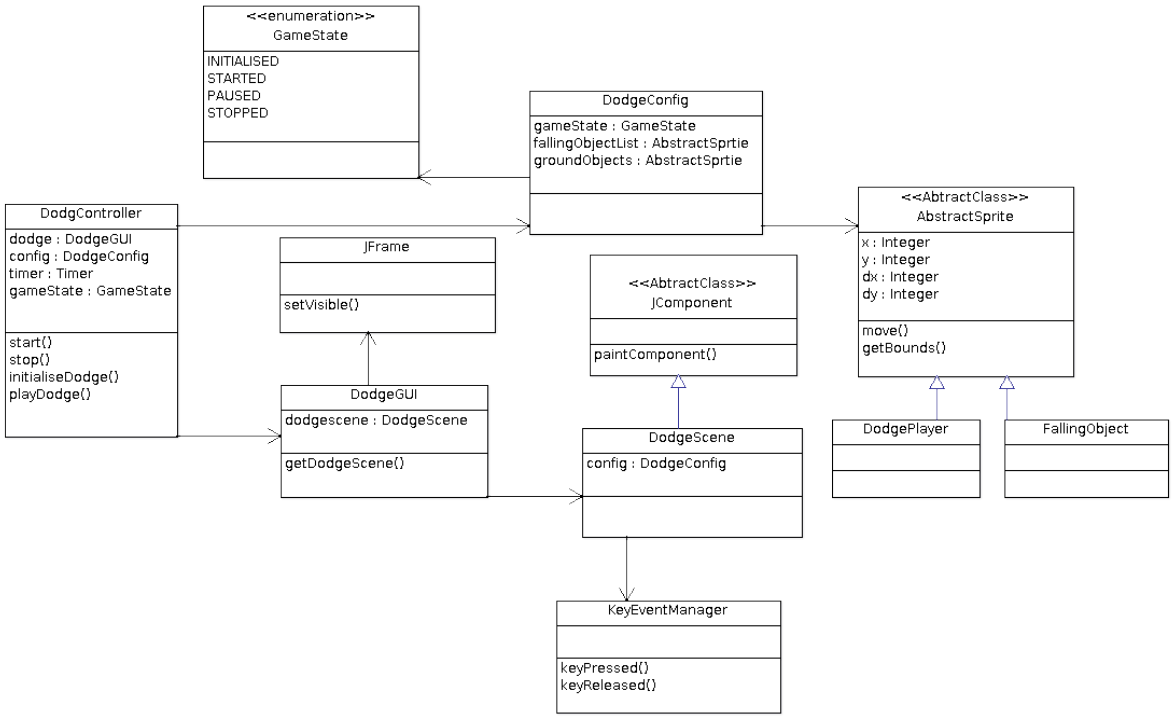

Figure 1. The Dodge Game highlevel design class diagram showing the relationship between the classes

2.1 Key Classes and Objects

2.1.1 DodgeController.java

The class coordinates all game classes to work together in providing the game functionality. The class initialises, starts and stops the game based on the users instruction or if the game ends. This is the entry point to the game functionality.

The controller employs *DodgeGUI.java* to initialise the game by creating the user interface (UI) and display the game window.

The controller uses java.swing.Timer (Timer) to start the game which also help provide the animation functionality. The class fires action events at specified intervals allowing the controller which implements the action listener interface to respond through the `actionPerformed(ActionEvent e)` method.

The `playDodge()` method is called for each action event fired by the Timer thus making the game playable by calling helper methods to change game character properties and repainting them on the UI. The repetition of this functionality enables game animation[1] functionality.

2.1.2 DodgeGUI.java

The class creates the UI of the game using DodgeScene.java a class extending JComponent abstracts class and the JFrame class object [2-3] of the Java Swing API[1]. An instance of DodgeScene.java (the container) acts as a container to all game characters. The objects loads and paints the object to UI. JFrame (the window) acts as the main window of the game, providing the title bar and borders on the game. The container is thus added to the window where the combination constitutes the UI presented by the DodgeGUI.java as shown in figure 2.

2.1.2.1 DodgeScene.java

The class creates the game scene by loading and painting game characters. The core responsibility of the class is managing the game scene by repainting the game characters for each action event triggered by the Timer. The method `paintComponent(Graphics g)` is overridden to allow custom drawing of game characters and is indirectly called through the repaint method called in the controller

Figure 2. The Dodge Game UI depicting the game after initialisation

1 Java Swing API provides all components for building Graphical User Interface application

2.1.3 AbstractSprite.java

The Abstract class [5] is a super class for all game characters, it represents and holds all their common functionality. The class provides the gamer the ability to to interact (play the game) with the game characters. The class is extended by DodgePlayer.java and FallingObject.java.

2.1.3.1 DodgePlayer.java

The class implements the player functionality. With the inherited common functionality, the class will add all player specific functionality.

2.1.3.2 FallingObject.java

The class implements the rainfall functionality. The rain droplets will fall vertically down from the top of the screen by manipulating the class's inherited common properties.

2.1.4 KeyEventManager.java

The inner class manages all game keyboard events, it links the gamer with the game making the game playable. The class gives life to the game characters by listening to KeyEvents[2] and processing each key accordingly. The class facilitates the updating of game character properties specifically game player and overall state of the game. The class is added as an key listener [6-7] for the container

The methods keyPressed(KeyEvent e) and keyReleased(KeyEvent e) are invoked every-time the gamer makes the action where further process is done based on the action taken.

2.1.5 DodgeConfig.java

The class keeps the game state during the entire session when the game is played. The class is used across the entire game carrying information and making it available where it is required.

3. Game Modifiability

The game design and implementation is flexible and open to extension of the following game functionality. All the modifications are done with minimal existing code changes.

The design employs the use of Object Oriented (OO) concepts. The use on these concepts allows the application designed to be implemented in a modular, reusable and extensible way, exhibiting qualities of a good software design.

3.1 Evaluation of scenarios for future changes

3.2.1 Addition of new game elements

Additional falling objects in a form of Umbrellas, will be added to the game by adding instances of FallingObject.java using umbrella images.

Additional dodge players are added by adding instances of DodgePlayer.java class also with different player images

The change will introduce an upgrade to the Abstract class in a form of object identification (OID). OID will be used to identify falling objects which will help in the game collision detection [8] method where an umbrella colliding with a player will not end the game but increase the player life. The OID will also help in making sure player objects are identified and paired with the caught umbrella.

3.2.2 Configurable player key controls

Player key controls are controlled in GameConfig.java, new KeyEvents will be added to the ground object list which will be automatically picked up in the controller. The KeyEventManager.java will also be changed to effect the processing of the new KeyEvents.

2 KeyEvents Is an object representing a keyboard event invoked

The change will extend the existing KeyEventManager.java events processing functionality.

3.2.3 The game scoring functionality

Game scoring will be added by including a new key value map list in the DodgeConfig.java class. The map will be of a player key with the score object/value. The list will be used to persist the score at the end of the game.

The change will upgrade the DodgeConfig.java design in that it might change to implement serialization[3] which will help further in storing the game state. This creates an opportunity for more extension to the current game design e.g. Pausing the game, saving the current state, closing it only to resume later.

3.2.5 Adopting game functionality for another version

A new version of the game explained in Example 1 will be achieved by creating a new class extending the JComponent, HunterScene.java. The class will replace DodgeScene.java to provide the new version of the game and will necessitate changes to the controller. The controller will be upgraded to take a parameter in the constructor which will be the JComponent. Depending on which version is in required the relevant JComponent extending class will be passed.

3.2.6 Using 3D Library

Using a 3D library will introduces a lot of changes to the current game design. The design will have to be planned from start taking into account the constrains of the 3D library. Most of the current game classes will be affected.

3.2.7 Use of a different graphics library

To replace the current graphic library, changes will be applied to the DodgeGUI.java and DodgeScene.java class**.** The classes will change from using JFrame and JComponent to components providing similar functionality in the new library.

Graphics being core to game functionality, the change will introduce a change to the current design which will affect most key classes in how they handle graphics. The change will extend the current design to have an elegant solution for game graphics by introducing a common interface to be implemented by all UI creating classes for the game.

3.2.8 Running the game on Linux Operating System

Java as the language of choice for the game implementation is portable and runs on different operating systems (OS) including Linux. The game will run on any operating systems that support the Java Virtual Machine. Recently other languages can also be ported on different OS but Java was build for portability thus the change to run the dodge game on a different OS comes naturally.

4. Conclusion

The game design choice was basic and simple for the scope of the game requirements. The simple design kept to known game patterns of using a controller that initialises, start, play and stops the game.

The extended abstract classes allow for the game to have polymorphic behavior. This behaviour enables the game to adapt different version using the same core functionality of dodging falling objects.

FallingObject.java and DodgePlayer.java were reused without changing a single line of code in them. The game functionality is thus extended by adding new classes or code instead of replacing existing functionality.

3 Serialization is an interface of the Java API that allows implementing classes to be persisted

References

[1] How to Use Swing Timers
http://docs.oracle.com/javase/tutorial/uiswing/misc/timer.html
Last Accessed: May 5, 2014

[2] Basic Race with Animation

http://www.java2s.com/Code/Java/Swing-Components/BasicRacewithAnimation.htm
Last Accessed: May 5, 2014

[3] How to Make Frames (Main Windows)
http://docs.oracle.com/javase/tutorial/uiswing/components/frame.html
Last Accessed: May 5, 2014

[4] The JComponent Class
http://docs.oracle.com/javase/tutorial/uiswing/components/jcomponent.html
Last Accessed: May 5, 2014

[5] Abstract Methods and Classes
http://docs.oracle.com/javase/tutorial/java/IandI/abstract.html
Last Accessed: May 6, 2014

[6] Java games moving sprites
http://zetcode.com/tutorials/javagamestutorial/movingsprites/
Last Accessed: May 6, 2014

[7] How to write a Key Listener
http://docs.oracle.com/javase/tutorial/uiswing/events/keylistener.html
Last Accessed: May 6, 2014