The Dodge Game design and modifiability

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

Abstract

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. The use of interfaces and abstract classes to allow modifiability is discussed and applied to allow for future game functionality extension. The Dodge Game will meet new requirements by making minimal existing code changes and adding new code.

1. Introduction

The design uses the separation of concerns principle. Each class's role is concerned with one functionality and that functionality alone. The game follows a simple strategy in using a controller class, ***DodgeController.java (the controller)***, that coordinates game objects and classes to provide functionality. The controller's role is to run the game.

The game uses ***DodgeGUI.java*** to manage and create the game user interface. The user interface uses the ***KeyEventManager.java*** to listen to keyboard events. The class 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 whole game state is carried by a single plain old java object (POJO) in ***DodgeConfig.java***

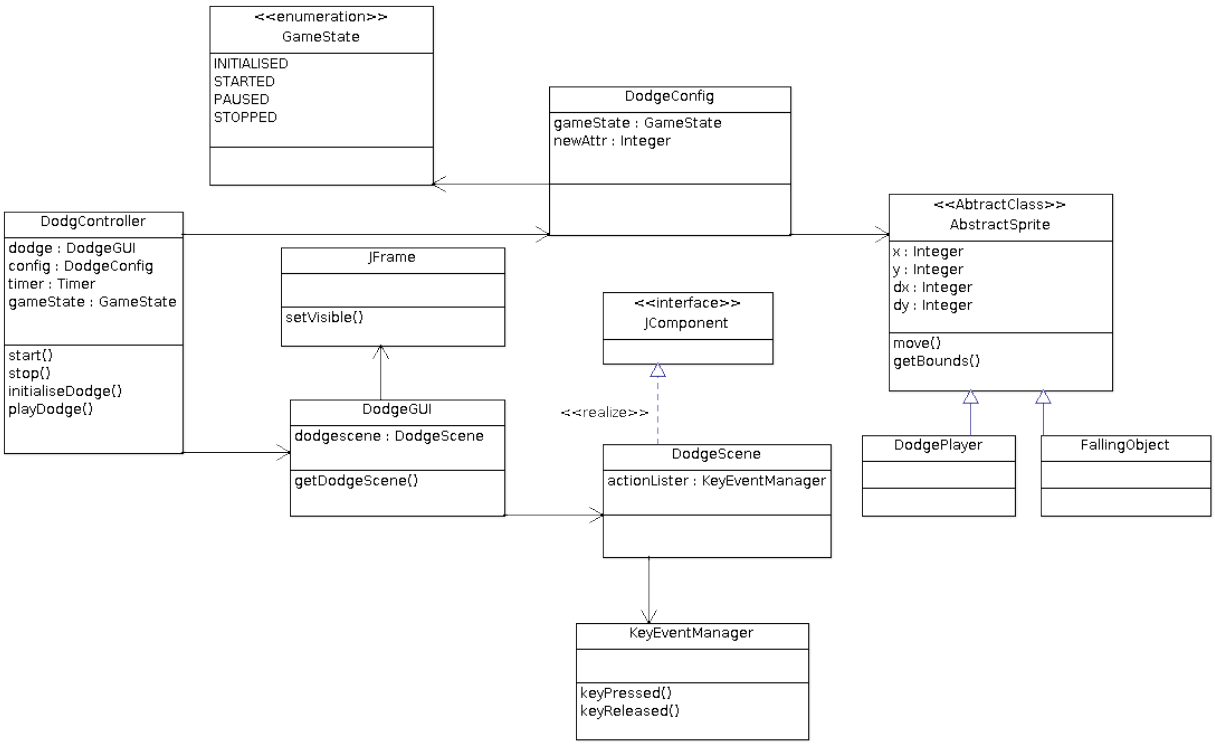
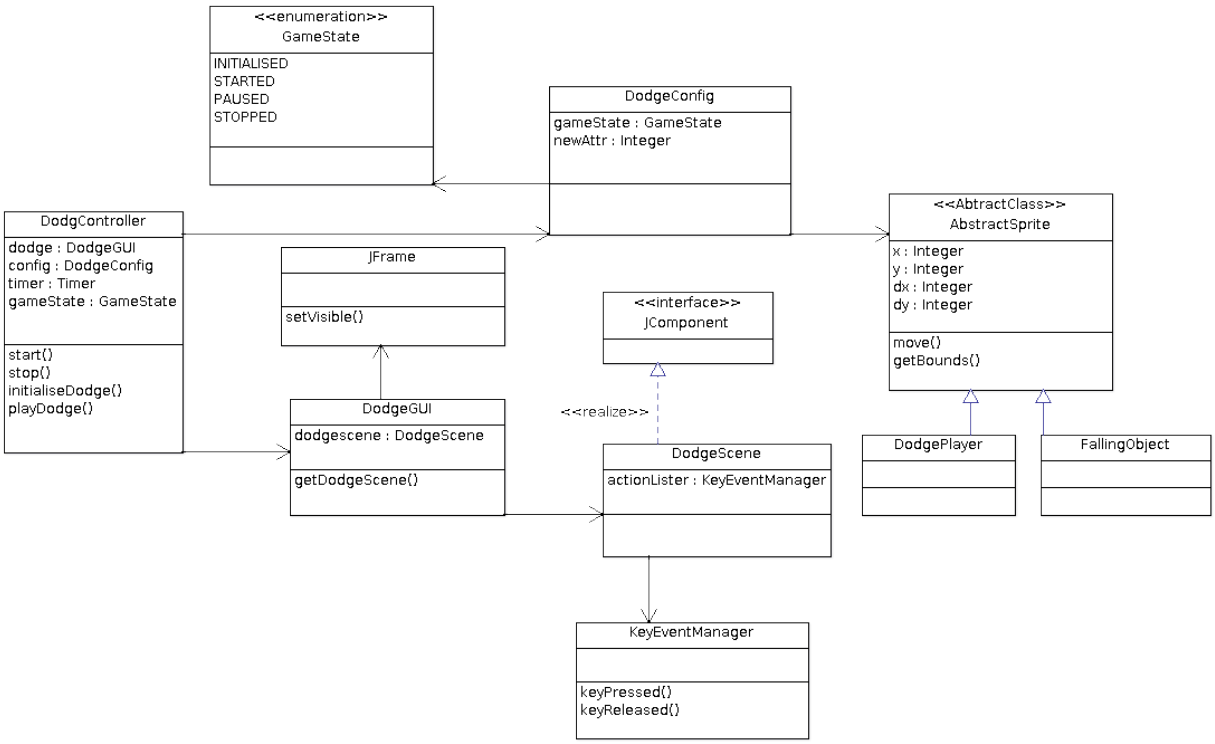


Figure 1. The Dodge Game highlevel design class diagram

Using the same example in section 2.2.1, hunter game arrows will be instances of the class, FallingObject.java which extends AbstractSprite.java, used to represent rain drops. The difference will be the image representing the arrow. Adding to the example, the hunter arrows changes to pointed stones which rotate down the hill. The AbstractSprite.java is extended by a new class representing the pointed stones, PointedStone.java. The class add the rolling stone functionality thus adding on to the already inherited behaviour from the super class.

The pointed stone class is still usable withing the game without the need to change any of the existing game classes.

2.1 Core 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 implementing JComponent interface [] and the JFrame class object. 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

2.1.3 AbstractSprite.java

The Abstract class [] 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 class manages all game key events, it links the gamer with the game making the game playable. The class gives life to the game characters by listening to key-event and processing each key accordingly. The class facilitates the updating of game characters specifically game player and overall state of the game. The class is added as an actionListener [] for the container

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

2.2 Applied Object Oriented Concepts

The design employs the use of the following 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.

2.2.1 Interfaces

Allows contracts to be set for implementing classes to exhibit certain expected behavior. In the case of the game DodgeScene.java implements the JComponent interface of SWING /AWT API. The API is used for building UI applications and thus help implementation of the UI functionality of the game.

Example 1. Hunter Game

A new game sharing the exact dodging functionality can be added by adding a new JComponent implementing class that will add different characters. The scene can be easily changed to a hunter game where arrows shot at the top, replacing rain drops, of the hill down to a Duck, representing the player, in the river below.

The use of interfaces allows the game to be open for extension without changing exiting functionality. In our example, DodgeScene.java will be replaced by the new hunter game scene class, HunterScene.java. The DodgeGUI.java will be able to use the new class without any modification.

2.2.2 Abstract Classes

Using the same example in section 2.2.1, hunter game arrows will be instances of the class, FallingObject.java which extends AbstractSprite.java, used to represent rain drops. The difference will be the image representing the arrow.

Adding to the example, the hunter arrows changes to pointed stones which rotate down the hill. The AbstractSprite.java is extended by a new class representing the pointed stones, PointedStone.java. The class add the rolling stone functionality thus adding on to the already inherited behaviour from the super class.

The pointed stone class is still usable withing the game without the need to change any of the existing game classes. The class is implemented as a KeyAdapter to avoid adding implementation for method that are not required [] .

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. Flexibility to expected future changes

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.

3.1 Addition of new game elements

Additional falling objects in a form of Umbrella, 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

3.2 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

3.4 The game scoring functionality

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

3.5 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 be the only change to existing functionality.

Using 3D Library

Use of a different graphics library

To replace the current graphic library, changes will be applied to the DodgeGUI.java class

Running the game on Linux Operating System

4. Conclusion

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

The choice of Interfaces was to 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.

Abstract classes allows re-usability, FallingObject.java and DodPlayer.java were reused without changing a single line of code in them. The abstract classes are also extended to add more functionality. The game's functionality is thus extended by adding new classes or code instead of replacing existing functionality.

References

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

[2] Java games collisions detection
http://zetcode.com/tutorials/javagamestutorial/collision/
Last Accessed: May 5, 2014

[3] Game Engine and Framework in Java
http://www.ntu.edu.sg/home/ehchua/programming/java/J8d_Game_Framework.html
Last Modified: September 4, 2008

[4] 2D Graphics & Java 2D
http://www3.ntu.edu.sg/home/ehchua/programming/java/J8b_Game_2DGraphics.html
Last Modified: April, 2012