Platform 51: Description of Design

By: Matt Gottsacker, Matt Gannon, Will Higgins

Overview:

Platform 51 is a platform-based game that follows the character BB-8 and its adventures to find Rey, who is the Resistance’s last hope. The user controls the movement of BB-8 with the left and right arrow keys, presses the spacebar to jump, and presses the ‘S’ key to shoot lasers. The lasers are necessary to eliminate the Tie Fighters that fly toward and can kill BB-8. BB-8 gets three lives in addition to the one he starts with before he has to start the game over. There are three levels to the game, and each increases in difficulty using a process that will be described later. The positions and widths of the platforms are randomly generated at the start of every level. The last platform in each level is colored bright yellow, and it marks the end of the level. Upon completion of a level, the user is taken to a frame with a prompt asking whether the user wants to continue or quit. Upon completion of the third level, a frame appears congratulating them on their victory and asking if they would like to play again.

Class breakdown (in alphabetical order):

Bullet: A Bullet is one laser shot that is created when the user presses the ‘S’ key. This class controls how each bullet moves and its collisionBox, which is a rectangle that is used to detect when it collides with an Enemy. When the Protagonist fires a Bullet, it is added to an ArrayList of Bullets in the Protagonist class to give the Protagonist maximum control over its bullets.

Enemy: An Enemy is a Tie Fighter that attacks the Protagonist. This class controls how each Enemy moves and also checks if it collides with a Bullet. If it does, it removes the bullet from the Protagonist’s list of bullets. In levels 2 and 3, it has AI that gets the Protagonist’s location and moves toward it. The Enemy class’s diffY instance variable controls this process, and it is set upon creation of each level. Because the Enemy uses the Protagonist as a targeting point for levels 2 and 3, each Enemy takes the Protagonist as a constructor parameter.

GameFrame: This frame extends JFrame because it is responsible for displaying the game’s GamePanel. It is not resizable so that the user cannot alter its dimensions and potentially affect gameplay.

GameManager: This class is responsible for controlling all of the nonphysical elements of the game. Since it accesses all of the physical elements of the game created in a GamePanel, it takes a GamePanel as an argument. It initializes a Protagonist because the game protagonist’s main functionality is managed here (movement, updating collisions, etc.). The GameManager also checks if the Protagonist has died or has completed the level, and it displays a new JFrame with the proper messages. When this happens, the GameManager also updates non-physical entities like the game’s level, or resets the entire game if necessary. The GameManager controls the Protagonist’s movement by reading the active keyboard keys from the KeyboardController class. The GameManager is responsible for repainting the game. It does so by extending the Thread class and repainting its GamePanel (and therefore everything in it) every 10 milliseconds.

GamePanel: This class contains a PlayPanel and a StatsPanel and displays them. In its constructor, it instantiates a GameManager so that there is always a GameManager to manage its non-physical components. Because the GameManager takes a GamePanel as an argument and is responsible for managing its non-physical elements, GamePanel contains several methods to pass values from the GameManager to the PlayPanel or StatsPanel (i.e. accessing/changing the Protagonist’s lives, which is managed in the StatsPanel).

KeyboardController: This class is responsible for listening for the user’s key strokes and updating the HashSet of activeKeys accordingly. We chose to use a HashSet to manage the data because we found that it has the fastest operation time for the contains() method we use to track the keys pressed by the user.

Level: This class is responsible for generating the platforms at the beginning of every level and managing the set of Enemies. The algorithm for generating the platforms varies for each level by making the height gap between adjacent platforms greater. The later levels also have more platforms. This class generates the platforms at the beginning of each level so that the GameManager and PlayPanel classes are not bogged down with running another algorithm and creating new Platforms at runtime.

Main: This class displays the instructions to the user and then instantiates a new GameFrame when the user clicks the “Start” button. The instructions are only needed the first time the user starts the game and not at every retry, and they also do not impact any part of the GameFrame or its components, so that is why the instruction frame is created at the program’s initial runtime.

Platform: This class controls the position where each platform starts and how each platform moves. It also contains a collisionBox, which is used to detect when the Protagonist collides with each Platform.

PlayPanel: This class is responsible for all of the physical aspects of the game. This class extends JPanel because it contains every gameplay item that is repainted by the GameManager. Its constructor instantiates a Level so that it can display the platforms and so that the Protagonist can easily interact with them. At every repaint, the PlayPanel paints the background image, the Protagonist, his Bullets, the Enemies, and the Platforms in their respective updated locations. This panel moves the Platforms, the Bullets, and the Enemies because their movements are independent of the Protagonist’s.

Protagonist: This class manages everything about the Protagonist. In its constructor, it first loads the images of its rolling animation. Then, it instantiates its collisionBox and other points in relation to the Protagonist that are used to track specific collisions. This class defines how the Protagonist moves and jumps, and it updates the location of its collisionBox every time. This class manages what happens when the Protagonist collides with the moving Platforms, and it handles each specific collision case (i.e. colliding from one of the sides, landing on a Platform, jumping and hitting a head on a Platform, etc.). It also checks if the Protagonist dies by colliding with an Enemy, or falling off screen. Protagonist also contains an instance variable reference to the current Level in order for the GameManager to control the Protagonist’s state in different situations.

StatsPanel: The StatsPanel contains the Protagonist’s lives left and its current level. This panel is also redrawn when the GamePanel is repainted by the GameManager so that it stays updated.

PATTERNS

Observer patterns: KeyboardListener observes the keys that the user presses and communicates them to the GameManager.

ActionListeners on the buttons in the Main and in the frames created by level advancing/dying.

The PlayPanel observes the current positions of all of the objects on the screen and draws their locations.

The StatsPanel observes the Protagonist’s current level and lives and displays them to the user.

The Protagonist and Enemy classes observe when they come into contact with something else and determine the proper course of action after they collide.

The GameManager observes the state of the Protagonist and the GamePanel and determines what to display or change based on that information.

Encapsulation: We encapsulated most of the classes well. If we had more time, we could encapsulate the classes even more. A lot of model Java game tutorials used a lot of static variables, and we tried to shy away from that when possible.

Strategy pattern: We used this in almost every frame we created to manipulate how the information should be displayed to the end user.

Iterator pattern: We iterated through the lists of Platforms, Enemies, and Bullets in the program. While we do not use a common iterator in our program, it is something we could look to implement going forward.

EXTRA CREDIT

Random generation: The platforms are randomly generated but kept within jumping distance at the start of each level.

Images for characters/objects: Used 8-bit gifs for the BB-8 character, its lasers, the Tie Fighters, and the background image.

AI for computer-controlled creatures: The Tie Fighters move toward their target’s position in levels 2 and 3. They move toward their target faster in level 3.

Multiple levels: There are 3 levels with different platform generators and enemy types.

Keyboard shortcuts: They game is controlled using the user’s keyboard.