

**Course Experiment Report**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Course:** | Java Language | | | | | | |
|  |  | | | | | | |
| **Semester:** | 1-18th | **week** | 2nd | **year** | | 1st | **term** |
|  |  |  |  |  | |  |  |
| **Major:** | Software Engineering | | | | | **Class:** | 2019 |
|  |  | | | | |  |  |
| **Student name:** | Fu Ruoxuan | | **Student No.:** | | 222019321062060 | | |
|  |  | |  | |  | | |
| **Teacher:** | Wang Xiaomeng | | | | | | |

College of Computer and Information Science

|  |  |  |  |
| --- | --- | --- | --- |
| Project | Exp7 GUI | | |
| Time | 12.23 | Type | □Verification □Design ☑Synthetical |
| 1. Answer the questions  (1) What else can be improved in the program?  ① When the person pushes the box to the border, it should not prompt "hahah" but should prompt that the player can no longer go in the direction of the border.  ② When the game can no longer be played, the player should be prompted that the game has been lost.  ③ When you click next at level2, the program will report an array out-of-bounds exception, which should catch the exception and then indicate that the current level is already the highest level.  (2) How to develop a map editing program for this game?  In summary, to create a new map you need to create a new file with the .map extension in the maps folder. In the first line, write the map's row size. In the second and subsequent lines, write the row value and type of each element in the form of "row, column, type". Create a new two-dimensional array of Imageview objects using the row size read into the first row. Create an ArrayList to store the stars, and then create the corresponding objects according to the types of the elements read in. If not, fill the object into the corresponding row value position in the Imageview object array.  2. All Codes  Elements.java   1. **package** sokoban; 3. **import** javafx.scene.image.Image; 4. **import** javafx.scene.image.ImageView; 6. **public** **abstract** **class** Element **extends** ImageView { 7. // Image element types 8. **public** **final** **static** **int** MAN = 0; 9. **public** **final** **static** **int** BOX = 1; 10. **public** **final** **static** **int** WALL = 2; 11. **public** **final** **static** **int** TARGET = 3; 12. **public** **final** **static** **int** BACKGROUND = 4; 13. **protected** Element(Image img) { 14. **super**(img); 15. // Set the display size 16. **this**.setFitHeight(MapPane.CELL\_SIZE); 17. **this**.setFitWidth(MapPane.CELL\_SIZE); 18. } 19. }   Box.java   1. **package** sokoban; 3. **import** javafx.scene.image.Image; 5. **public** **class** Box **extends** MovingElement { 6. **public** Box(Image img) { 7. **super**(img); 8. } 9. }   Man.java   1. **package** sokoban; 3. **import** javafx.scene.image.Image; 5. **public** **class** Man **extends** MovingElement { 6. **public** Man(Image manImg) { 7. **super**(manImg); 8. } 9. }   Target.java   1. **package** sokoban; 3. **import** javafx.scene.image.Image; 5. **public** **class** Wall **extends** Element { 6. **public** Wall(Image img) { 7. **super**(img); 8. } 9. }   Wall.java   1. **package** sokoban; 3. **import** javafx.scene.image.Image; 5. **public** **class** Wall **extends** Element { 6. **public** Wall(Image img) { 7. **super**(img); 8. } 9. }   MovingElement.java   1. **package** sokoban; 3. **import** javafx.scene.image.Image; 5. **public** **class** MovingElement **extends** Element { 6. **protected** MovingElement(Image img) { 8. **super**(img); 9. } 10. **public** **void** left() { 11. **this**.setX(**this**.getX() - MapPane.CELL\_SIZE); 12. } 13. **public** **void** right() { 14. **this**.setX(**this**.getX() + MapPane.CELL\_SIZE); 15. } 16. **public** **void** up() { 17. **this**.setY(**this**.getY() - MapPane.CELL\_SIZE); 18. } 19. **public** **void** down() { 20. **this**.setY(**this**.getY() + MapPane.CELL\_SIZE); 21. } 22. }   MapPane.java   1. **package** sokoban; 3. **import** com.sun.glass.ui.Screen; 4. **import** javafx.scene.image.Image; 5. **import** javafx.scene.image.ImageView; 6. **import** javafx.scene.layout.\*; 8. **import** java.io.File; 9. **import** java.io.IOException; 10. **import** java.util.ArrayList; 11. **import** java.util.List; 12. **import** java.util.Scanner; 14. **public** **class** MapPane **extends** Pane { 15. // The cell size 16. **public** **final** **static** **int** CELL\_SIZE = 64; 17. // All images for element displaying 18. **private** Image[] icons; 19. // Logical storage space for all elements 20. **private** ImageView[][] map; 21. // Controlled object、 22. **private** Man man; 23. // Mission points 24. **private** List<ImageView> targets; 25. // Number of cells 26. **private** **int** xlength; 27. **private** **int** ylength;  30. // Constructor. 31. **public** MapPane(Image[] iconList, File mapFile) { 32. icons = iconList; 33. // Load map 34. **this**.loadMap(mapFile); 35. // Set background 36. Background bg = **new** Background( 37. **new** BackgroundImage( 38. iconList[Element.BACKGROUND], 39. BackgroundRepeat.REPEAT, 40. BackgroundRepeat.REPEAT, 41. BackgroundPosition.DEFAULT, 42. **new** BackgroundSize( 43. BackgroundSize.AUTO, 44. BackgroundSize.AUTO, 45. **true**, **true**, **false**, **false**) 46. ) 47. ); 48. setBackground(bg); 49. } 50. // Clear old data and load new map data 51. **public** **void** loadMap(File mapFile) { 52. **try**(Scanner input = **new** Scanner(mapFile)){ 53. // Get length information from the first line in the map file 54. String[] items = input.nextLine().split(","); 55. **int** xlen = Integer.parseInt(items[0]); 56. **int** ylen = Integer.parseInt(items[1]); 57. xlength = xlen; 58. ylength = ylen; 59. // Initialize and clear the map pane 60. **this**.map = **new** ImageView[xlen][ylen]; 61. targets = **new** ArrayList<ImageView>(); 62. **this**.getChildren().clear(); 63. // Read file and add elements to the map pane 64. **while**(input.hasNextLine()) { 65. // Get information of a element from string parsing 66. items = input.nextLine().split(","); 67. **int** x = Integer.parseInt(items[0]); 68. **int** y = Integer.parseInt(items[1]); 69. **int** type = Integer.parseInt(items[2]); 70. // Create an element 71. Element e = **null**; 72. **switch**(type) { 73. **case** Element.MAN: e = **this**.man = **new** Man(icons[Element.MAN]); 74. **break**; 75. **case** Element.BOX: e = **new** Box(icons[Element.BOX]); 76. **break**; 77. **case** Element.WALL: e = **new** Wall(icons[Element.WALL]); 78. **break**; 79. **case** Element.TARGET: e = **new** Target(icons[Element.TARGET]); 80. **break**; 81. **default**:**break**; 82. } 83. // Set the position and display the element 84. e.setX(x \* CELL\_SIZE); 85. e.setY(y \* CELL\_SIZE); 86. getChildren().add(e); 87. // Add element to the logical map 88. **if**(e **instanceof** Target) { 89. targets.add(e); 90. }**else** { 91. **this**.map[x][y] = e; 92. } 93. } 94. }**catch**(IOException e) { 95. e.printStackTrace(); 96. } 97. } 98. // Judge the state of the game 99. **public** **boolean** judge() { 100. **boolean** win = **true**; 101. ImageView img; 102. **for** (ImageView target : targets) { 103. img = target; 104. **int** x = (**int**) (img.getX() / CELL\_SIZE); 105. **int** y = (**int**) (img.getY() / CELL\_SIZE); 106. **if** (!(map[x][y] **instanceof** Box) ) { 108. win = **false**; 109. **break**; 110. } 111. } 112. **return** win; 113. } 114. // How to move the man and boxes in the upward direction 115. **public** **void** moveManUp(){ 116. **int** manX = (**int**) (man.getX() / CELL\_SIZE); 117. **int** manY = (**int**) (man.getY() / CELL\_SIZE); 118. **if**(manY > 0) { 119. **if**(map[manX][manY - 1] == **null**) { 120. man.up(); 121. map[manX][manY] = **null**; 122. map[manX][manY - 1] = man; 123. }**else** **if**(map[manX][manY - 1] **instanceof** Box) { 124. System.out.println("hahah"); 125. **if**(manY - 1 > 0 && map[manX][manY - 2] == **null**) { 126. Box b = (Box) map[manX][manY - 1]; 127. b.up(); 128. map[manX][manY - 2] = b; 129. man.up(); 130. map[manX][manY] = **null**; 131. map[manX][manY - 1] = man; 132. } 133. } 134. } 135. } 136. // How to move the man and boxes in the downward direction 137. **public** **void** moveManDown() { 138. **int** manX = (**int**) (man.getX() / CELL\_SIZE); 139. **int** manY = (**int**) (man.getY() / CELL\_SIZE); 140. **if**(manY < ylength-1) { 141. **if**(map[manX][manY + 1] == **null**) { 142. man.down(); 143. map[manX][manY] = **null**; 144. map[manX][manY + 1] = man; 145. }**else** **if**(map[manX][manY + 1] **instanceof** Box) { 146. System.out.println("hahah"); 147. **if**(manY + 2 < ylength && map[manX][manY + 2] == **null**) { 148. Box b = (Box) map[manX][manY + 1]; 149. b.down(); 150. map[manX][manY + 2] = b; 151. man.down(); 152. map[manX][manY] = **null**; 153. map[manX][manY + 1] = man; 154. } 155. } 156. } 157. } 158. // How to move the man and boxes in the leftward direction 159. **public** **void** moveManLeft(){ 160. **int** manX = (**int**) (man.getX() / CELL\_SIZE); 161. **int** manY = (**int**) (man.getY() / CELL\_SIZE); 162. **if**(manX > 0) { 163. **if**(map[manX - 1][manY] == **null**) { 164. man.left(); 165. map[manX][manY] = **null**; 166. map[manX - 1][manY] = man; 167. }**else** **if**(map[manX - 1][manY] **instanceof** Box) { 168. System.out.println("hahah"); 169. **if**(manX - 1 > 0 && map[manX - 2][manY] == **null**) { 170. Box b = (Box) map[manX - 1][manY]; 171. b.left(); 172. map[manX - 2][manY] = b; 173. man.left(); 174. map[manX][manY] = **null**; 175. map[manX - 1][manY] = man; 176. } 177. } 178. } 179. } 180. // How to move the man and boxes in the rightward direction 181. **public** **void** moveManRight() { 182. **int** manX = (**int**) (man.getX() / CELL\_SIZE); 183. **int** manY = (**int**) (man.getY() / CELL\_SIZE); 184. **if**(manX < xlength-1) { 185. **if**(map[manX + 1][manY] == **null**) { 186. man.right(); 187. map[manX][manY] = **null**; 188. map[manX + 1][manY] = man; 189. }**else** **if**(map[manX + 1][manY] **instanceof** Box) { 190. System.out.println("hahah"); 191. **if**(manX + 2 < xlength && map[manX + 2][manY] == **null**) { 192. Box b = (Box) map[manX + 1][manY]; 193. b.right(); 194. map[manX + 2][manY] = b; 195. man.right(); 196. map[manX][manY] = **null**; 197. map[manX + 1][manY] = man; 198. } 199. } 200. } 201. } 202. }   Sokoban.java   1. **package** sokoban; 3. **import** javafx.application.Application; 4. **import** javafx.scene.Scene; 5. **import** javafx.scene.control.Alert; 6. **import** javafx.scene.control.Button; 7. **import** javafx.scene.control.Label; 8. **import** javafx.scene.control.ToolBar; 9. **import** javafx.scene.image.Image; 10. **import** javafx.scene.layout.VBox; 11. **import** javafx.stage.Stage; 13. **import** java.io.File; 14. **import** java.util.ArrayList; 15. **import** java.util.List; 17. **public** **class** Sokoban **extends** Application { 18. // File root paths 19. **private** **final** String mapDir = "maps"; 20. **private** **final** String imgDir = "imgs"; 21. // All map files 22. **private** List<File> mapFiles; 23. // Current game level 24. **private** **int** currentLevel = 0; 25. // Current map 26. **private** MapPane currentMap = **null**; 27. // Load all map files in the application 28. **public** **void** loadMapFiles() { 29. mapFiles = **new** ArrayList<File>(); 30. File dir = **new** File(mapDir); 31. **for**(File f:dir.listFiles()) { 32. mapFiles.add(f); 33. } 34. } 35. **public** **void** start(Stage primaryStage) **throws** Exception { 36. loadMapFiles(); 37. VBox vb = **new** VBox(); 38. ToolBar tb = **new** ToolBar(); 39. Label label = **new** Label("Current Level:" + currentLevel); 40. // Click button for changing to prevoies level map 41. Button preBtn = **new** Button("Previous"); 42. preBtn.setOnAction(e -> { 43. **if**(currentLevel > 0) { 44. currentLevel--; 45. currentMap.loadMap(mapFiles.get(currentLevel)); 46. label.setText("Current Level:" + currentLevel); 47. primaryStage.sizeToScene(); 48. } 49. }); 50. // Click button for changing to next level map 51. Button nextBtn = **new** Button("Next"); 52. nextBtn.setOnAction(e -> { 53. **if** (currentLevel < 3){ 54. currentLevel++; 55. currentMap.loadMap(mapFiles.get(currentLevel)); 56. label.setText("Current Level:" + currentLevel); 57. primaryStage.sizeToScene(); 58. } 59. }); 60. // Click button for resetting this map 61. Button resetBtn = **new** Button("Reset"); 62. resetBtn.setOnAction(e -> { 63. currentLevel = 0; 64. currentMap.loadMap(mapFiles.get(currentLevel)); 65. label.setText("Current Level:" + currentLevel); 66. primaryStage.sizeToScene(); 67. }); 68. tb.getItems().addAll(label, preBtn, nextBtn, resetBtn); 69. // Add an alert dialog box 70. Alert a = **new** Alert(Alert.AlertType.CONFIRMATION); 71. a.setHeaderText("You have won"); 72. a.setContentText("Next level?"); 73. a.setOnCloseRequest(e -> { 74. **if**(currentLevel < mapFiles.size() - 1) { 75. currentLevel++; 76. currentMap.loadMap(mapFiles.get(currentLevel)); 77. label.setText("Current Level:" + currentLevel); 78. primaryStage.sizeToScene(); 79. } 80. }); 81. // Add a map pane and load the first level map data 82. Image[] icons = **new** Image[5]; 83. icons[Element.MAN] = **new** Image("file:" + imgDir + "/man.png"); 84. icons[Element.BOX] = **new** Image("file:" + imgDir + "/box.png"); 85. icons[Element.WALL] = **new** Image("file:" + imgDir + "/wall.png"); 86. icons[Element.TARGET] = **new** Image("file:" + imgDir + "/target.png"); 87. icons[Element.BACKGROUND] = **new** Image("file:" + imgDir + "/floor.png"); 88. currentMap = **new** MapPane(icons, mapFiles.get(currentLevel)); 89. vb.getChildren().addAll(tb, currentMap); 90. Scene scene = **new** Scene(vb); 91. scene.setOnKeyPressed(e -> { 92. **switch**(e.getCode()) { 93. **case** A: currentMap.moveManLeft(); 94. **break**; 95. **case** D: currentMap.moveManRight(); 96. **break**; 97. **case** W: currentMap.moveManUp(); 98. **break**; 99. **case** S: currentMap.moveManDown(); 100. **break**; 101. **default**: **break**; 102. } 103. **if**(currentMap.judge()) { 104. a.show(); 105. } 106. }); 107. primaryStage.sizeToScene(); 108. primaryStage.setResizable(**false**); 109. primaryStage.setScene(scene); 110. primaryStage.getIcons().add(**new** Image("file:" + imgDir + "/man.png")); 111. primaryStage.setTitle("Sokoban"); 112. primaryStage.show(); 113. } 114. **public** **static** **void** main(String[]args){ 115. launch(args); 116. } 117. } | | | |

|  |  |  |
| --- | --- | --- |
| Evaluation | Code Correctness (60%): |  |
| Experience (40%): |  |
| Score： | |