Cory W. Cordell &
Andrew Markley
February 19, 2015

Assignment IV
Frogger 

 Project Design

Classes and Methods

The project design was influenced by the books implementation of the animation program which contained the MoveableShape interface and CarShape and IconShape classes.

The frogger package contains the classes Main, Frogger, and Menu. The Main class has one method. The static main method is used only to generate an instantiation of the Frogger class.

The Menu class provides flexibility and playability in the program by allowing the user to select a difficulty level. The menu object uses an option dialog to display an explanation of five options and allow the user to choose between those five options. The menu relays the user's selection with a getReply method.

The Frogger class is the primary class of the program and instantiates the BufferedImage and ImageIcon used in the dialogs and in the game board. The Frogger class uses the generateBoard method as the primary driver method in which all objects converge. The method instantiates the frame, labels, and game pieces. There are two labels that are contained in the frame. The gameLabel encompasses the play area of the game while the statusLabel displays the score, high score, frogs saved and most frogs saved. KeyListeners are used to move the frog icon on the screen while an action listener argument for a timer object is used to animate non-user controlled game pieces. Collision detection is implemented in the collision method by acquiring the current lane that the frog icon is currently in using the getLane method then comparing the icon's left and right bounds against the car shape's bounds in that lane using the checkLane method. The frog icon and cars are not allowed outside the frame. The carBoundsCheck is used to determine if a car is in violation of the frame boundary in which the calling method can take appropriate action. The frog's position is checked within the key listener which limits the user's

movements based on the position. If the user action is valid then a call to the frog object is made to move the piece. If the frog icon enters the final lane, user control is temporarily revoked while animation is carried out and the frog icon is reset to its start position. The "Frogs Saved" and "Score" text is updated to reflect a successful attempt in the levelUp method. If the frog icon encounters a car the user is prompted to play again. If the user chooses yes, the high score and most frogs saved text is updated and the game pieces are reset. If the user chooses no then the program is closed.

The shape package contains CarShape, FrogShape, MultiShapeIcon classes as well as the MoveableShape interface. The book code for CarShape and MoveableShape was used as a basis for the Frogger program's class and interface with the same name. As suggested in the assignment documentation, a MultiShapeIcon class was used to contain the moveable objects to be added to the JLabel. The CarShape class manages the cars appearance and position on the display. The color of the car is updated randomly with the newColor method. The MultiShapeIcon class is merely a container for the shapes so that they may be added to the gameLabel.

Guidelines

Encapsulation of each class was strictly enforced. Each class is responsible for managing its own parameters. Accessors were used to provide data to client objects and mutators were used on a limited basis..

The law of Demeter states that a method should only use instance fields of tits class, parameters, and objects it constructs with new. This law was used whenever possible but the nature of the program made it difficult to strictly enforce Demeter's law.

The five c's were carefully studied and implemented in the Frogger program. Each class has a singular purpose and it is that class' sole responsibility to manage its affairs. Each class manages its operations and data to provide completeness in purpose. Each class has methods of supplying and manipulation of data as it requires fulfilling the convenience part of the five c's. The program is well documented and method names are self explanatory. The methods provide single call solutions for most requests. This provides the necessary requirements for clarity and consistency.

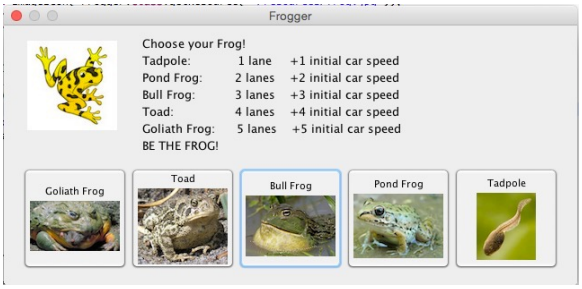
Programming by contract is a simplistic but efficient approach to programming. Each class has its set of responsibilities and each class expects certain details and information to be in proper order. The concept is novel but the power of the concept is driven by a simple set of rules.

Worked and Not Worked

The program was initially created to use independent icons but was quickly changed to use the MultiShapeIcon class due to implementation issues.

Generating a jar file was not part of the assignment but due to the popularity of the program it is beneficial for sharing among family and friends thus Frogger.class.getResourceAsStream("/resources/frog.png") was used so that the jar file could contain the resource image in lieu of new File("frog.jpg") .

The original start menu was used but a more aesthetic menu was made with picture icons. A return value from a user selection in the newer menu was not being returned and was replaced with the original.



The original animation for when the frog icon entered victory lane didn't actually show that the frog was in victory lane before reseting the frog icon back to start. This issue was corrected by adding a gameLabel.repaint call under the up key switch. Additional animations were added to provide a more visual cue and add to the user experience. The background of the frame is painted green and the frog icon travels off of the screen to the left.

Testing

Testing of the minigame was achieved in two separate phases. The first category of tests involved trials of the program as each stage of completion was reached in order to contribute to a more iterative design process. Early testing came in the capabilities and limitations present in using JFrame objects, Icons, and our own classes that implemented the Icon class. From there, movement of both the cars and of the frog were debugged extensively to ensure that neither could leave the bounds of the game board and that both behaved appropriately at any given location. Following this, collision detection became the next priority. Once the base game was

established and relatively solid, tests involving splash screens, score keeping, and flexible board generation were conducted. Finally, when all personal tests were conducted we released the program, exported as an executable Java archive file, to a fair-sized group of testers between the ages of 4 and 47. This enabled us to conduct black-box testing and solve any issues that we might not have encountered during our design and implementation process.

Checkstyle Comments

Checkstyle caught many of the trailing or missing spaces in our program, as well as errors in structure. That said, it also demanded Javadoc comments for private methods and fields and was unable to be convinced that such a thing is unnecessary. Checkstyle also insisted that single-line for-loops require { } brackets and sought to correct the "magic numbers" used in a point-slope formula that our program uses to establish base car speed based on their lane number. The last issue that was encountered was the fact that anonymous class length was limited to have a scope of only 30 lines, which is a heavy constraint for an anonymous KeyListener class, for example.

Collaborators

Andrew Markley and Cory W. Cordell authored the assignment.