Gregory Jerian, Jerry Huang, Sid Srinivasan
Period 2, 4
APCS
Kuszmaul

Hivolts Readme

HOW TO DOWNLOAD THIS GAME:

1. Make a project and put the .java files into the project (you could name it hivolts!)
2. Put the image files in the project source directory (this is the folder called hivolts)
3. Congratulations!

I: Introduction:

This original Hivolts was created by Thomas Ahasic, Douglas Jones, and Dirk Pellett in the 1970s for the PLATO system. When it was created, this awe-inspiring, legendary masterpiece rivaled the grandeur of the Apple computer, but for unknown and mysterious reasons it soon faded into the ashes of history. This all changed forty years later when Aikido master Mr. Kuszmaul found it and revived it from the dead, introducing it to an audacious team of talented, ambitious students named Gregory, Jerry, and Sid.

This trio created a version of Hivolts where the evil Lord Farquaad has cloned himself twelve times in order to catch the young ogre Shrek. Shrek, however, realizes this and runs around his swamp in an attempt to subdue them. Because Lord Farquaad is not the sharpest knife in the drawer, he blunders after Shrek in more-or-less straight line paths, zapping himself on electric fences and eliminating himself from the game.

This version of the game won critical acclaim and was featured on youtube.com/top10/top10legendarygamesofalltime. Inspired by this, aikido master Mr. Kuszmaul continues the legacy of this game by requiring his students in his APCS class to recreate the game while modernizing it. This game is a masterpiece and should be played by everybody on the planet.

II: How This Fulfills the Specification

This project fulfills the specifications for a Hivolts game with modern graphics. The core gameplay is the same with updated graphics inspired by the *Shrek* movies. The game gives feedback upon completion with “Game Over” and “You Win” screens and tracks the keys the user has pressed in the console.

III: Explanation of Current Errors

Currently, there is some code in the program that has not been encapsulated. This code is located in the method updateMhosPositions() which is used to move the mhos. We are aware of this lack

of encapsulation but unfortunately, we were not able to encapsulate the code in time. Next time, we will definitely watch out for this problem and make sure we encapsulate code when necessary.

IV: Overview of Code

Our version of the Hivolts game contains 3 objects. A player, a mho, a grid, and a hivolts frame. The main game loop occurs in the HivoltsFrame class. When a hivolts frame object is created, the HivoltsFrame class is called and the game begins. The code is split into three main parts:

1. The gameboard is created by creating a two dimensional array of objects of type Cell. We defined type Cell in the class Cell. When we create the 14 by 14 double array called grids, we are essentially creating 14 * 14 or 196 items - each of type Cell. The array grids simply holds 196 Cell objects. Each cell represents a position on the gameboard. For example, grids[1][1] represents the cell at the (1, 1) position on the gameboard. Each cell object has one single characteristic - a token. A token is used to determine what is on a particular cell at the current moment. The possible tokens for a cell are ' ', 'F', 'O', 'B', 'E', and 'P'. ' ' represents a cell without a token, meaning there are no objects on it at the moment. An empty cell is automatically painted with the background picture. 'F' represents a fence inside the borders of the gameboard. 'O' represents the border fences. 'B' represents the cells bordering the outline around the border fences. 'E' represents a cell that has a mho on it. And lastly, 'P' represents a cell that the player is currently on.
2. The init function adds the actual grid to the JFrame and paints it. It also uses the random number generator to set the token of random cells to either 'E' (enemy), 'P' (player), or 'F' (fence). Whenever the token of a single cell is changed, it repaints only that cell so that a new image correctly reflects the token value. For example, if the token of a cell is changed from ' ' (background) to 'E', the image on the cell will change from the background image to an enemy mho. After the random number generators run and put 12 mhos on random cells, 20 fences on random cells inside border, and 1 player on a random cell, the player can begin playing.
3. The KeyListener is used to track keyboard inputs and run methods that update the positions of the mhos and move the player. Whenever a valid key is pressed, the player is moved in the corresponding direction and the isGameOver() method is used to determine whether the player has lost yet. If not, the token of the new cell is set to 'P' and the token of the old is set back to empty. The updateMhosPositions() method is then used to update the position of the mhos. This loop is repeated throughout the game until either the player loses, or all the mhos are removed from the gameboard. At the end of the game, a isGameWon() method is run if the player wins and an isGameOver() method is run if the game is lost. From there, a newGame() method that resets the game is run if the user wishes to play another game.

V: Major Challenges

Major challenges in creating this code were debugging the mho movement, encapsulating the code, and creating a logical game loop that performed all game over checks correctly. However, through dedicated hard work, we were able to overcome these challenges as a group and create a working Hivolts game.

VI: Acknowledgements

Special thanks to Sean Chapman and Dick Agarwal for moral support. Also, thanks to the Oracle Java documentation for providing us with the methods required to add pictures to our JFrame window and BrandonioProductions for his KeyListener tutorial.