Project 9

Added by Christopher Hoder, last edited by Christopher Hoder on Nov 22, 2011

Project 9 Write Up

Project Goal

For this project the goal was to create a game / simulation that used data structures we have learned about in class as an integral part of the program. I chose to implement a version of the millipede game similar to the atari version that can be found online. My rules and implementation were based off of the following page: http://www.atari.com/play/atari/millipede. The basic rules are as follows: You are a spaceship tasked with destroying the attacking millipede. The millipede will enter the screen from the top left as one giant animal. The millipede will continue to snake back and forth down the landscape until it reaches the bottom, where it will snake back up the screen again. There are also obstacles on the landscape that cause the millipede to move one unit vertically and then switch direction. When you hit a section of the millipede, that section becomes and obstacle and the millipede splinters into two fragments. This happens with every hit of the millipede. The level is complete when all sections of the Millipede have been killed. You are also only allowed to fire one bullet on the landscape at a time and are given 4 lives (in the default mode). After you die, the user is prompted to enter their name to be added to the score list. All scores where somebody enters a name are saved and only the top 5 are displayed on the right.

Implementation

To implement this game I continued to use the Landscape, SimObject system that we have used over the past few projects. I modified the SimObjectList class to make it a double linked list, which allows for an easier remove method (can just go to node instead of the parent node). Additionally, I made the SimObjectList extend SimObject to it too would have an updateState method. I did this to make the divisions of the millipede easier to implement. To implement the millipede's motion I changed the resources on the landscape to be an ArrayList of SimObjectLists. At first there is only one SimObjectList holding the long millipede segment. However, when the millipede gets cut into segments the new segment is cut out of the old list (why i chose a double-linked list over an array) and stored as a new SimObjectList. This new list is then added to the resources ArrayList. Therefore, the resource ArrayList is holding an array where each element is each segment of the millipede.

Another part of my code, which I implemented as an extension, uses another data structure from class to hold highscores. After a user looses, he/she is prompted to input a name to save their score. All input scores are saved in a text file (highscore.txt). During the initialization this file is read and parsed where each score/user name combo is saved in a ScoreData object. The ScoreData object implements the Comparable interface which allowed me to add these objects to a PriorityQueue which sorts them based on their score. Furethermore there is appropriate error check so that if the highscore text file is deleted it will be detected and created.

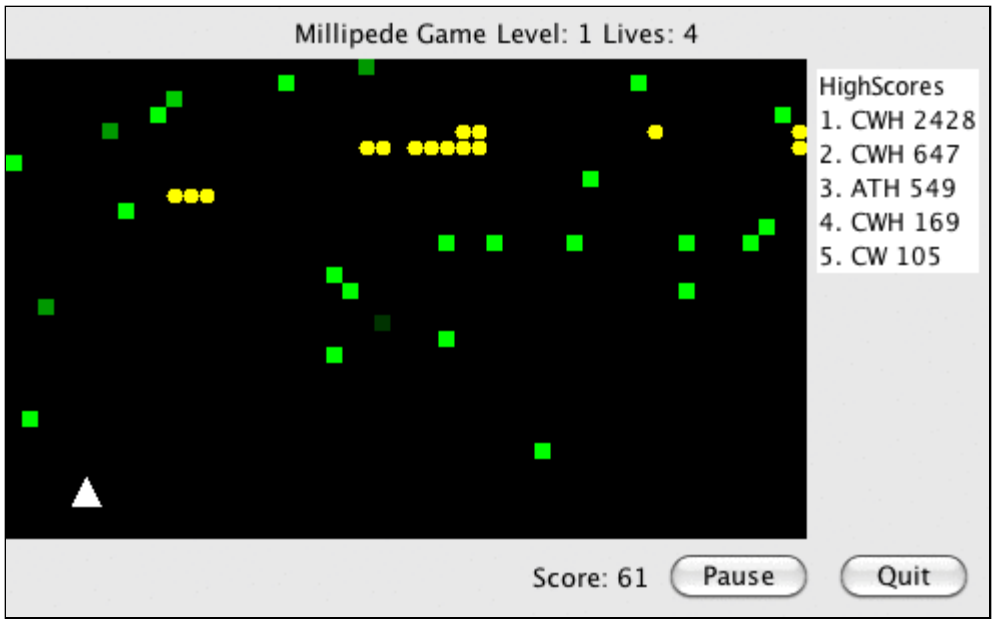
The game is controlled almost completely through the keyboard and the display. The defender is controlled using the arrow keys to move and the space bar to fire. There are pause and quit buttons on the display but you can also start/pause by hitting the p button. The "w" button speeds up the simulation, "s" slows it down. "r" will start/stop saving images every 50 iterations. Some of the defaults such as lives, millipede size, and

number of obstacles in the game can be set using command line input parameters. The syntax can be seen in the javadoc created documentation which was included in the solution.

I also modified the LandscapeDisplay code so that the image will change when the game is paused. Instead of just showing a static image of the current Landscape situation the landscape changes to just show a static picture of the Millipede game cover (taken from http://www.emuparadise.me/Nintendo_Game_Boy_ROMs/Millipede_&_Centipede_%28USA,_Europe%29/69215). Additionally, I worked to use generics, try/catch /exceptions, appropriate use of statics and the ?: operator when needed in the code instead of just ignoring them. This included some modifications to the SimObjectList code.

Results

Below are some gif's of the game in action. The game gets increasingly hard after only the first few levels because it is currently set to double the size of the starting millipede with each level. I was able to make it to level 4 but not past, with a high score of 2428. The display will also change when the game is paused so that the user cannot see the current situation, instead it just shows a picture of the Millipede game logo. Other information such as lives, score and high scores are still displayed. There is also an image of this below.



This image below is of the start screen. The pause screen looks exactly the same but the score, lives and level information would be updated to match what is going on in the game.

Conclusions

This project taught me how to use the data structures that we have learned about to create a fun, interactive game using the java swing components. I used both double linked lists and priority queues in this project to achieve the effects in the games such as the Millipede's and the highscore list. After completing the game i realized that I used a lot of repeated code that could be eliminated with come code refactoring. I don't think I need to an array of SimObjectLists to deal with the different Millipede segments and additionally I think i could combine the Millipede class with the Obstacle class and store them as the same data set. However with clarity of my code in mind I decided to leave the code as it was. I would have liked to add more interactive components if I had had the time to continue learning more about the swing components of java.

Labels

cs231f11project9