Hivolts Pre-Plan + Progress Report

Introduction

For this assignment, we are rewriting a game from the PLATO system called hivolts, originally written by either Thomas Ahasic or Douglas Jones and later modified by Dirk Pellett. Our goal is to create computer graphics superior to the 1970's graphics (pictured below) the game originally had. We plan to create different icons to represent the fence, the Mhos (monsters), and the user (as a smiley face), and assign them with a key. Our game should create a 12 by 12 grid of icons, and the user should be able to navigate through with their keyboard keys, dying by both Mho's and Fences, and also be able to win by killing off all of the Mho's. This project will be done in collaboration with Zage Strassberg-Phillips and Sabrina Martin, and with help from Chris Kuszmaul. In order to achieve these goals, I plan to break the project into four parts.

Part 1: Board

First, I plan to create a 12 by 12 board, printed in output section of Eclipse, as a representation of the final product. The board will be printed using a Fence class, a Mho class, and a SmileyFace class. These three classes will store separate positions for each of their individual components. For example, the SmileyFace class will have one x and one y variable storing the position of the SmileyFace, or user, at that time. The Mho class will have a double array storing the locations of the Mho's. It has not been decided whether there will be one two dimensional array storing the placement of each of the items, or three separate dimensional arrays for the Fence class, the Mho class, and the overall placement. Once the board both randomizes and prints everything correctly, we will move on the next step. The status on this part is about halfway done. All of the classes have been created, and two main methods have been created. The position two dimensional array has also been created, but the use of it is tentative. The level of effort is about a 5, mainly because of the randomization required. The classes and variables themselves are simple, but the code that prevents Mho's from interfering with Fences and the SmileyFace landing on a fresh spot is more complicated, which bumps the level of effort up to a 5.

Part 2: Graphics

The next part of the plan is to create the graphics. This means assigning icons to each one of the classes, linking the main to JFrame, and following other graphic, init, and paint code. We are not sure whether this will mean creating entirely different classes then the ones we created with the Board, but with the same concepts, or we will reuse them. This part will mainly consist of placing the icons in their correct spaces on the screen, and creating a correctly randomized 12 by 12 grid with a key. The interaction with the user will come later. The status on this part is not started. The level of effort on this part will be about a 4, because we will be using only topics learned previously, and be using the same grid structure practiced in step 1. The icon design itself raises the level of effort, but other then that, this part is tedious but simple.

Part 3: Mho's and Fences

For the 3rd part, I plan to make the substance of the Mho and the Fence. For the Mho class, this means creating methods to allow the Mho to be movable to coordinates, to react by disappearing, and the interaction between it and the Fence, and it and the user. More methods might be necessary, and will be added. Next, for the Fence class, this means creating methods that allow the fence's coordinates to be easily accessible, the restarting of the fence placement, and other important methods not yet known. This part is mainly creating the fundamental parts of both classes before adding the final piece of user interaction. The status on this part is not started. The level of effort is about a 7, because of the amount of thought that has to go into each part, and the prior organization. The Mho and Fence should be designed, so that the next step of user interaction can be easily interacted. This required deliberation over the process in which we run the program.

Part 4: User Interaction

Finally, for the last part, I plan to add all parts of the user interaction, which consist of user play, user outcome, and user restart. For user play, this is the most difficult part of the game. This means, based on whether the user decides to jump or move, we have to link the keys to the commands. Next, we have to execute on the commands, and have the Mho's and Fences act accordingly. This part is a large unknown right now, and will require research and help from outside sources. Next, the user outcome will happen if the user either runs into an Mho or a Fence (death) or the user defeats all the Mho's (win). A user death should end the game, and ask the user if they want to start a new game or stop playing. A user win should congratulate the user, and also ask them if they would like to continue. Finally, the user restart will be if the user decides to continue playing, in which the entire game has to be reset. We are not sure whether we want to add a button during the game that allows the user to restart, but either way restart means that all of the randomizing functions must be called again, new Fences and Mho's placed, and a fresh starting position for the user must be calculated. The status on this part is not started. Overall, these are the three components of Part 4, which are very difficult, and therefore put the level of effort at a staggering 10. Many of these parts are unknown to me, which will require research and learning new CS topics, along with implementing them, which will require a large level of effort from me.

Overview

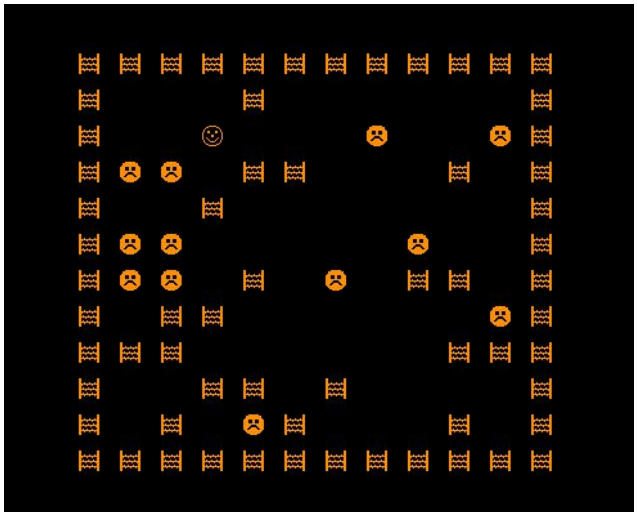
Overall, the plan is to first create the board as an outline for the placement of each icon. Next, we plan to create the graphics on the window through icons, and then set the fundamentals and other methods for both the Mho class and the Frame class. Finally, we will add user interaction, to allow the user to move, play, die, restart, and win.

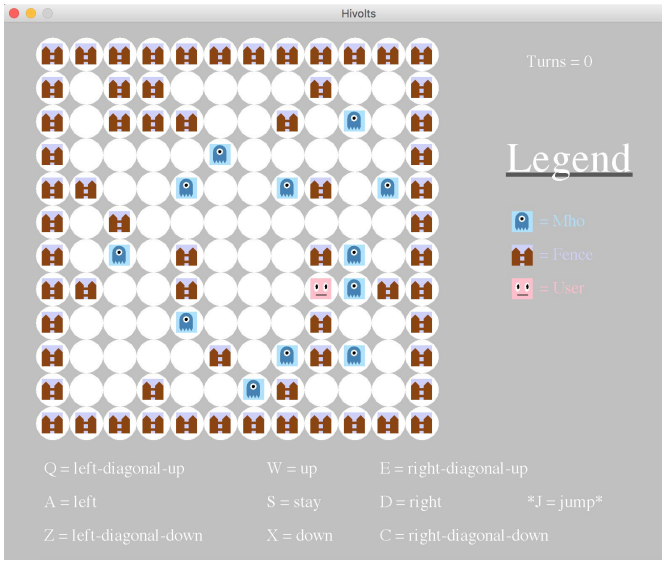
Timeline

Tuesday, September 26th: Start Part 1, setting out the board
Thursday, September 28th: Finish Part 1, and have a full fledge board that randomizes
Monday, October 2nd: Start Part 2, if Part 1 is complete, and set each icon
Tuesday, October 3rd: Set the randomizing algorithm and system from Part 1 into Part 2
Thursday, October 5th: Start Part 3, with the fundamentals of both the Mho and Fence class
Monday, October 9th: Work on Part 3, and finish the parts of Mho and Fence class
Tuesday, October 10th: Start Part 4, and begin working with user interaction
Thursday, October 12th: Continue working on user jumping and moving
Monday, October 15th: Work on integrating the Mho and fence with jumping and moving
Tuesday, October 16th: Finish integrating the Mho and fence
Thursday, October 18th: Work on user outcome (death and win)
Monday, October 23rd: Finish user outcome (death and win)
Tuesday, October 24th: Work on user reset
Thursday, October 26th: Finish user reset
Monday, October 30th: Align all parts directly with the instructions, and make changes as necessary
Tuesday, October 31st: Work on documentation
Thursday, November 2nd: Finish documentation, turn it in.

Hivolts Documentation

The purpose of the hivolts project is to create a 12 by 12 grid game according to the specification file provided by Chris Kuszmaul of the game named “Hivolts” created by either Thomas Ahasic or Douglas Jones, which requires movement connected to keyboard keys, fences, mhos (monsters), a user, and interactive gameplay. “Hivolts” in its full 1970’s graphics is displayed in the first image below (the code created by group is the second image) in which the game consisted of a single user, 12 randomly distributed mhos, 10 randomly distributed interior fences, and a border of outside fences. This original version represents the user as a happy face, the mhos as a sad face, and the fences as two horizontal lines with three squiggly vertical lines in between. The game had keys corresponding to user movement for stay, up, down, right, left, diagonal up-right, diagonal up-left, diagonal down-right, diagonal down-left, and jump, and gave the mho’s flexibility of all of these directions as well. Gameplay consisted of the user moving in one direction, in which the mho’s would respond by gravitating towards the user. A specific set of rules are set for this gravitation, allowing the mho to die in a fence with pure vertical and horizontal movement, but restricting a more intelligent movement of the mho’s when diagonal movement was considered. At every move the user makes, the mhos move towards the user, including the movement of stay. The one exception to this is the user movement of jump, which places the user on a random point of the inner 10 by 10 grid, but guarantees that the user will not land on a fence, but can lend on an mho (instant death) or an empty space. The specifications given included these requirements, including some ambiguity surrounding the movement of the mhos, and also asked for creativity in the appearance of the game board. I completed this project with two other people, Zage Strassberg-Phillips and Sabrina Martin. The code we created fulfills these specifications to the best of my ability, but still stands for improvement. There is one main point that stands out, for diagonal movement, the mho, finding it’s two main pathways blocked, chooses the vertical/horizontal pathway. This change was created intentionally for ease of user play, and increased the chances of winning the game and enjoyment of it. Based on the ambiguity of the project and the creativity and uniqueness asked in the instructions, this is one of the changes we made to our code. As explained in the pre-plan, we broke hivolts into four parts: the board, the graphics, the mho’s & fences, and the user interaction. To break up the work between the three of us, I tackled the graphics, instantiating the board, the user's movement, and the organization. Zage completed the Mhos move method, the fences, and the mho’s interaction. Sabrina coded the work for the KeyListener, the game over and wining methods, and the resetting of the game. Lastly, we faced many challenges throughout completing hivolts, specifically with the creation of the Mhos move method and the KeyListener, but were able to overcome them with help from other classmates, Kuszmaul, Tutorial Point, and Stack Overflow. Through these challenges, the plan, analysis, and errors, we have been able to create a replica of the 1970’s “Hivolts” game following the details in the instructional file.





My Java code successfully fulfills the requirements from the specification file (picture shown to the left) with exact movements, the correct user, mho, and fence interaction, a game board, and an 10 by 10 randomized grid with a border of fences. The mho's move as corresponding to the specification file and the 1970's game, except for the modification aforementioned. The user moves correctly with each of the 9 movement keys, and jumps to a random position with the touch of the jump key. Also, we added interaction with the "p" key to restart the game. The user appropriately dies when landing on the same space as a fence or an mho, the reason cited as "you ran into something". If an mho reaches the user, then the reason cited is "an mho ate you". And, lastly, if the jump method places the user on an mho, the reason cited is "you landed on an mho". The game board contains a fully fenced border, and the inside 10 by 10 contains the 10 fences, 12 mhos, and one user randomly placed on the board, but all in separate places. This entire setup is represented with the two dimensional array ps[][] (position array), where an "x" represents an empty space, an "S" represents the user, an "M" represents an mho, and an "F" represents a fence.

Our hivolts game meets all of the expectations detailed in the project, and therefore falls short in no areas but in the area of purposeful change, as aforementioned. However, as beginning coders, our code could contain loopholes and other smaller kinks we did not address or find. This possibility leads us hesitant to conclude that our code that follows the specifications perfectly, but instead claim to have code that follows the general idea of the instruction file and adds a bit of creativity into a few elements, as asked. For this possibility of error, I would like to learn how to find and create methods to search for problems or loopholes in a set of methods or classes. Overall, the one major parts of the flag code I would improve is the lack of experience, and perhaps creating a separate level where the mho dies diagonally to increase the difficulty of the game.

First, the code creates a two dimensional String array called the position array (ps[][]), which is initialized to contain all "x"s, and then the user, fences, and mhos are added with their specific requirements and with no overlap. First, following my pre-plan, this position array is printed on the console as a game board. This board was created by me, with help from Zage. The level of effort was around a 3, the game board was much easier than expected. Next, following the pre-plan, came the graphics. I was able to use the classes from the game board, which greatly increased the efficiency, and I found that printing the graphical board only required running a loop across the positional array, with separate distinctions (color, shape) for the different String that was in question. I also created white circles in the back of each space to better distinguish them, and used these as placemats for creating and matching the colors. To start off, I only had light pastel colors assigned to the characters (chosen by the group). The icons part of the graphics was pushed to later on. This creation of the graphical game board was also completed by me. The next part of the plan was the mhos and the fences. Zage contributed the most to this part. She created an mhos move method for all of the actions an mho carried out, which consisted of following the user in many different ways. This part was tedious, and unorganized, but ended up turning out well! The mhos death was also complicated, as we were uncertain how and when to confirm the death. Zage also created the mpa position array, which held the coordinates of each of the mhos. Zage also created the code for the the interaction and the creation of the fences, which consisted of many "else" and "else if" statements. This level of effort was definitely around a 9, because of the confusion with the user coordinates, and how to implement the seconds steps of the mhos following. The last part of the plan was user interaction. Sabrina both found (through the internet two sources credited below) and implemented the KeyListener program into our code. Through that code, she linked the correct keys, and linked that to the user (SmileyFace) method with our moveOrDie method, where she inputted the correct coordinates for each letter move. Next, Sabrina created with game over and the you won methods. There were many issues with these at first, because we wanted to use the graphics outside of the paint method, but were unable to. It was difficult for us to figure out how to account for game over and you won requirements in the paint method, and make sure that paint method was called at the right times with repaint. Eventually, we figured out a forEachTurn method that ran repaint after each turn, whether it was a moveOrDie or a jump method, and therefore we were able to implement game over and you won. Also, Sabrina created the reset method for after game over and you won, which carried out many similar functions to the initializing methods, and so we combined most of their components. Lastly, I carried out the organization by creating helper methods for every available scenario. However, I did not create helper methods for mhoMove because of the iteration through the mpa array, and the amount of code that had to be

written to redefine x and y outside in a helper method and then again in mhoMove. Also, a part of the beginning pre plan that got pushed to the end, I created the icons for each character, using the limited graphical knowledge I had of polygon, oval, and rectangle creation. The user is a content face, a play on the happy and sad face in the original design, while the mho is a cyclops ghost, a play on Pacman, which was created around the same time. The fence is a regular clipart fence, and the Legend is created with the same icons. The other legend, keys, and turns were created by Zage and I. Overall, our code structure mainly follows both the pre-plan and the specification file.

There were two major challenges I faced in the code, the repaint method and the mho movement. With the repaint method, I could not figure out how the issue of my code, where repaint was running constantly and causing the screen to flash repeatedly. Everything I changed had no effect, and I was left frustrated and at square one. I ended up using an older version of the code, and never figuring out why what I had done was incorrect until the end of the hivolts project. Turns out, the setBackground method had been the reason for the continuous flashing, and it didn't have to do with repaint, which relieved the struggle. Next, Zage, Sabrina, and I all struggled to interpret the meaning of the specification file when it asked about the direction of the mho after diagonal facing the user is restricted. After many attempts on the whiteboard, and three different mho methods, we found a solution. Then, we added our own element to make the game easier, by allowing the mhos to die either vertically or horizontally without a diagonal dying element. This is shown by our else statements after checking the absolute value of the difference of the x and y coordinates. Overall, these were my group's worst struggles, and I'm glad we overcame them and learned from them.

Thank you to
https://docs.google.com/document/d/15wIsV7iMtdKvFh86yfYU549kk3w7CLjjCXu89KUU5l8/edit
which provided us with the exact requirements for hivolts. Additionally, I would like to thank Zage, Sabrina, for completing this project with me, my other classmates for helping me struggle through each one of the areas of improvement and challenges, Kuszmaul for pushing me through the repaint code, and Tutorial Point, and Stack Overflow for providing me with sufficient java information to troubleshoot through difficult areas and implement the essential graphic and key parts of hivolts.