Wyatt Smith
March 2014

Independent Java Project Narrative

I guess the start of my project really began when the idea of generating and solving mazes was brought up momentarily in class when talking about interesting problems and solutions, and for some reason this idea got stuck in my head. The idea was subsequently hammered down into my skull because some labyrinth building dude came to punahou and gave lectures to the effect of "Labyrinths are orgasmic" or something, and because labyrinths and mazes to my admittedly absent head were like po-tay-to, po-tah-to during his lectures, I eventually chose the subject of mazes for my Java Independent Project. I started out super ambitious--I thought that I could knock out this project in a few weeks and add a bunch of cool features to increase the challenge of each maze, and so I planned on having 2.5d and 2d displays, and almost like a text adventure feature for each maze where the user could try to solve the maze using only text prompts, but as I began to investigate in 2d and 2.5d game engines like game, I realized how learning an entirely new engine would be both tedious and inefficient, so I decided to code using GridWorld rather than to use game engines or to create my own interface/display. I'm not sure I was exactly hoping to learn anything, but I was definitely intrigued at the possibility of using recursion in my code, which actually became a major challenge for me.

To really get started in the coding process, I basically googled "How to generate mazes" and found a bunch of great formulas (Thanks, Wikipedia) which I went through, selecting ideal ones for use in GridWorld. I then imported a bunch of stuff from the GridWorld folders already on my computer, and set up a bunch of skeleton classes that I thought I would use for the running of the game and all its methods and whatnot. I brainstormed ways of converting the terminology used in the maze generation algorithms into classes for use in GridWorld, making my own versions of walls, and began on the meat and potatoes of my project: the generation. I knew that the recursive backtracker used for finding the shortest possible route through a maze was doable, as I had seen a bug going through a GridWorld maze in a youtube video that LeClaire had found in the early stages of my project, and even if I wasn't able to duplicate that maze-running bug I could always implement the classic "always turn right" strategy for finding the exit in a maze, so I focused on generating a maze that didn't suck.

To start maze generation I just sat down and hard-coded what I thought would be the easiest of the various maze generations to implement, depth first search, and hit compile expecting the worst. My original code had a maze generation class in the from of a bug run through a solid block of wall-type actors to flesh out the maze, but unfortunately I was plagued by errors and misplaced semicolons . I'd like to say here that I heroically cut a path through to a perfect recursive solution to my maze generation problems within a week, but sadly this was not the case. At all. Honestly, 80% of my project was figuring out how to generate a maze in Grid World, and so 80% of my time was spent being utterly frustrated with Grid World's limitations. I occasionally helped and consulted with Nolan and Jaz, because they were also working in GridWorld, but aside from that I was mostly error checking and testing the generation code. Eventually, I got it to the point where it compiled no problem, and so I proceeded as if the maze generation was pristine, starting on a backtracker class that would in a perfect world compare the number of steps a player took to complete the maze to the bare minimum number of steps, as well as showing the correct path, but as I got to what I thought was a workable solution for backtracking, my tests for the backtracker didn't go through because of problems with the maze runner class that I had supposedly fixed days earlier. I had already made a bare minimum for a Player class to navigate towards a blue picture of Nolan that was functioning as the end of the maze, which thankfully, worked, but it turned out my maze generator was creating something that belonged more in a game of civ 5 than it did in a maze game. At this point, deadlines for beta testing were becoming more and more menacing, as I had scarcely a week left, so I had to scrap my plans for fancier mazes and for arrow-key dependent movement and really get my hands dirty with the dysfunctional code on my computer.

The absolute first thing I did was to check the formulae I found on the internet to see if they actually generated a maze as they should have, because for some reason, not everything posted on the internet is true. So I got out a piece of printer paper, drew a five by five grid and went through the recursion in depth-first search maze generation manually. What I discovered was that the problem with my code wasn't that there was a problem with my recursion or with my interpretation of the formulae found on the internet, but that the problem lay in the way GridWorld itself generated a grid. Every formula found on the internet depended on a grid full of WALLS, but grid world is a grid made of BLOCKS. What was really frustrating though, was that while this distinction was visible within three steps of my five by five maze on paper, I didn't think to check the formula on paper until I was three weeks into my project. At this point, I decided that it was way too late to convert all of my code into code workable with other displays or grid types, so I was forced into finding a way to convert my code into a maze in a grid of blocks.

To work my code into a maze generating savant, I went through each section of my code methodically until I hit the spot where it ran into problems, and added a restriction to that section. Prior to my meddling, the code was "eating" squares in blocks around certain spots because there were no restrictions on whether or not it could destroy parts of the maze that were already "walls" or blocks upon which there were two or more adjacent cells in cardinal directions which were already empty. Just to see what would happen, I implemented a checking aspect in the bug to make it so that if a wall had two or more empty cardinal adjacent cells, it would remove that cell from the list of cells to recurse through. This made a long winding path that went through the maze without coinciding with itself to make an actual maze, so I caused the recurse method that made winding paths loop itself in the act method of the bug until there was a little more than half of the entire number of blocks defined in the grid empty. This actually made something that resembled a maze, but I realized after testing some of the mazes that they weren't all solvable, so I implemented a "destroy up to two blocks" mechanic to make the mazes 100% solvable. unfortunately, this mechanic also broke my backtracker class, which basically made me remove it from the entire program. At this point I guess my project was really finished, with only a few user-interface messages changed or updated at later dates.

In conclusion, I guess the only really thing I really learned was not to just work out the grander concepts of a program theoretically, but also to CHECK THAT THEY WORK because the potential loss of time spent exploring down avenues that are already dead ends is much greater than the time it takes to do a simple check on a formula, even if it takes no more than five minutes with a piece of paper and a pencil.