Dylan Maloy
Prof. Pfaffmann
CS150 Project 2 Report
10/29/19

Project 2 Report

Summary:
The project revolves around a game map. The map can be created by the user, and will be imported from a text file. The player will get to choose which amount of dwarfs (each with their given attributes (digger, harvester, builder)) to buy with the starting balance, and the game will then simulate until all of the gold has been harvested - the goal is for the dwarfs to get all of the gold on the map, however, there are obstacles like lava, pits, and underground rivers that they can fall into and either moved or killed. If killed, the player will have to pay the dwarf's family a sum of gold. The final goal of the game is to get as much gold as possible with the greatest efficiency possible. This efficiency will be based off of the amount of "moves" made by the dwarfs and how much gold they have collected. Iterating through each dwarf in the game once is equal to one move.

Major Features:

- Allow player to choose their lineup of dwarfs via dwarfs.txt
- Different dwarf types with different abilities
- Ability to retrace steps to return to home base
- Each dwarf has the ability to see what other dwarfs have been to a given tile
- Ability to add obstacles like rivers, pits, and lava to hinder performance
- Ability to create a custom map (map.txt)
- Dwarf productivity logging / statistics

Assumptions:

- The player isn't going to start with an absurd amount of money
    - This would ruin the point of the game. Obviously the gold will be found quicker with 100 dwarfs in comparison to 10 dwarfs.
- Dwarfs won't be overpowered / can't have all attributes

    - This would allow 1 dwarf to go around the map without calling builders / harvesters to get the gold : would greatly increase the efficiency & would ruin the point of having different lineups to improve overall efficiency.
- The map will have a limited number of obstacles (this is up to the user though)
    - A map with too many obstacles might not allow the dwarfs to get to the gold (if its surrounded by obstacles that cannot be built over)
- The harvester dwarfs can see gold as long as there is nothing in the way (can see through tunnels)
- Score mechanism will be based off of a time variable and the amount collected
    - Creates a way to score each simulation with different maps and dwarf setups
- No families will have to be paid because dwarfs cannot die (they avoid obstacles)

Dwarf Abilities:

- Digger Ability
    - Can dig dirt
    - Can see pits and gold
    - Can send its own object to harvesters and builders (not location, just name)
- Builder Ability
    - Can build over pits so that diggers don't fall in
    - Can see if the digger who has found the pit has been on a specific tile
- Harvester Ability
    - Can harvest gold
    - Can see if the digger who has found the gold has been on a specific tile
    - Can see through tunnels

Game Experience Goals:

The user will be able to adjust the map to their liking in the map.txt file, and they will also be able to change the lineup of dwarfs in the dwarfs.txt file. Once running, the program will show them the map after every move / every dwarf iteration. After all of the gold has been collected, the user will get a final value, "moves" which represents the amount of iterations through each dwarf or ticks that the program has been through before harvesting all of the gold.

Classes:

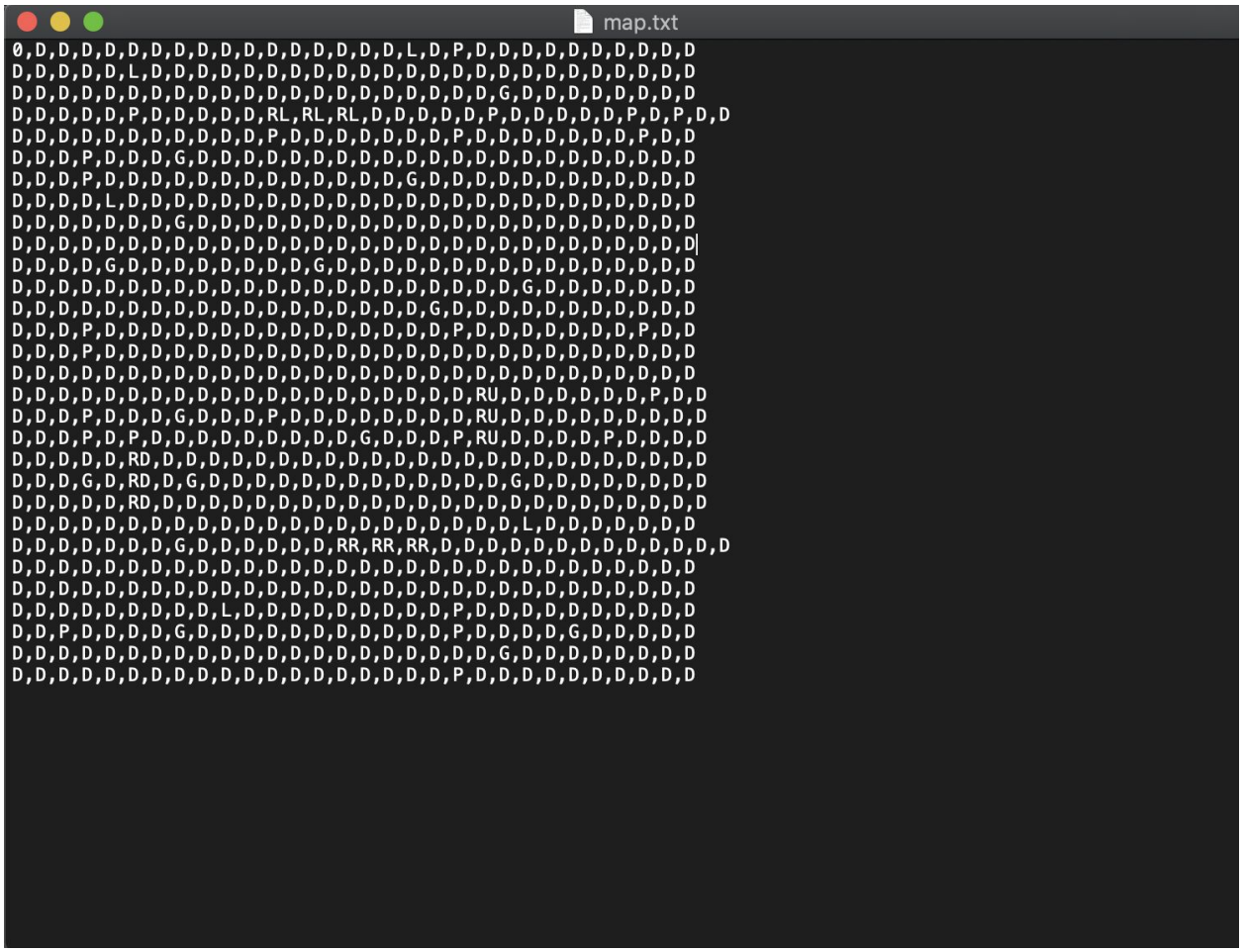
- Controller
    - Creates map && runs simulation on it given dwarfs
- Logger

    - Controls all log related aspects of the program (see important code components)
    - Appends messages to the log file
- Map
    - Controls all map changes / methods (see important code components)
    - Has methods which return the index's on each side of the input index
- Tile
    - Contains hashmap of dwarf objects that have gone to this tile (and the move integer when it was there) (log(n) search time)
    - Contains type (dirt, river, lava, etc.)
- Game
    - Iterates through the dwarfs
    - Controls all of the games main variables and structure
    - Controls what do do with each dwarf at its current state
- Dwarf
    - Main dwarf object (features inherited from subclasses)
    - Contains abstract methods for submethods to inherit
    - Contains methods that all dwarfs have in common (basic mobility)
- Builder (pit and lava)
    - Builder object (idle, moving) extends dwarf
        - Uses recursion to find the pit node which the digger has been to (see important code components)
        - Build up
        - Build down
        - Build left
        - Build right
- Harvester
    - Harvester object (idle, moving) extends dwarf
        - Uses recursion to find the gold node which the digger has been to (see important code components)
        - Harvest left
        - Harvest right
        - Harvest down
        - Harvest Up
- Digger
    - Digger object (moving) extends dwarf
        - Find closest dirt node (for all for sides) and goes in the direction of the closest. (see important code components)
        - Dig left
        - Dig right

- Dig down
- Dig up

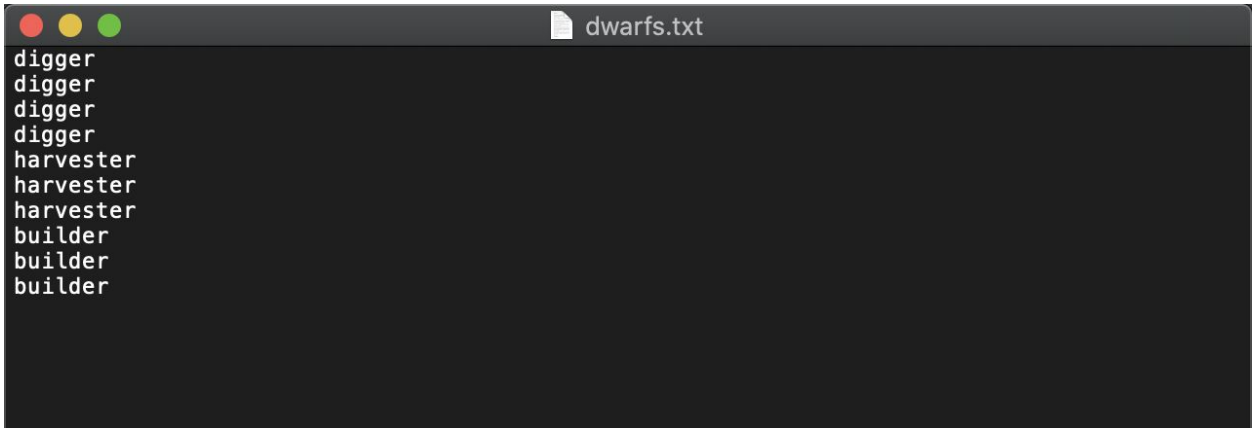
Important Code Components:

Text File input:



- Map.txt (map input file)

The map text file input is pretty simple. In terms of the map, each map tile is separated by a comma - during the import phase, the map is then split by the comma and made into a tile based off of its type. Then each tile object is added to an arraylist which is the map. The **Map** class contains functions which allow dwarfs to “move” though the map as a 2D space.



- Dwarfs.txt (dwarf input file)

The dwarf text file input is more simple than the maps. In the **Game** class, before the simulation is ran, the importer goes through each line in the input file and creates and adds the dwarf object that corresponds to the line in the file to a priority queue "dwarfs".

Map Manipulation:

Depending on the type of dwarf, a dwarf will only be able to manipulate certain types of tiles on the map. In the code, each dwarf type has a set of methods that corresponds with its abilities. These methods are named "left", "right", "up", and "down". Using the location of the dwarf, the getRight, getLeft, getAbove, and getBelow methods will be able to retrieve the index's of the dwarf's surrounding tiles respectively. Depending on the type of this tile, the dwarf will either dig, move in that direction, or "call" for another type to either build or harvest said tile. This process will be done by passing the dwarf object (basically used as an ID) an idle harvester / builder - they can then use their own methods to find the location of the gold/pit via the dwarf's object that found the gold/pit in the first place.

Digger movement:

The digger's movement is contained mostly in the move() method of the **Digger** class. It is an algorithm that finds the closest dirt block to the current digger dwarf and creates a tunnel at that block. If there are 2 blocks at the same distance it will go in this order - left, down, right, up. This way it will keep going across unless there is an obstacle. This movement algorithm also makes sure that diggers don't bunch up all at once making the efficiency worse.

Harvester / Builder movement:

At first, the harvester and diggers only move when they are needed. Once needed, they will recursively find the location of the node that which they intend on harvesting/building over via searching the hashmaps for the digger object that found the gold/pit tile in the first place. Once at the desired location, they will then fill/harvest the tile. Once the game progresses (less than 30 dirt blocks left on the entire map aka mostly tunneled out) than the harvesters will go to the remaining gold once the map has been dug out if there is any remaining. This will help efficiency because they will not have to wait for the digger to find the remaining gold blocks and go through their entire path to the gold which gets larger and larger over time - becoming less efficient.

System Clock:
The system clock isn't much of a clock. It is more of a move counter - one move is considered all dwarfs making one advancement / running through their move methods. This move counter will stop when all of the gold in the map has been harvested by the dwarfs and will provide insight on the efficiency of the lineup of dwarfs that the user chose.

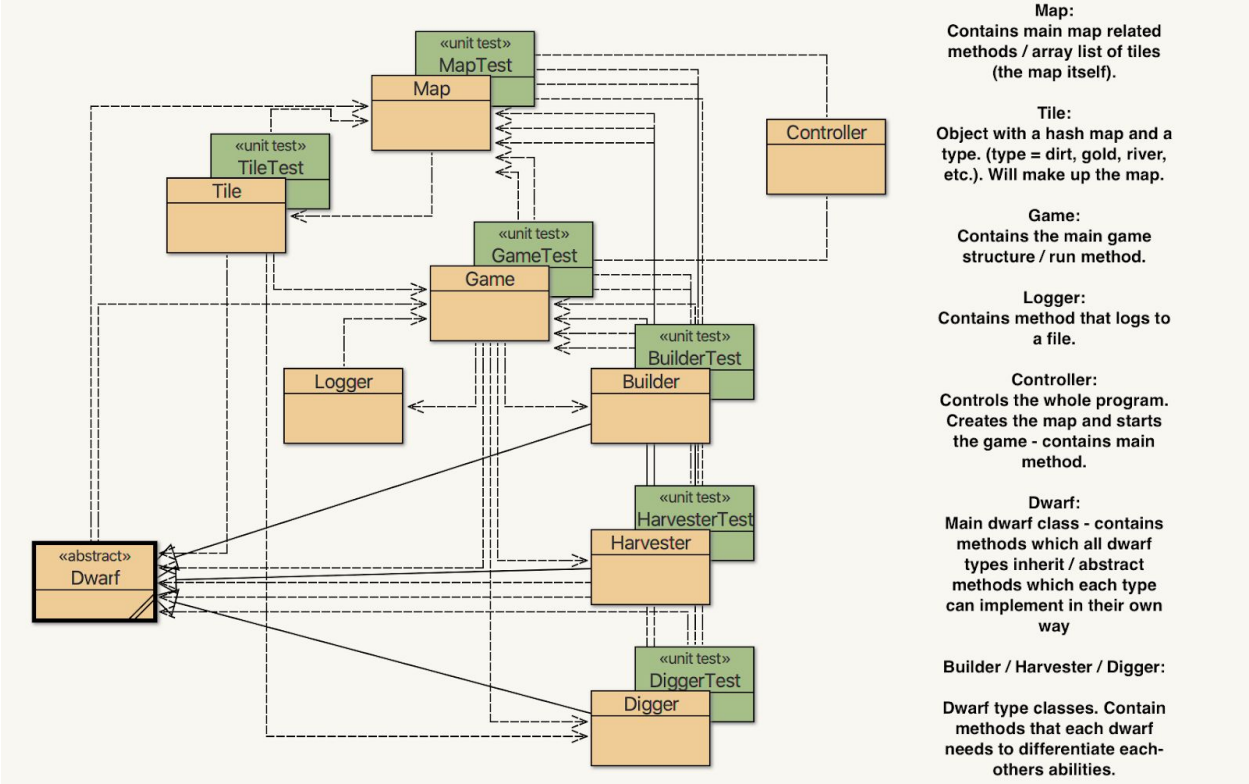
Generating a Map:
Map generation is done through the **Map** class. A filereader reads the map.txt file and iterates through each line. For each line the program splits the elements separated by commas into an array. Once split, each element (a new element created with the type from the file) is added to the ArrayList of tiles named map. This map ArrayList can then be used by the game.

Logger:
The logger is pretty simple. It creates a file when the object is created (before the game is created), and whenever called with a specific message, it then appends the message to the log file. This is useful for debugging and seeing what actually is going on over the course of the programs runtime.

Dwarf Implementation:
Each ability class (**Harvester, Digger, Builder**) implement both the abstract methods from the **Dwarf** class, and also their own methods that are specific to that ability. The abstract methods include basic movement methods (left, right, up, down) which uses the getLeft, getRight, getAbove, and getBelow methods from the **Map** class in order to get the tile index (and the tile type) of the blocks around the dwarf given the dwarfs current position. The **Builder** and

Harvester classes have their own methods (fill and dig) which harvest the gold at the location / build over the pit at the location.

Game Analysis:



In terms of Game analysis, the user won't do much other than choosing dwarfs and map layout - after the simulation starts they will be able to see a visual representation of the map in the terminal as it changes, and they will be given a score based off of the amount of moves that the dwarfs made over the course of the game / amount of ticks that passed before harvesting all of the gold nodes. After running a simulation, the logger file will contain a breadth of information corresponding to each dwarf and their movement - this is more useful for debugging program functionality, however, it could be used by the player to possibly figure out a more efficient lineup of dwarfs.

Data:

Input Data:

- Amount of each dwarf (via file)
- Map contents (via file)

Output Data:

- Total moves made

    - Basically a score for the simulation
- Total gold collected
    - Variable for reference when analyzing the simulation
- Efficiency factor (gold collected per move)

Data Importance:
Overall, the only important data to the player will be the total moves made. However, if the player wanted to optimize their simulation by finding the best lineup of dwarfs given their budget, other data may be a help to them. The experiments below help understand the inner workings of the program, and the analysis may help the user create a more successful setup without trial and error.

Experiment Information:
See below each graph to understand each experiment.

Graphs & Experiments:

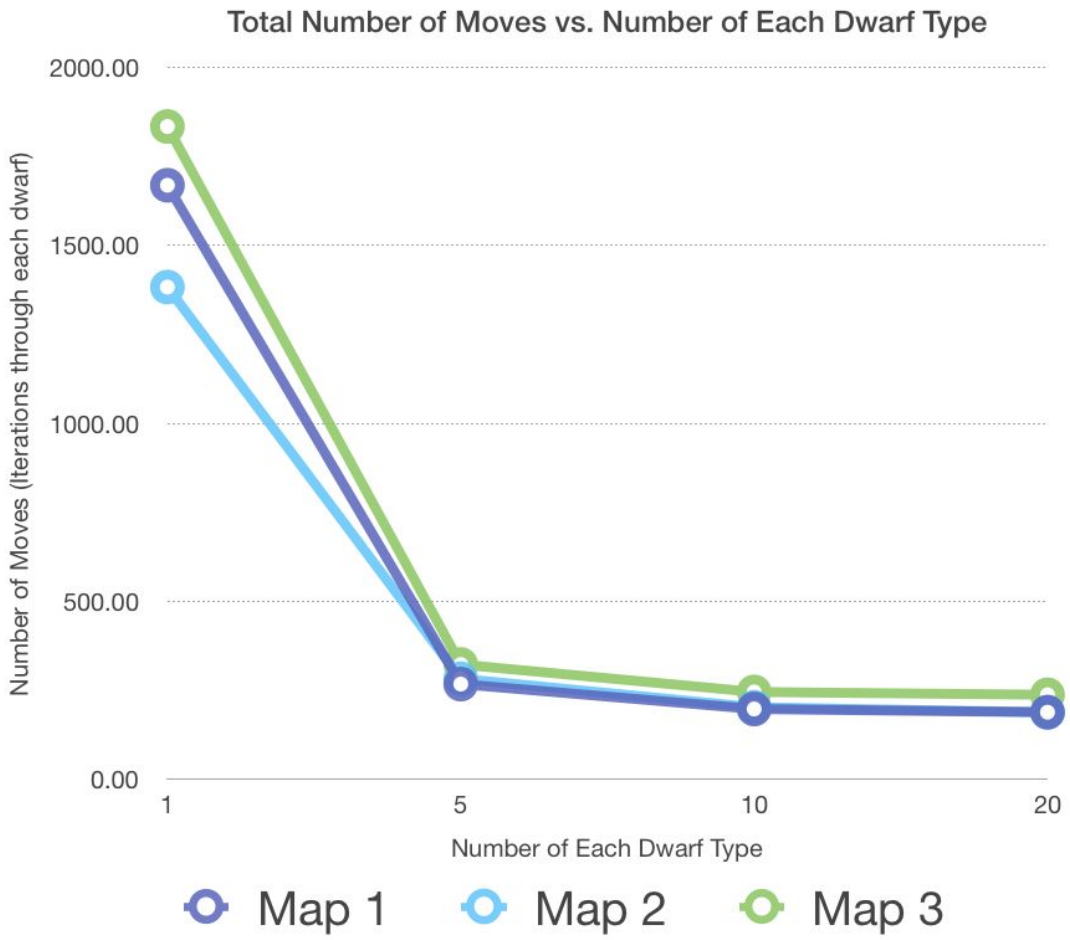
[MAP KEYS]

Map 1 data:
Total gold: 17
Total pits: 25
Total rivers: 5
Total lava: 8

Map 2 data:
Total gold: 25
Total pits: 26
Total rivers: 6
Total lava: 9

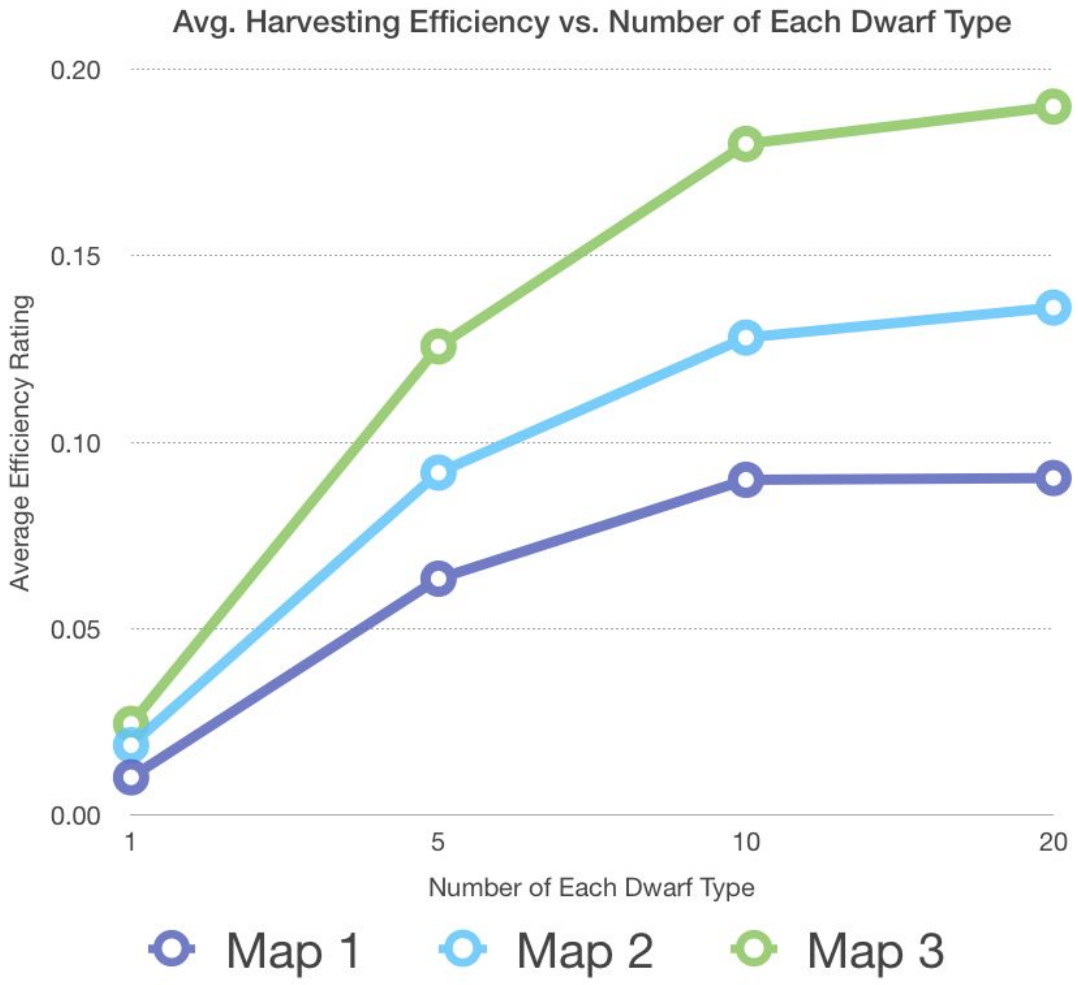
Map 3 data:
Total gold: 45
Total pits: 25
Total rivers: 4
Total lava: 18

Test 1: Amount of Dwarfs vs. Total Moves



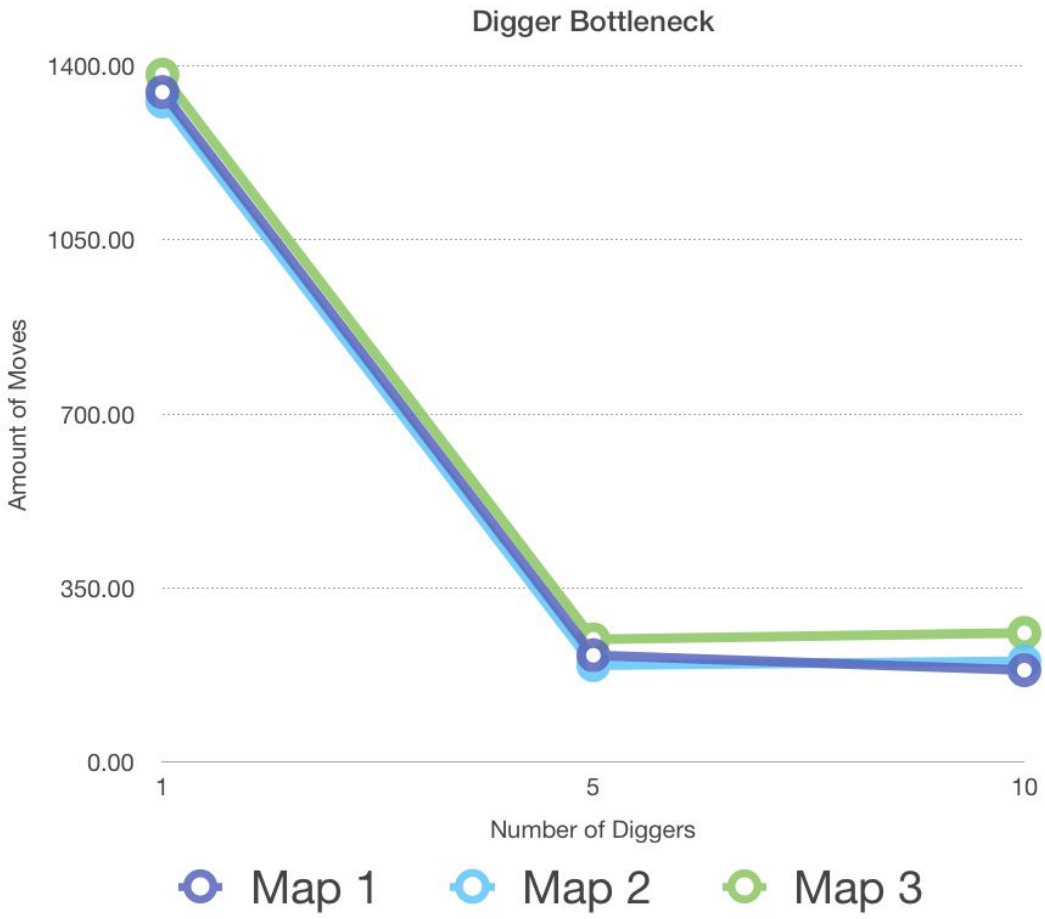
- This graph represents the number of each dwarf (consistent throughout each dwarf type) vs the number of moves needed to harvest all of the gold. The data does seem to represent the movement algorithms well even though it does start to flatten out at 10-20 dwarfs.

Test 2: Efficiency Factor vs. Amount of Each Dwarf



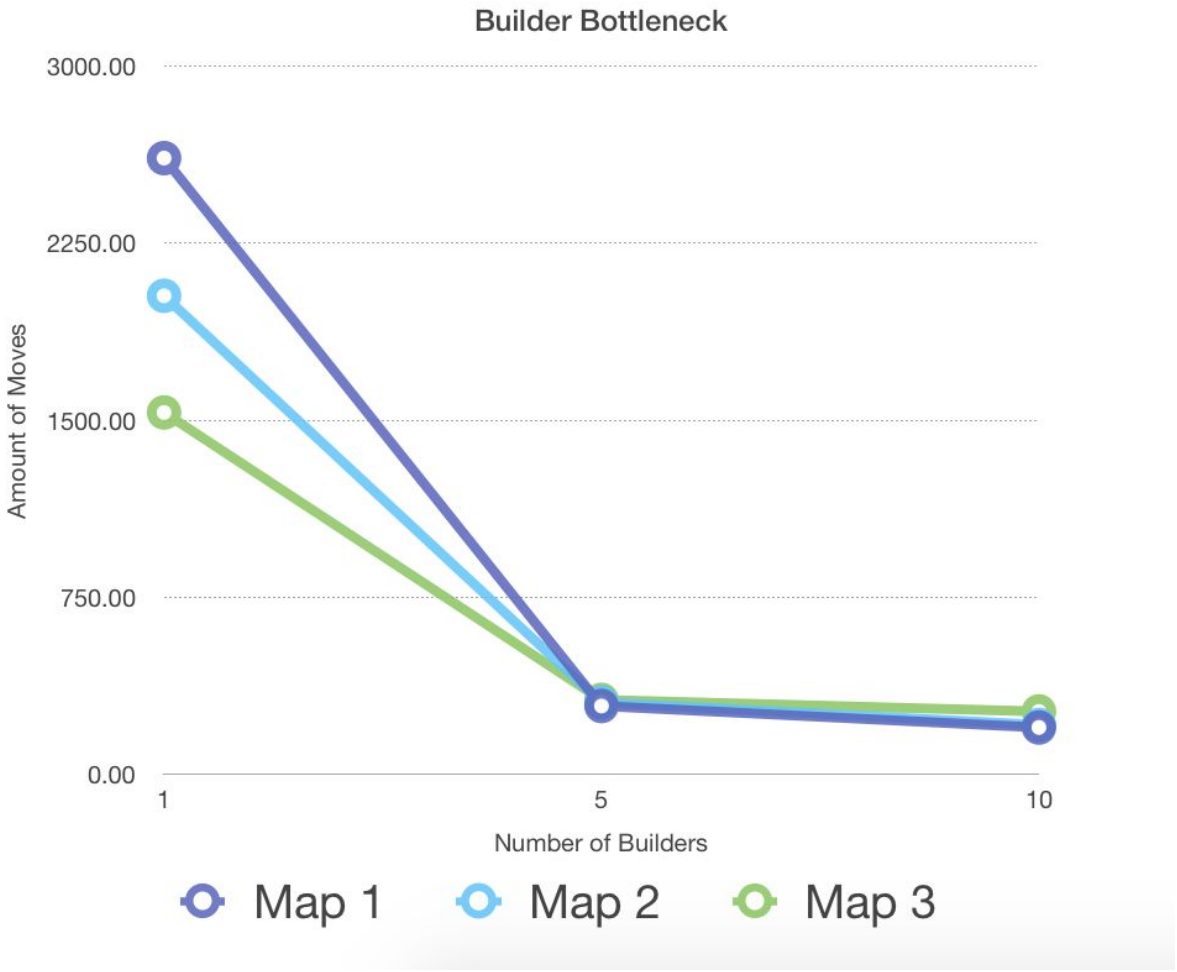
- This graph represents the harvesting efficiency for all dwarfs vs. the number of each dwarf (consistent throughout each dwarf type). Per each map, as the amount of each dwarf increased, so did the efficiency. This makes sense because the amount of moves would be less but the amount of gold will stay the same - which would result in an increase in efficiency.

Test 3: Digger Bottleneck case



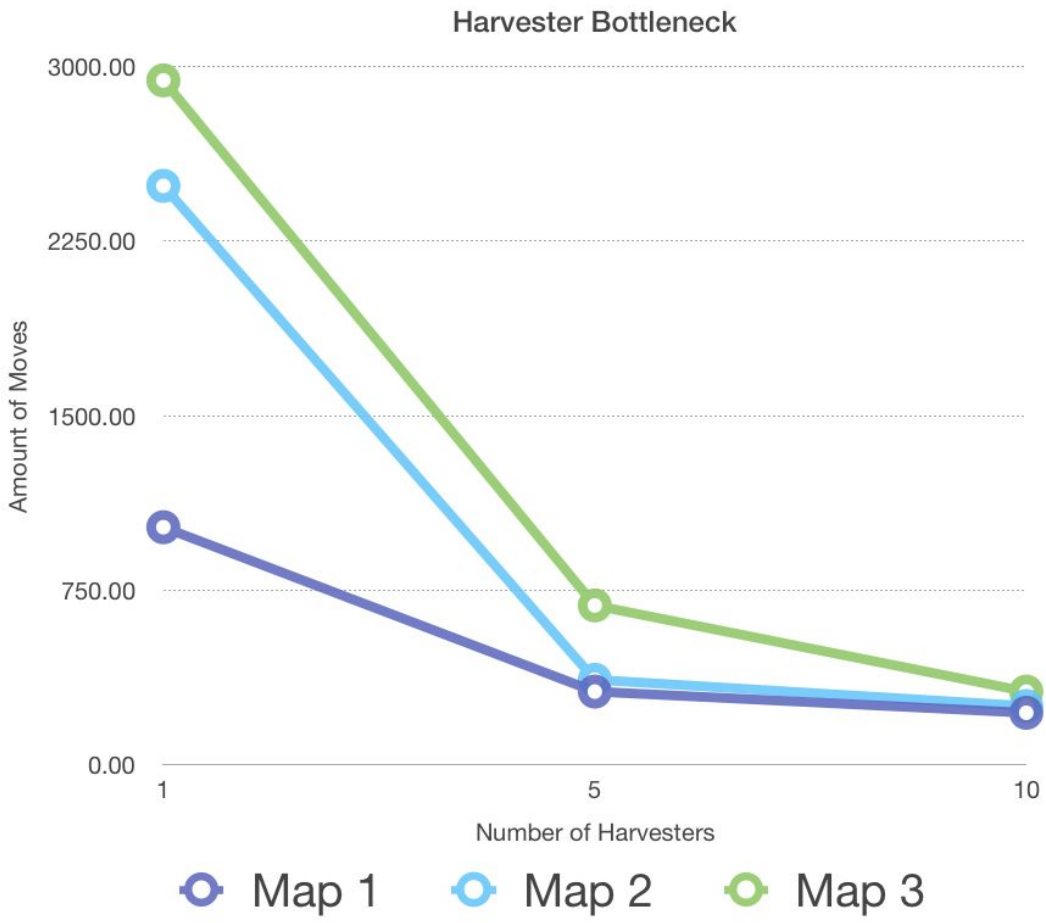
- The digger bottleneck experiment shows the effect that the amount of diggers have on the program runtime / number of moves. All bottleneck cases were conducted with the 20 of each other dwarf type to avoid bottlenecking from other types. From this data we can see that above 5 diggers, there is not much of a change, however, there is a large change from 1-5.

Test 4: Builder Bottleneck case



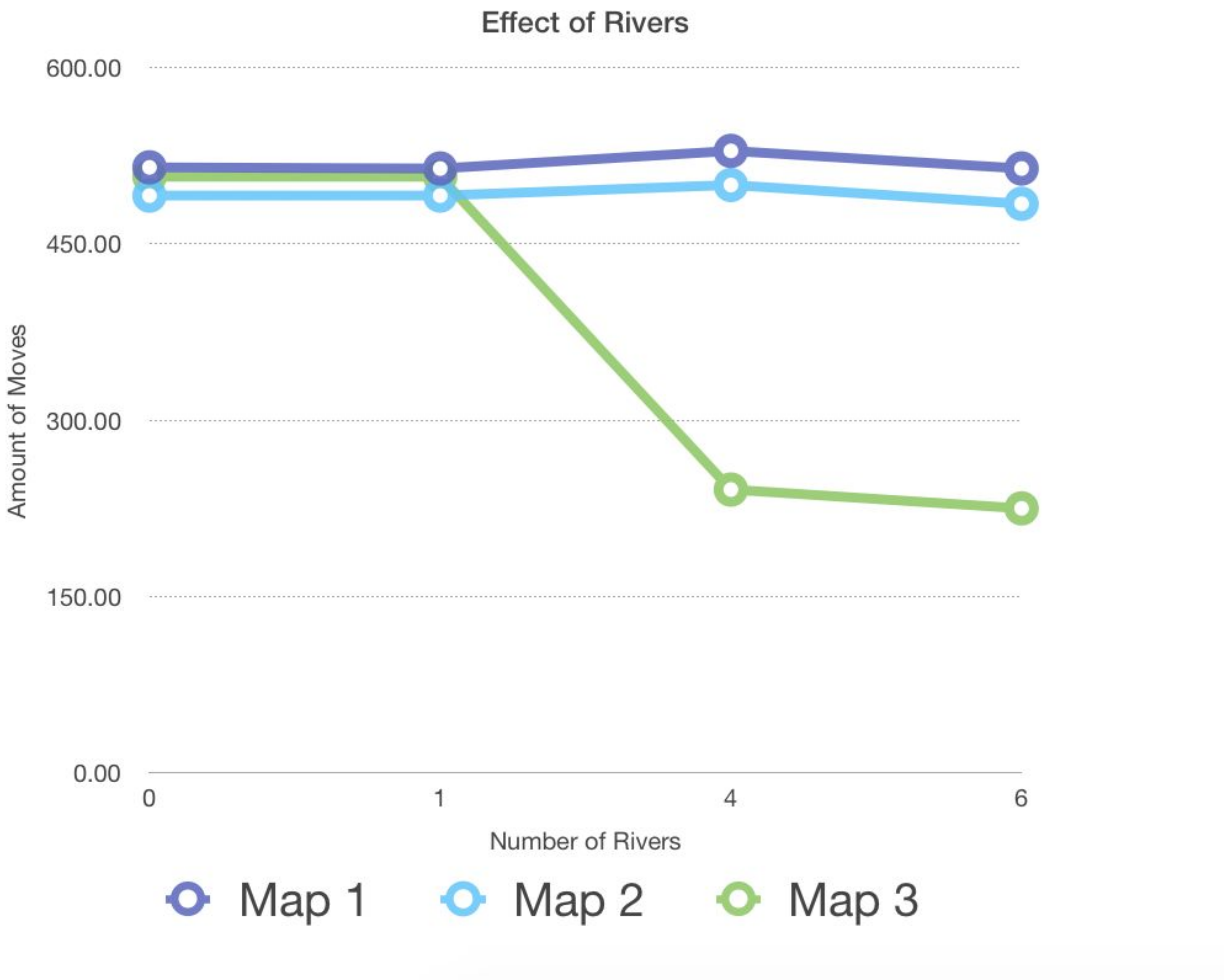
- The builder bottleneck experiment shows the effect of the amount of builders on the game runtime. All bottleneck cases were conducted with the 20 of each other dwarf type to avoid bottlenecking from other types. Considering each map has a similar amount of pits, the data is similar throughout (other than the fact that some maps have more gold / will take longer to complete). At the 5 builder mark, all of the data points are near identical which shows that it could become redundant to have more than that amount of builders.

Test 5: Harvester Bottleneck Case



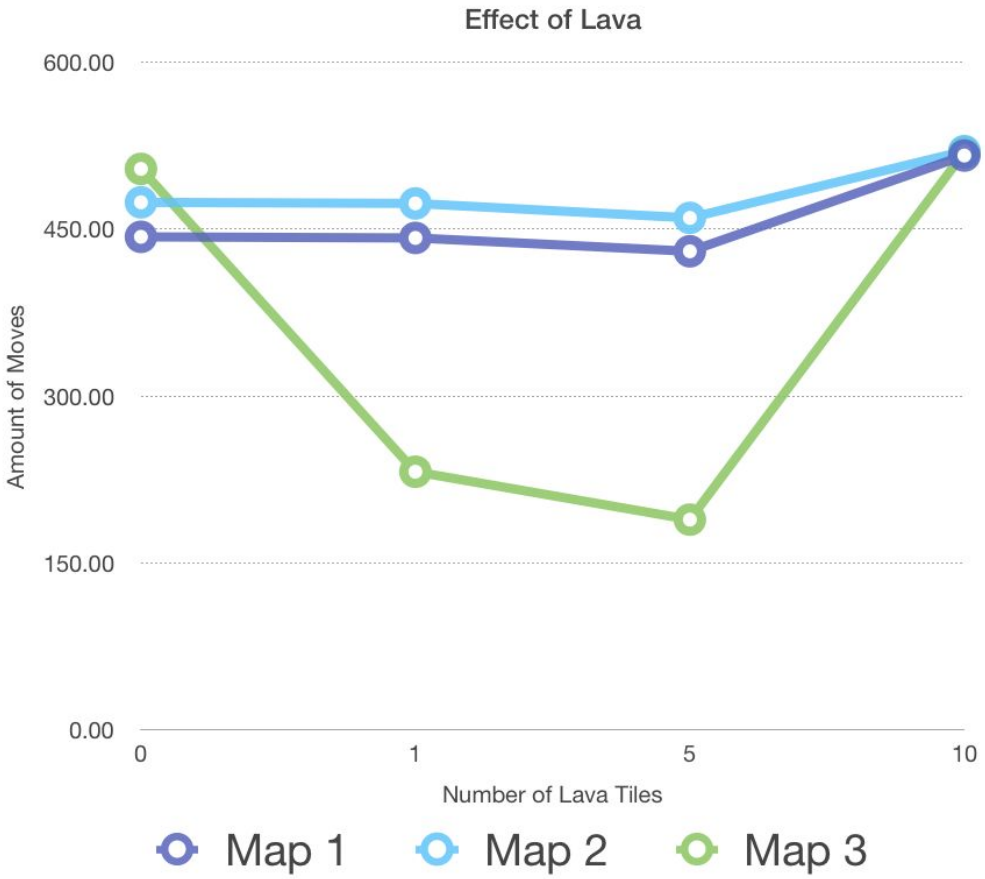
- The harvester bottleneck experiment shows the effect on the game runtime starting with 1 harvester. All bottleneck cases were conducted with the 20 of each other dwarf type to avoid bottlenecking from other types. From this data we can conclude that the amount of harvesters has a large effect on the time that the program will take to finish. However, once the amount of harvesters got higher (around 5-10), the maps with less gold saw a smaller difference in the overall performance.

Test 6: Effect of rivers on total number of moves



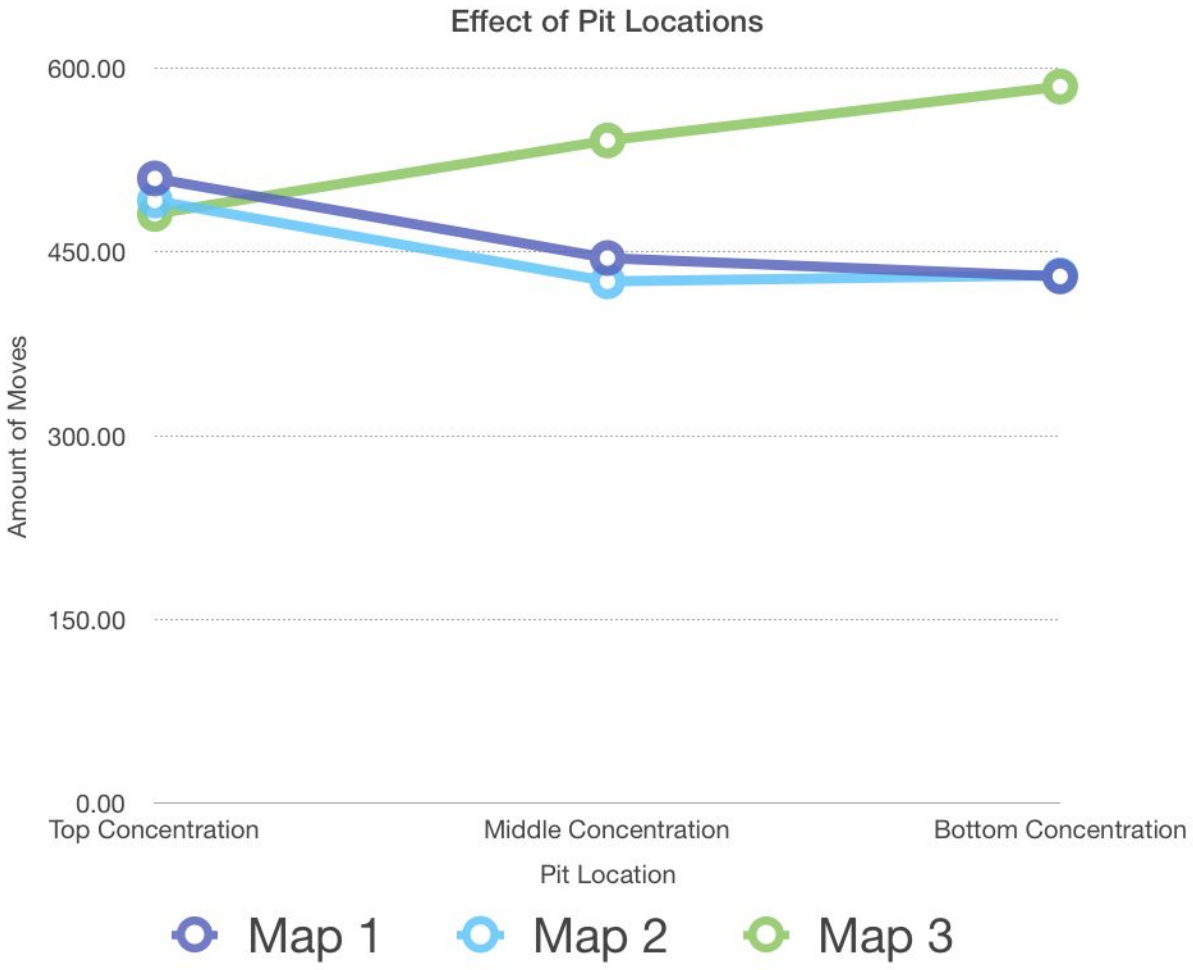
- The experiment was conducted with a generous amount of each dwarf (over 10 each). This helped make sure that there were no bottlenecks present. As a result, the output is relatively random depending on which rivers were removed - this makes sense because it would change the path of the digger dwarf and therefore could change the total amount of moves drastically. This was clear in Map 3 between the 1 river and 4 river data points.

Test 7: Effect of lava on total number of moves



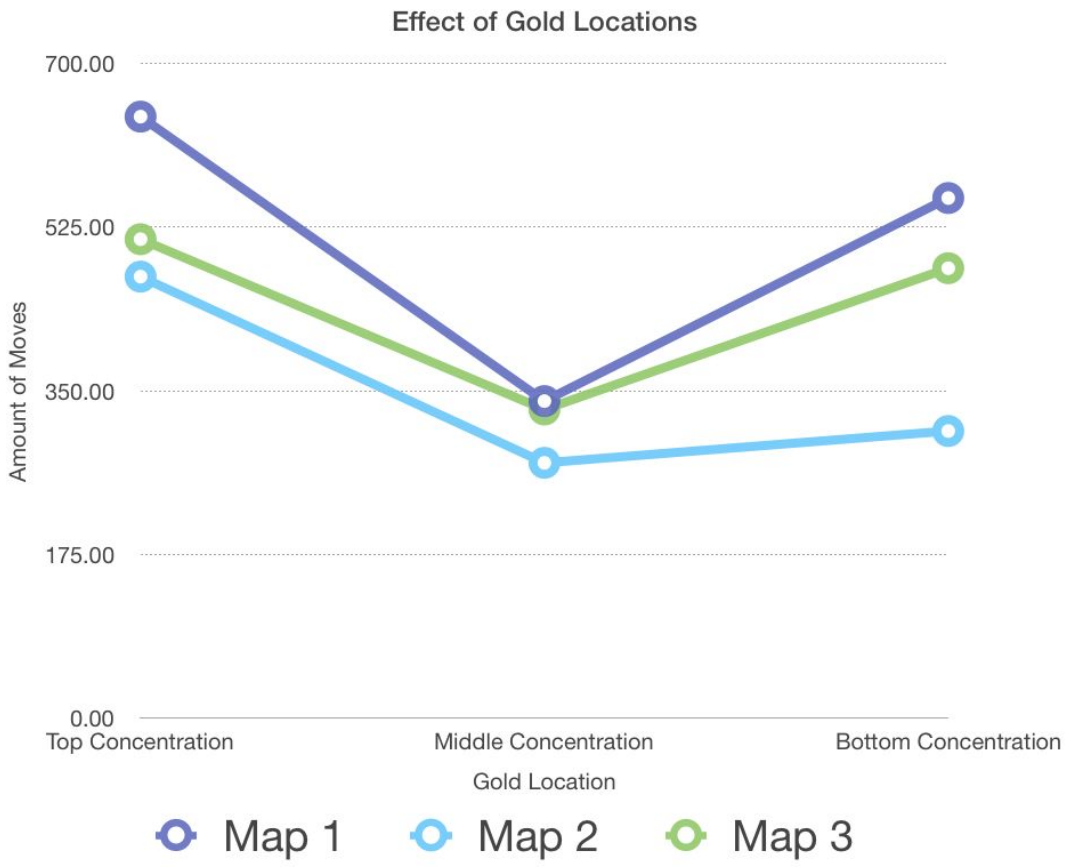
- The experiment was conducted with a generous amount of each dwarf (over 10 each). This helped make sure that there were no bottlenecks present. As a result, the output data seemed to depend on the map given. This would make sense because the absence / addition of a lava node will change the digger's algorithm - in turn creating a totally different outcome in some cases.

Test 8: Effect of Pit Placement:



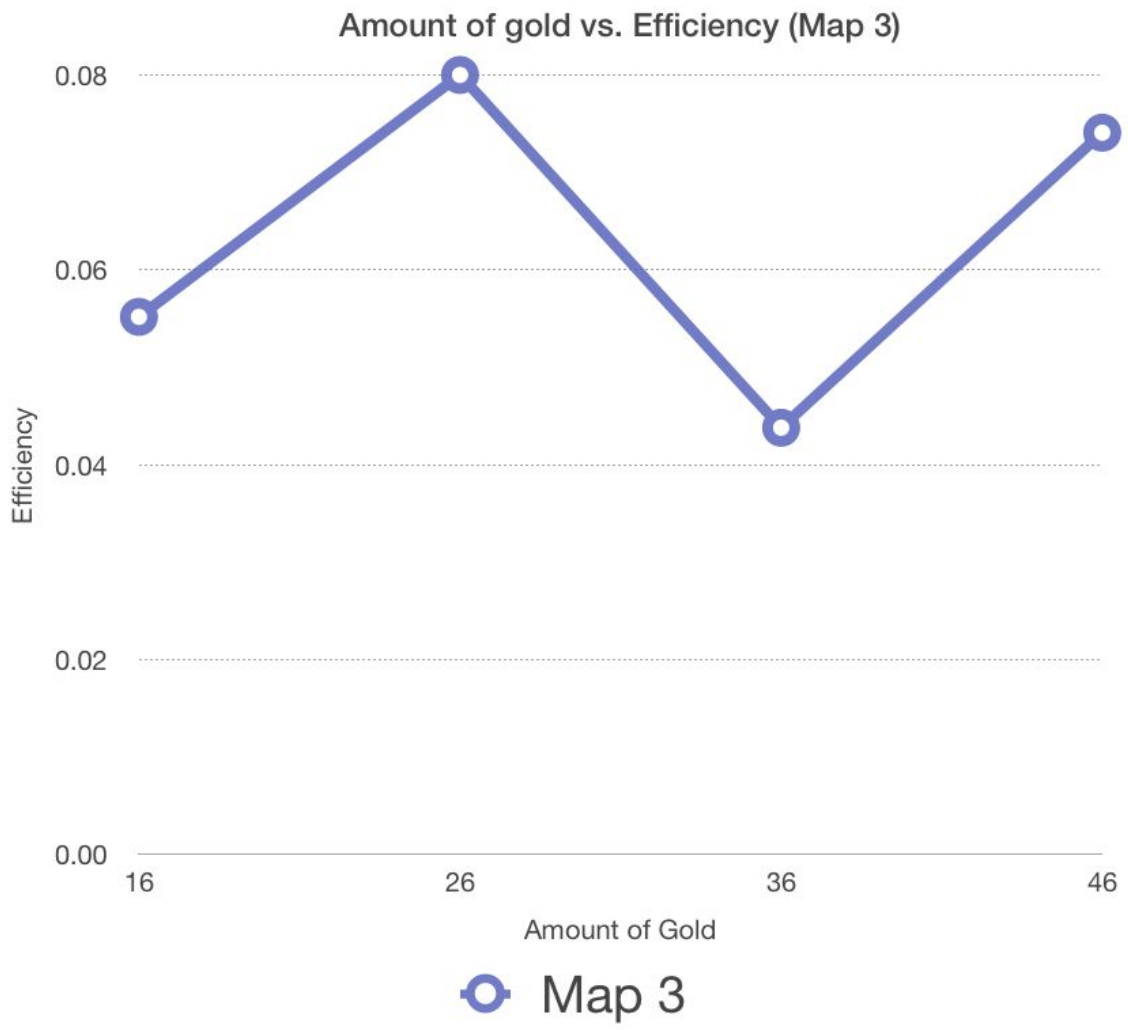
- In this experiment the effect of relative pit concentration was tested. Throughout the tests, the section with the pits (all other sections had pits replaced with dirt) was moved from the top of the map, to the middle, and lastly to the bottom of the map. From the data, there seems to be no relation; the output depends entirely on the layout of the map.

Test 9: Effect of Gold location



- In this experiment the effect of relative gold concentration was tested. Throughout the tests, the section with the gold nodes (all other sections had gold replaced with dirt) was moved from the top of the map, to the middle, and lastly to the bottom of the map. From the results, we can see that the final amount of moves was lower when the gold was concentrated in the center of the map. This gives insight on the movement algorithm's as well as how the dwarfs like to move through the map.

Experiment 10: Amount of gold vs efficiency



- This experiment tested the efficiency as a result of the amount of gold on a given map. This test was conducted using test map 3 as a base adding or removing gold nodes randomly to get the new result. Given the data, there seems to be no clear correlation between the two variables in this setting.

Data Analysis:
If you are a player playing the game, it all depends on the budget that is set for you. From the data that has been collected it is clear that both the diggers and harvesters play an important role in the efficiency of the program.

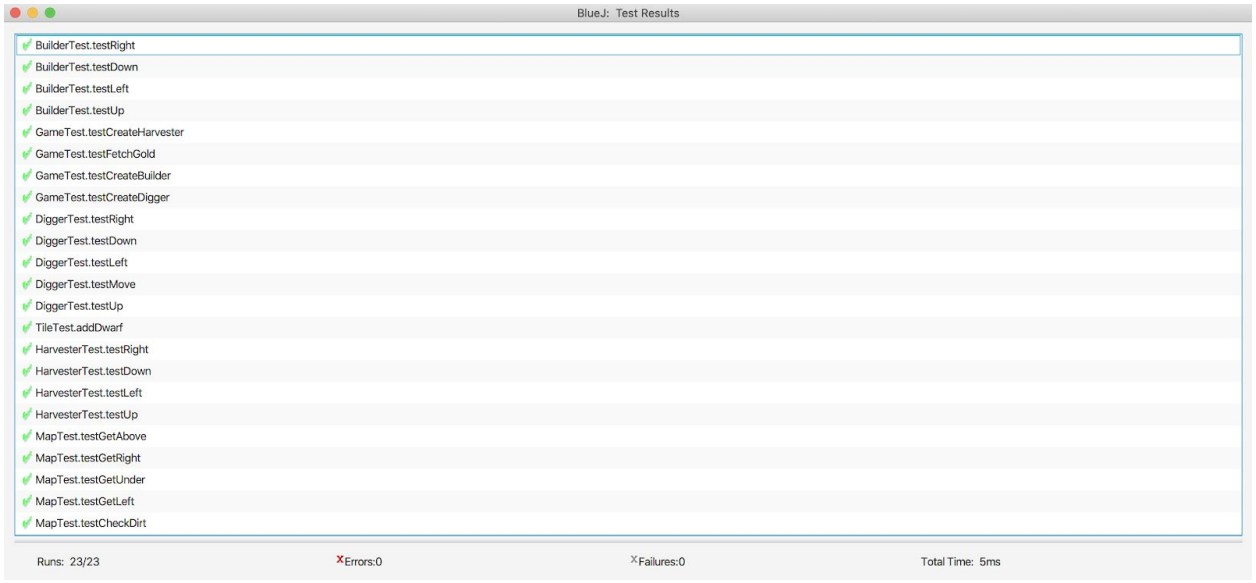
In a situation with a higher budget on a map with only a few gold nodes, having more diggers would expedite the process. However, in a situation where the map has a lot of gold nodes, more harvesters will be able to shave a few more moves off of the total.

In contrast, a situation with a lower budget - having more builders may be more beneficial due to the fact that they hit their plateau earlier. Harvesters are still needed, and may be more beneficial in cases with more gold, but the diggers should be a first pick.

In both situations diggers play a role, but their value is totally up to the setup of the map in relation to the locations of each pit. It MAY help out in some cases, however, they are not nearly as important as the diggers or harvesters because the pits can always be avoided - they might just open up more efficient paths for the diggers / harvesters in certain situations.

Keep in mind the prices of each dwarf type:
Digger: 300
Harvester: 500
Builder: 200

Unit Testing:



- BlueJ unit test result window

Trouble report:
This section is not applicable because the program works as expected.

References:

Tree Map. (2019, September 11). Retrieved from https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html.

Priority Queue. (2018, October 6). Retrieved from https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html.

Stack. (2018, October 6). Retrieved from https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html

Zybooks. (n.d.). Retrieved from https://learn.zybooks.com/zybook/LAFAYETTECS150PfaffmannFall2019.

Pausing Execution with Sleep. (n.d.). Retrieved from https://docs.oracle.com/javase/tutorial/essential/concurrency/sleep.html.

Reversing a Stack. (n,d,). Retrieved from https://www.careercup.com/question?id=12689669