Dwarf Mining Game Simulation

Tafita Rakoto

November 10,2019

This project consists of a dwarf mining game in which dwarves search and collect gold. Each one of them will have their own abilities whether building a bridge, digging or searching for gold. An initial amount of gold will be assigned to the player to hire dwarves and their hiring price changes based on their abilities. The game will be over when all the gold is found or the dwarves die from the different obstacles.

1. Features

This program will simulate the overall interaction between every participant of the game in order to achieve the goal of the game. Each one of the dwarves has their own specific abilities which make them dependant on achieving the common goal . A GUI is used to visualize the overall interaction of the game. It is also used to understand the program behind. Each location in the GUI will be presented as a grid and every participant will be presented as a specific form.

2. Approach

Criteria

The game has a predefined rules that are described in the project description:

- The game will use an event driven simulation to run the overall program.

- The game will start with an initial amount of gold
- There will be a danger for the dwarves such as : rock , pit , underground river or lava.
- The game should use a reversible memory to track their way back
- The dwarves can mark location
- The dwarves use an indexing mechanism
- A dwarf can ask another dwarf for information

3. Assumption

Based on the project description, few assumptions were made before the development:

- 4 types of different dwarf will be implemented : searcher,digger, carrier and builder. Their hiring price will differ based on their respective ability
- Searchers are in charge of searching for the gold and calling the different types of dwarves. It is the most expensive. It costs 10 gold to hire one.
- Digger is in charge of digging the tunnel when a searcher finds a rock . It takes 5 gold to hire a digger
- Builder is in charge of building the bridge when the searcher makes a call . It takes 8 dollars to hire a builder. It has a high qualification and deserves a higher pay but in this game, the searcher has even more important roles than the builder. So, it is fair to assume that it is not the most expensive.
- Carrier is in charge of taking the gold from the digging place and carries it back to the initial point which is the top upper left. It is the least expensive as it has just to bring stuff. It takes 3 gold to hire one of them.
- It is assumed that one gold in a specific location counts 10.

- We assume that the dwarves can never get tired, especially for the carriers. In real life, carrying a lot of gold can be tiring !
- The time taken to build a bridge depends on the number of builders.

4. Description of the classes

Different types of classes were implemented in this project, they are interconnected with 2 other classes.

Dwarf: It holds the common characteristics that all of the dwarf can make. This class will have 4 subclasses: digger, builder, carrier and searcher
The common characteristic of those dwarfs is for example their change in location system. They implement the same method that converts their raw coordinates into the GUI map coordinate. They all have their reversible memory system. They can also mark their path.
Searcher: Being the most expensive of all dwarves, it has more methods on it. The method that calls the 3 other for example.
Carrier: The carrier method has that allows it the ability to join a specific position based on the information sent by the searchers.

Builder : has a method that only allows them to build a bridge which can be translated by removing a pit

Digger: has method that allows them to remove a rock

Obstacle : it regroups all the common characteristics of each obstacle : location system used, the conversion of the raw location to map location. What differs between the different obstacles are their color. That differentiation helps to visualize the behavior of each obstacle when runned in the GUI.

Gold: the class gold is a little similar to obstacle. However by design choice, they were not connected to each other because in reality, dwarves are looking for gold and are avoiding obstacles. It would be inappropriate to combine but we can do that.

Map: the map class holds the place where all the process of data exchange is performed. It will be a branch that provides the user interface graphic.

Clock: Is where all the conditions for every condition take place to realize the overall programs.

Budget: it is the class the handle the earning and expense

5. Program description :

By default , the program has no fixed number of builders, carriers , searchers and diggers. The user will need to enter that information in order to run. It will read the input entered .
After all the initialisation, the time clock starts. The searchers will start to move. The searchers only know one type of movement: Zig Zag : start from the leftmost part and go straight until the rightmost part and go down and do the opposite.

As the searcher moves, a test is made to see whether or not the searcher step on the lava or river. If that is the case, they will be removed. Money will be automatically deducted from the budget to pay the dwarf' s family if any of them die.
Other conditions are also checked if the searcher has found Gold or not. When the gold is found, a call function is called to call the carrier in the location.

Similarly, the conditions are also made to check if the searchers found any rock or pit . The builder is called if we have to build a bridge for the pit while a digger is called if it is for digging a tunnel.

The predefined budget on the game was set to 100 golds.

GUI interface:

The GUI interface is very important to understand how the overall program works. The interface will look like the following

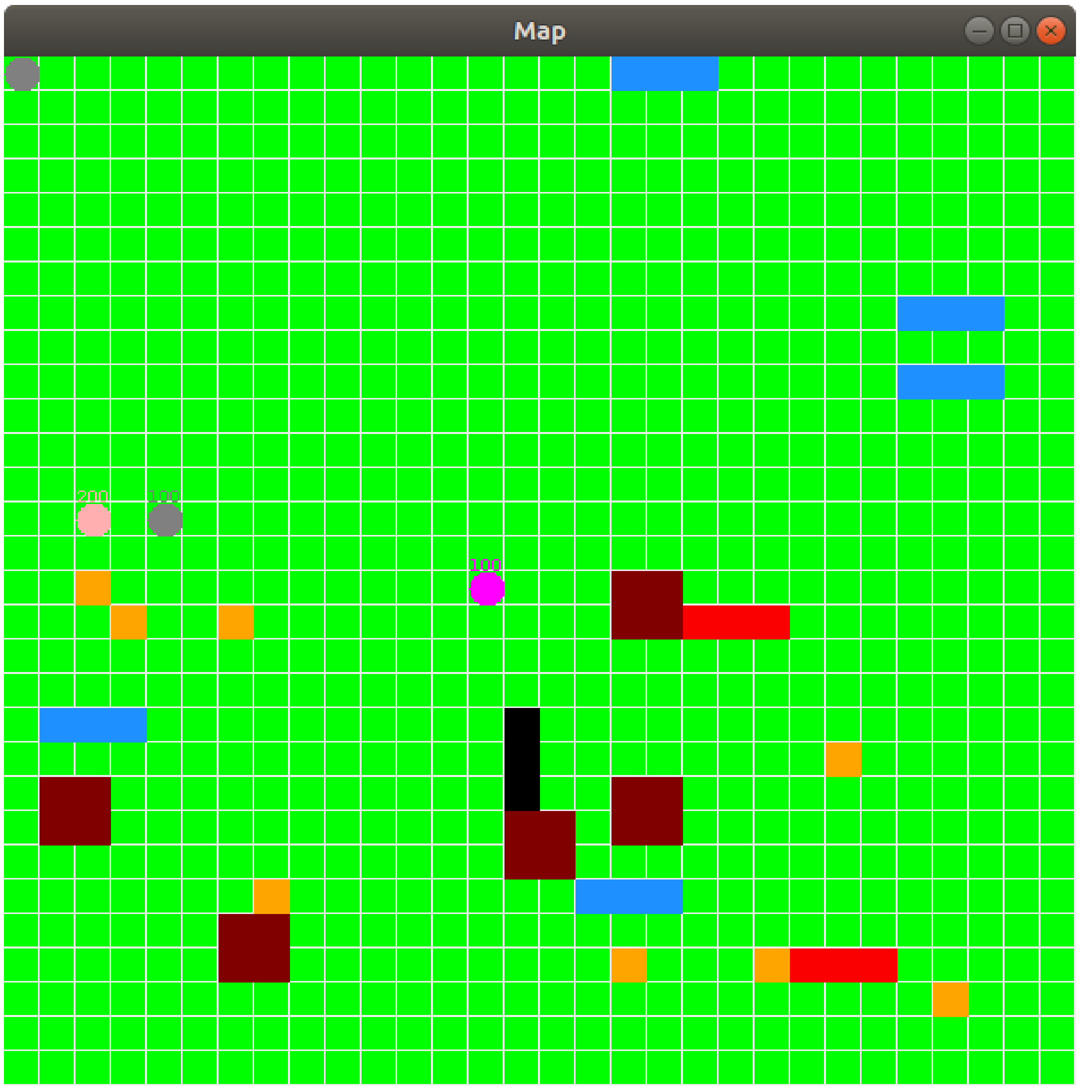

Figure: User interface map representation

Each grid is represented by the green box. If it is filled with other elements, that means that something else occupies the space. The legends for that map is the following:

- Orange box: represents a gold
- Black box: represents a pit
- Brown box: represents a rock
- Blue box: represents a river
- Red box: represents a river
- Purple circle: represents a dwarf searcher
- Pink circle : represents a dwarf carrier
- Digger circle: represents a dwarf builder

6. Use of data structure.

As part of the learning in this class, a data structure should be used to demonstrate their use in a real project. 3 different types of data structure was used for a very specific task.

- **Binary Tree**

Each grid possesses a binary tree. The binary tree is used to mark who passed in a specific place of the map. The dwarves have the ability to mark those trees. In this particular project, I implemented the binary tree in each of the grid no matter if there is an obstacle in that place or not. It was possible to initialise the tree in the constructor of the obstacle . That would make more sense the most for an object oriented project. However, in this particular case, if a bridge is built over a pit, that pit itself can be considered as a new position that can be marked. And, in my particular case, building a bridge over a pit means removing the pit. Therefore, the binary tree has been already assigned in that specific position. A TreeSet class was used in order to achieve it . The website GeektoGeek gives a good explanation of how it works.

- **Priority Queue**

The priority queue was used to handle the searchers call. Since it is important to use the carriers efficiently, it was important to allow them to take in the closest place first. In order to realize that , a priority queue is made to store the location of the gold found in a priority order. The most important data is the gold nearby's position. Therefore , the first carrier that receives the call will start the collection with the gold nearby (when you peek the priority queue, you will get the closest found gold) . A comparator function was build in order to identify the position of the gold based on its location index on the game's memory.

- **Stack**

Carriers are the ones that use stack the most. Their job is to take the gold in found location and bring it back to the place where every dwarf started to move. They can only take another gold after finishing that part. The Stack on Java API was used in order to realize that.

7. Data analysis

Effect of hiring multiple dwarves on final revenue and time

The first experiment realized was to understand the effect of increasing the hiring of multiple dwarves on the final budget and the time spent . The table below represents the experiment made.

Initial budget	N° Searcher	N° Carrier	N° Builder	N° Digger	Final Budget	Time spent
100	2	2	2	2	208	1355
100	3	3	3	3	172	1047
100	4	4	4	4	136	1147

Table: Experimentation on increasing dwarf hiring

Note: 18 golds were located on the map in this experimentation

Interpretation: The more you hire dwarves for each respective abilities, the lesser the total budget revenue at the end will be. However , that increase on dwarves also decreases the time spent to find gold but it is not efficient as our goal is to make profit.

Average time and budget

In this second experiment, the idea was to understand which dwarves with a specific abilities need to be hired in order to maximise profit in terms of money and time spent. The number of the dwarves with different abilities were tested based on the different random seed combinations . The random generator of the map will provide different configurations of maps. However, the number of kept players obstacles were kept the same. An average of final budget and time spent were calculated in order to achieve that. The result is presented in the table below.

Initial budget	N° Searcher	N° Carrier	N° Builder	N° Digger	Average Final Budget	Average Time spent
100	2	1	1	1	208.557	1355.146
100	1	2	1	1	241.245	1583.564
100	1	1	2	1	239.002	1513.365
100	1	1	1	2	236.245	1583.413

Table: Permutation of the number of dwarves

Based on the permutation of their number, having extra dwarves with specific abilities provide different results in terms of time and final budget. For example, having a digger allows us to gain about 236 gold which is small compared to 239 golds for the rising number of builders. Yet, it is also very time

consuming as it takes 1583 time clock to make the overall experiment while on the other hand with an increase of builders, it only takes 1513. Therefore it is not preferable to invest in a digger.
On the other hand by increasing the searchers number, we can learn from the table, we can lose up to 30 golds compared to the others (the other average final budget : 241.245,239.002,236.245). However, the reduction of time is very significant compared to others.

8. Limitation

There are a lot of limitations in this program that need to be discussed.

- Searchers zig zag motion

The searchers zig zag motion is a very low level way to look for the gold in a way that they cannot avoid the obstacle. They always need the other dwarves to make way for them.
For example, if there is a pit in front of them, they cannot escape it and try another way.

- Zig Zag motion scan

In order to go down, the zig zag motion allows us to test if another dwarf has already passed in the specific line. If yes, a recursion is made so that will test the line after. As a result, when initiating new searchers, the last one will join directly the position where no dwarf has ever passed

- Only searchers can die

One of the failures of my program is that only the searchers can die directly from the obstacles which can contradict the game criteria. Due to the limitation of time and the original design , such implementation could not have been made before the deadline.

9. Conclusion

It is hard to conclude how to be the winner in this exam because the hiring system will depend on how fast you will achieve the goal and at what cost you will rik for that. There is no choice in between. However, there is a discovery that we found in the experimentation that can be helpful in the game. For example buying less digger has a positive effect on price and time. However, the limitation of experimentation might be the fact that the number obstacles in the experiment were kept the same.

One of my personal problems in this game was that I had a bad design in terms of data organization at the beginning. As the projects went on , I realized that it was hard to return back because my program relied on each other. As a result, debugging took most of my time in this program. Rather than working on a program unit testing, I relied very much on the GUI to test every method.

References

GeektoGeek(2019) Tree Set in Java, Retrieved from https:
https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html

Oracle(2019) Stack Java, Retrieved from https:
https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html
Oracle(2019) Priority Queue, Retrieved from https:
https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html