Data Structures and Algorithms

Taxi Pooling

Computer and systems department
Third Year

2015-2016

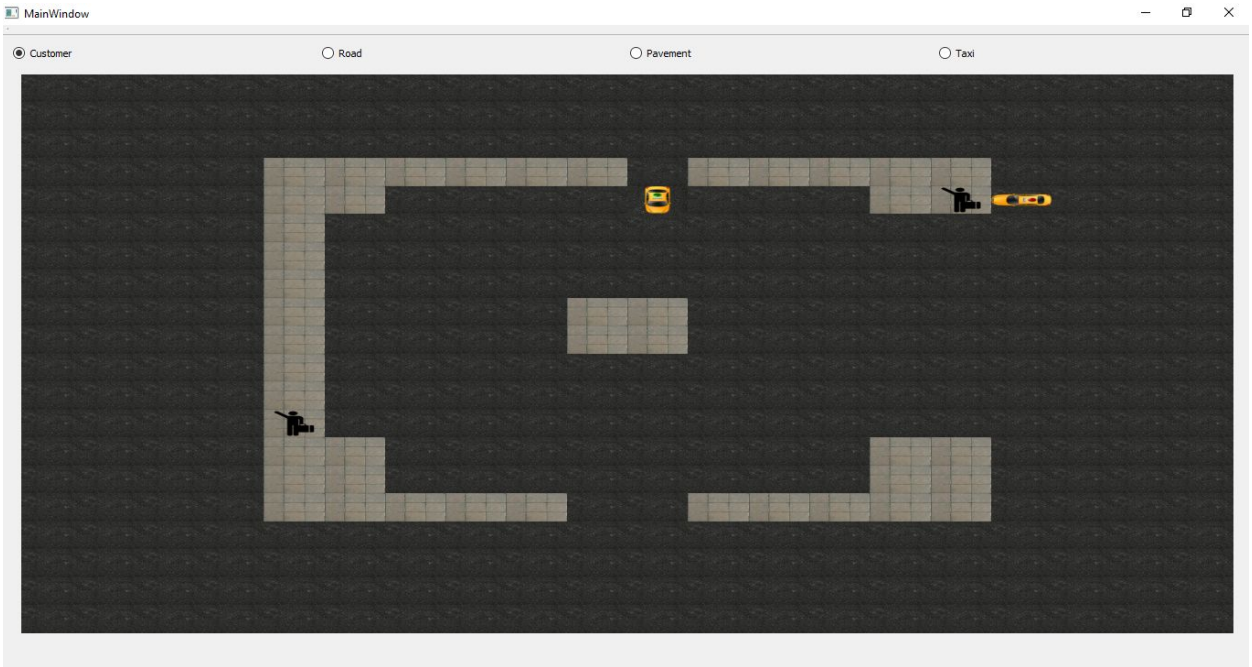
Team Members:

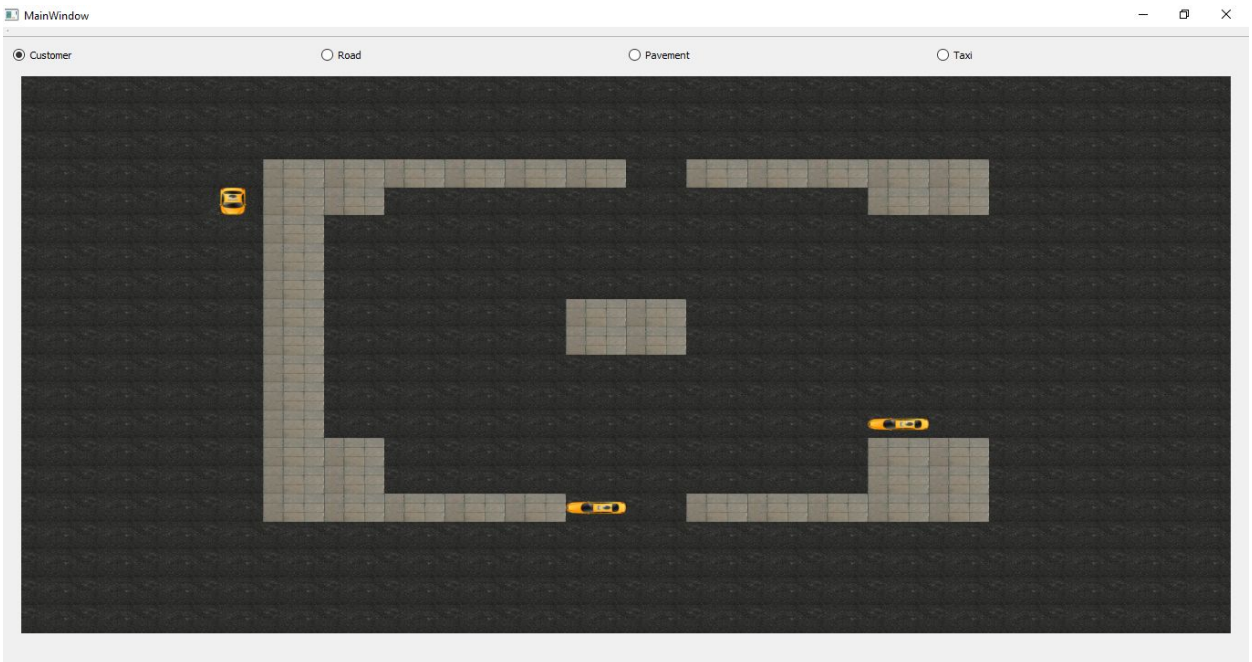
Ahmed Mostafa Soliman

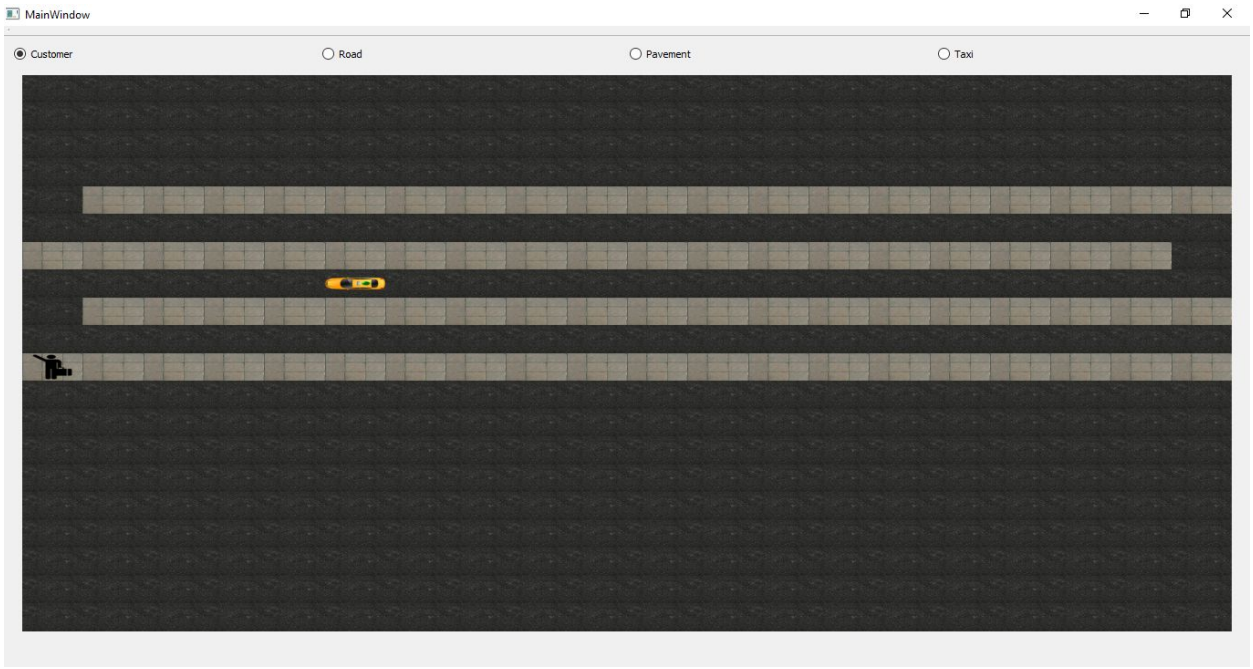
Amr Mohamed Hosny

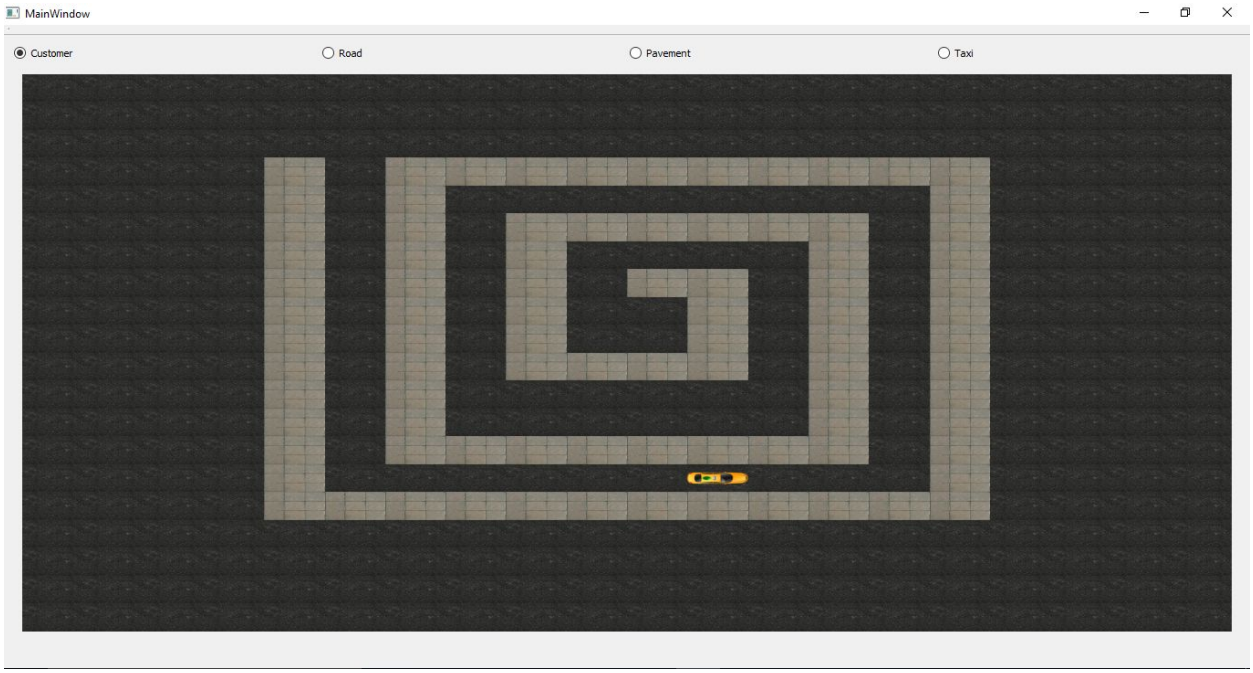
Mohamed Ahmed Thabet

1. Screenshots






MainWindow
Customer
Road
Pavement
Taxi


MainWindow
Customer
Road
Pavement
Taxi

2. Main Data Structures / Algorithms

- All the taxis are stored in a **vector**.
- New customers are added to a waiting **queue**.
- The grid is stored as a 2D array.

- **Breadth First Search** is used to find the shortest path from each vacant taxi to a customer, among them, the nearest one is assigned to the customer, then **Breadth First Search** is performed again to find the shortest path to the customer's destination.
- The path itself is stored as a stack attribute in each taxi object to ease the construction of the path/simulation process after finding the target in the BFS, the steps are stored in the stack as characters ('u', 'l', r', 'd').

- Each cell is assigned a number, this number is mapped to a pair of two integers representing the coordinates of the cell, this is done to aid sending the click signals through the Qt signal mapper.

3. Main Code Parts

- **Implementation notes**
  - The project is built with Qt and C++.
  - Various entities in the project are modeled as classes, including Cells, Customers and Taxis.
  - GUI updates are done with the help of a dedicated thread to avoid freezing.

- **Find the shortest path between a taxi and a customer**

```
std::stack<char> Taxi::findPath(int desx,int desy)
{
  std::stack<char> taxi_path;
  std::vector<std::vector<cell*>> v=*Grid::getgrid();
  int numRows=v.size();
  int numCols=v[0].size();

  std::vector<std::vector<char>>par(numRows,std::vector<char>(numCols,'s'));
  par[x][y]='T';

  std::queue<std::pair<int,int>>BFS_Q;
  BFS_Q.push(std::make_pair(this->x,this->y));

  while(!BFS_Q.empty())
  {
      int curx = BFS_Q.front().first,cury=BFS_Q.front().second; BFS_Q.pop();
      if((abs(curx-desx)+abs(cury-desy))==1)
      {
          while(!BFS_Q.empty())
              BFS_Q.pop();

          if(customerLocationx==-1 ||customerLocationy==-1)
               taxi_path.push('x'); // a char that is used to delay the taxi for a
moment when reaching the
                                        //customer for the first time
          while(par[curx][cury] != 'T')
          {
              if(par[curx][cury]=='u')
              {
                 curx++;
                 taxi_path.push('u');
              }
              else if(par[curx][cury]=='d')
              {
```

```
                curx--;
                taxi_path.push('d');
            }


            else if (par[curx][cury]=='l')
            {
                cury++;
                taxi_path.push('l');
            }
            else if(par[curx][cury]=='r')
            {
                cury--;
                taxi_path.push('r');
            }
         }
        return taxi_path;
     }

    if(curx+1 <numRows)
    {
        if(par[curx+1][cury]=='s' && v[curx+1][cury]->isRoad()){
            BFS_Q.push(std::make_pair(curx+1,cury)); par[curx+1][cury]='d';
        }
    }
    if(curx-1 >=0)
    {
        if(par[curx-1][cury]=='s' && v[curx-1][cury]->isRoad()){
            BFS_Q.push(std::make_pair(curx-1,cury)); par[curx-1][cury]='u';
        }
    }
    if(cury+1 <numCols)
    {
        if(par[curx][cury+1]=='s' && v[curx][cury+1]->isRoad()){
            BFS_Q.push(std::make_pair(curx,cury+1));par[curx][cury+1]='r';
        }
    }
    if(cury-1 >=0)
    {
        if(par[curx][cury-1]=='s' && v[curx][cury-1]->isRoad()){
            BFS_Q.push(std::make_pair(curx,cury-1));par[curx][cury-1]='l';
        }
    }

   }
  return std::stack<char>();
}
```

Grid singleton class

```
class Grid
{
private:
  static std::vector<std::vector<cell*>> * cells;
  const static int numRows=20;
  const static int numCols=20;
public:
  static std::vector<std::vector<cell*>> * getgrid();
};
```

```
#include "grid.h"
std::vector<std::vector<cell*>> * Grid::cells = nullptr;
std::vector<std::vector<cell*>> *Grid::getgrid() {
 if (!cells)
 {
   Grid:: cells=new std::vector<std::vector<cell*>>
(numRows,std::vector<cell*>(numCols));
 }
 return cells;
}
```

Add a new taxi

The program has a dynamic array (vector) of taxis that tracks all the taxis' activities.

```
void MainWindow::addTaxi(int x,int y)
{
   if((*cells)[x][y]->isRoad())
   {
       taxis.push_back(new Taxi(x,y));
       (*cells)[x][y]->setState(cell:: VacantTaxi);
   }
}
```

- **Search for vacant taxis to serve waiting customers then make all the taxis move in their defined paths to their destinations**

```
void MainWindow::onWakeUp()
{
   while(!customers.empty())
   {
       Customer * curcustomer=customers.front();
       int bestTaxi=-1;
       std::stack<char>curpath,bestpath;
       for(int i=0;i<taxis.size();i++)
       {
           if(!taxis[i]->isOccupied())
           {
curpath=taxis[i]->findPath(curcustomer->getCurrentX(),curcustomer->getCurrentY());
               if(bestpath.size()==0 && curpath.size()>0)
               {
                   bestpath=curpath;
                   bestTaxi=i;
               }
               else if(bestpath.size()>curpath.size() && curpath.size()>0)
               {
                   bestpath=curpath;
                   bestTaxi=i;
               }
           }
       }
       if(bestTaxi!=-1)
       {
           customers.pop();
           taxis[bestTaxi]->setPath(bestpath);
taxis[bestTaxi]->setCustomer(curcustomer->getCurrentX(),curcustomer->getCurrentY(),
curcustomer->getDestinationX(),curcustomer->getDestinationY());
       }
       else
       {
           break;
       }
   }
   for(auto t:taxis)
       t->move();
}
```

- **Add a new customer**

  The program has a queue of customers that provides a FCFS service algorithm.

```
void MainWindow::addCustomer(int curx,int cury,int desx,int desy)
{
   customers.push(new Customer(curx,cury,desx,desy));
}
```

4. Future Work

- Add some traffic
- Handle multiple taxis in one cell
- Substitute the grid with a real map
- Calculate the fair
- Improve the algorithm to maximize the throughput

5. Class Diagram