به نام خدا

گزارش پروژه میانی درس برنامه سازی پیشرفته (AP)

سپهر قمری 9623090

Github : https://github.com/sepehrgh98/Maze.git

استاد درس : استاد امیر جهانشاهی

بهار 1400

ساختار این پروژه از یک کلاس به نام **MazeGUI** تشکیل شده است که کلاس **Node** به صورت کلاس تودر تو تشکیل شده است.در این پروژه هرخانه در ماز یک نود در نظر گرفته شده که دارای ویژگی های خاص خود می باشد.مسلما هر ماز که ساخته شود ، شامل تعدادی نود خواهد بود که تعداد آن توسط کاربر وارد می شود.

کلاس Node

```
class Node
{
public:
    Node(size_t _id);

    //class methods
    void Update_availableDirections();
    void Update_availableDirections_end();
    void set_dir_flags(std::shared_ptr<Node> select);

    //logical fields
    bool visited = false;
    bool end_visit = false;
    bool is_start = false;
    bool is_end = false;
    bool is_true_dir = false;
    std::vector<std::pair<std::shared_ptr<Node>,int>> availableDirections;
    std::unordered_map<const char*,std::pair<std::shared_ptr<Node>,int>> Directions{
        {"North" , std::pair<std::shared_ptr<Node>,int>{nullptr,0}},
        {"South" , std::pair<std::shared_ptr<Node>,int>{nullptr,0}},
        {"East" , std::pair<std::shared_ptr<Node>,int>{nullptr,0}},
        {"West" , std::pair<std::shared_ptr<Node>,int>{nullptr,0}}
    };
    std::shared_ptr<Node> parent{nullptr};
    std::shared_ptr<Node> end_parent{nullptr};
    size_t id;

    //graphical fields
    QGraphicsItemGroup* recGroup;
    QGraphicsItemGroup* mainNode;
    QGraphicsItemGroup* top;
    QGraphicsItemGroup* right;
    QGraphicsItemGroup* left;
    QGraphicsItemGroup* down;
    QGraphicsEllipseItem * Me{nullptr};

};
```

همان طور که در تصویر بالا مشاهده می شود ، هر نود دارای یک id است که از طریق کانسراکتور مقدار دهی میشود.هر نود دارای Directions از جنس unordered_map می باشد و در نتیجه دارای چهار جهت اصلی است. در ولیو هر جهت یک متغیر از جنس pair تعریف شده است که عضو اول به پوینتر نودی اشاره میکند که درآن جهت خاص قرار دارد و عنصر دوم اینتیجری است که وضعیت بین دو نود از نظر وجود مسیر یا دیوار را بررسی میکند(-1 : دیوار و 1: مسیر)

به علاوه هر نود دارای متغیری از جنس vector با نام availableDirections خواهد بود که در هر لحظه مسیر های در دسترس هر نود را درون خود ذخیره میکند.

برای بررسی وضعیت هر نود چند فلگ به کار برده شده است ، end_visit ، is_true_dir ، is_end ، is_start ، visited که وظیفه هر کدام از اسم کاملا مشخص است. در مورد end_visit ، این فلگ و همه متغیر ها و توابعی که با end آغاز میشوند ، برای پیش برد هم زمان الگوریتم ها هم از سمت نقطه شروع و هم از سمت نقطه استارت و برای ایجاد تمایز بین نود هایی است که از سمت پایان مورد بررسی قرار گرفته اند (فقط مخصوص الگوریتم Bidirectional)

به علاوه هر نود ، یک نود پرنت هم دارد که اشاره دارد به نودی که در مرحله قبلی مورد ویزیت قرار گرفت .

گرافیک نود:

برای بخش گرافیکی برای هر نود یک مربع به عنوان بخش اصلی نود و چهار مستطیل در چهار طرف نود برای نمایش جهات چهارگانه اطراف هر نود در نظر گرفته شده است.

تابع کانسراکتور Node

```
MazeGUI::Node::Node(size_t _id)
    : id{_id}
{
     MazeGUI::GMaze = new QGraphicsItemGroup();
     recGroup = new QGraphicsItemGroup(MazeGUI::GMaze);
     mainNode = new QGraphicsItemGroup(recGroup);
     QGraphicsRectItem *mainBox = new QGraphicsRectItem(QRectF(-10,-10,20,20),mainNode);
     mainBox->setBrush(redBrush);
     mainBox->setPen(redPen);
}
```

در تابع سازنده نود ، در ابتدا id مقداردهی می شود سپس 5 مستطیل مربوط به هر نود ساخته شده و در یک گروه گرافیکی قرار میگیرند.

کلاس MazeGUI

: logical fields

```
//logical fields
int row;
int col;
std::vector<std::vector<std::shared_ptr<Node>>> Board;
std::shared_ptr<MazeGUI::Node> start{nullptr};
std::shared_ptr<MazeGUI::Node> end{nullptr};
char DFS_or_BFS_or_BS;
int visited_counter = 0;
size_t step{0};
std::stack<std::shared_ptr<Node>> True_Dir;
std::queue<std::shared_ptr<Node>> frontier;
std::queue<std::shared_ptr<Node>> end_frontier;
std::vector<std::shared_ptr<Node>> visitedNodes;
```

هر ماز دارای ابعادی است که در row و col ذخیره می شوند. برای هر ماز یک vector دو بعدی به اسم Board در نظر گرفته شه است که هر نود مربوط به ماز به صورت shared_ptr درآن ذخیره می شود. به علاوه پوینتر های مربوط به نقاط شروع و پایان هر ماز در متغیر های start و end ذخیره میگردند.

برای نمایش تعداد مراحل حل هر ماز از متغیر step استفاده شده است. برای حل به روش dfs از متغیری از جنس stack به نام True_Dir استفاده شده است برای روش bfs از متغیری از جنس queue به نام frontier استفاده می شود. وکتور visitedNodes هم در هر مرحله همه نود هایی که ویزیت شده اند درون خود ذخیره میکند تا برای پاک کردن آنها در وسط اجرای الگوریتم مشکلی نداشته باشیم.

: graphical fields

```
//graphical fields
Ui::MazeGUI *ui;
static inline QGraphicsItemGroup* GMaze;
QGraphicsScene* scene{nullptr};
QGraphicsEllipseItem* Qstart;
QGraphicsEllipseItem* Qend;
std::vector<QGraphicsRectItem *> masir;
```

GMaze متغیری است که همه گرافیکی همه نود هارا درون خود ذخیره میکند تا برای حذف کردن آنها در تابع clear مشکلی نداشته باشیم. علاوه بر تعریف scene برای رسم ماز ، برای هر ماز کل مسیر های تولید شده در وکتور گرافیکی masir ، دایره مربوط به استارت در متغیر Qstart و دایره مربوط به نقطه پایان هم در متغیر Qend ذخیره می گردد.

صفحه welcome

همان طور که مشاهده می شود برای صفحه خوش آمد گویی از لیبل برای لوگو واسم برنامه استفاده شده است.

```
class Thread : public QThread
{
    Q_OBJECT

    signals:
        void progress( int value );

    private:
        void run()
        {
            for(int i = 0; i <= 100; i++ )
            {
                emit progress( i );
                QThread::msleep(50);
            }
        }
};
```

برای پر شدن progressbar از یک کلاس Thread استفاده شده است که از QThread ارث برده است.محدوده برای progressbar از صفر تا 100 در نظر گرفته شده و هر عدد با تاخیر 50 میلی ثانیه ای اضافه می گردد.

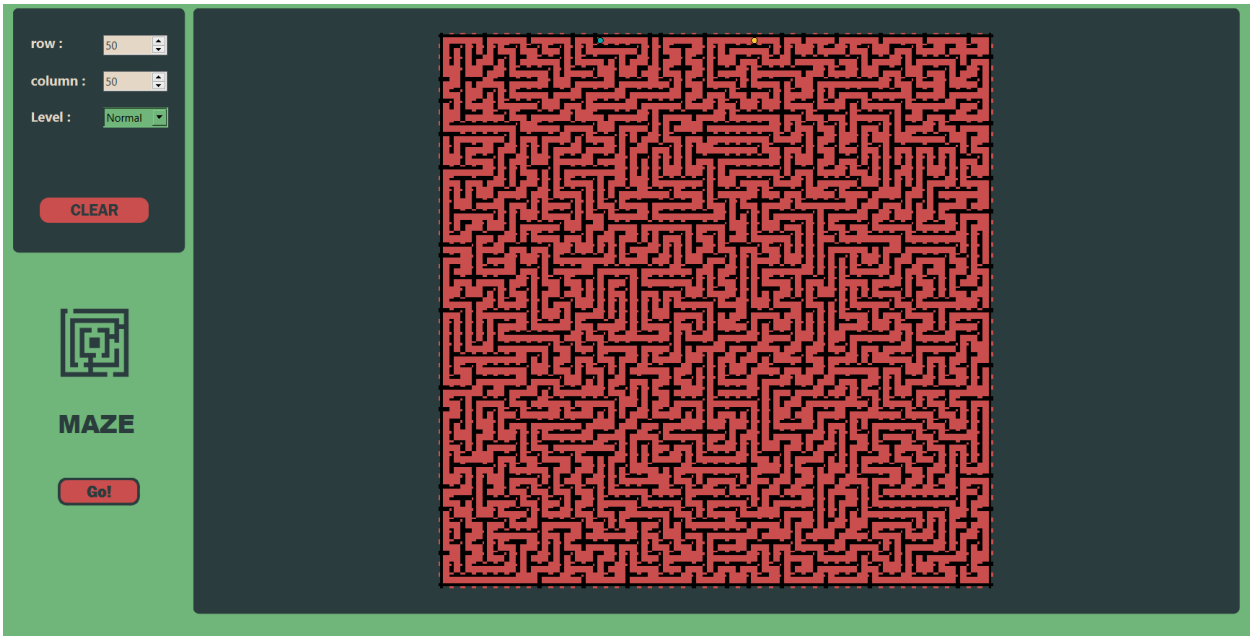
صفحه اصلی بازی

صفحه اصلی از سه فریم تشکیل شده است:

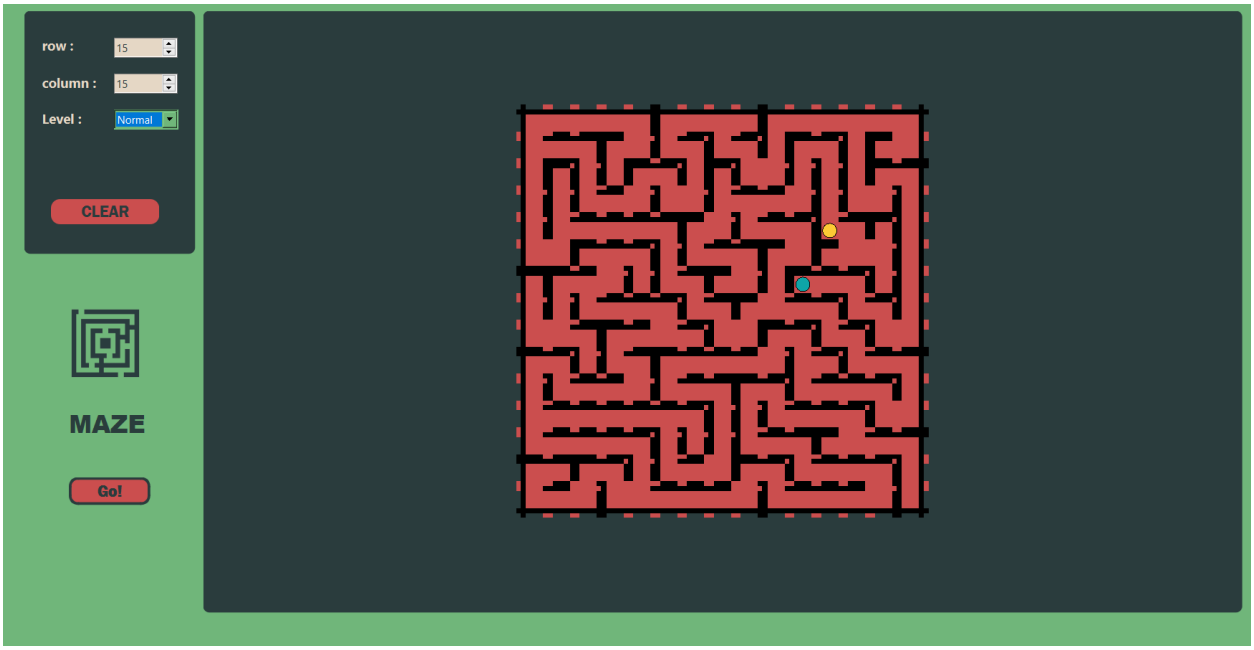
1- graphicsView: درون این فریم graphicsView قرار گرفته که با تعیین scene ، میتوان اشکال و طرح هایی را در آن رسم نمود. در این پروژه برای نمایش ماز از شیوه رسم آن در یک بوم استفاده شده است.

```
//set Zoom ability to my graphicsView
Graphics_view_zoom* z = new Graphics_view_zoom(ui->graphicsView);
z->set_modifiers(Qt::NoModifier);
```

از آن جایی که ابعاد ماز متغیر است و برای هر نود ابعاد ثابتی در نظر گرفته شد ، بنابراین برای ابعاد بزرگ ماز به طور کامل داخل صفحه جای نمی گرفت . به همین دلیل قابلیت بزرگ نمایی و کوچک نمایی صفحه نمایش به پروژه اضافه شده است.



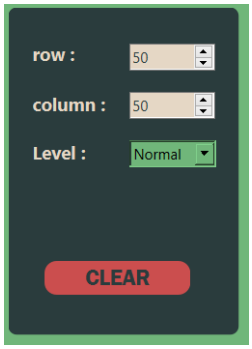
50*50



15*15

setBar -2

این فریم برای انجام تنظیمات اولیه ماز ساخته شده است.



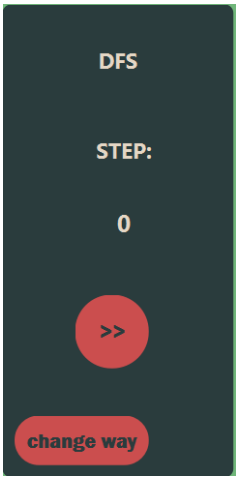
پس از تعیین ابعاد و سطح بازی با زدن دکمه Generate ماز ساخته و نمایش داده میشود. سپس کلید Generate ناپدید می شود و کلید Clear ظاهر میگردد که در هر لحظه بازی با زدن آن کل زمین بازی و متغیر ها به حالت اولیه باز میگردند و کلید Generate دوباره ظاهر می گردد.

logoWidget -3

این فریم در ابتدای بازی و با هر بار clearکردن ماز مشاهده می گردد .

controlBar -4

در این فریم روند بازی کنترل می شود. در ابتدا کاربر باید روش حل ماز را انتخاب کند. پس از انتخاب شکل زیر ظاهر می گردد.



همان طور که در تصویر مشاهده میشود با فشردن دکمه next هر مرحله از بازی پیش میرود و در هر مرحله مقادیر متغیر step نمایش داده میشوند. در ضمن در هر مرحله از بازی میتوان با فشردن دکمه change way تمامی تغییرات مربوط به راه حل انتخابی را حذف کرد و راه حل دیگری انتخاب نمود.

Maze Generation

برای تولید ماز در این بازی از الگوریتم dfs استفاده شده است .

```
void MazeGUI::Go_to_Generation(std::stack<std::shared_ptr<Node>> myDirection)
{
    myDirection.top()->Update_availableDirections();
    if (visited_counter < row*col)
    {
        if (!myDirection.top()->availableDirections.empty())
        {
            size_t mysize = (myDirection.top()->availableDirections).size();
            std::pair<std::shared_ptr<Node>,int> selected = make_pair(myDirection.top()->availableDirections[ MazeGUI::Random_generator(mysize)].first,myDir
            selected.first->visited = true;
            visited_counter++;
            selected.first->Update_availableDirections();
            myDirection.top()->set_dir_flags(selected.first);
            myDirection.push(selected.first);
            myDirection.top()->Update_availableDirections();
            Go_to_Generation(myDirection);
        }
        else
        {
            myDirection.pop();
            myDirection.top()->Update_availableDirections();
            Go_to_Generation(myDirection);
        }
    }
}
```

به این صورت که پس از پر کردن وکتور Board توسط تابع fill و اختصاص دادن مستطیل های گرافیکی به هر نود ، به صورت رندوم یک نقطه به عنوان start انتخاب می شود. سپس این نقطه داخل ویزیت میشود و با توجه به availableDirections بین نود هایی که امکان ادامه مسیر دارند به صورت رندوم یکی انتخاب میشود و داخل یک stack قرار میگرد. و این کار تا جایی ادامه پیدا میکند تا همه نقاط ویزیت شوند .

Find_end

در این مرحله لازم است نقطه ای به عنوان نقطه پایانی در نظر گرفته شود. این نقطه با توجه به سطحی که کاربر در برنامه برای بازی تعیین می کند ، قرار داده می شود.

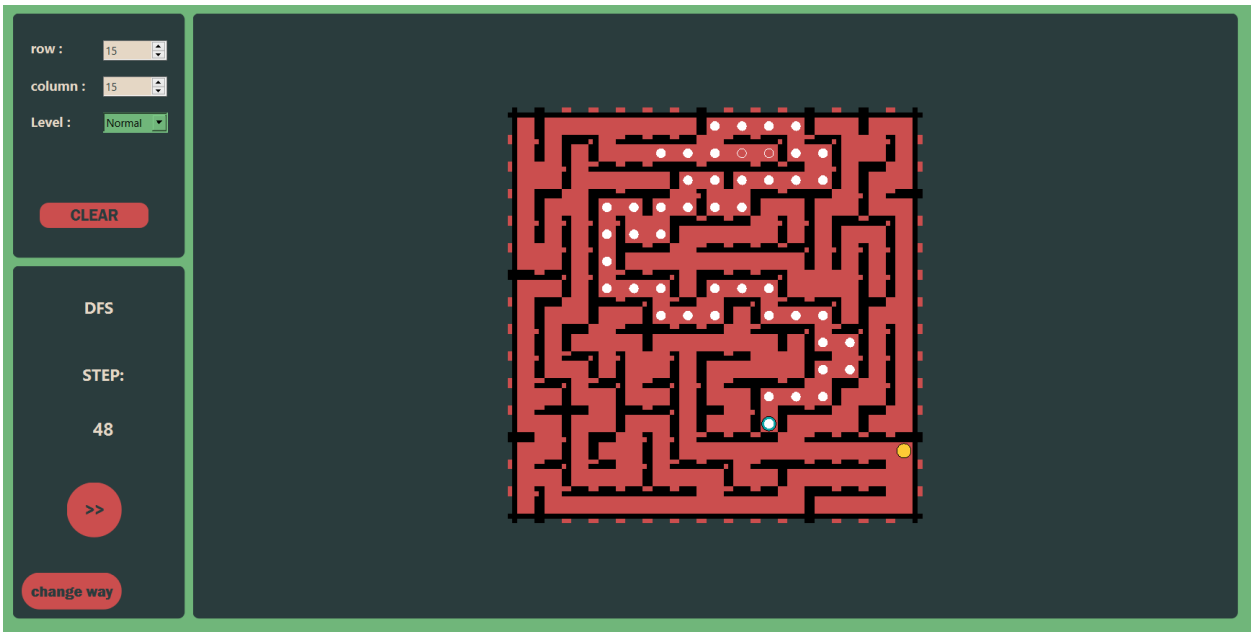
```
void MazeGUI::find_end()
{
    std::queue<std::shared_ptr<Node>> fr;
    fr.push(start);
    int visited_node_counter=0;
    int levelRange{};
    if(ui->levelBox->currentText() == "Easy")
        levelRange = row*col/4;
    else if(ui->levelBox->currentText() == "Normal")
        levelRange = row*col/2;
    else if(ui->levelBox->currentText() == "Hard")
        levelRange = 3*row*col/4;
    while(visited_node_counter < row*col-1)
    {
        fr.front()->visited = true;
        visited_node_counter++;
        fr.front()->Update_availableDirections();
        std::shared_ptr<Node> p = fr.front();
        fr.pop();
        for(auto& ch : p->availableDirections)
        {
            ch.first->parent = p;
            fr.push(ch.first);
            fr.front()->Update_availableDirections();
        }
        if (visited_node_counter == levelRange)
        {
            if(p->availableDirections.empty())
                end = p;
            else
                end = p->availableDirections[MazeGUI::Random_generator(p->availableDirections.size())].first;
            end->is_end = true;
            break;
        }
    }
}
```

همان طور که در کد بالا مشاهده می گردد ، برای تعیین نقطه end از روش bfs استفاده شده است. یعنی از نقطه start شروع میکنیم و با الگوریتم bfs پیش می رویم . اگر سطح بازی easy باشد ، وقتی به نقطه ای رسیدیم که ¼ کل نود ها ویزیت شده باشند ، یکی از child های باقی مانده در stack به صورت رندوم به عنوان نقطه end در نظر میگیریم. همین روند برای سطح Normal برای ½ کل نود ها و برای سطح Hard برای ¾ کل تعداد نود ها در نظر گرفته می شود.

DFS

```
if (True_Dir.top() != end)
{
       if (!True_Dir.top()->availableDirections.empty())
        {
            size_t mysize = (True_Dir.top()->availableDirections).size();
            std::pair<std::shared_ptr<Node>,int> selected = make_pair(True_Dir.top()->availableDirections[ MazeGUI::Random_generator(mysize)].first,
            selected.first->visited = true;
            visitedNodes.push_back(selected.first);
            selected.first->Update_availableDirections();
            selected.first->parent = True_Dir.top();
            True_Dir.push(selected.first);
            True_Dir.top()->Update_availableDirections();
            True_Dir.top()->Me = new QGraphicsEllipseItem(True_Dir.top()->recGroup->pos().rx()-5,True_Dir.top()->recGroup->pos().ry()-5,10,10);
            True_Dir.top()->Me->setBrush(whiteBrush);
            True_Dir.top()->Me->setPen(whitePen);
            scene->addItem(True_Dir.top()->Me);
        }
        else
        {
           delete True_Dir.top()->Me;
           True_Dir.top()->Me = new QGraphicsEllipseItem(True_Dir.top()->recGroup->pos().rx()-5,True_Dir.top()->recGroup->pos().ry()-5,10,10);
           True_Dir.top()->Me->setBrush(redBrush);
           True_Dir.top()->Me->setPen(whitePen);
           scene->addItem(True_Dir.top()->Me);
           True_Dir.pop();
           True_Dir.top()->Update_availableDirections();
        }
```

همان طور که در کد بالا مشاهده میشود ، برای الگوریتم DFS یک stack در نظر گرفته میشود. در ابتدا نود استارت داخل آن وارد میشود. هر نود که انتخاب میشود ابتدا ویزیت میگردد ، سپس با توجه به availableDirections و به صورت رندوم نود بعدی انتخاب میگردد و داخل stack وارد میگردد. در صورتی که به نودی رسیدیم که availableDirections خالی باشد و نود end نباشد ، آخرین عضو stack حذف می گردد تا به نودی برسیم که مسیر در دسترس داشته باشد.بازی تا جایی ادامه پیدا میکند که به نود end برسیم.

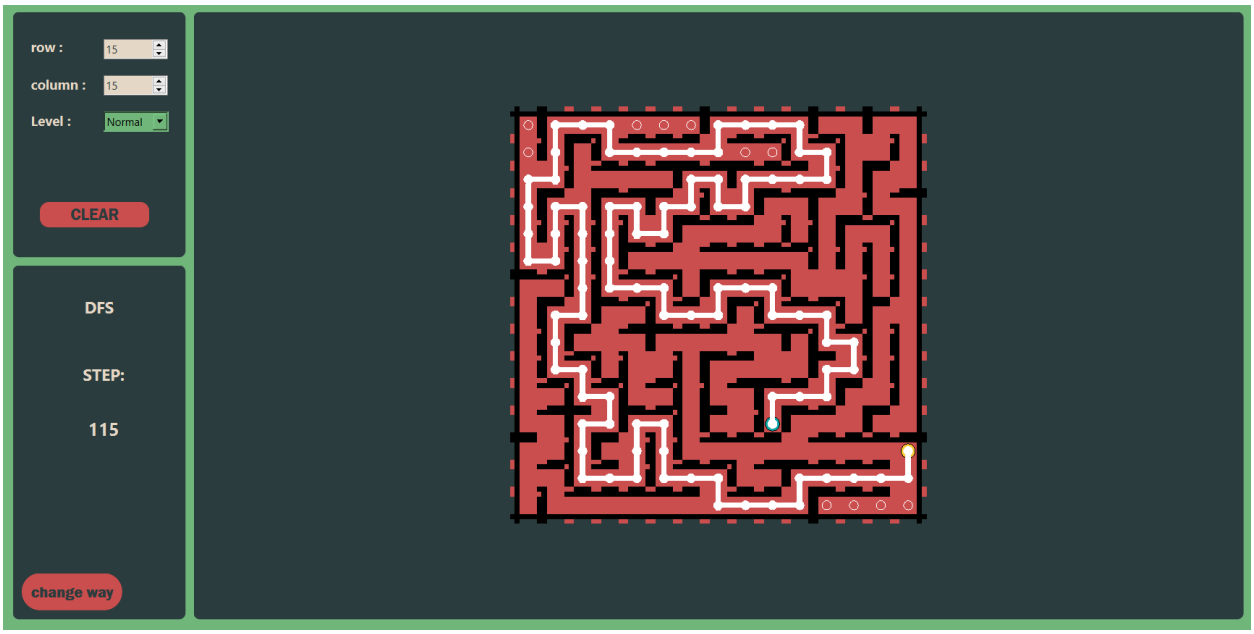


همان طور که در شکل مشاهده می گردد در هر مرحله به ازای هر نود ویزیت شده یک دایره در خانه مربوطه قرار میگیرد و به ازای هر نود backtrack شده هم یک دایره توخالی قرار میگیرد.

```
}else{

    ui->nextstepbtn->setEnabled(false);
    QGraphicsRectItem *Dir;
    while (!True_Dir.empty()) {
        True_Dir.top()->is_true_dir = true;
        if(True_Dir.top()->parent){
            if((True_Dir.top()->Directions)["North"].first == True_Dir.top()->parent)
            {
                Dir = new QGraphicsRectItem(QRectF(True_Dir.top()->recGroup->pos().rx()-2.5,True_Dir.top()->recGroup->pos().ry()-25,5,30));
                masir.push_back(Dir);
            }else if((True_Dir.top()->Directions)["South"].first == True_Dir.top()->parent){
                Dir = new QGraphicsRectItem(QRectF(True_Dir.top()->recGroup->pos().rx()-2.5,True_Dir.top()->recGroup->pos().ry(),5,30));
                masir.push_back(Dir);
            }else if((True_Dir.top()->Directions)["East"].first == True_Dir.top()->parent){
                Dir = new QGraphicsRectItem(QRectF(True_Dir.top()->recGroup->pos().rx()-2.5,True_Dir.top()->recGroup->pos().ry()-2.5,30,5));
                masir.push_back(Dir);
            }else{
                Dir = new QGraphicsRectItem(QRectF(True_Dir.top()->recGroup->pos().rx()-30,True_Dir.top()->recGroup->pos().ry()-2.5,30,5));
                masir.push_back(Dir);
            }
            Dir->setBrush(whiteBrush);
            Dir->setPen(whitePen);
            scene->addItem(Dir);
        }
        True_Dir.pop();
    }
    ui->nextstepbtn->setHidden(true);
    ui->nextstepbtn->setEnabled(false);
}
```

در نهایت و پس رسیدن به نقطه end هر آنچه که در stack باقی مانده است ، مسیر درست خواهد بود .

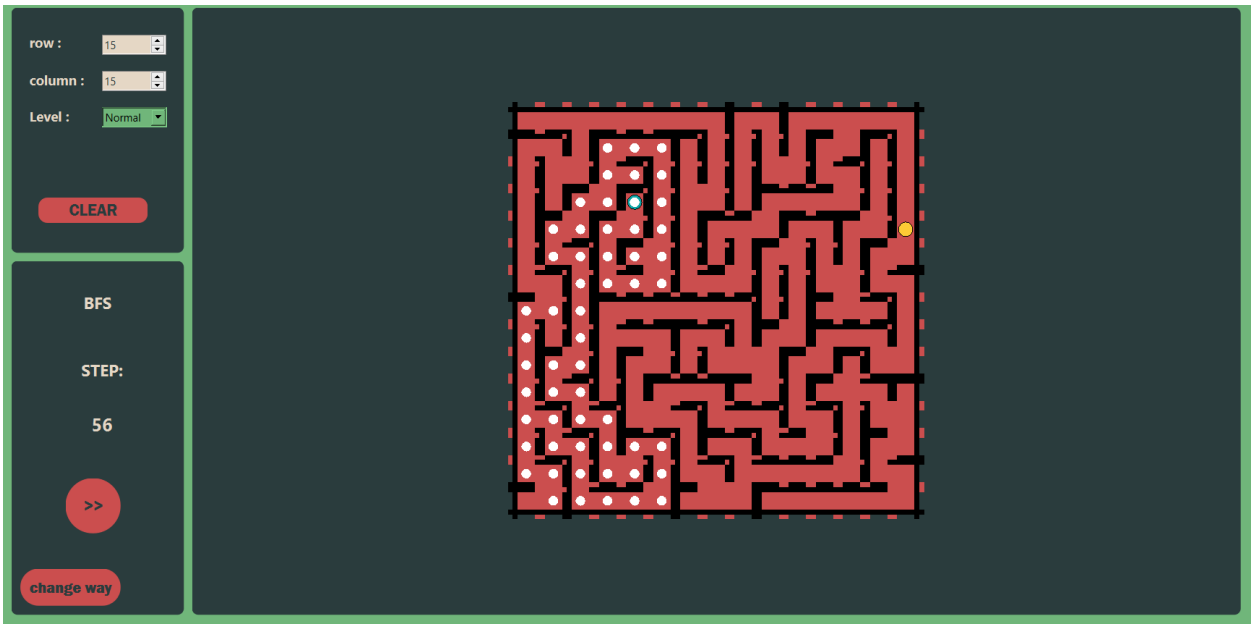


BFS

```
if(frontier.front() != end)
{
    frontier.front()->visited = true;
    visitedNodes.push_back(frontier.front());
    frontier.front()->Update_availableDirections();
    std::shared_ptr<Node> p = frontier.front();
    if(frontier.front() != start)
    {
        p->Me = new QGraphicsEllipseItem(p->recGroup->pos().rx()-5,p->recGroup->pos().ry()-5,10,10);
        p->Me->setBrush(whiteBrush);
        p->Me->setPen(whitePen);
        scene->addItem(p->Me);
    }
    frontier.pop();
    for(auto& ch : p->availableDirections)
    {
        ch.first->parent = p;
        frontier.push(ch.first);
        frontier.front()->Update_availableDirections();
    }
```

در کد مشاهده می گردد که برای این روش از یک queue استفاده میگردد. در ابتدا نود start وارد queue میشود.

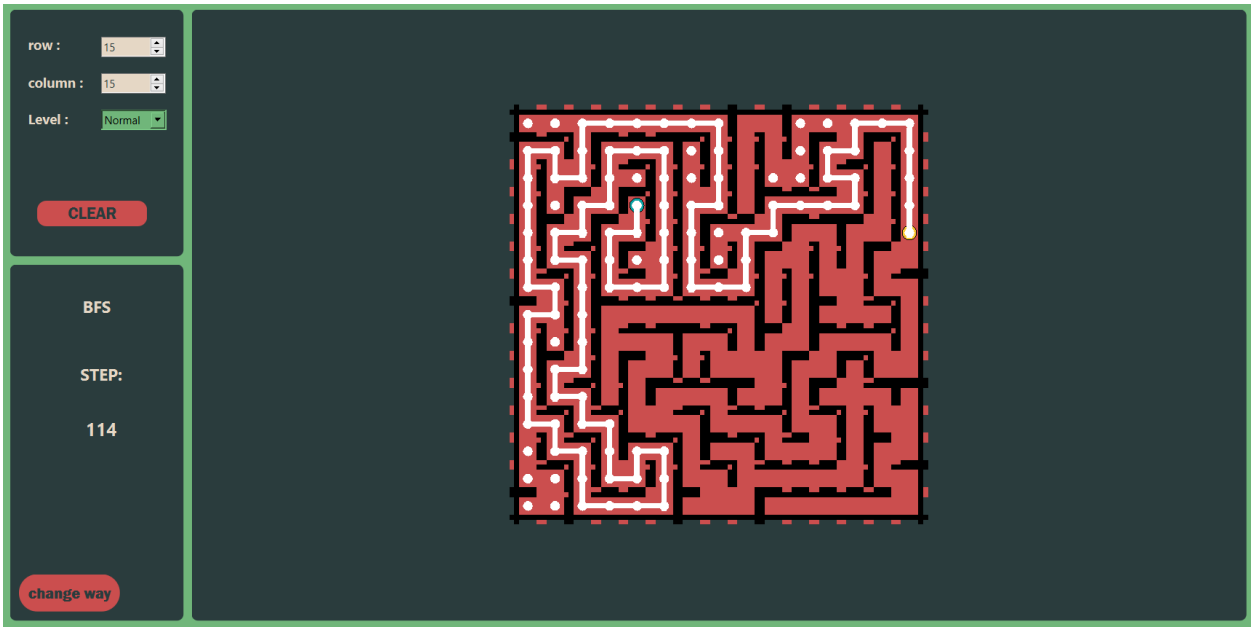
در هر مرحله همیشه اولین عضو queue خارج میشود ، ویزیت می گردد و در نهایت همه child هایش دوباره داخل queue اضافه میگردد. این روند تا جایی ادامه پیدا میکند که اولین عضو queue همان نقطه end باشد.



در هر مرحله به ازای هر نود ویزیت شده یک دایره در خانه مربوطه قرار می گیرد.

```
}else{
    frontier.front()->visited = true;
    visitedNodes.push_back(frontier.front());
    frontier.front()->Me = new QGraphicsEllipseItem(frontier.front()->recGroup->pos().rx()-5,frontier.front()->recGroup->pos().ry()-5,10,10);
    frontier.front()->Me->setBrush(whiteBrush);
    frontier.front()->Me->setPen(whitePen);
    scene->addItem(frontier.front()->Me);
    std::stack<std::shared_ptr<Node>> short_dir;
    make_dir(short_dir,  end, false);
    QGraphicsRectItem *Dir;
    while (!short_dir.empty()) {
        short_dir.top()->is_true_dir = true;
        if(short_dir.top()->parent){
            if((short_dir.top()->Directions)["North"].first == short_dir.top()->parent)
            {
                Dir = new QGraphicsRectItem(QRectF(short_dir.top()->recGroup->pos().rx()-2.5,short_dir.top()->recGroup->pos().ry()-25,5,30));
                masir.push_back(Dir);
            }else if((short_dir.top()->Directions)["South"].first == short_dir.top()->parent){
                Dir = new QGraphicsRectItem(QRectF(short_dir.top()->recGroup->pos().rx()-2.5,short_dir.top()->recGroup->pos().ry(),5,30));
                masir.push_back(Dir);
            }else if((short_dir.top()->Directions)["East"].first == short_dir.top()->parent){
                Dir = new QGraphicsRectItem(QRectF(short_dir.top()->recGroup->pos().rx()-2.5,short_dir.top()->recGroup->pos().ry()-2.5,30,5));
                masir.push_back(Dir);
            }else{
                Dir = new QGraphicsRectItem(QRectF(short_dir.top()->recGroup->pos().rx()-30,short_dir.top()->recGroup->pos().ry()-2.5,30,5));
                masir.push_back(Dir);
            }
            Dir->setBrush(whiteBrush);
            Dir->setPen(whitePen);
            scene->addItem(Dir);
        }
        short_dir.pop();
    }
    ui->nextstepbtn->setHidden(true);
    ui->nextstepbtn->setEnabled(false);
}
```

نکته قابل توجه در ارتباط با این روش مسیریابی است. در این روش برای مسیریابی نمیتوان از queue استفاده کرد ، چرا که در هر لحظه آخرین عضو حذف میشود. به همین جهت برای هر نود یک parent در نظر گرفته شد و در هر مرحله با انتخاب هر نود ، نود قبلی به عنوان parent قرار داده میشود. بنابراین پس از رسیدن به نقطه end با استفاده از تابعی با نام make_dir از آخرین نود به سمت اولین نود پرنت ها در یک stack ذخیره می گردند تا مسیر درست یافت شود. در نهایت مسیر به صورت گرافیکی نمایش داده میشود.

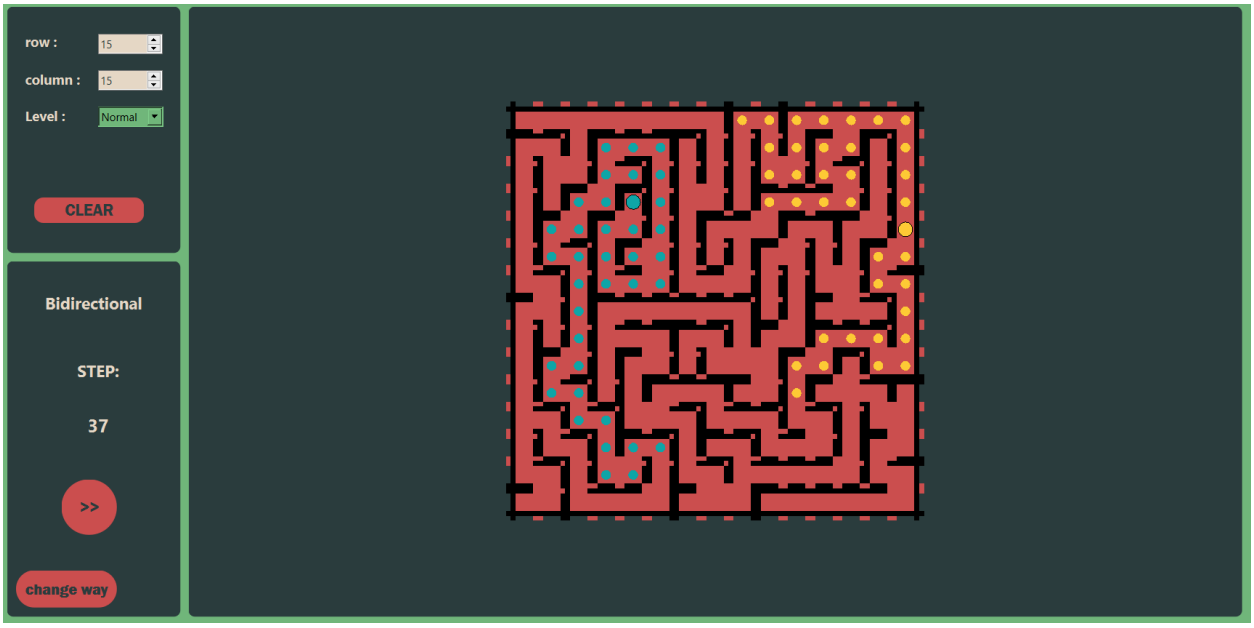


Bidirectional

```
if(DFS_or_BFS_or_BS == 'B')
{
    find_bidirectional(frontier, false);
    find_bidirectional(end_frontier, true);
    if(frontier.front()->end_visit)
    {
        paint_dir(frontier.front());
    }else if(end_frontier.front()->visited)
    {
        paint_dir(end_frontier.front());
    }else if(frontier.front() == end_frontier.front())
    {
        frontier.front()->Me = new QGraphicsEllipseItem(frontier.front()->recGroup->pos().rx()-5,frontier.front()->recGroup->pos().ry()-5,10,10);
        frontier.front()->Me->setBrush(yellowBrush);
        frontier.front()->Me->setPen(yellowPen);
        scene->addItem(frontier.front()->Me);
        paint_dir(frontier.front());
    }
}
```

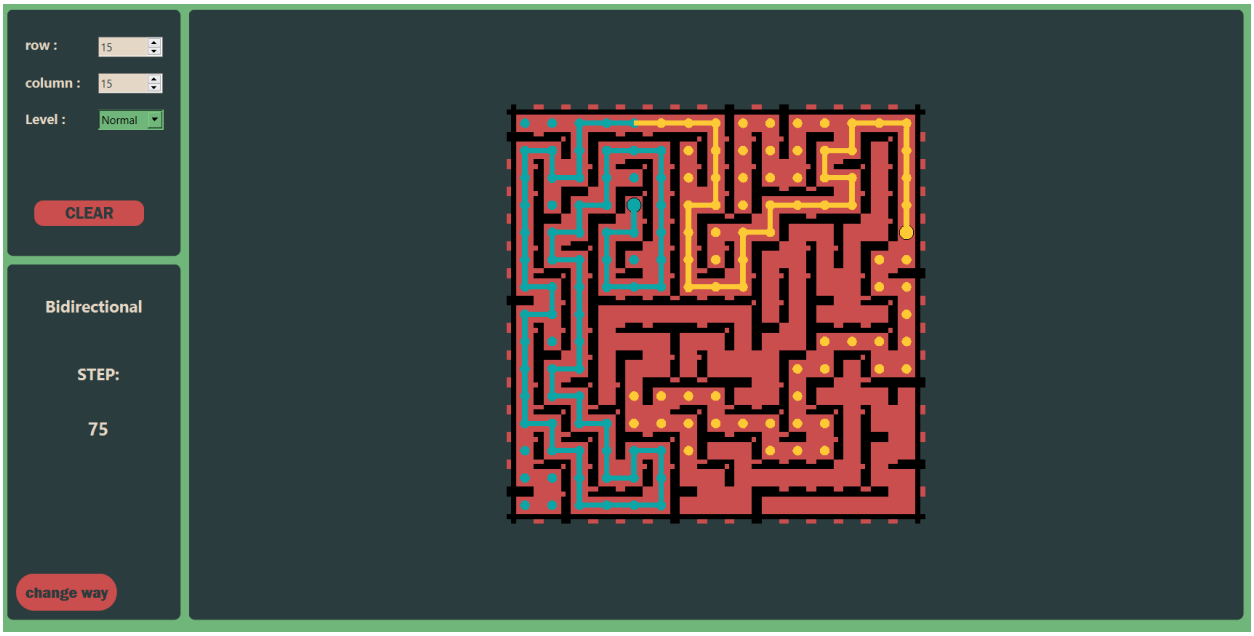
در این روش به صورت هم زمان الگوریتم bfs از هردو طرف نقطه start و نقطه end انجام میشود. متغیر های end که در ابتدا معرفی شدند برای دنبال کردن این الگوریتم بودند.

```
if(middle->end_parent != nullptr)
{
    std::stack<std::shared_ptr<Node>> short_dir_end;
    make_dir(short_dir_end,  middle, true);
    QGraphicsRectItem *Dir;
    while (!short_dir_end.empty()) {
        short_dir_end.top()->is_true_dir = true;
        if(short_dir_end.top()->parent){
            if((short_dir_end.top()->Directions)["North"].first == short_dir_end.top()->parent)
            {
                Dir = new QGraphicsRectItem(QRectF(short_dir_end.top()->recGroup->pos().rx()-2.5,short_dir_end.top()->recGroup->pos().ry()-25,5,30));
                masir.push_back(Dir);
            }else if((short_dir_end.top()->Directions)["South"].first == short_dir_end.top()->parent){
                Dir = new QGraphicsRectItem(QRectF(short_dir_end.top()->recGroup->pos().rx()-2.5,short_dir_end.top()->recGroup->pos().ry(),5,30));
                masir.push_back(Dir);
            }else if((short_dir_end.top()->Directions)["East"].first == short_dir_end.top()->parent){
                Dir = new QGraphicsRectItem(QRectF(short_dir_end.top()->recGroup->pos().rx()-2.5,short_dir_end.top()->recGroup->pos().ry()-2.5,30,5));
                masir.push_back(Dir);
            }else{
                Dir = new QGraphicsRectItem(QRectF(short_dir_end.top()->recGroup->pos().rx()-30,short_dir_end.top()->recGroup->pos().ry()-2.5,30,5));
                masir.push_back(Dir);
            }
            Dir->setBrush(yellowBrush);
            Dir->setPen(yellowPen);
            scene->addItem(Dir);
           }
           short_dir_end.pop();
    }
}
```



در حالت کلی دو مسیر به سه شکل ممکن است به هم برسند یا نود بعدی هر دو هم زمان یک نود است ، یا نود بعدی هرکدام یکی از نود های ویزیت شده توسط دیگری است.الگوریتم جایی پایان می یابد که یکی از این سه حالت اتفاق بیفتد.

در نهایت وقتی دو مسیر بهم رسیدند ، با استفاده از روش به کار برده شده در bfs مسیر از نقطه برخورد به هردو سمت رسم می گردد.



تابع clear

از آنجایی که المان های گرافیکی پروژه new شده بودند ، لازم است تا تک به تک دلیت گردند. به همین دلیل در ابتدا برای هر نود همه مستطیل ها و همچنین دایره مربوط به راه حل ها دلیت می گردند. سپس دوایر مربوط به start و end و سپس مسیر به طور کامل حذف می گردند. در نهایت هم فلگ ها و شمارنده ها هم به حالت اولیه باز میگردند و وکتور دوبعدی Board خالی می گردد.