دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده برق

به نام خدا

پروژه میانترم برنامه نویسی پیشرفته

استاد درس

دکتر جهانشاهی

دانشجو

مصطفی حمیدی فرد

بهار ۱۴۰۰

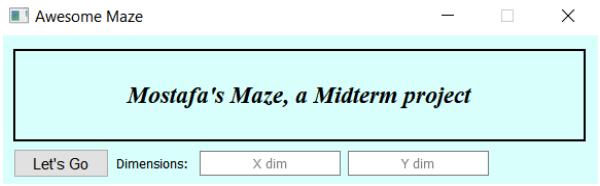
توضیحات کلی

هدف این پروژه ساخت یک برنامه ایجاد و حل ماز است. این پروژه به قسمت های کوچکتر زیر تقسیم بندی می‌شود:

- طراحی صفحات و رابط کاربری
- مدل سازی یک ماز تصادفی
- تحقیق در مورد DFS و BFS
- ایجاد طراحی مناسب در کد نویسی برای حل یک ماز براساس DFS و BFS
- استفاده از روش Thread برای جلوگیری مشغول شدن main thread هنگام حل ماز

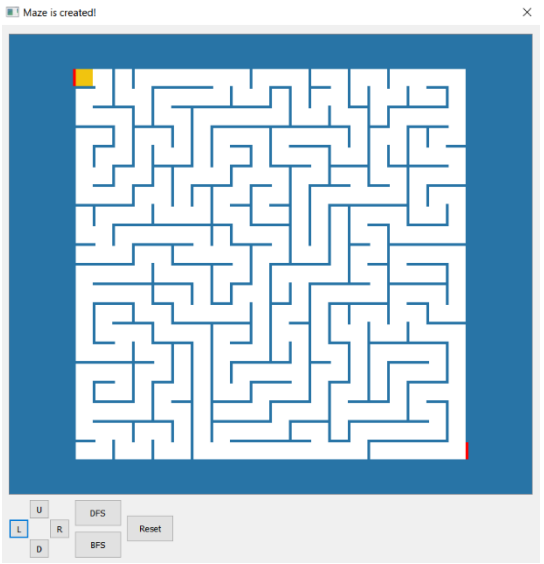
طراحی رابط کاربری:

جهت طراحی رابط کاربری از نرم افزار QT استفاده شده است و رابط کاربری از دو صفحه تشکیل شده است که صفحه اول برای گرفتن ابعاد از ۱*۱ تا ۹۹*۹۹ است.



کاربر فقط می تواند در قسمت edit text ها عدد وارد کرد.

سپس با زدن دکمه وارد صفحه دوم که وظیفه نشان دادن ماز و حل آن را برعهده دارد می رویم.



در این صفحه با زدن دکمه های U,L,D,R می توانیم راس ماز را بالا یا پایین یا چپ یا راست ببریم. همچنین با زدن دکمه های DFS و BFS، یک thread اجرا می شود تا ماز را حل کند. توجه شود که اگر از thread استفاده نشود، صفحه قفل می شود زیرا تنها main thread مسئول UI است و اگر آن را مشغول کنیم دیگر نمی تواند به UI جواب بدهد. همچنین با زدن RESET صفحه ماز دوباره به حالت اولیه بر میگردد.

مدل سازی یک ماز تصادفی:

ابتدا یک کلاس برای هریک از مستطیل های داخل ماز درست می کنیم تا برای ساخت ماز از آن استفاده کنیم.

اسم این کلاس RectNode است که نگهدارنده ۹ شیء از جنس QGraphicsRectItem است که متد هایی برای مدیریت رنگ این مستطیل ها براساس همسایگی با جهات های مختلف و در گوشه قرار داشتن، تشکیل شده است که هرکدام را توضیح می دهیم:

- Constructor: ۳ ورودی می گیرد که دو تا از آن ها مختصات و دیگری متغیری است که نشان می دهد که آیا از این مستطیل عبور کرده ایم یا خیر.
- در دیستراکتور، به پاک کردن شیء های مستطیلی می پردازیم.
- متد add_node این مستطیل را به Scene برای نمایش دادن در QGraphicsView اضافه می کند. ورودی آن scene است که قرار است به آن اضافه شود.
- متد make_neighbour از نوع استاتیک است که نیاز نیست تا از طریق یک شیء خاص صدا زده شود. این متد دو مستطیل را که در کنار یکدیگر هست، با هم همسایه می کند و باعث می شود تا خطی بین آن ها قرار نگیرد.
- متد update_rect_nei_colors باعث آپدیت شدن رنگ این مستطیل براساس ویژگی های وجودی این مستطیل از ماز است. مثلا براساس همسایگان این سلول، شروع به رنگ کردن آن ۹ مستطیل می کند.
- Set_passed متغیر passed را تنظیم می کند تا مربع وسطی سفید یا رنگی شود. این نشان دهنده عبور از آن سلول می شود.
- متد real_neighbours یک مجموعه را برمیگرداند که همسایگان واقعی این سلول را نشان می دهد. زیرا برای مدل کردن همسایگان اینگونه عملکرده ایم که اگر در متغیر neighbours ، اگر key=up و value=nullptr باشد یعنی اینکه این متغیر از بالا در گوشه صفحه قرار ندارد ولی با سلول بالایی همسایه نیست. اگر key=up وجود نداشته باشد یعنی در بالاترین سطر قرار دارد و اصلا سلولی بالای آن نیست. اگر key=up و ...=value وجود داشته باشد، یعنی با سلول بالایی همسایه است و در ماز می تواند از این سلول به سلول بالایی برویم. متد real_neighbours مجموعه ای را برمیگرداند که value=nullptr را ندارد.
- متغیر parent برای نگهداشتن و وصل کردن node ها در روش BFS است تا بتوانیم node ها را به هم وصل کنیم.

در گام بعدی یک کلاس به نام Maze ایجاد کرده ایم که وظیفه آن ایجاد و نگهداشتن ماز مورد نظر ما است.

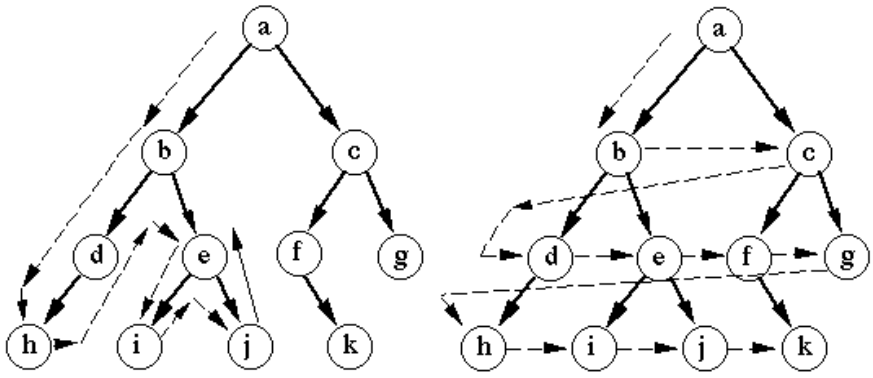
روش ساخت یک ماز: ابتدا از خانه [0][0] شروع می کنیم به طور تصادفی یکی از همسایگان بالقوه آن را انتخاب می کنیم و آن را با این خانه همسایه می کنیم سپس خانه جدید را مبنا قرار می دهیم و یکی از همسایگان بالقوه را انتخاب می کنیم و با ان همسایه می کنیم. به همین شکل جلو می رویم تا به جایی برسیم که در نوک این path دیگر نتوانیم همسایه ای را انتخاب کنیم، در این لحظه از نوک شروع می کنیم و به عقب می رویم تا بتوانیم دوباره همسایه انتخاب کنیم و دوباره همین الگوریتم را ادامه می دهیم. مزیت این روش آن است که همه خانه ها به هم وصل هستند اما عیب آن نیز این است که تا به اولین node موجود در path که قابلیت گسترش دارد می رسیم، از آن شروع به گسترش می کنیم در حالی که باید بین نود های موجود، دوباره نیز به صورت رندوم شروع به گسترش کنیم.

توضیح کلاس Maze: در کلاس maze متد ها و متغیر های زیر وجود دارد.

- کانستراکتور: scene مربوطه برای رسم ماز و ابعاد آن را می گیرد براساس ابعاد، شروع به ساخت یک ماتریس [m][n] می کند. همچنین همسایگانی که خیالی هستند و از شکل ماز بیرون است را حذف می کند. این Matrix تمامی *RectNode ها وجود دارند. سپس متد create_maze را صدا می زند تا الگورتم ساخت ماز را شروع کند.
- Add_maze: تمام node های ماتریس را در داخل scene وارد می کند.
- Clear_path: صفحه را و path تا الان نشان داده شده را پاک می کند و صفحه را به حالت اول بر میگرداند.
- Change_head هنگامی که یکی از کلید های R,L,U,D زده می شود، این متد اجرا می شود تا نوک پس براساس جهت گفته شده، در صورت امکان، جابه جا شود.
- متد های solveBFS و solveDFS باعث استارت شدن thread های مربوط حل BFS و DFS می شود.
- اسلات do_update_scene باعث آپدیت کردن رنگ وسط سلول ورودی به آن هنگام حل توسط thread ها است. در این متد باید QEventloop::processEvent() صدا زده شود تا این آپدیت صفحه صورت گیرد. برای توضیحات بیشتر به مستند مربوط به آن مراجعه کنید.
- متد neighbour_available مجموعه ای را برمی گرداند که همسایگان بالقوه یک node را در بر دارد.

BFS و DFS:

برای توضیح آن صرفا به یک عکس بسنده می کنیم و در قسمت threadها به توضیح این بحث با توجه محتوای ماز می پردازم:



حل ماز براساس BFS:

برای حل از طریق BFS از bfsSolverThread استفاده می کنیم. Path و matrix و ابعاد وارد کانستراکتور میکنیم تا بتوانیم آنان را ذخیره کنیم و از آن استفاده کنیم. متد run را override کرده ایم و کد خودمان را در آن نوشته ایم.

الگوریتم به این صورت است که ابتدا همسایگان [0][0] را بررسی می کنیم سپس، همسایگان این همسایگان را به عنوان بچه های آن ها در یک container وارد می کنیم و دوباره آن را بررسی می کنیم و به همین صورت پیش می رویم تا به [m-1][n-1] برسیم. با توجه به اینکه parent ها در این حلقه ها مشخص شده اند. با حرکت کردن رو به عقب مسیر [m-1][n-1] تا [0][0] را می یابیم.

متد get_children وظیفه پس دادن همسایگان یک Node را برعهده دارد البته این همسایه نباید خود parent این node باشد و آن را از این children باید حذف کند تا دچار لوپ بینهایت نشویم.

حل ماز براساس DFS:

برای حل از طریق BFS از DFSsolverThread استفاده کرده ایم. قسمت های کانستراکتور، مانند bfsSolverThread است.

الگوریتم به این صورت است که در متد Depth_first_search ابتدا نود ورودی را بررسی می کنیم، سپس تمامی همسایگان آن نود را دوباره به همین متد ارجاع می دهیم. اگر نود مورد نظر [m-1][n-1] بود، true برمی گردانیم و اگر نبود و هیچ همسایه ای نیز نداشت(انتهای درخت) false بر میگردانیم. در این متد اگر به در لوپ به [m-1][n-1] برسیم یعنی اینکه خود این نود اولیه نیز جزئی از path هست و true برمیگردانیم و ان را از path حذف نمی کنیم ولی اگر از لوپ بیرون بیاییم یعنی از طریق همسایگان این node نتوانسته ایم به [m-1][n-1] برسیم پس خود این node نیز جزء path نیست و باید آن را از path بیرون بیندازیم و false برگردانیم. با استفاده از این روش بازگشتی به عمق درخت می رویم و با رسیدن به برگ ها، دوباره به بالای درخت بر می گردیم.

استفاده از thread:

چون تنها main thread وظیفه پاسخگویی به اجزای درحال نمایش را دارد، پس هیچگاه نباید به آن وظیفه سنگینی سپرد تا main thread بتواند صفحه را رفرش نگه دارد. برای ان که صفحه به هنگام حل فریز نشود، یکی از بهترین کار ها استفاده از Qthread ها است که کلاس های bfsSolverThread و DFSsolverThread از کلاس Qthread ارث بری کرده اند. یکی از مشکلات اصلی برنامه من این است که thread های مربوط به حل همچنان پس از بسته شدن برنامه نیز درحا کار هستند و حتی با اینکه متد های exit یا terminate را صدا می زدم ولی باز هم تاثیری نداشت که احتمالا این موضوع به خاطر عدم پیاده سازی درست thread یا نکته ای در event ها است.

نتیجه گیری

در این گزارش پروژه maze و الگوریتم ها و توابع مهم آن گلوگاه های کدنویسی آن را بررسی کردیم و از ضعف ها و نقاط قوت آن سخن گفتیم. این پروژه می تواند نشان دهنده نقش thread ها در برنامه نویسی باشد. همچنین دانشجو را با مفهوم کلی برنامه نویسی آشنا می کند.

عکس هایی از محیط برنامه: