به نام خدا

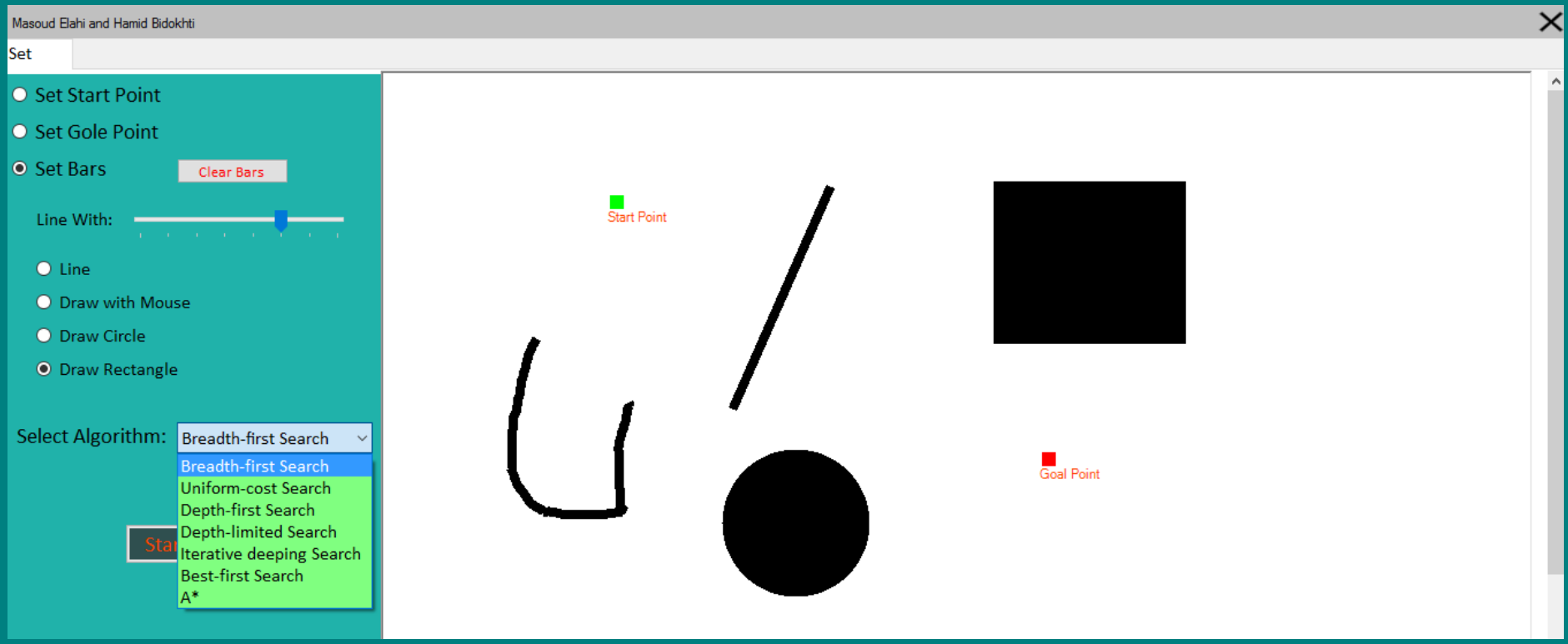
پیاده سازی الگوریتم های :

اول سطح ، هزینه یکنواخت ، اول عمق ، عمق محدود ، عمیق شونده تکراری ، حریصانه و A*

تهییه کننده : مسعود الهی

برنامه رابط کاربری:

برای تست الگوریتم ها یک فضای ۱۰۰۰*۱۰۰۰ در نظر گرفته شده است (Bitmap Bm) و با استفاده از شی گرافیکی روی این تصویر نقطه آغاز ، نقطه پایان و موانع روی تصویر رسم میشوند و کار بر عملیاتی که میخواهد انجام دهد را از منوی سمت چپ با توجه به تصویر زیر انتخاب میکند و در آخر بعد از تعیین موانع و نقاط آغاز و پایان روی دکمه Start کلیک میکند والگوریتم انتخاب شده کار خود را شروع میکند.



رویکرد کلی:

بعد از در یافت اطلاعات وارد شده ابتدا اطلاعات تصویر را که فضای عملیاتی الگوریتم است در یک آرایه به نام array[1000,1000] میریزیم به طوری که array[i,j] نشان دهنده اطلاعات pixel سطر iام و ستون jام از تصویر است و خانه های این آرایه به این صورت است:

Arry[x,y]=0 اگر رنگ پیکسل Bm[x,y] غیر سیاه باشد یعنی تله نباشد

Arry[x,y]=3 اگر رنگ پیکسل Bm[x,y] سیاه باشد یعنی تله باشد

مختصات نقاط آغاز و پایان هم در متغییرهای StartX و StartY و GoalX و GoalY نگه داری میشوند.

رویکرد کلی:

گره های جستجو شده در یک آرایه به نام table[1000000,6] که ۶ ستون دارد که

ستون اول X مختصات گره

ستون دوم Y مختصات گره

ستون سوم g گره که نشان میدهد از آغاز تارسیدن به این گره چند گره آمده ایم. همان g در f(هزینه مسیر)

ستون چهارم h گره که هیوریستیک فاصله مستقیم از گره تا هدف میباشد

ستون پنجم ۱ یا صفر است که نشان میدهد که این گره برای گسترش دادن انتخاب شده است یا نه

ستون ششم که شماره سطر گره پدر این گره را در همین آرایه نشان میدهد ، همان PID

تابع پسین: Around_Point()

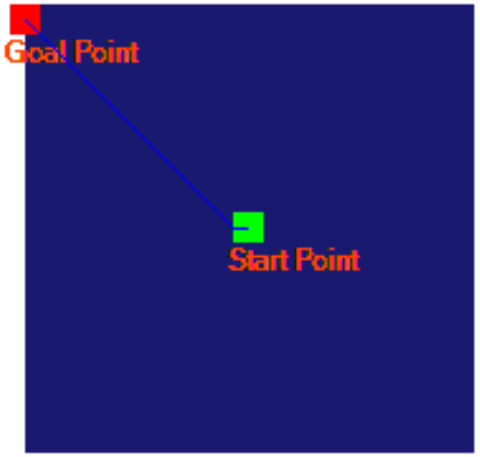
این تابع مختصات یک نقطه را میگیرد و هر یک از ۸ نقطه ی اطراف آن را اگر تله نباشند و با توجه به شرایط الگوریتم ها به آرایه table اضافه میکند. و همه ۶ ستون آن را با توجه به شرایط پر میکند.

در بعضی از الگوریتم ا به علت زیاد شدن فضای جستجو و کندی ۴ نقطه اطراف آن یعنی بالا ، پایین ، چپ و راست به آرایه اضافه میشوند.

الگوریتم اول سطح: Breadth_first_Search()

در این الگوریتم همه ی نقاط (گره های) اطراف یک نقطه جستجو میشوند و به آرایه table اضافه میشوند

در هنگام اضافه کردن نقاط اطراف یک نقطه در این روش توجه میشود که این نقطه از قبل براش تابع پسین نخورده باشد یعنی نقاط اطراف این نقطه تولید نشده باشند، برای این کار لازم نیست جستجویی انجام شود برای این کار هر موقع برای نقطه ای خواستیم تابع پسین فراخوانی کنیم array[x,y]=4 میشود (x,y مختصات گرهی است که خواستیم برای این گره تابع پسین فراخوانی کنیم است) و در مراجعات بعدی اگر مختصات مربوطه ۴ بود این نقطه برایش نقاط اطراف تولید نمیشود

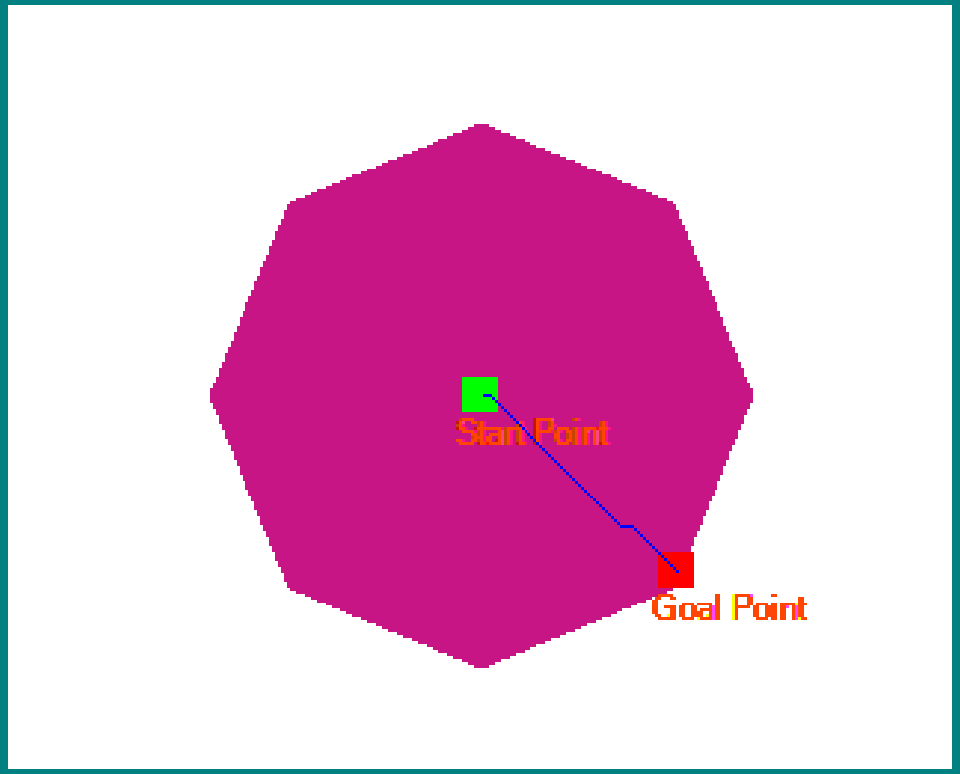


با توجه به شکل این الگوریتم همه ی سطوح اطرافش را تا رسیدن به هدف دنبال میکند.

این الگوریتم کامل وبهینه است

الگوریتم هزینه یکنواخت: Uniform_cost_Search()

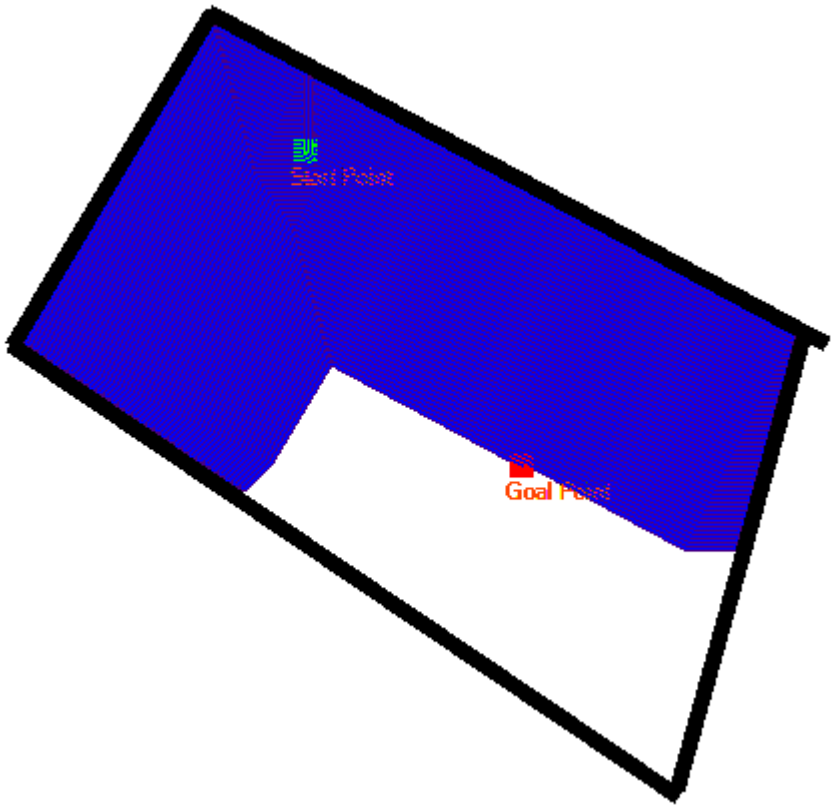
در این الگوریتم به این صورت عمل میشود که هر کدام از گرهها که کمترین میزان مسیر را داشتند برای ادامه انتخاب میشوند (هزینه حرکت اریب جذر عدد ۲ است) تابع min_g() اندیس گرهی که کمترین g را دارد برمیگرداند. چون در این فضای جستجو هزینه مسیر تغریبا یکسان است بجز در حرکت اریب شبیه اول سطح جیتجو میکند.



این الگوریتم کامل وبهینه است

الگوریتم اول عمق: Depth_first_Search()

در این الگوریتم به این صورت عمل میشود که اولا برای هر کدام از گرهها فقط ۴ نقطه اطرافش تولید میشود در نتیجه هزینه مسیر ۱ است و عمق برابر با g است وقتی هر با انتخاب بیشترین g پیش میرویم (تابع max_g() بیشترین g را بر میگرداند) اول عمق حاصل میشود.

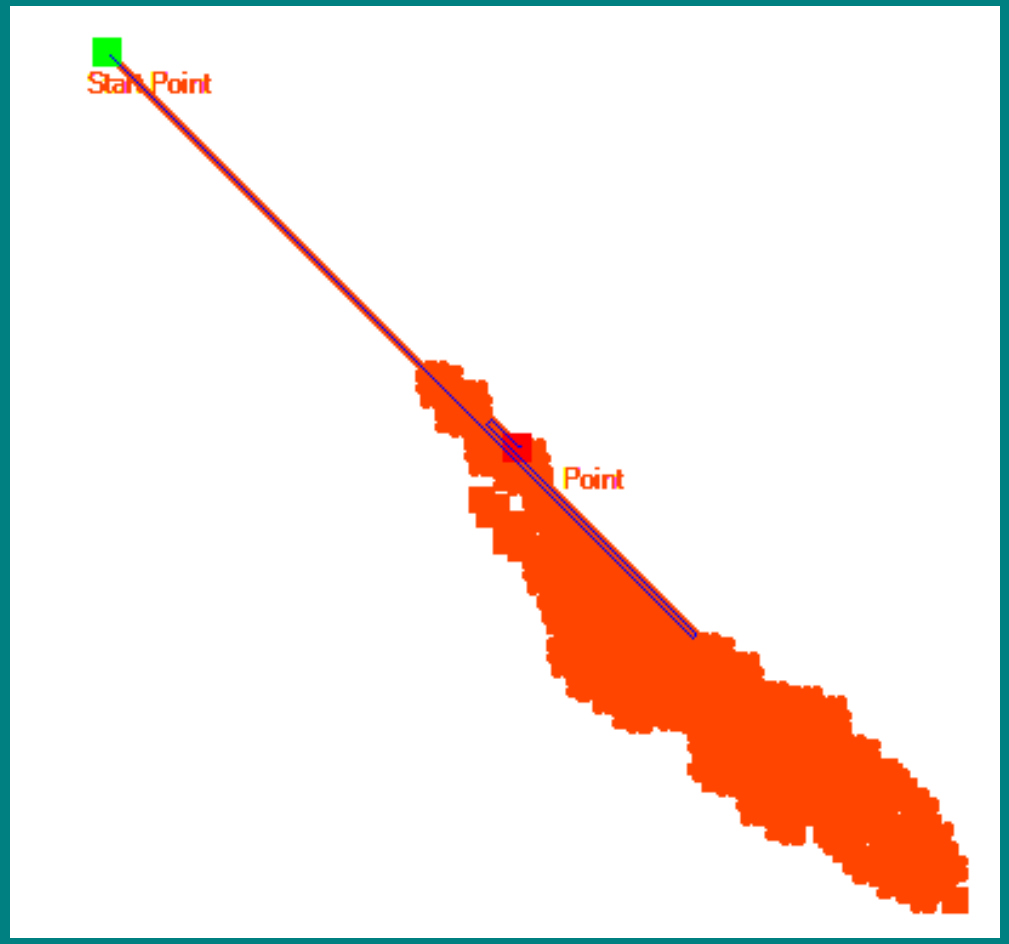


این الگوریتم در این فضای جستجو کامل است

ولی بهینه نیست

الگوریتم عمق محدود: Depth_limited_Search()

این الگوریتم هم مانند اول عمق است با این تفاوت که اگر تابع max_g مقداری بیشتر از عمق مسخص شده نداشته باشد.شکل زیر یک مثال با عمق محدود ۵۰۰ را نشان میدهد.



این الگوریتم کامل نیست چون ممکن است جواب در عمق پایین تری باشد وبهینه هم نیست

الگوریتم عمیق شونده تکراری: Iterative_deeping_Search()

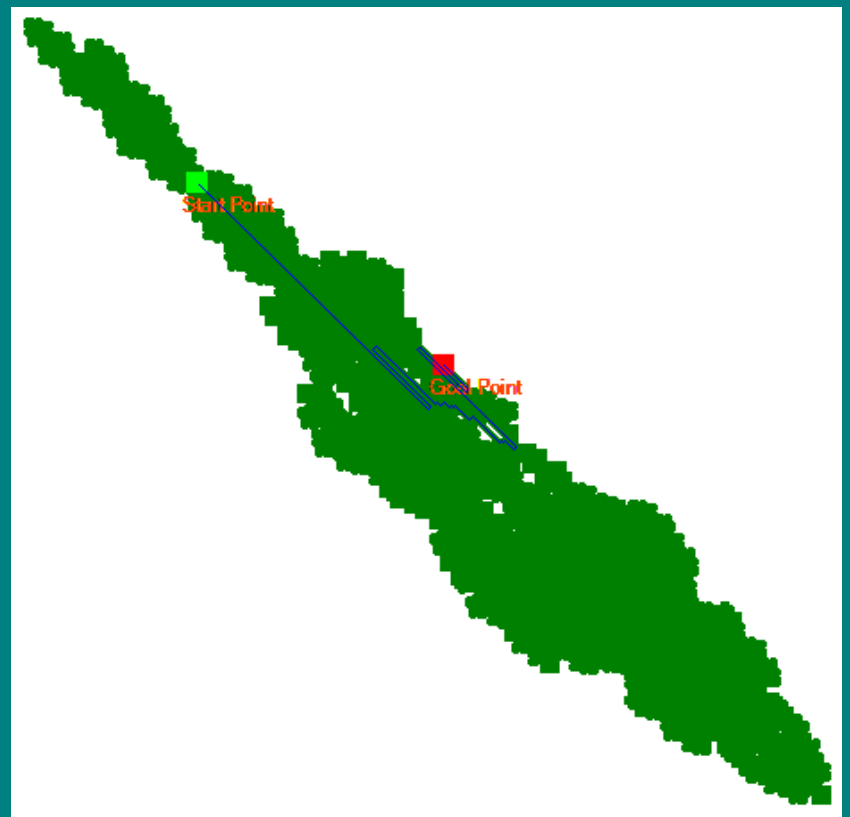
این الگوریتم عمق محدود را پی در پی از اول تکرار میکند و هر بار به عمق جستجو اضافه میکند (ما در این مورد در هر بار ۱۰۰ تا به عمق اضافه کردیم) این عمل تا وقتی که الگوریتم به جواب برسد ادامه میباید.

این الگوریتم کامل است

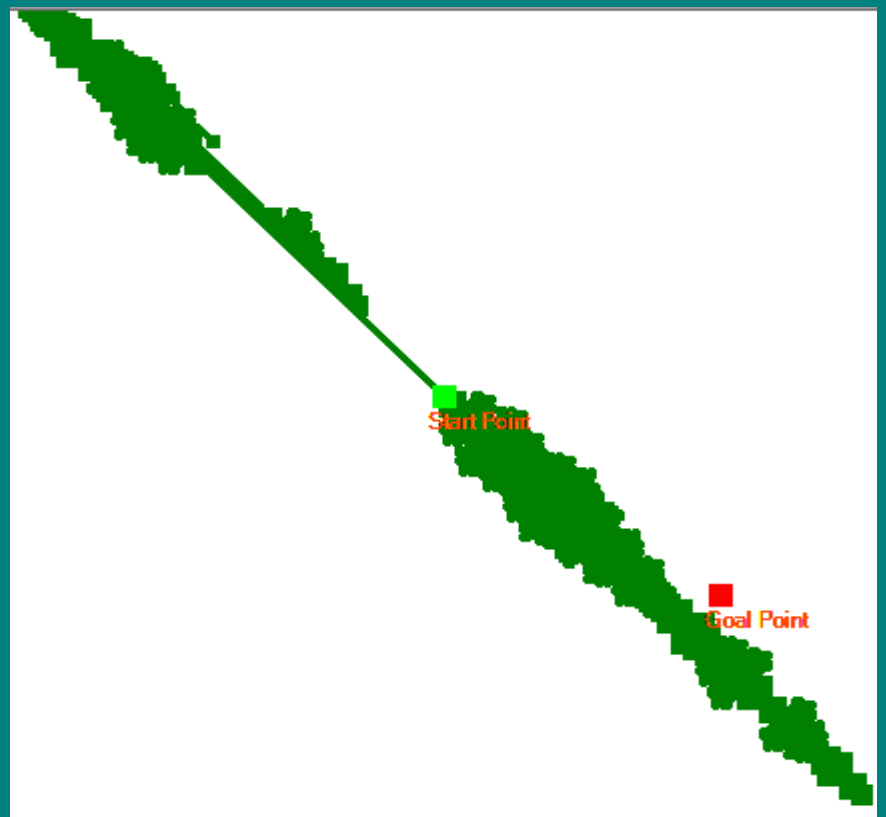
ولی بهینه هم نیست

اگر هر بار به عمق یکی اضاف میکردیم بهینه بود.

عمق ۵۰۰



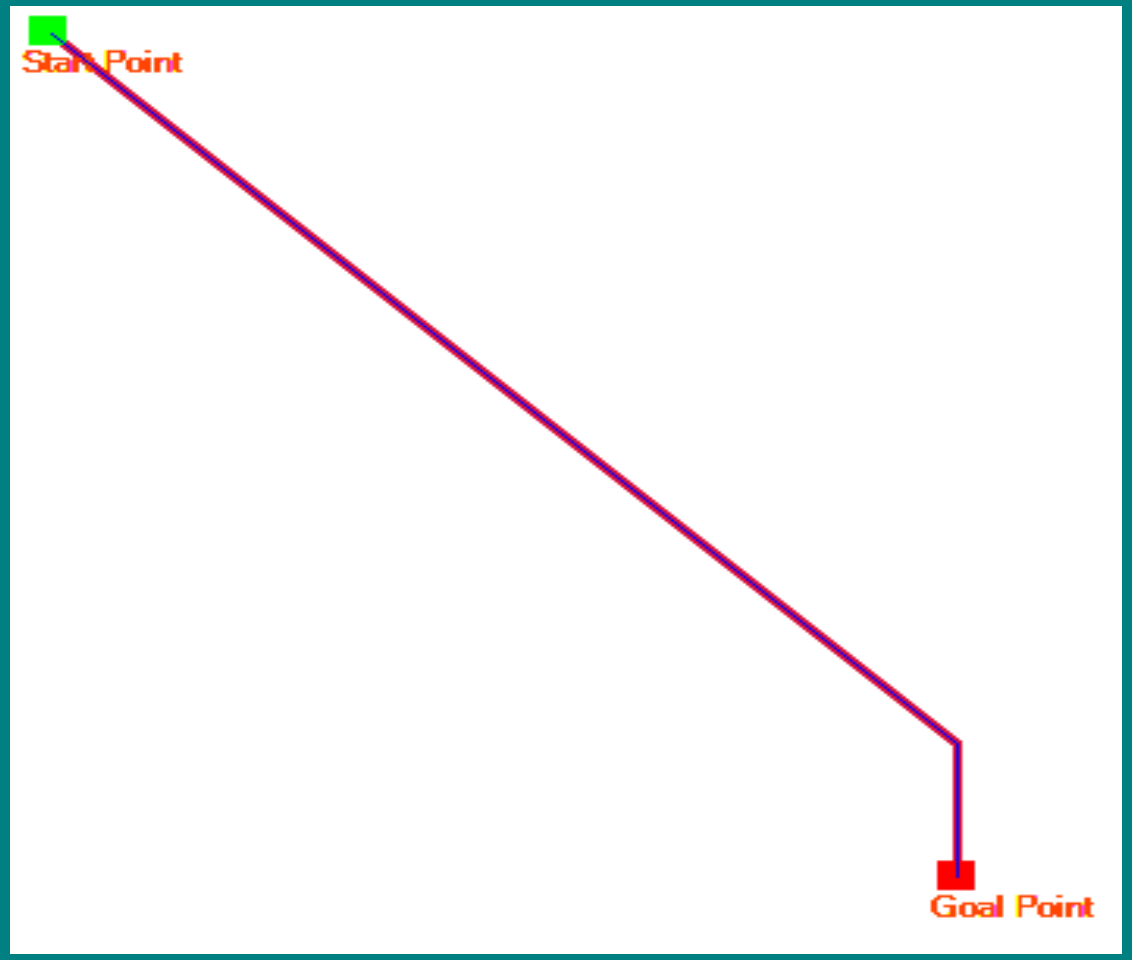
عمق ۲۰۰



الگوریتم حریصانه: Best_first_Search()

این الگوریتم گرهی را برای تدامه انتخاب میکند که دارای کمترین میزان هیوریستیک فاصله مستقیم

همان h ستون چهارم جدول جستجو را داشته باشد تابع min_H() اندیس این گره را بر میگرداند.

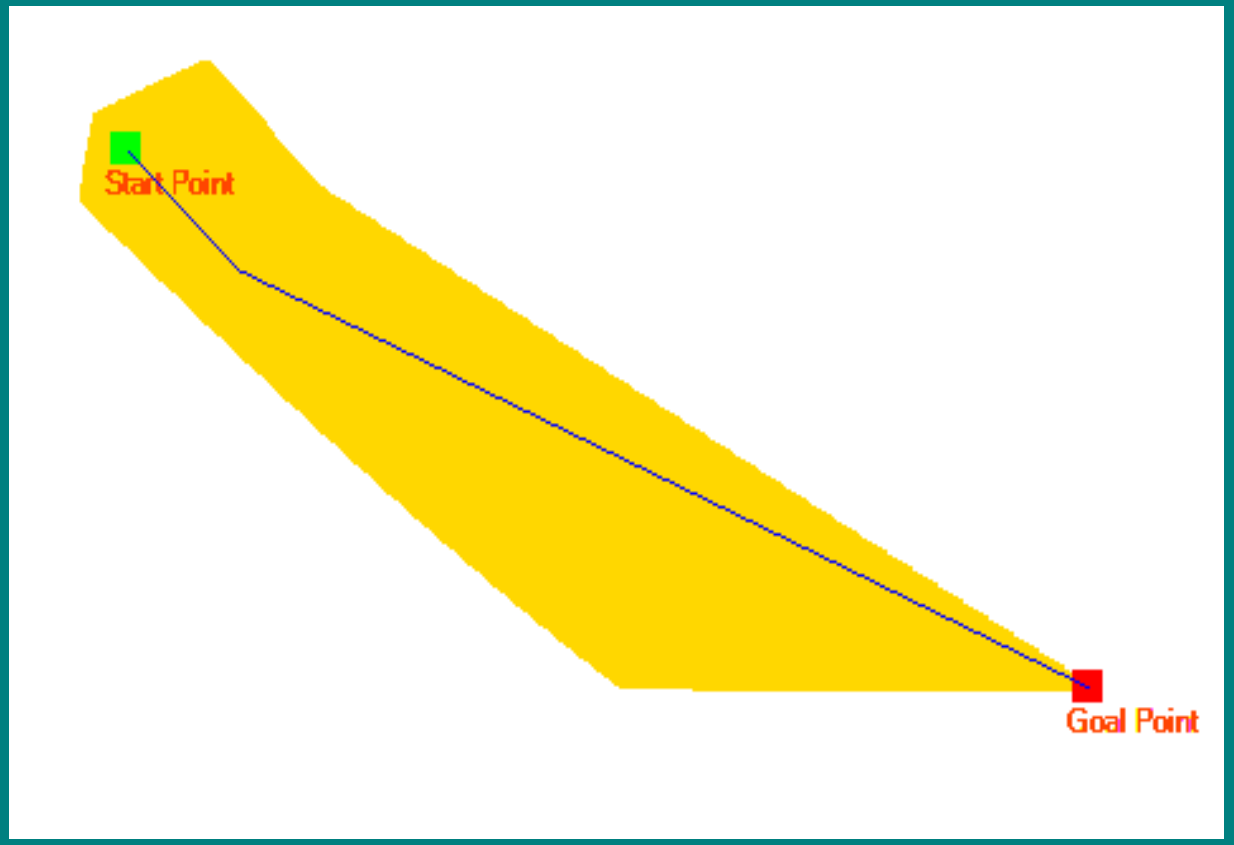


این الگوریتم در این فضای جستجوکامل است

ولی بهینه هم نیست.

الگوریتم A*: A()

این الگوریتم گرهی را برای ادامه انتخاب میکند که دارای کمترین میزان h+g باشد یعنی گرهی که مجموع هزینه مسیر از آغاز بعلاوه هیوریستیک فاصله مستقیم تا هدف آن کمترین باشد. تابع min_F() اندیس این گره را برمیگرداند.



این الگوریتم کامل است و بهینه

با تشکر