تقرير مشروع خوارزميات البحث الذكية

2020-2021

المسألة الثانية:

اسم اللعبة: Blob War Game

اللغة المستخدمة:C++

المسؤول عن تنفيذها: برزاني برمجة

أولاً - البنية المستخدمة :

تم اختيار مصفوفة ثنائية كبنية لتمثيل رقعة اللعب لما توفره من سرعة وصول لأي موقع داخله بتعقيد يصل إلى O(1) , تتكون المصفوفة الثنائية من Cells حيث يعبر كل Cell عن احداثيات الكرة حيث يكون X يعبر عن مكان الكرة على مستوى الصفوف و Y يعبر عن مكان الكرة على مستوى الاعمدة ويكون الشكل النهائي كالتالي : Grid[x][y]= player.symbol وتم تنفيذ المصفوفة الثنائية ك vector<vector<char>> بدلا من مصفوفة ثنائية عادية لسهولة التعامل مع الفيكتور في لغة ال c++ والابتعاد عن المشاكل التي تسببها ال pointers عند استخدام مصفوفات عادية. قمنا بانشاء struct Player يحوي على الخواص التالية id,score , symbol والتي ستساعد في تحديد رمز ونتيجة ومحدد كل لاعب في اللعبة في حال تم اللعب ب two players او againt AI . كما أنشئنا struct Move يحوي الخواص التالية x,y,nextX,nextY والتي ستساعد في عملية البحث عن افضل حل في الخطوات التي تليها وتجمع بين الاحداثي الحالي للكرة والاحداثي القادم المحتمل للكرة نفسها.

ثانياً – التوابع المستخدمة:

في هذا القسم سيتم شرح كل تابع قمنا باستخدامه لتنفيذ المطلوب داخل اللعبة مع شرح موجز لما يقوم به التابع.

1-Void initializePlayers()

يقوم هذا التابع بعمل initialize للاعبين حيث يعطي قيم id, score , symbol للاعبين قبل بداية كل لعبة ويمكن التعديل على ايا كان من خواص اللاعبين داخل هذا التابع .

2-Grid makeGrid(int rows,int cols)

يقوم هذا التابع ببناء رقعة اللعب بناءاً على قيم البارميترز المرسلة إليه كما يقوم باسناد الاحداثيات الابتدائية لكرات كل من اللاعبين على الشكل التالي :

```
Choose your Board Dimensions:
4 4
1 - - 0
- - - -
- - - -
1 - - 0
```

3-Void printGrid(Grid grid)

يقوم هذا التابع بطباعة رقعة اللعب الحالية كما هو موضح أعلاه .

4-bool isFinished(Grid grid)

يقوم هذا التابع بإرجاع قيمة true اذا كانت اللعبة قد انتهت او false اذا كانت لم تنتهي بعد وفق الخوارزمية التالية:

تنتهي اللعبة بثلاث حالات :1- اذا وصل Score احد اللاعبين إلى صفر 2- اذا وصل score احد اللاعبين إلى عدد الرقع داخل اللعبة 3- اذا كان مجموع scores اللاعبين يساوي عدد الرقع داخل اللعبة.

ملاحظة: تعبر score كل لاعب عن عدد الكرات التي لديه في الوقت الحالي.

5-bool isInBoard (Grid grid,int x,int y) & bool isEmpty (Grid grid,int x,int y)

يقوم التابع الاول بمعرفة ان كان الاحداثي المعين داخل حدود الرقعة ام لا أم التابع الثاني يقوم بمعرفة ان كان الاحداثي المعين فارغ ام لا (اي يوجد كرة في تلك الاحداثيات ام لا)

6- Coords getRaduis(Grid grid, int x, int y)

يقوم هذا التابع بمساعدة التابعين السابقين بإيجاد الرقع الممكن الانتقال إليها المحيطة به درجة واحدة فقط اي الرقع المحيطة به مباشرة , ويعيد التابع predfiened data type Coords و المكون من vector<pair<x,y>>

7- bool inRaduis(Grid grid, int x, int y, int nextX,int nextY)

يقوم هذا التابع بالاعتماد على التابع السابق بمعرفة ان كان الاحداثي التالي الممرر داخل محيط الرقعة الحالية مباشرة ام لا

8- Coords getDoubleRaduis(Grid grid, int x, int y)

يقوم هذا التابع بإيجاد الرقع الممكن الانتقال إليها المحيطة به درجتين اي الرقع المحيطة به مباشرة والرقع المجاورة لها , ويعيد التابع predfiened data type Coords و المكون من vector<pair<x,y>>

9- vector<Move> getPossibleMoves(Grid grid, int x, int y,Player player)

يقوم هذا التابع بإرجاع جميع الحركات الممكنة للاعب ما , حيث يقوم بالمرور على الرقعة باكملها وعند وجود كرة ل اللاعب المطلوب يقوم بايجاد جميع الحركات الممكنة له عبر تابع getDoubleRadius ويقوم بارجاع جميع الحركات الممكنة من كل احداثي والى كل احداثي ممكن عن طريق ال struct Move

10- bool canMove(Grid grid, int x, int y, int nextX,int nextY,Player player)

يقوم هذا التابع بمعرفة ان كانت الحركة ممكنة من احداثي إلى احداثي اخر عن طريق التأكد من عدم وجود كرة في الرقعة التالية .

11- Grid convertBlobs(Grid grid,int x,int y,Player player)

اعتمادا على قواعد اللعبة المذكورة يقوم هذا التابع بتحويل كل الرقع المحيطة بالكرة و التي تحوي كرة للخصم بالكرة الى كرات لنفس اللاعب , اي تحويل الوان الكرات الى لون كرة اللاعب .

12- int getScores(Grid grid,Player player)

يقوم هذا التابع بالمرور بإرجاع ال score اي عدد كرات اللاعب المعين داخل الرقعة بالمرور على الرقعة وزيادة العداد عند كل كرة تنتمي الى اللاعب .

13- Void printScores()

يقوم هذا التابع بطباعة نتائج اللاعبين على شاشة الاستخدام .

14- Grid makeMove(Grid grid, int x,int y,int nextX,int nextY,Player player)

يقوم هذا التابع بالقيام بالحركة الفعلية للكرة من احداثي إلى احداثي اخر والقيام بالتغييرات بالاعتماد على التوابع السابقة وتكون خوارزمية الحركة اعتمادا على قواعد اللعبة كالتالي :

1- يتم التأكد من كون الحركة قانونية عن طريق تابع canMove

2- اذا كان الاحداثيات التالية داخل محيط الكرة بشكل مباشر inRadius يتم بنسخ الكرة الى الاحداثيات التالية ويتم استدعاء تابع convetBlobs للقيام بالتغييرات الممكنة اي تحويل كرات الخصم المحيطة بالاحداثي الجديد الى كرات للاعب نفسه.

3- اذا كان الاحداثيات التالية خارج محيط الكرة بشكل مباشر اي داخل ال getDoubleRadius يتم نقل الكرة من الاحداثي الأول إلى الأحداثي المطلوب بدون نسخ ويتم استدعاء تابع convetBlobs للقيام بالتغييرات الممكنة اي تحويل كرات الخصم المحيطة بالاحداثي الجديد الى كرات للاعب نفسه.

4- يتم تحديث scores اللاعبين عن طريق تابع getScores

15- int getWinner()

يقوم هذا التابع بإرجاع ال id الخاص بالرابح وذلك عن طريق المقارنة بين score كل لاعب .

16- int evaluate(Grid grid)

يقوم هذا التابع بالقيام بعملية evaluate اي تقييم حالة الرقعة ويتم ذلك عن طريق الخوارزمية التالية:

1- اذا انتهت اللعبة يقوم عن طريق تابع getWinner الذي يعيد ال id الخاص بالفائز فاذا كان الفائز هو الخصم فيعيد قيمة سالبة كبيرة والا يعيد قيمة موجبة كبيرة .

2- اذا لم تنتهي اللعبة يعيد فرق عدد الكرات بين اللاعب والخصم .

17-int minimax(Grid grid ,int depth,bool isMax)

يقوم هذا التابع العودي بايجاد افضل قيمة ممكنة من حركة ما عن طريق محاكاة لعب اللعبة الى عمق محدد يحدده المستخدم اي كل ما زاد العمق المحدد يزداد نسبة ايجاد الحركة الافضل ولكن يزداد زمن التنفيذ ايضا(زمن البحث)

وتعمل الخوارزمية العودية هذه بالنسبة اللعبة كالتالي :

1- اذا وصلنا الى العمق 0 او اللعبة قد انتهت قبل الوصول الى العمق 0 قم بأرجاع قيمة ال evaluate لهذه الرقعة.

2- اذا كان عدد الحركات الممكنة لكل لاعب تساوي الصفر قم بإرجاع قيمة التابع ال evaluate لهذه الرقعة .

3- اذا لم يتحقق احد ال base cases السابقين يتم التفاضل بين ان كان العمق الحالي maximizer او minimizer عن طريق القيمة الممررة كابرميتر isMax :

I. اذا كان maximizer نقوم باسناد قيمة سالبة كبيرة الى المتغير best و نوجد جميع الحركات الممكنة للاعب ونمر عليها بحلقة for ونقوم بالقيام بكل حركة ممكنة عن طريق تابع makeMove ونمرر الرقعة الجديد الناتجة الى الاستداعاء العودي التالي بالطريقة التالية:

best=max(best,minimax(grid,depth-1,!isMax))

اي نأخذ القيمة الأكبر بين ال best وبين ناتج الأستداعاء العودي ونقلل ال depth للاستدعاء التالي مع التأكيد على كون الاستدعاء التالي هو minimizer والذي يقلل فرص فوز الخصم .

II. اذا كان minimizer نقوم باسناد قيمة موجبة كبيرة الى المتغير best و نوجد جميع الحركات الممكنة للخصم ونمر عليها بحلقة for ونقوم بالقيام بكل حركة ممكنة عن طريق تابع makeMove ونمرر الرقعة الجديد الناتجة الى الاستداعاء العودي التالي بالطريقة التالية:

best=min(best,minimax(grid,depth-1,!isMax))

اي نأخذ القيمة الأَصغر بين ال best وبين ناتج الأستداعاء العودي(لاننا هنا نريد ان يكون ناتج ال evaluate للخصم اقل ما يمكن) ونقلل ال depth للاستدعاء التالي مع التأكيد على كون الاستدعاء التالي هو maximizer والذي يزيد فرص فوز اللاعب .

4- بكلا الحالتين نعيد قيمة best والذي سيتخدم للاستدعاءات العودية التالية.

18-int alphaBeta(Grid grid ,int depth,bool isMax,int alpha ,int beta)

يقوم هذا التابع بما يقوم به التابع السابق minimax من ناحية طريقة العمل والهدف من استدعائه ولكن يختلف اختلاف جوهري يجعل منه تحسين لتابع ال minimax , من ناحية خوارزمية الحل الخاصة به فهي تطابق خوارزمية حل التابع السابق مع اختلافات معينة لذلك لن يتم كتابة الخوارزمية مرة وأنما ذكر الاختلافات فقط :

يختلف هذا التابع عن التابع السابق بكونه اسرع اي يكون زمن تنفيذه أقل بكثير و يعزى هذا الاختلاف إلى القيام بقطع في شجرة البحث عندما تكون النتيجة المتوقعة القادمة من شجرة البحث لن تغير النتائج فيقوم التابع بتجاهل ال branch باكمله اما تابع ال minimax فيقوم بالمرور على كامل الشجرة , ويتم هذا ال cut branch عن طريق 2 parameters وهم alpha , beta

حيث عوضا عن متغير best الموجود سابقا والذي يتم اعادة اسناده بقيمة كبيرة ان كانت سالبة او موجبة , يتم استخدام alpha, beta الممررتان خلال الاستدعاء العودي وتتم عملية القطع بالخوارزمية التالية :

1- اذا كان maximizer نتحقق من قيمة المعادة من الاستدعاء العودي التالي اذا كان اكبر او تساوي من beta فتوقف عن البحث وارجع beta اي قم بعمل cut لذلك ال branch والا اسند القيمة المعادة الى ال alpha وقم بارجاعها .

2- اذا كان minimizer نتحقق من قيمة المعادة من الاستدعاء العودي التالي اذا كان اصغر او تساوي من alpha فتوقف عن البحث وارجع alpha اي قم بعمل cut لذلك ال branch والا اسند القيمة المعادة الى ال beta وقم بارجاعها .

ملاحظة : تكون قيمة alpha عند أول استدعاء لتابع قيمة سالبة كبيرة , وتكون قيمة beta قيمة موجبة كبيرة .

19- Move findBestMove(Grid grid, int depth,Player player,int algo)

يقوم هذا التابع باستدعاء احد التابعين السابقين والحصول على القيمة المرجعة من احدهم لمعرفة ان كانت الحركة الممكنة جيدة ام لا حيث يعيد افضل حركة والتي حصلت على افضل قيمة مرجعة من احد الخوارزميتين السابقتين وتكون الخوارزمية بالشكل التالي:

1- initialize ال bestMove وbestVal

2- ايجاد جميع الحركات الممكنة للاعب والمرور عليها بحلقة for

3- القيام بالحركة وتمرير الرقعة الى احد الخوارزميتين السابقتين (يتم تحديد الخوارزمية عن طريق البارميتير algo)

4- اذا كان ناتج الخوارزمية اكبر من ال bestVal نقوم باسنادها الى ال bestval ونقوم باسناد الحركة ال bestMove

5- قم باعادة bestMove .

20 -void twoPlayers(Grid grid)

يقوم هذا التابع بتشغيل اللعبة على وضع لاعبين اثنين اي بدون تدخل الكومبيوتر AI وذلك عن طريق الخوارزمية التالية:

1- طالما اللعبة لم تنتهي (!isFinished()

2- اطبع الرقعة والنتائج واطلب من اللاعب ادخال الاحداثيات from والاحداثيات To

3- قم بالحركة المطلوب (makeMove()

4- قم بتغيير اللاعب الى اللاعب التالي .

5- اذا انتهت اللعبة اطبع الرقعة والنتائج .

21 -void AIplayer(Grid grid,int algo)

يقوم هذا التابع بتشغيل اللعبة على وضع لاعب و AI ان كان minmaxاو AI alphabeta وذلك عن طريق الخوارزمية التالية:

1- طالما اللعبة لم تنتهي !isFinished()

2- اطبع الرقعة والنتائج واطلب من الخصم(الانسان) ادخال الاحداثيات from والاحداثيات To

3- قم بالحركة المطلوب makeMove()

4- قم بتمرير الرقعة الناتجة الى التابع findBestMove() .

5- قم بالحركة الناتجة عن التابع السابق makeMove()

5- اذا انتهت اللعبة اطبع الرقعة والنتائج .

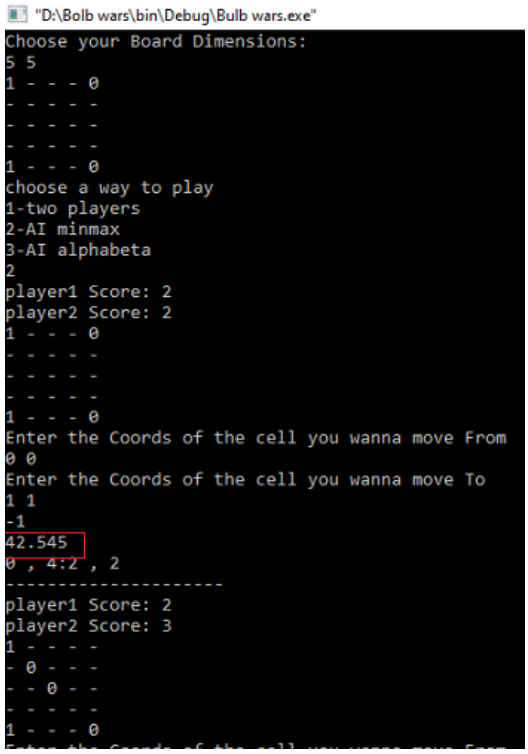
ملاحظة : يتم تحديد الخوارزمية المطلوب عن طريق البارميتر algo قيمة 1 تعبر عن minmax و 2 تعبر عن alphabeta

21 - Game()

مهمة هذا التابع هو تشغيل اللعبة باكملها ومساعدة المستخدم باختيار ابعاد الرقعة والطريقة اللعب .

ثالثا - مقارنة في وقت التنفيذ والسرعة بين خوارزمية minmax و alpha Beta :

MinMax : 42 s



AlphaBeta : 5.5 s