جامعة دمشق

كلية الهندسة المعلوماتية

السنة الرابعة(برمجيات)

خوارزميات البحث الذكية

مشروع خوارزميات بحث ذكية

Smart Algorithm

تقديم:

ندیم احمد عیروطه أحمد خالد الخوالده

عبد الهادي اسماعيل الهلال أحمد عبد الحكيم شباط

❖ شرح وظائف الكلاسات مع الدوال التي تحويها :

1) الكلاس public class Stone{} :

هذا الكلاس المسؤول عن تخزين معلومات احجار اللعب و هو الذي يمكنا من التعامل مع الحجر كقيمة نصية او لون ممثل بدائرة و تكون وظيفة الباني في هذا الكلاس .

• public Stone(char color,int row){}:

هذه الدالة المسؤولة عن تهيئة القطع البيضاء و السوداء في بداية اللعبة بحيث تقوم بتحديد لون و موقع كل قطعة في رقعة اللعب.

2) الكلاس public class Board {} :

هذا الكلاس يمثل اللعبة يمثل رقعة اللعب بكامل تفاصيلها مثل مكان كل حجر و محتوى كل خانة من خانات اللعب بالاضافة الى قواعد اللعب و الفوز و كيفية ادوار اللعب و سنقوم بشرح وظيفة كل دالة داخل هذا الكلاس :

• public Board(){}:

هذا الباني الخاص بالكلاس و الذي يتم فيه تهيئة الرقعة للعب حيث يقوم بوضع قطع اللعب في اماكنها الصحيحة اي توزيع القطع البيضاء و السوداء على اول 14 خانه و من خلال الدالة intilize() يقوم بتوزيع العلامات المخصصة على المربعات بحيث لكل واحدة مكان محدد وصفة معينة و في الاخير يقوم بتوزيع علامة مخصصة و التي تدل على مربع فارغ ويضعها في امكنتها الصحيحة .

• public void intilize (){}:

ذكرنا شرحها سابقا من خلال المنشئ فهي تحدد العلامات المخصصة و تضعها في اماكنها الصحيحة في الرقعة.

• public Board cloneBoard(){}:

هذه الدالة مسؤولة عن نسخة الرقعة الحالية بكامل تفاصيلها المهمه بحيث نقوم بانشاء كائن جديد و لكن ليس بالقيم الافتراضية و انما بقيم الحالة الحالة اي هذا الكائن سيكون مستقل عن الكائن السابق بحيث يقوم في البداية بانشاء الرقعة الجديدة ثم يقوم بنسخ القطع و نسخ العدادات للقطع البيضاء و السوداء و في النهايه يقوم بنسخ اماكن القطع الخاصة .

- **public int ValueStick(){}:**

هذه الدالة هي المسؤولة عن معالجة رمي الاعواد الاربعة و القيم الناتجة عنها بحيث تستخدم دالة تولد قيمة عشوائية لكل عود اما صفر او واحد و يأخذ الوجه الابيض قيمة صفر و الوجه الاسود قيمة واحد ثم تحسب عدد الوجوه االبيضاء و السوداء للحصول على القيمة النهائية لعملية الرمي و تكون الحالتين الخاصتين في حال جميع الاعواد الناتجة بيضاء تكون النتيجة النهائية 5 و إذا كانت جميعها سوداء تكون النتيجة 4 و في ما عدا ذالك تكون النتيجة بعدد الوجوه السوداء الناتجة.

- **public void print(){}:**

هذه الدالة تقوم بطباعة رقعة اللعب في كل خطوة و بما اننا ممثلين الرقعة بمصفوفة احادة لسهولة الوصول و نريد في الطباعة ان تكون الرقعة من ثلاث صفوف اي مثل الرقعة الحقيقة فتقوم هذه الدالة بقسيم الصف الذي يمثل 30 مربع الى ثلاث صفوف كل صف 10 مربعات بحيث نقوم بعكس اتجاه الطباعة عن انتهاء كل صف و ذلك حتى يبقى الترتيب بنفس ترتيب الرقعة الحقيقية بالاضافة الا ان الدالة تقوم بطباعة عدد القطع التي خرجت لكل لاعب لتوضيح الاعب المتقدم بالنتيجة على الاخر.

- **public void home(Stone stone){}:**

هذه الدالة تقوم بإدارة حركات اللعب التي تخالف قواعد اللعبة بحيث يتم ارسال القطعة المخالفة سوداء او بيضاء الى المربع 14 بيت الفرصة الجديدة و في حال كان بيت الفرصة الجديدة شاغر اي يحوي قطعة يتم ارسال القطعة الى اول مربع فارغ و يوضع الحجر الجديد الذي خالف احد القواعد في بيت الفرصة الجديدة و تكون القطعة مخالفة في حال كان لدي انتقال و لم يمر على المربع 26 بيت الحاجز او كان لدي انتقال و توقف في المربع 27 بيت الحفرة او اذا كان الحجر يقف عند 28 بيت الخطوات الثلاث و في الدور التالي لم يتم الحصول على رمية مقدارها 3 و حاولنا تحريك هذا الحجر او اذا كان الحجر يقف عند 29 بيت الخطوتين و في الدور التالي لم يتم الحصول على رمية مقدارها23 و حاولنا تحريك هذا الحجر او اذا كان الحجر يقف عند البيت 30 بيت الخطوة الإجبارية و في الدور التالي لم يتم تحريك الحجر لاخراجه.

- **public String getShape(Stone stone){}:**

هذه الدالة تقوم بتحويل الرمز المعبر عن اللون الى شكلة الحقيقي فاذا كان الحرف الراجع W تعيد الدالة دائرة بيضاء و في ما عدا ذالك يتم اعادة دائرة سوداء فهذه الدالة توفر كتابة الكثير من الجمل الشرطية للتحقق من نوع الحجر.

- **public boolean CanMove(Stone stone,int to){}:**

هذه الدالة المسؤولة من التحقق من صحة حركة الاحجار قبل تحريكها ففي البداية تقوم الدالة من التحقق من تواجد الحجر الذي نريد تحريكة ضمن حدود الرقعة و من ثم تتحقق من ان الحركة توافق قواعد اللعبة بحيث يمنع الهبوط على حجر من نفس اللون و يمنع المرور من فوق الحجر 25 و يمنع الخروج العشوائي دون تحقيق شروط البيوت .

- **public boolean NextStep(Stone stone,int value){}:**

هذه الدالة المسؤولة عن إدارة التحريك الفعلي للاحجار و كيفية التبادل فيما بينها و ذلك ضمن حدود قواعد اللعبة اي تطبيق قواعد اللعبة واقعيا بحيث تقوم باخذ القيمة الناتجة عن رمي العصي و تطبيقها على رقعة اللعب و ذلك مع مراعاة الحركات الممكنة للعب بحيث تحدد حالة الحجر بناء على المربع الذي تقف عليه ايا كانت الحركة سواء حركة عادية او حركة فوز و خروج ايضا بالاضافة الى انها المسؤولة عن الحركات الخاصة و ذلك عند تجاوز قواعد البيوت و الجزء الاهم ادارة المبادلات بين احجار الخصوم و التدافعات.

- **public boolean Move(Stone stone,int value) {}:**

نقوم بهذا التابع بتحقيق الحركة للحجر المحدد مع قيمة رمي العصي حيث نتحقق أولا من أمكانية الحركة فاذا كانت ممكنة نقوم بالتحقق من أن الحجر داخل الرقعة بعدها نقوم ثم نقوم بأنشاء متغير من نوع البوليان ونسند إليه القيمة المعادة من التابع (nextstep) ونتحقق اذا كان المتغير (true) ندخل إلى الشرط و نمر على جميع الأحجار في المصفوفة (stone1) ونحققلا أنها يجب أن يكون نفس لون الحجر المحرك و أن يكون داخل الرقعة فإذا حقق ذلك وكان في أحد الأماكن 27او28او29 يعو إلى الرقع رقم 14 ثم يقوم التابع بأعاده قيمة المتغير .

- **public char IsWin (){}:**

نتحقق اذا كان عدد الأحجار التي هي خارج الرقعة تساوي 7 للون معين يفوز وإلا يرجع c والتي تعني الاستمرار .

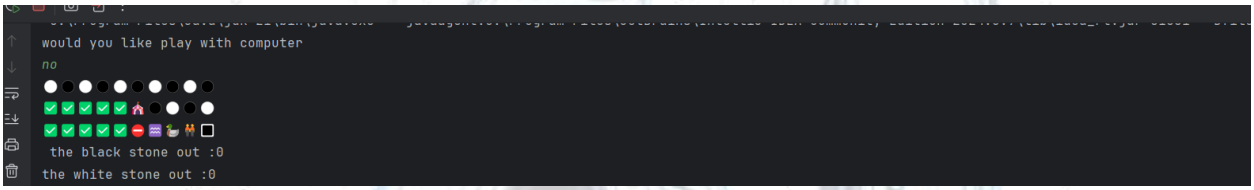
- **public void PlayGame() {:**

هذا التابع نقوم به بممارسة العبة بين لاعبين بالتناوب اللاعب 0 هو الذي يملك الأحجار البيضاء و اللاعب 1 هو الذي يملك الأحجار السوداء يقوم اللاعب باختيار رقم الرقعة الذي يوجد الحجر عليه

فأذا كان اللاعب اختار رقم رقعة تقع عليها احداثيات حجر معاكس للاعب فيطلب أعادة تعين رقم رقعة حجر ليتوافق مع احجاره .

- **public void PlayGameWithAI() {:**

نفس عمل التابع السابق لكن هنا الاعب الثاني هو الحاسوب حيث نستخدم خوارزمية Expectiminmax .



الجزء الثاني: Class Computer

في هذا الجزء سوف يتم شرح كلاس Computer وهو المسؤول عن لعب الكمبيوتر وعن الخوارزميات التي تختار افضل نتيجة, سوف نشرح التوابع المستدعاة بالتفصيل ثم شرح كيفية لعب الكمبيوتر:

public int bestmove(Board b, int val):

الهدف من كتابتنا لهذا التابع هو جعل الكمبيوتر يختار الحركات بشكل منطقي. فقمنا بعمل حلقة تمر على قطع الكمبيوتر وتختبر كل قطعة أنها متاحة للتحريك وتم عمل نسخة عن اللوحة الأصلية لتجربة الاحتمالات عليها وليس التعديل على اللوحة الاصلية. بعد ذلك تم استدعاء خوارزمية البحث expectiminimax لتوقع قوة هذه الحركة ، وتمت المقارنة بين النتائج لاختيار الحجر الذي يحقق أعلى درجة. وبالتالي التابع يرجع موقع القطعة الأفضل ويطبع ملخصاً يوضح الأحجار الممكن تحريكها مع احتمالاتها لاختيار الأفضل بينها.

private double expectiminimax(Board b, int d, boolean isMax):

هذا التابع هو الذي يجعل الكمبيوتر يتوقع الخطوات القادمة، فأول شيء يتأكد إذا اللعبة خلصت أو إذا وصلنا لأقصى عمق حددته للتفكير، وفي هذه الحالة يتم الطلب من تابع التقييم أن يعطي النتيجة الحالية للوحة. أما إذا لسا في مجال للبحث، ينظر الدور لمن؛ سواء كان دور الكمبيوتر الذي يحاول الربح (Max) أو دور الخصم الذي نفترض أنه يلعب ضدنا(Min) ، ويحول اللوحة لتابع chancenode لكي يحسب الاحتمالات بناءً على رميات العصي المتوقعة. وتم استخدام عداد الـ count هنا لمراقبة عدد المرات التي يتم فيها المرورعلى العقد.

private double chancenode(Board b, int d, boolean isMax):

هذا التابع تم انشاءه ليتعامل مع منطق الحظ في اللعبة، لأن رمي العصي ليس دايمًا يعطي نفس النتيجة، وصعب ان نعتمد على رقم واحد واحتمال واحد. وبالتالي تم تحديد مصفوفة فيها الاحتمالات لكل رمية، وتم عمل حلقة لتجريب كل الأرقام الممكنة من 1 لـ 5 , وفي كل مرة يتم طلب التابع moveNode لفحص افضل حركة، ثم يتم ضرب القيمة باحتمالية حدوثها. لضمان أن الكمبيوتر ما يبني قراره على رمية نادرة الحدوث، ويركز أكثر على الحركات اللي احتمالها أكبر.

رمية العصي	الاحتمال
5	$^{1}/_{16}$
1	$^{4}/_{16}$
2	$^{6}/_{16}$
3	$^{4}/_{16}$
4	$^{1}/_{16}$

private double moveMax(Board b, int d, int val):

هذا التابع مسؤول عن تحديد أفضل حركة ممكنة للكمبيوتر (الأحجار السوداء) في هذا الدور، يقوم التابع بالمرور على جميع الأحجار السوداء ومحاولة تحريك كل حجر يمكن تحريكه حسب قيمة العصي الحالية، لكل حركة محتملة، يتم إنشاء نسخة من اللوحة وتنفيذ الحركة عليها ثم تقييم الحالة الناتجة باستخدام الدالة expectiminimax() يتم مقارنة نتائج التقييم واختيار الحركة التي تعطي أعلى قيمة لأنها الأفضل للكمبيوتر. في حال لم تكن هناك أي حركة ممكنة، يتم تقييم وضع اللوحة الحالي مباشرة، في النهاية يعيد التابع أفضل نتيجة تم الحصول عليها.

private double moveMin(Board b, int d, int val):

هذا التابع يمثل دور اللاعب الخصم (الأحجار البيضاء) في اللعبة، حيث يحاول اختيار الحركة التي تعطي أسوأ نتيجة ممكنة للكمبيوتر، يقوم التابع بالمرور على جميع الأحجار البيضاء ومحاولة تحريك كل حجر يمكن تحريكه حسب قيمة العصي الحالية، لكل حركة صالحة يتم إنشاء نسخة من اللوحة وتنفيذ الحركة عليها لتجنب التأثير على الحالة الأصلية. بعد ذلك يتم تقييم الوضع الناتج باستخدام الدالة expectiminimax() مع تبديل الدور إلى الكمبيوتر، يتم اختيار الحركة التي تعطي أقل قيمة تقييم لأنها الأسوأ بالنسبة للكمبيوتر. في حال عدم وجود أي حركة ممكنة، يتم تقييم وضع اللوحة الحالي مباشرة وإرجاع نتيجته.

private double moveNode(Board b, int d, boolean isMax, int val):

هذا التابع يعمل كحلقة وصل بين دالتي moveMax() و moveMin() يقوم هذا التابع بتحديد أي دالة يجب استدعاؤها بناءً على الدور الحالي في اللعبة. إذا كان الدور للكمبيوتر (Max) يتم استدعاء moveMax() لاختيار أفضل حركة ممكنة. أما إذا كان الدور للاعب الخصم (Min) يتم استدعاء moveMin() لاختيار الحركة التي تقلل من نتيجة الكمبيوتر. بهذه الطريقة يتم تنظيم منطق اتخاذ القرار داخل خوارزمية اللعب بشكل واضح وبسيط.

private void printsummary(double bestscore, int bestmove):

هذا التابع يُستخدم لعرض ملخص عن عملية البحث أثناء اختيار الحركة، يقوم بطباعة عدد الحالات (العُقد) التي تم فحصها خلال تنفيذ الخوارزمية، كما يعرض أفضل قيمة تقييم تم الوصول إليها، بالإضافة إلى ذلك يتم طباعة موقع الحجر الذي تم اختياره لتنفيذ الحركة.

private double evaluate(Board b):

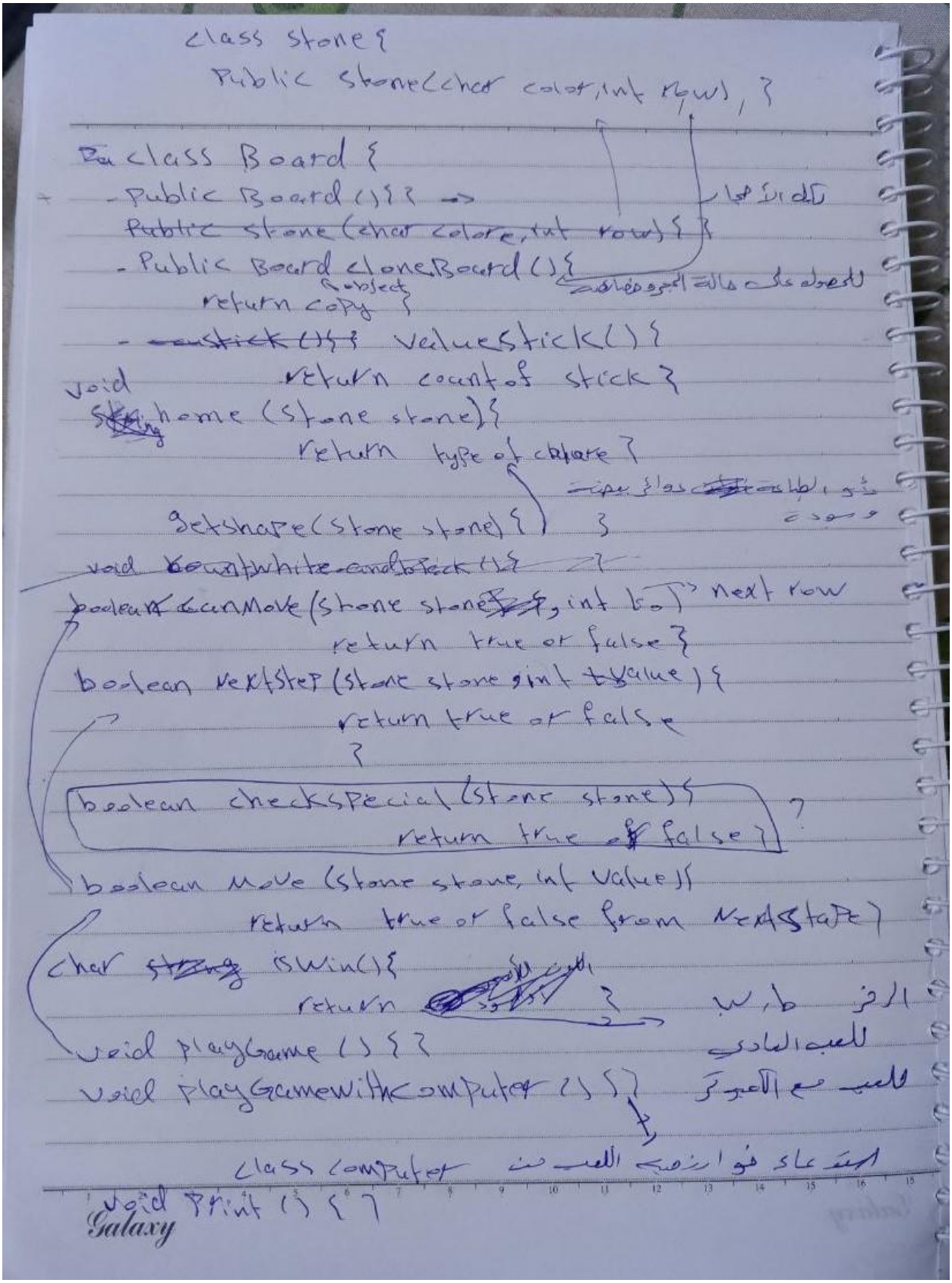
هذا التابع مسؤول عن إعطاء قيمة رقمية تعبّر عن حالة اللوحة الحالية من وجهة نظر الكمبيوتر. يقوم التابع بالمرور على جميع الأحجار الموجودة في اللعبة وحساب تأثير كل حجر على النتيجة النهائية. الأحجار السوداء تعطي قيمة موجبة لأنها تخص الكمبيوتر، وكلما تقدم الحجر أكثر على اللوحة زادت قيمته. أما الأحجار البيضاء فتعطي قيمة سالبة لأنها تخص الخصم وتقلل من نتيجة الكمبيوتر. في حال كان الحجر قد خرج من اللعبة، يتم إعطاؤه قيمة أكبر لأنه أقرب للفوز و في النهاية يتم جمع جميع القيم وإرجاع الناتج كتقييم عام للوضع الحالي في اللعبة.

سيناريو لعب الكمبيوتر:

تبدأ الرحلة عندما يحين دور الكمبيوتر ويستلم قيمة رمية العصي، ولنفرض أنها الرقم ثلاثة، يبدا الكمبيوتر بمسح كامل الأحجار الخاصة ،ويتأكد منها انها قابلة للتحريك، وعند إيجاد قطعة قابلة للتحريك يقوم بإنشاء نسخة ثانية كاملة من اللوحة الحالية لتجربة حركته عليها بحيث لا تؤثر على الرقعة الاصلية.

ثم يبدأ البرنامج بتخيل ردود فعل الخصم المحتملة، مفترضاً أن الإنسان سيلعب بأفضل طريقة ممكنة ليضره، وهذا ما يدفعه للغوص في شجرة الاحتمالات لعدة خطوات قادمة.

ولأن اللعبة اعتمادها على الحظ، فالكمبيوتر يوازن كل رمية باحتمالها الحقيقي؛ فلا يعامل الرمية النادرة بنفس أهمية الرمية المتكررة، بل يضرب كل نتيجة في نسبة حدوثها ليحصل على قيمة متوازنة. وعندما تصل شجرة البحث إلى نهايتها، نستخدم تابع eveulate لتقييم اللوحة، فيعطي score كبير للقطع التي خرجت من اللوحة، ويمنح نقاطاً أقل للقطع التي تقدمت لمربعات متقدمة قريبة من خط النهاية، مع خصم نقاط إذا كان وضع الخصم قريباً من الخروج. وبعد أن يتم فحص آلاف الاحتمالات، يقارن الكمبيوتر بين النتائج النهائية لكل قطعة قام بتجربتها، ليختار في النهاية القطعة التي حققت أعلى رصيد من النقاط والتي حصلت على اعلى score، ثم يتم عرض ما اختاره الكمبيوتر مع عرض احتمال كل حجر.


class stone{
Public stone(char color, int row,){
class Board {
- Public Board (){}
Public stone(char color, int row){}
- Public Board cloneBoard (){
return copy }
- valueStick(){
return countof stick}
void
home (stone stone){
return type of colore }
getshape(stone stone){ }
boolean canMove(stone stone, int to) next row
return true or false}
boolean nextStep(stone stone, int tvalue){
return true or false
}
boolean checkSpecial(stone stone){
return true or false}
boolean Move(stone stone, int value){
return true or false from NextState}
char isWin(){
return
void playGame (){}
void playGamewithComputer (){}
class computer
void Print (){}

/ /

* class computer → depth, count

حسنا نقيم اللوحة الحالية evaluate(Bord):

يرجع قيمة ال evaluate للوحة .

لما عمر الكمبيوتر يفكر .

وبعدين تابع هذا يجيبلنا افضل حركة للكمبيوتر

ويرجع اندكس الحجر يلي رح يحركو bestmove(Bord, stick)

خوارزمية اللعبة expectiminimax(Bord, depth, comp)

بترجع قيمة ال evaluate بعد ما عمق معين

~~calculateProbs()~~

chancenode() فيلزمنا تابع لحساب احتمالية هذه الحركة ←

board, depth, comp

calculateProbs() لحساب احتمالية كل رمية من العصي

نستدعيه في chancenode ← α لانه استخدام تطلب احتمالات ما جرى

bestmovemax(board, depth, stick) تجربة لعب الكمبيوتر لاختيار

اعادة score بعد تستدعي expectiminimax()

بشكل عودي

maxmin نفسها بس للاعب الثاني

minmaxAlphabeta() لتسرع عملية البحث ← نستخدم α β

print() طباعة