Logic magnets

الاسم: محمد فاعل

الفئة: 3

- ## توصيف الحالات:

تمثيل المسألة:

تم تمثيل المسألة على شكل شبكة من الخلايا الرقمية كل خلية بداخلها رقم يمثل ما حالة الخلية
تم استخدام بنية معطيات vector of vector
الرقم 0: الخلية فارغة
الرقم 1: الخلية تحوي على حلقة بيضاء
الرقم 2: الخلية تحوي على قطعة ذات لون رمادي
الرقم 3: الخلية تحوي قطعة ذات لون بنفسجي
الرقم 4: الخلية تحوي على قطعة ذات لون أحمر
الرقم 13: الخلية تحوي على حلقة بيضاء بداخلها قطعة ذات لون بنفسجي
الرقم 14: الخلية تحوي على حلقة بيضاء بداخلها قطعة ذات لون أحمر

شكل التخزين:

```
{0, 0, 0, 0, 0},
{1, 2, 1, 2, 0},
{0, 0, 3, 0, 1},
{0, 0, 4, 0, 1},
{0, 0, 0, 0, 0}
```

بنية الكود:

Utils.cpp:

يحتوي على التوابع المستخدمة في الحل والمساعدة في بناء منطق اللعبة

- **Display(vector)**

تابع لعرض الشبكة على الكونسول حيث يقوم بتحويل كل رقم الى رمز بلون معين

- **isSolved(vector)**

تابع التحقق من المرحلة اذا كانت محلولة ام لا
وذلك من خلال التأكد أن جميع الخلايا لا تحوي على الرقم 1 الذي يمثل حلقة فارغة

- **isValidPos(x, y, val)**

تابع يقوم بالتحقق فيما اذا كان الموقع المراد نقل القطعة فارغ
او يحوي حلقة بيضاء فارغة

- **isMovable(val)**

تابع يتحقق فيما اذا كانت القطعة قابلة للنقل ام لاز

- **replace(x, y)**

تابع للتبديل بين مواقع قطعتين

movements.cpp:

الملف الذي يحوي على توابع تطبيق الانتقالات و تأثير التنافر والتجاذب

- **move(x1, y1, x2, y2, vector)**

يقوم التابع بنقل القطعة من مكان لمكان بعد التحقق من إمكانية النقل من خلال التوابع المستخدمة في ملف utils.cpp الذي ذكر سابقا

يقوم بتطبيق التوابع التالية بالاعتماد على رمز القطعة(تنافر أو تجاذب)

- **repel(x, y, vector)**

يقوم بتطبيق تأثير تنافر القطع بشكل افقي وعمودي على القطعة المختارة

- **attract(x, y, vector)**

يقوم بتطبيق تأثير تجاذب القطع بشكل افقي وعمودي على القطعة المختارة

LEVELS.cpp:

ملف يخزن جميع المراحل وعدد الحركات المسموح بها في كل مستوى

طرق الحل

- **الحل من خلال المستخدم:**

في كل مرحلة يتم عرض عدد الحركات المسموح بها ويتم تحديث العدد بشكل ديناميكي بعد كل حركة.

يتم تحديث الشبكة وعرضها بعد كل حركة

يتم الانتقال الى المستوى التالي بعد انهاء المستوى السابق

- **الحل عن طريق خوارزميات البحث (BFS):**

بداية تم الحل عن طريق خوارزمية Breadth-First search

تم تمثيل كل حالة على شكل Struct يحوي:

- **الشبكة الحالية**
- **سجل التحركات الذي أدى الى الشبكة الحالية**

```
struct State
{
	vector<vector<int>> v;
	vector<vector<int>> moveHistory;
};
```

Search_Algorithm:

utils_functions.cpp:

ملف يحوي على مجموعة من التوابع المساعدة في حل وتطبيق خوارزمية BFS

- **getValidMoves(x, y, vector)**

تابع يقوم بإرجاع مجوعة من الأماكن التي يمكن الانتقال لها

- **applyMove(state, x1, y1, x2, y2)**

تابع يقوم بتطبيق انتقال معين

- **convertBoardToString(vector)**

تابع يقوم بتحويل الشبكة الى String لكي يتم تخزينها في Uunordered_set والتي قمنا باستخدامها لتمثيل الحالات التي تمت زيارتها لمنع التكرار

آلية الحل:

بنى المعطيات المستخدمة في الخوارزمية:

- **queue<State> q;**

تستخدم لتخزين مجموعة الحالات لإنها تدعم مبدا FIFO

- **unordered_set<string> visited;**

تستخدم لتخزين الحالات التي تمت زيارتها

تم اعتماد هذه البنية لإنها تخزن بالاعتماد على ال Hash_Tables

حيث يكون تعقيد البحث والحذف والإضافة بتعقيد O(1)

- **الحالة البدائية:**

يكون عدد الحالات البدائية في البداية هو جميع الحالات التي يمكن تحريكها من كل حالة بدائية يتم اتخاذ مسار جديد والنظر فيما اذا كان يؤدي الى حل معين

- **الحالة النهائية:**

تكون الحالة النهائية هي عندما لا تحوي الشبكة على أي خلية تحوي حلقة بيضاء

Thanks for reading…