ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Kỹ thuật Lập trình - CO1027

Bài tập lớn 2

BÍ KÍP LUYỆN RỒNG
Phần 2: Cuộc chiến với Chúa tể Rồng

Phiên bản 1.0

TP. HỒ CHÍ MINH, THÁNG 08/2025

ĐẶC TẢ BÀI TẬP LỚN

Phiên bản 1.0

1 Chuẩn đầu ra

Sau khi hoàn thành bài tập lớn này, sinh viên ôn lại và sử dụng thành thục:

- Hàm và lời gọi hàm.
- Các thao tác đọc/ghi tập tin.
- Con trỏ và cấp phát động.
- Lập trình hướng đối tượng.
- Danh sách liên kết đơn.

2 Dẫn nhập

Sau hành trình huấn luyện cam go, các chiến binh rồng đã dần thấu hiểu và gắn bó sâu sắc với bạn đồng hành của mình. Tuy nhiên, trong một buổi luyện tập bay đặc biệt, Răng Sún – chú rồng trung thành của Hiccup – đã bất ngờ dẫn cậu bay xuyên qua những tầng mây dày đặc và vô tình tiến vào một nơi chưa từng được nhắc đến trong các bản đồ: Hang rồng khổng lồ.

Tại đây, Hiccup chứng kiến một sự thật kinh hoàng: hàng ngàn con rồng từ khắp nơi đã bị khuất phục và đang sống như nô lệ dưới quyền điều khiển của một con Rồng Chúa khổng lồ, độc ác và tàn nhẫn. Chính con rồng này đã cưỡng ép các loài rồng khác phải rời tổ, bay đi khắp nơi cướp bóc lương thực từ các ngôi làng – bao gồm cả Berk. Nếu không mang được thức ăn về, chúng sẽ trở thành nạn nhân tiếp theo – trở thành món mồi cho Rồng Chúa.

Mặc dù rồng vốn là những sinh vật hòa bình và trung thành, nhưng trước sự đe doạ tàn bạo của Rồng Chúa, chúng buộc phải tuân phục. Bản chất hoà ái của rồng bị bóp méo thành bạo lực và hủy diệt, khiến mối quan hệ giữa loài người và loài rồng trở nên căng thẳng.

Nhận ra sự thật ẩn sau những cuộc tấn công vào làng Berk và những mất mát kéo dài suốt bao năm qua, các chiến binh trẻ tuổi cùng những chú rồng đồng hành đã cùng nhau đưa ra một quyết định táo bạo: mở cuộc tổng tiến công vào hang ổ Rồng Chúa, với mục tiêu không chỉ để tiêu diệt ác long đầu đàn, mà quan trọng hơn là giải phóng các loài rồng khỏi xiềng xích nô lệ, khôi phục hoà bình và niềm tin giữa người và rồng.

Chiến dịch Giải Phóng Rồng chính thức được khởi động. Mỗi chiến binh sẽ dẫn đầu một

đội, vượt qua các khu vực hiểm trở, đối mặt với các loại rồng tay sai, giải mã những mê cung dày đặc trong hang ổ, vượt qua bẫy và cạm bẫy nguy hiểm để tiến sâu vào trung tâm căn cứ đen tối. Chỉ khi đánh bại được Rồng Chúa, ánh sáng mới có thể chiếu rọi trở lại vùng đất của Berk và các giống loài rồng.

Trong bài tập lớn này, các bạn được yêu cầu hiện thực các lớp (class) để mô tả lại quá trình diễn biến của cuộc chiến. Chi tiết của các class cần hiện thực sẽ được nêu chi tiết trong các nhiệm vụ bên dưới.

3 Các lớp trong chương trình

Bài tập lớn này sử dụng Lập trình Hướng đối tượng để tái hiện cuộc chiến đấu với Chúa tể Rồng. Các đối tượng trong BTL được biểu diễn thông qua class và được mô tả như bên dưới.

3.1 Thành phần bản đồ

Mê cung trên hòn đảo bí ẩn – nơi được cho là nơi trú ngụ của Rồng Chúa – được biểu diễn bởi một bản đồ có kích thước (nr, nc). Bản đồ này là một mảng hai chiều gồm nr hàng và nc cột, mỗi phần tử đại diện cho một vùng địa hình cụ thể trên đảo, được mô tả bằng lớp MapElement. Bản đồ gồm ba loại phần tử chính:

- **PATH**: biểu diễn lối đi, các đối tượng có thể di chuyển trên phần tử này.
- **OBSTACLE**: biểu diễn chướng ngại vật, các đối tượng không được di chuyển trên phần tử này.
- **GROUND_OBSTACLE**: biểu diễn những chướng ngại vật nằm sâu dưới mặt đất – như rễ cây cổ đại, đá tảng, hay dấu tích của rồng cổ xưa. Những chướng ngại vật này chỉ ảnh hưởng đến đội Chiến binh mặt đất, nên hoàn toàn không gây trở ngại cho Chúa tể Rồng hay đội Chiến binh bay. Cụ thể, các Chiến binh mặt đất chỉ có thể vượt qua ô GROUND_OBSTACLE nếu họ sở hữu lượng sát thương đủ lớn, tức là `damage` phải vượt quá mức yêu cầu của ô chướng ngại đó. Nếu không đạt ngưỡng, họ không thể di chuyển theo hướng này.

Bên cạnh đó, các đối tượng chỉ được di chuyển bên trong bản đồ.

Cho trước định nghĩa của enum **ElementType** như sau:

```
enum ElementType { PATH, OBSTACLE, GROUND_OBSTACLE };
```

Yêu cầu: **Hiện thực** class **MapElement** với các mô tả sau:

1. Thuộc tính private tên **type** có kiểu là **ElementType** biểu diễn kiểu của thành phần bản đồ.
2. Thuộc tính private tên **req_dmg** có kiểu là **int** biểu diễn lượng sát thương yêu cầu của `GROUND_OBSTACLE`.
3. Phương thức khởi tạo (public) có 1 tham số truyền vào kiểu ElementType. Phương thức khởi tạo gán giá trị của tham số cho thuộc tính **type** và **req_dmg**.

```
MapElement(ElementType type, int in_req_dmg = 0);
```

4. Phương thức hủy ảo (virtual destructor) với quyền truy cập public.
5. Phương thức **getType** (public) được định nghĩa bên trong class như bên dưới:

```
ElementType getType() const;
```

6. Phương thức **getReqDmg** (public) được định nghĩa bên trong class như bên dưới:

```
int getReqDmg() const;
```

Yêu cầu: **Hiện thực** phương thức **getReqDmg** với mô tả sau:

MapElement còn cần gán giá trị của **in_req_dmg** cho **req_dmg**. Class **MapElement** có phương thức **getReqDmg** trả về giá trị của thuộc tính **req_dmg**. **in_req_dmg** được tính bằng công thức

```
in_req_dmg = (r * 257 + c * 139 + 89 ) % 900 + 1
```

với r và c lần lượt là vị trí theo hàng và cột của ô `GROUND_OBSTACLE`.

3.2 Bản đồ

Yêu cầu: **Hiện thực** Class **Map** để biểu diễn bản đồ theo các mô tả sau:

1. Thuộc tính private **num_rows** và **num_cols** đều có kiểu int, lần lượt lưu trữ số hàng và số cột của bản đồ.
2. Thuộc tính private **map** để biểu diễn một mảng 2 chiều, mỗi phần tử của mảng là một MapElement. SV lựa chọn kiểu dữ liệu phù hợp cho **map** để thể hiện được tính đa hình (polymorphism). Gợi ý: cân nhắc giữa các kiểu dữ liệu: **MapElement**** và **MapElement***.
3. Phương thức khởi tạo Constructor (public) với khai báo như sau:

```
Map(int num_rows, int num_cols, int num_obstacle, Position *
    array_obstacle, int num_ground_obstacle, Position *
    array_ground_obstacle);
```

Trong đó:

- **num_rows**, **num_cols** lần lượt là số hàng và số cột
- **num_obstacle** là số lượng đối tượng Wall
- **array_obstacle** là mảng các Position, biểu diễn một mảng các vị trí của các Wall. Mảng này có **num_obstacle** phần tử. Thông tin về class **Position** sẽ được mô tả trong các mục sau.
- **num_ground_obstacle** là số lượng đối tượng FakeWall
- **array_ground_obstacle** là mảng các Position, biểu diễn một mảng các vị trí của các GROUND_OBSTACLE. Mảng này có **num_ground_obstacle** phần tử.

Constructor cần tạo ra một mảng 2 chiều mà mỗi phần tử là các đối tượng phù hợp. Nếu phần tử có vị trí nằm trong mảng **array_obstacle** thì phần tử đó là đối tượng OBSTACLE. Nếu phần tử có vị trí nằm trong mảng **array_ground_obstacle** thì phần tử đó là đối tượng GROUND_OBSTACLE. Các phần tử còn lại là đối tượng PATH.

4. Phương thức hủy Destructor (public) thu hồi các vùng nhớ được cấp phát động.
5. Một số phương thức khác được yêu cầu trong các mục sau.

3.3 Vị trí

Class **Position** biểu diễn vị trí trong chương trình. Position có thể được sử dụng để lưu trữ vị trí của thành phần bản đồ (như vị trí của OBSTACLE, GROUND_OBSTACLE) hoặc vị trí của các đối tượng di chuyển trên bản đồ.

Yêu cầu: Hiện thực class **Position** với các mô tả sau:

1. Hai thuộc tính private có tên **r_** và **c_** đều có kiểu int, lần lượt mô tả vị trí theo hàng và theo cột.
2. Phương thức khởi tạo Constructor (public) với hai tham số **r** và **c** lần lượt gán cho hai thuộc tính hàng và cột. Hai tham số này có giá trị mặc định là 0.

```
Position(int r=0, int c=0);
```

3. Phương thức khởi tạo Constructor (public) với 1 tham số **str_pos** biểu diễn một vị trí ở dạng chuỗi. Định dạng của **str_pos** là `"(<r>,<c>)"` với <r> và <c> lần lượt là giá trị cho hàng và cột.

```
Position(const string & str_pos); // Example: str_pos = "(1,15)"
```

4. 4 phương thức get, set cho 2 thuộc tính hàng và cột với khai báo như sau:

```
int getRow() const ;
int getCol() const ;
void setRow(int r) ;
void setCol(int c) ;
```

5. Phương thức **str** trả về một chuỗi biểu diễn thông tin vị trí. Định dạng trả về giống như định dạng của chuỗi **str_pos** trong Constructor. Ví dụ với $r = 1, c = 15$ thì **str** trả về giá trị **"(1,15)"**.

```
string str() const
```

6. Phương thức **isEqual** có hai tham số truyền vào **in_r** và **in_c** biểu diễn cho một vị trí. **isEqual** trả về giá trị **true** nếu vị trí truyền vào trùng với vị trí của đối tượng này. Ngược lại, **isEqual** trả về **false**.

```
bool isEqual(int in_r, int in_c) const
```

7. Một số phương thức khác có thể được yêu cầu trong các mục sau.

3.4 Đối tượng di chuyển

Abstract class **MovingObject** được sử dụng để biểu diễn các đối tượng di chuyển trong chương trình như FlyTeam, GroundTeam, DragonLord.

Yêu cầu: Hiện thực abstract class **MovingObject** với các mô tả sau:

1. Các thuộc tính protected:
   - **index**: kiểu *int*, vị trí của đối tượng di chuyển trong mảng các đối tượng di chuyển, mảng này sẽ được mô tả sau.
   - **pos**: kiểu *Position*, vị trí hiện tại của đối tượng di chuyển.
   - **map**: kiểu *Map **, bản đồ cho đối tượng này di chuyển trong đó.
   - **name**: kiểu *string*, tên của đối tượng di chuyển.
2. Phương thức khởi tạo Constructor (public) với các tham số **index**, **pos**, **map**, **name** có ý nghĩa giống với thuộc tính có cùng tên. Phương thức gán giá trị của tham số cho thuộc tính cùng tên. Riêng tham số **name** có giá trị mặc định là "".

```
MovingObject(int index, const Position pos, Map * map, const string &
    name="")
```

3. Phương thức hủy ảo (virtual destructor) với quyền truy cập public.

4. Phương thức ảo thuần tố (pure virtual method) **getNextPosition** trả về Position tiếp theo mà đối tượng này di chuyển đến. Mặt khác, trong trường hợp không có Position nào để đối tượng di chuyển đến, ta định nghĩa một giá trị để trả về cho phương thức này và lưu trong biến **npos** của class **Position**. Khi không có Position để di chuyển đến thì phương thức trả về **npos**.

```
virtual Position getNextPosition() = 0;
```

5. Method **getCurrentPosition** trả về Position hiện tại của đối tượng di chuyển.

```
Position getCurrentPosition() const;
```

6. Method **move** thực hiện 1 bước di chuyển của đối tượng.

```
virtual void move() = 0;
```

7. Pure virtual method **str** trả về chuỗi biểu diễn thông tin của đối tượng.

```
virtual string str() const = 0;
```

Yêu cầu: Định nghĩa biến **npos** (not position) trong class **Position** biểu diễn rằng không có vị trí nào để đối tượng di chuyển đến. Biến **npos** có $r = -1$ và $c = -1$. Khai báo của biến như sau:

```
static const Position npos;
```

Yêu cầu: Trong class **Map**, định nghĩa phương thức **isValid** kiểm tra vị trí **pos** có phải là một vị trí hợp lệ cho đối tượng **mv_obj** di chuyển đến. Một vị trí hợp lệ cho việc di chuyển phải phụ thuộc vào đối tượng di chuyển là gì và thành phần bản đồ. Ví dụ, FlyTeam có thể di chuyển trên GROUND_OBSTACLE nhưng GroundTeam thì cần thỏa yêu cầu về DMG. SV cần tìm đọc mô tả trong BTL này để hiện thực phương thức cho đúng.

```
bool isValid(const Position & pos, MovingObject * mv_obj) const;
```

3.5 Warrior

Class **Warrior** biểu diễn cho nhóm Chiến binh trong chương trình. Chúng ta có hai loại nhóm chiến binh trong cuộc chiến này: **FlyTeam** và **GroundTeam**, mô tả chi tiết sẽ nằm ở phần sau.

Class **Warrior** nhận class **MovingObject** là lớp tổ tiên (ancestor class). Do đó, class Warrior phải hiện thực các pure virtual method của class MovingObject.

Yêu cầu: SV hiện thực class Warrior với yêu cầu bên dưới. **SV có thể tự đề xuất thêm các thuộc tính, các phương thức hoặc các class khác để hỗ trợ cho việc hiện thực các class trong BTL này.**

1. Constructor (public) được khai báo như bên dưới. Constructor này có thêm một số tham số khác bên cạnh tham số đã có trong **MovingObject**:

```
Warrior(int index, const Position & pos, Map * map, const string & name,
    int init_hp, int init_damage);
```

   - **init_hp**: HP ban đầu của Warrior. HP nằm trong khoảng $[0, 500]$. Nếu HP vượt quá 500 thì được cài đặt về 500, nếu HP bằng 0 thì coi như Warrior đã hết thể lực và không thể di chuyển tiếp trong mê cung.
     - Nếu HP của cả FlyTeam và GroundTeam đều bằng 0, thì các chiến binh rồng bị thua trong cuộc chiến với Chúa tể Rồng.
     - Nếu HP của FlyTeam = 0 và GroundTeam khác 0, thì các chiến binh rồng cũng bị thua trong cuộc chiến với Chúa tể Rồng.
     - Nếu HP của FlyTeam khác 0 và GroundTeam bằng 0, thì FlyTeam vẫn có thể tiếp tục cuộc chiến.
   - **init_damage**: Damage ban đầu của Warrior. Damage nằm trong khoảng $[0, 900]$. Nếu Damage vượt quá 900 thì cài đặt về 900, nếu Damage bằng 0 thì Warrior cũng sẽ không di chuyển tiếp trong mê cung.
   - Tham số **name** của Constructor MovingObject được truyền vào giá trị tên của nhóm chiến binh.

2. Warrior có thêm các thuộc tính **hp**, **damage**, và **bag** thuộc class **BaseBag**.
3. 4 phương thức get, set cho 2 thuộc tính hp và damage với khai báo như sau:

```
int getHP() const ;
int getDamage() const ;
void setHP(int hp) ;
void setDamage(int damage) ;
```

4. Phương thức **getBag** class **BaseBag** được mô tả trong các mục sau.

```
BaseBag* getBag() const;
```

3.6 FlyTeam

Class **FlyTeam** biểu diễn cho nhóm chiến binh trên không trong chương trình. Class **FlyTeam** nhận class **Warrior** làm lớp tổ tiên (ancestor class).

Yêu cầu: SV hiện thực class FlyTeam với yêu cầu bên dưới. **SV có thể tự đề xuất thêm các thuộc tính, các phương thức hoặc các class khác để hỗ trợ cho việc hiện thực các class trong BTL này.**

1. Constructor (public) được khai báo như bên dưới. Constructor này có thêm một số tham số khác bên cạnh tham số đã có trong **Warrior**:

```
FlyTeam(int index, const & string & moving_rule, const Position & pos,
    Map * map, int init_hp, int init_damage);
```

   - **moving_rule**: mô tả cách thức mà FlyTeam di chuyển. Đây là một chuỗi mà các ký tự chỉ có thể là một trong 4 giá trị: 'L' (Left - đi sang trái), 'R' (Right - đi sang phải), 'U' (Up - đi lên trên), 'D' (Down - đi xuống dưới). Ví dụ về **moving_rule** là "LU".

2. FlyTeam có thêm các thuộc tính private **(string) moving_rule** và **(int) moving_index**.
   - **moving_rule**: như mô tả trên
   - **moving_index**: lưu chỉ số của ký tự hiện tại trong moving_rule đang được sử dụng để xác định hướng di chuyển tiếp theo.
3. Phương thức **getNextPosition** (public) trả về vị trí di chuyển tiếp theo của FlyTeam. FlyTeam di chuyển dựa theo **moving_rule**. Mỗi lần gọi phương thức, một ký tự tiếp theo được sử dụng để làm hướng di chuyển. Lần đầu tiên gọi phương thức thì ký tự đầu tiên sẽ được sử dụng. Khi ký tự cuối cùng được sử dụng thì sẽ quay lại bắt đầu quá trình này từ ký tự đầu tiên. Ví dụ với **moving_rule** = "LR" thì thứ tự các ký tự được sử dụng là: 'L', 'R', 'L', 'R', 'L', 'R',... Nếu Position được trả ra không phải là một vị trí hợp lệ cho đối tượng này di chuyển thì trả về **npos** thuộc class **Position**.
4. Phương thức **move** (public) thực hiện một bước di chuyển của FlyTeam. Trong **move** có lời gọi đến **getNextPosition**, nếu nhận được giá trị trả về khác **npos** là bước đi hợp lệ thì FlyTeam sẽ di chuyển đến đó. Nếu không phải bước đi hợp lệ thì FlyTeam sẽ đứng im.
5. Phương thức **attack** (public) thực hiện bước tấn công và kết liễu của FlyTeam khi đã đối đầu với Chúa tể Rồng. Chỉ có FlyTeam mới có thể kết liễu Chúa tể Rồng.

```
bool attack(DragonLord *dragon);
```

6. Phương thức **str** trả về chuỗi có định dạng như sau:

```
FlyTeam[index=<index>;pos=<pos>;moving_rule=<moving_rule>]
```

   Trong đó:

   - <index> là giá trị của thuộc tính *index*.
   - <pos> là chuỗi biểu diễn của thuộc tính *pos*.
   - <moving_rule> là giá trị của thuộc tính *moving_rule*.

3.7 GroundTeam

Class **GroundTeam** biểu diễn cho nhóm chiến binh mặt đất trong chương trình. Class **GroundTeam** nhận class **Warrior** làm lớp tổ tiên (ancestor class).

Yêu cầu: SV hiện thực class GroundTeam với yêu cầu bên dưới. **SV có thể tự đề xuất thêm các thuộc tính, các phương thức hoặc các class khác để hỗ trợ cho việc hiện thực các class trong BTL này.**

1. Constructor (public) được khai báo như bên dưới. Constructor này có thêm một số tham số khác bên cạnh tham số đã có trong **Warrior**: tương tự với FlyTeam

```
GroundTeam(int index, const & string & moving_rule, const Position & pos
    , Map * map, int init_hp, int init_damage);
```

2. GroundTeam có thêm các thuộc tính private **(string) moving_rule**, **(int) moving_index**, và **(int) trap_turns**.
   - **trap_turns**: GroundTeam có nhiệm vụ chính là khống chế thay vì kết liễu Chúa tể Rồng. Khi chạm trán, họ sẽ tìm cách vây bắt và bẫy Rồng trong vòng `N` lượt, tạo cơ hội cho FlyTeam tiếp cận. Biến `trap_turns` đại diện cho số lượt GroundTeam có thể cầm chân Rồng Chúa tể.
3. Các phương thức **getNextPosition**, **move**, và **str** tương tự với FlyTeam.
4. Phương thức **trap** (public) thực hiện bước khống chế khi chạm trán với Chúa tể Rồng.

```
bool trap(DragonLord *dragon);
```

5. 2 phương thức get, set cho thuộc tính `trap_turns` với khai báo như sau:

```
int getTrapTurns() const;
void setTrapTurns(int turns);
```

3.8 Chúa tể Rồng

Class **DragonLord** đại diện cho Chúa tể Rồng trong chương trình. Lớp này kế thừa từ lớp tổ tiên **MovingObject**.

Chúa tể Rồng sở hữu khả năng biến cảm – cho phép cảm nhận sự "biến động" trong không gian do sự hiện diện của các cá thể rồng khác gây ra. Nhờ đó, nó có thể định vị được các nhóm chiến binh, đặc biệt là các chiến binh cưỡi rồng trên bầu trời.

Do đặc tính này, cách di chuyển của Chúa tể Rồng khác biệt so với các nhóm chiến binh: Vị trí tiếp theo mà Chúa tể Rồng sẽ di chuyển đến là một vị trí đặc biệt, được xác định dựa

trên tọa độ của hai nhóm chiến binh bầu trời: **FlyTeam1** và **FlyTeam2**.

Gọi (x1,y1) là tọa độ hiện tại của **FlyTeam1**, và (x2,y2) là tọa độ hiện tại của **FlyTeam2**.

Lưu ý: Trong mỗi trận chiến, luôn có hai nhóm chiến binh bầu trời và một nhóm chiến binh mặt đất tham gia.

- Toạ độ x của DragonLord = |x1 - x2|
- Toạ độ y của DragonLord = |y1 - y2|
- Sau khi có được toạ độ (x,y) của DragonLord, chương trình cần kiểm tra Khoảng cách Manhattan giữa (x,y) lần lượt với (x1,y1) và (x2,y2). Khoảng cách Manhattan giữa 2 điểm P1 có tọa độ $(x1, y1)$ và P2 có tọa độ $(x2, y2)$ là:

$$|x1 - x2| + |y1 - y2|$$

- Dựa vào hai khoảng cách này:
  - Nếu cả hai khoảng cách đều lớn hơn 5, giữ nguyên (x, y) và sử dụng nó làm vị trí di chuyển tiếp theo của Chúa tể Rồng.
  - Nếu một trong hai (hoặc cả hai) khoảng cách nhỏ hơn hoặc bằng 5, đảo tọa độ (x, y) thành (y, x). Vị trí mới (y, x) sẽ được dùng làm bước di chuyển tiếp theo.

Yêu cầu: Hiện thực class **DragonLord** tương tự như class **Warrior** với các sự thay đổi như sau:

1. Constructor (public) được khai báo như bên dưới. Một số tham số có ý nghĩa tương tự như class **Warrior**. Một số điểm khác nhau là:

```
DragonLord(int index, const Position & init_pos, Map * map, FlyTeam *
    flyteam1, FlyTeam *flyteam2, GroundTeam *groundteam);
```

   - **flyteam1**, **flyteam2**, **groundteam** lần lượt là con trỏ đến đối tượng FlyTeam1, FlyTeam2, và GroundTeam. Thông qua 2 con trỏ, ta có thể lấy được vị trí hiện tại của hai nhóm chiến binh này.
   - Tham số **name** của Constructor MovingObject được truyền vào giá trị "DragonLord".

2. Phương thức **str** trả về chuỗi có định dạng như sau:

```
DragonLord[index=<index>;pos=<pos>]
```

3. **SV có thể tự đề xuất thêm các thuộc tính, các phương thức hoặc các class khác để hỗ trợ cho việc hiện thực các class trong BTL này.**

3.9 Mảng các đối tượng di chuyển

Class **ArrayMovingObject** biểu diễn một mảng các đối tượng di chuyển. Khi chương trình chạy, mảng này được duyệt từ đầu đến cuối và gọi phương thức **move** của mỗi phần tử để mỗi đối tượng thực hiện 1 bước đi.

Yêu cầu: Hiện thực class **ArrayMovingObject** với các yêu cầu sau:

1. Các thuộc tính private:
   - **arr_mv_objs**: mảng các đối tượng di chuyển (MovingObject). Mỗi phần tử trong mảng cần thể hiện được tính đa hình. SV tự đề xuất kiểu dữ liệu cho biến.
   - **count**: kiểu *int*, số lượng phần tử hiện tại của mảng.
   - **capacity**: kiểu *int*, số lượng phần tử tối đa của mảng.
2. Các phương thức public:
   - Constructor **ArrayMovingObject** nhận vào một tham số để khởi tạo cho thuộc tính **capacity**. Đồng thời phương thức cần thực hiện cấp phát cho phù hợp.
   - Destructor của class cần thu hồi vùng nhớ được cấp phát động.
   - Phương thức **isFull** trả về giá trị **true** nếu mảng đã đầy, ngược lại thì trả về **false**. Mảng đã đầy nếu số lượng phần tử hiện tại bằng số lượng phần tử tối đa của mảng.

     ```
     bool isFull() const;
     ```

   - Phương thức **add** thêm một đối tượng di chuyển mới vào cuối mảng nếu mảng chưa đầy, sau đó trả về **true**. Ngược lại, phương thức sẽ không thêm đối tượng mới vào và trả về **false**.

     ```
     bool add(MovingObject * mv_obj)
     ```

   - Phương thức **str** trả về chuỗi biểu diễn thông tin cho **ArrayMovingObject**.

     ```
     string str() const
     ```

     Định dạng của chuỗi trả về là:

     ```
     ArrayMovingObject[count=<count>;capacity=<capacity>;<MovingObject1>;...]
     ```

     Trong đó:

- count, capacity: Lần lượt là số lượng và số phần tử tối đa của đối tượng **ArrayMovingObject**
- MovingObject1,...: Lần lượt là các MovingObject có trong mãng. Mỗi MovingObject được in theo định dạng tương ứng của loại đối tượng đó.

Ví dụ về chuỗi trả về của phương thức **str** của **ArrayMovingObject**:

```
ArrayMovingObject[count=3;capacity=10;DragonLord[index=0;pos=(8,9)];
    ↪ FlyTeam1[index=1;pos(1,4);moving_rule=RUU];FlyTeam2[index=2;
    ↪ pos=(2,1);moving_rule=LU];GroundTeam[index=3;pos(8,8);
    ↪ moving_rule=UUL]]
```

3.10 Cấu hình cho chương trình

Một tập tin được sử dụng để chứa cấu hình cho chương trình. Tập tin gồm các dòng, mỗi dòng có thể là một trong các định dạng như bên dưới. Lưu ý rằng thứ tự các dòng có thể thay đổi.

1. MAP_NUM_ROWS=<nr>
2. MAP_NUM_COLS=<nc>
3. MAX_NUM_MOVING_OBJECTS=<mnmo>
4. ARRAY_OBSTACLE=<ao>
5. ARRAY_GROUND_OBSTACLE=<ago>
6. FLYTEAM1_MOVING_RULE=<f1mr>
7. FLYTEAM1_INIT_POS=<f1ip>
8. FLYTEAM1_INIT_HP=<f1ih>
9. FLYTEAM1_INIT_DAMAGE=<f1id>
10. FLYTEAM2_MOVING_RULE=<f2mr>
11. FLYTEAM2_INIT_POS=<f2ip>
12. FLYTEAM2_INIT_HP=<f2ih>
13. FLYTEAM2_INIT_DAMAGE=<f2id>
14. GROUNDTEAM_MOVING_RULE=<gmr>
15. GROUNDTEAM_INIT_POS=<gip>
16. GROUNDTEAM_INIT_HP=<gih>
17. GROUNDTEAM_INIT_DAMAGE=<gid>
18. DRAGONLORD_INIT_POS=<dip>
19. NUM_STEPS=<ns>

Trong đó:

1. <nr> là số hàng của bản đồ, ví dụ:
   MAP_NUM_ROWS=10
2. <nc> là số cột của bản đồ, ví dụ:
   MAP_NUM_COLS=10
3. <mnmo> là số lượng phần tử tối đa của mảng các đối tượng di chuyển, ví dụ:
   MAX_NUM_MOVING_OBJECTS=10
4. <ao> là danh sách các vị trí của các OBSTACLE. Danh sách được đặt trong một cặp dấu "[]", gồm các vị trí được phân cách bởi 1 dấu ';'. Mỗi vị trí là một cặp mở đóng ngoặc "()", bên trong là số hàng và số cột phân cách bởi 1 dấu phẩy. Ví dụ:
   ARRAY_OBSTACLE=[(1,2);(2,3);(3,4)]
5. <ago> là danh sách các vị trí của các GROUND_OBSTACLE. Danh sách này có định dạng giống <ao>. Ví dụ:
   ARRAY_GROUND_OBSTACLE=[(4,5)]
6. <f1mr> là chuỗi biểu diễn **moving_rule** của FlyTeam1. Ví du:
   FLYTEAM1_MOVING_RULE=RUU
7. <f1ip> là vị trí ban đầu của FlyTeam1. Ví du:
   FLYTEAM1_INIT_POS=(1,3)
8. <f2mr> là chuỗi biểu diễn **moving_rule** của FlyTeam2. Ví du:
   FLYTEAM2_MOVING_RULE=LLD
9. <f2ip> là vị trí ban đầu của FlyTeam2. Ví du:
   FLYTEAM2_INIT_POS=(4,8)
10. <gmr> là chuỗi biểu diễn **moving_rule** của GroundTeam. Ví du:
    GROUNDTEAM_MOVING_RULE=LU
11. <gip> là vị trí ban đầu của GroundTeam. Ví du:
    GROUNDTEAM_INIT_POS=(2,1)
12. <dip> là vị trí ban đầu của Chúa tể Rồng. Ví du:
    DRAGONLORD_INIT_POS=(7,9)
13. <ns> là số vòng lặp mà chương trình sẽ thực hiện. Trong mỗi vòng lặp, chương trình sẽ duyệt qua toàn bộ các MovingObject và để các đối tượng này thực hiện 1 bước di chuyển. Ví dụ:
    NUM_STEPS=100
14. <f1ih>, <f1id> lần lượt là HP và DMG ban đầu của FlyTeam1.
15. <f2ih>, <f2id> lần lượt là HP và DMG ban đầu của FlyTeam2.
16. <gih>, <gid> lần lượt là HP và DMG ban đầu của GroundTeam.

Yêu cầu: Hiện thực class **Configuration** biểu diễn cho một cấu hình của chương trình

thông qua việc đọc tập tin cấu hình. Class **configuration** có các mô tả sau:

1. Các thuộc tính private:
   - **map_num_rows**, **map_num_cols** lần lượt là số hàng và số cột của bản đồ.
   - **max_num_moving_objects**: kiểu *int*, tương ứng với `<mnmo>`.
   - **num_obstacles**: kiểu *int*, số lượng đối tượng OBSTACLE.
   - **arr_obstacles**: kiểu *Position**, tương ứng với `<ao>`.
   - **num_ground_obstacles**: kiểu *int*, số lượng đối tượng GROUND_OBSTACLE.
   - **arr_ground_obstacles**: kiểu *Position**, tương ứng với `<ago>`.
   - **flyteam1_moving_rule**: kiểu *string*, tương ứng với `<f1mr>`.
   - **flyteam1_init_pos**: kiểu *Position*, tương ứng với `<f1ip>`.
   - **flyteam1_init_hp**: kiểu *int*, tương ứng với `<f1ih>`.
   - **flyteam1_init_damage**: kiểu *int*, tương ứng với `<f1id>`.
   - **flyteam2_moving_rule**: kiểu *string*, tương ứng với `<f2mr>`.
   - **flyteam2_init_pos**: kiểu *Position*, tương ứng với `<f2ip>`.
   - **flyteam2_init_hp**: kiểu *int*, tương ứng với `<f2ih>`.
   - **flyteam2_init_damage**: kiểu *int*, tương ứng với `<f2id>`.
   - **groundteam_moving_rule**: kiểu *string*, tương ứng với `<gmr>`.
   - **groundteam_init_pos**: kiểu *Position*, tương ứng với `<gip>`.
   - **groundteam_init_hp**: kiểu *int*, tương ứng với `<gih>`.
   - **groundteam_init_damage**: kiểu *int*, tương ứng với `<gid>`.
   - **dragonlord_init_pos**: kiểu *Position*, tương ứng với `<dip>`.
   - **num_steps**: kiểu *int*, tương ứng với `<ns>`.
2. Constructor **Configuration** được khai báo như bên dưới. Constructor nhận vào **filepath** là chuỗi chứa đường dẫn đến tập tin cấu hình. Constructor khởi tạo các thuộc tính cho phù hợp với các mô tả trên.

```
Configuration(const string & filepath);
```

3. Destructor cần thu hồi các vùng nhớ được cấp phát động.
4. Phương thức **str** trả về chuỗi biểu diễn cho Configuration.

```
string str() const;
```

Định dạng của chuỗi này là:

```
Configuration[
<attribute_name1>=<attribute_value1>...
]
```

Ví dụ về một chuỗi trả về cho phương thức **str** của **Configuration**:

```
Configuration[
MAP_NUM_ROWS=10
MAP_NUM_COLS=10
MAX_NUM_MOVING_OBJECTS=10
NUM_OBSTACLE=3
ARRAY_OBSTACLE=[(1,2);(2,3);(3,4)]
NUM_GROUND_OBSTACLE=1
ARRAY_GROUND_OBSTACLE=[(4,5)]
FLYTEAM1_MOVING_RULE=RUU
FLYTEAM1_INIT_POS=(1,3)
FLYTEAM1_INIT_HP=250
FLYTEAM1_INIT_DMG=500
FLYTEAM2_MOVING_RULE=LLD
FLYTEAM2_INIT_POS=(1,3)
FLYTEAM2_INIT_HP=350
FLYTEAM2_INIT_DMG=400
GROUNDTEAM_MOVING_RULE=LU
GROUNDTEAM_INIT_POS=(2,1)
GROUNDTEAM_INIT_HP=300
GROUNDTEAM_INIT_DMG=350
DRAGONLORD_INIT_POS=(7,9)
NUM_STEPS=100
]
```

Trong đó mỗi thuộc tính và giá trị thuộc tính tương ứng sẽ được in theo thứ tự như đã liệt kê tại phần "Các thuộc tính private" của class này.

3.11 SmartDragon

Trong quá trình di chuyển của Chúa tể Rồng, cứ sau mỗi **5 bước** mà Rồng chúa di chuyển, một Rồng Trí tuệ sẽ được tạo ra. Lưu ý rằng khi phương thức **getNextPosition** của **DragonLord** không trả về được một vị trí di chuyển hợp lệ, phương thức **move** không thực hiện bước di chuyển, ta không tính đó là 1 bước di chuyển. Mỗi loại smartdragon sẽ là một nhận **MovingObject** làm ancestor class. Mỗi loại smartdragon đều có thể di chuyển trên **PATH**

hoặc **GROUND_OBSTACLE**, nhưng không thể di chuyển trên **OBSTACLE**. Sau khi được tạo ra, smartdragon sẽ được thêm vào mảng các đối tượng di chuyển (**ArrayMovingObject**) thông qua phương thức **add** của lớp **ArrayMovingObject**. Trong trường hợp số lượng của mảng này đã đầy, thì smartdragon không được tạo ra.

Sau mỗi **5 bước** đi của Rồng chúa, một smartdragon sẽ được tạo ra tại **vị trí trước đó mà Rồng chúa đang đứng**. Các loại smartdragon và điều kiện tạo ra tương ứng:

- Nếu là smartdragon đầu tiên được tạo ra trên bản đồ, đó sẽ là loại smartdragon **SD1**. Nếu không, ta xét khoảng cách từ SmartDragon đến FlyTeam1, FlyTeam2 và GroundTeam:
- Nếu khoảng cách đến FlyTeam1 gần hơn: Tạo ra loại smartdragon **SD1**
- Nếu khoảng cách đến FlyTeam2 gần hơn: Tạo ra loại smartdragon **SD2**
- Khoảng cách đến GroundTeam gần hơn: Tạo ra loại smartdragon **SD3**
- Khoảng cách đến các nhóm có sự tương đồng, ưu tiên tạo ra loại smartdragon hiện có số lượng trên bản đồ ít nhất (ít nhất tìm thấy đầu tiên).

Các loại smartdragon được định nghĩa thành enum **DragonType** như sau:

```
enum DragonType { SD1, SD2, SD3};
```

Yêu cầu: SV hiện thực các class cần thiết liên quan đến các đối tượng là smartdragon theo các yêu cầu như các nhân vật khác, có một số thay đổi như sau:

1. Các loại smartdragon có thuộc tính giống nhau bao gồm:
   - Thuộc tính **smartdragon_type** mang giá trị là loại smartdragon có kiểu **DragonType**
   - Thuộc tính **damage** có kiểu **int** mang giá trị là sát thương có thể gây ra của smartdragon.
   - Thuộc tính **item** có kiểu **BaseItem *** mang giá trị là loại thuộc tính nếu chiến thắng nhân vật sẽ nhận được. Chi tiết về các vật phẩm sẽ được trình bày tại phần sau.
   - Thuộc tính **target** có kiểu **MovingObject *** là con trỏ trỏ đến đối tượng smartdragon nhắm đến. Cụ thể:
     - **SD1**: Thuộc tính **flyteam1** là con trỏ trỏ đến FlyTeam1. Từ đó có thể lấy được vị trí hiện tại của FlyTeam1.
     - **SD2**: Thuộc tính **flyteam2** là con trỏ trỏ đến FlyTeam2. Từ đó có thể lấy được vị trí hiện tại của FlyTeam2.
     - **SD3**: Thuộc tính **groundteam** là con trỏ trỏ đến GroundTeam. Từ đó có thể lấy được vị trí hiện tại của GroundTeam.

2. Constructor (public) tương ứng nhận vào các tham số liên quan đến **MovingObject** và các tham số riêng cho từng loại smartdragon. Cụ thể constructor của từng loại được gọi như sau:

```
SD1: SmartDragon(int index, const Position & init_pos, Map * map,
DragonType type, MovingObject * flyteam1, int damage);

SD2: SmartDragon(int index, const Position & init_pos, Map * map,
DragonType type, MovingObject * flyteam2, int damage);

SD3: SmartDragon(int index, const Position & init_pos, Map * map,
DragonType type, MovingObject * groundteam, int damage);

```

3. Phương thức **getNextPosition** (public): Đối với mỗi loại smartdragon, quy tắc di chuyển cũng khác nhau, cụ thể:
   - **SD1:** Di chuyển tiếp cận gần hơn đến FlyTeam1 1 ô dựa theo khoảng cách Manhattan. Ưu tiên theo chiều kim đồng hồ: UP, RIGHT, DOWN, LEFT.
   - **SD2:** Di chuyển tiếp cận gần hơn đến FlyTeam2 1 ô dựa theo khoảng cách Manhattan. Ưu tiên theo chiều kim đồng hồ: UP, RIGHT, DOWN, LEFT.
   - **SD3:** Không di chuyển, đứng tại chỗ.
4. Phương thức **move** (public): Nếu vị trí tiếp theo không phải là **npos** thì di chuyển đến vị trí này.
5. Phương thức **str** (public): Định dạng chuỗi trả về như sau:

   ```
   smartdragon[pos=<pos>;type=<dragon_type>;tg=<target>]
   ```

   <**pos**> in ra vị trí hiện tại của smartdragon
   <**dragon_type**> in ra giá trị có thể là SD1, SD2, hay SD3
   <**target**> in ra đối tượng smartdragon đang tiếp cận (nhắm đến).

3.12 Vật phẩm

Class **BaseItem** là một abstract class biểu diễn thông tin cho 1 vật phẩm. Class có hai public pure virtual method là:

- **canUse**

```
virtual bool canUse(Warrior* w) = 0;
```

Method trả về `true` nếu Warrior có thể dùng được vật phẩm này, ngược lại trả về `false`.

- **use**

```
virtual void use(Warrior* w) = 0;
```

Phương thức này thực hiện tác động lên đối tượng Warrior, nhằm điều chỉnh các thuộc tính của họ sao cho phù hợp với hiệu ứng của vật phẩm. Trong một số trường hợp, vật phẩm sẽ đi kèm điều kiện sử dụng. Nếu điều kiện đó chưa được thoả mãn tại thời điểm nhận vật phẩm, vật phẩm sẽ tạm thời được lưu trữ trong túi đồ của nhóm chiến binh. Khi chiến binh đạt đủ điều kiện cần thiết, họ cần kiểm tra lại túi đồ và sử dụng vật phẩm tương ứng nếu phù hợp.

Tác dụng của từng loại vật phẩm được mô tả như sau:

Vật phẩm	**Tác dụng**	**Điều kiện sử dụng**
DragonScale	Chứa sức mạnh ma thuật của Rồng Trí tuệ giúp Warrior tăng cường sức mạnh cho họ một cách nhanh chóng nên dễ dàng hồi phục **DMG** tăng thêm 25% cho chỉ số hiện tại khi sử dụng.	**exp** ≤ 400
HealingHerb	Thảo dược phục hồi khi được sử dụng sẽ giúp nhóm chiến binh hồi phục **hp** tăng thêm 20% khi sử dụng.	**hp** ≤ 100
TrapEnhancer	Dụng cụ nâng cấp bẫy của nhóm Chiến binh mặt đất. Cộng trực tiếp thêm 1 lượt cho lần giữ chân Chúa tể Rồng.	Không có

Các vật phẩm này được chứa trong các smartdragon. Điều kiện như sau:

- **SD1**: nắm giữ các ma thuật DragonScale.
- **SD2**: nắm giữ các thảo dược HealingHerb.
- **SD3**: nắm giữ các công cụ TrapEnhancer. Tuy nhiên, TrapEnhancer chỉ rớt ra nếu người tiêu diệt SD3 là GroundTeam. Nếu người tiêu diệt là FlyTeam1 và FlyTeam2, lần lượt rớt ra HealingHerb và DragonScale.

Yêu cầu: Sinh viên hiện thực các class DragonScale, HealingHerb, và TranEnhancer được mô tả như trên.

3.13 Túi đồ

Mỗi nhóm chiến binh rồng được trang bị một túi đồ để chứa vật phẩm thu thập được trên đường di chuyển. Túi đồ được hiện thực dưới hình thức một danh sách liên kết đơn. Các hành động thực hiện với túi bao gồm:

- Thêm một món đồ vào túi đồ (phương thức **insert)**): Thêm món đồ vào đầu danh sách
- Sử dụng một món đồ bất kì (phương thức **get**): Tìm món đồ trong túi có thể sử dụng được và gần đầu túi nhất. Món đồ này sẽ được đảo với món đồ đầu tiên trong danh sách và sau đó được xóa ra khỏi danh sách.
- Sử dụng một món đồ cụ thể (phương thức **get(ItemType)**: Nếu trong túi đồ có món đồ có kiểu cần sử dụng, nhân vật có thể sử dụng món đồ này bằng cách đảo vị trí nó với món đồ đầu tiên (nếu đó không phải là món đồ đầu tiên) và xoá nó khỏi danh sách. Nếu có nhiều món đồ thì món đồ ở gần đầu túi nhất sẽ được thực hiện như trên.

Mỗi nhóm sẽ có một số lượng món đồ tối đa có thể chứa trong túi. Túi của FlyTeam có thể chứa tối đa **5 món đồ**, trong khi GroundTeam có thể chứa tối đa **7 món đồ**.

Yêu cầu: Sinh viên cần hiện thực các thông tin biểu diễn túi đồ bao gồm:

Class **BaseBag** là một class biểu diễn túi đồ. Class có một thuộc tính **warrior** kiểu **Warrior*** biểu diễn nhân vật đang sở hữu túi. Ngoài ra class còn có các public method cơ bản như mô tả, bao gồm:

```
1    virtual bool insert (BaseItem* item); //return true if insert
    successfully
2    virtual BaseItem* get(); //return the item as described above, if not
    found, return NULL
3    virtual BaseItem* get(ItemType itemType); //return the item as described
     above, if not found, return NULL
4    virtual string str() const;
```

Đối với method str(), định dạng chuỗi trả về là:

```
Bag[count=<c>;<list_items>]
```

Trong đó:

- <c>: là số lượng item hiện tại mà túi đồ đang có.
- <list_items>: là một chuỗi biểu diễn các vật phẩm từ đầu đến cuối của danh sách liên kết, mỗi vật phẩm được biểu diễn bằng tên kiểu của vật phẩm, các vật phẩm được ngăn

cách nhau bởi 1 dấu phẩy. Tên kiểu của các vật phẩm như tên class đã mô tả ở trên.

Yêu cầu: Sinh viên tự đề xuất một class kế thừa từ class BaseBag và để biểu diễn các loại túi đồ khác nhau cho các nhóm chiến binh. Tên class sẽ là **TeamBag**. Constructor của class này chỉ có 1 tham số truyền vào là **Warrior * warrior** (biểu diễn cho nhóm chiến binh).

3.14 DragonWarriorsProgram

Nếu trong quá trình di chuyển các nhóm chiến binh đụng độ với các loại smartdragon hay với Chúa tể Rồng sẽ có các vấn đề xảy ra. Các trường hợp bao gồm:

1. Nếu các chiến binh gặp Rồng Trí tuệ:
   - SD1: Nếu nhóm chiến binh đụng độ với Rồng Trí tuệ SD1, họ sẽ chiến đấu với nó. Nếu damage của chiến binh > damage của smartdragon, thì chiến binh thắng và họ nhận được vật phẩm mà SD1 nắm giữ. Nếu thua, chiến binh bị trừ 100HP.
   - SD2: Tương tự SD1.
   - SD3: Tương tự SD1. Tuy nhiên, nếu chiến binh thắng là GroundTeam, GroundTeam nhận được TrapEnhancer.
2. Nếu các chiến binh gặp Chúa tể Rồng:
   - FlyTeam: trực tiếp chiến đấu và kết liễu được Chúa tể Rồng. Nhóm chiến binh chiến thắng và kết thúc trò chơi.
   - GroundTeam: không thể chiến đấu với Rồng Chúa. Họ tung bẫy đã chuẩn bị, khống chế nó trong N lượt mà dụng cụ có thể chịu đựng. Nếu trong N lượt mà FlyTeam tiếp cận được Rồng Chúa, kết quả như trên. Nếu không, Chúa tể Rồng dịch chuyển GroundTeam đến vị trí đảo ngược (x,y) -> (y,x), và trừ họ 200 HP.

   Yêu cầu: SV tự đề xuất và hiện thực class **DragonWarriorsProgram** theo các yêu cầu sau:

   (a) Class có các thuộc tính: config (kiểu **Configuration***), flyteam1 (kiểu **FlyTeam***), flyteam2 (kiểu **FlyTeam***), groundteam (kiểu **GroundTeam***), dragonlord (kiểu **DragonLord***), map (kiểu **Map***), arr_mv_objs (kiểu **ArrayMovingObject**.

   (b) Constructor nhận 1 tham số là đường dẫn đến tập tin cấu hình cho chương trình. Constructor cần khởi tạo các thông tin cần thiết cho class. Riêng **arr_mv_objs** sau khi khởi tạo thì lần lượt thêm vào *dragonlord*, *flyteam1*, *flyteam2*, *groundteam* bằng phương thức **add**.

```
DragonWarriorsProgram(const string & config_file_path)
```

(c) Phương thức **isStop** trả về **true** nếu chương trình dừng lại, ngược lại trả về **false**. Chương trình dừng khi một trong các điều kiện sau xảy ra: **hp** của hai FlyTeam bằng 1; **hp** của tất cả nhóm chiến binh bằng 1; FlyTeam tiêu diệt được Rồng chúa.

```
bool isStop() const;
```

(d) Phương thức **printResult** và **printStep** in ra thông tin của chương trình. Các phương thức này được hiện thực sẵn, SV không thay đổi các phương thức này.

(e) Phương thức **run** có 1 tham số là **verbose**. Nếu **verbose** bằng **true** thì cần in ra thông tin của mỗi MovingObject trong ArrayMovingObject sau khi thực hiện một bước di chuyển và các cập nhật sau đó nếu có (ví dụ như FlyTeam1 gặp một smartdragon và thực hiện thử thách với smartdragon). SV tham khảo thêm initial code về vị trí của hàm **printStep** trong **run**. Phương thức **run** chạy tối đa **num_steps** (lấy từ tập tin cấu hình). Sau mỗi **step** nếu chương trình thỏa điều kiện dừng nêu trong phương thức **isStop** thì chương trình sẽ dừng lại. Mỗi **step**, chương trình sẽ lần lượt chạy từ đầu đến cuối một ArrayMovingObject và gọi **move**. Sau mỗi lần thực hiện di chuyển của một phần tử, tiến hành các thao tác sau theo thứ tự: thực hiện hành động nếu có các đối tượng gặp nhau, kiểm tra điều kiện dừng (stop), kiểm tra điều kiện tạo smartdragon và tạo smartdragon nếu cần.

```
void run(bool verbose)
```

3.15 TestDragonWar

TestDragonWar là một class dùng để kiểm tra các thuộc tính của các class trong BTL này. SV phải khai báo **TestDragonWar** là friend class khi định nghĩa tất cả các class trên.

4 Yêu cầu

Để hoàn thành bài tập lớn này, sinh viên phải:

1. Đọc toàn bộ tập tin mô tả này.
2. Tải xuống tập tin initial.zip và giải nén nó. Sau khi giải nén, sinh viên sẽ nhận được các tập tin: main.cpp, main.h, dragons.h, dragons.cpp, và các file dữ liệu đọc mẫu. Sinh viên phải nộp 2 tập tin là dragons.h và dragons.cpp nên không được sửa đổi tập tin main.h khi chạy thử chương trình.
3. Sinh viên sử dụng câu lệnh sau để biên dịch:

g++ -o main main.cpp dragons.cpp -I . -std=c++11

Sinh viên sử dụng câu lệnh sau để chạy chương trình:

./main

Các câu lệnh trên được dùng trong command prompt/terminal để biên dịch và chạy chương trình. Nếu sinh viên dùng IDE để chạy chương trình, sinh viên cần chú ý: thêm đầy đủ các tập tin vào project/workspace của IDE; thay đổi lệnh biên dịch của IDE cho phù hợp. IDE thường cung cấp các nút (button) cho việc biên dịch (Build) và chạy chương trình (Run). Khi nhấn Build IDE sẽ chạy một câu lệnh biên dịch tương ứng, thông thường câu lệnh chỉ biên dịch file main.cpp. Sinh viên cần tìm cách cấu hình trên IDE để thay đổi lệnh biên dịch: thêm file dragons.cpp, thêm option -std=c++11, -I .

4. Chương trình sẽ được chấm trên nền tảng Unix. Nền tảng chấm và trình biên dịch của sinh viên có thể khác với nơi chấm thực tế. Nơi nộp bài trên BKeL được cài đặt để giống với nơi chấm thực tế. Sinh viên phải chạy thử chương trình trên nơi nộp bài và phải sửa tất cả các lỗi xảy ra ở nơi nộp bài BKeL để có đúng kết quả khi chấm thực tế.
5. Sửa đổi các file dragons.h, dragons.cpp để hoàn thành bài tập lớn này và đảm bảo hai yêu cầu sau:
   - Chỉ có 1 lệnh **include** trong tập tin dragons.h là **#include "main.h"** và một include trong tập tin dragons.cpp là **#include "dragons.h"**. Ngoài ra, không cho phép có một **#include** nào khác trong các tập tin này.
   - Hiện thực các hàm được mô tả ở các Nhiệm vụ trong BTL này.
6. Sinh viên được khuyến khích viết thêm các class và các hàm để hoàn thành BTL này.
7. Sinh viên bắt buộc phải cung cấp phần hiện thực cho tất cả các class và phương thức được liệt kê trong BTL này. Nếu không thì SV sẽ bị 0 điểm. Nếu đó là class hoặc phương thức mà SV chưa thể giải quyết được, thì SV phải cung cấp phần hiện thực rỗng chỉ gồm cặp dấu mở đóng ngoặc nhọn {}.

5 Nộp bài

Sinh viên chỉ nộp 2 tập tin: dragons.h và dragons.cpp, trước thời hạn được đưa ra trong đường dẫn "Assignment 2 - Submission". Có một số testcase đơn giản được sử dụng để kiểm tra bài làm của sinh viên nhằm đảm bảo rằng kết quả của sinh viên có thể biên dịch và chạy được. Sinh viên có thể nộp bài bao nhiêu lần tùy ý nhưng chỉ có bài nộp cuối cùng được tính điểm. Vì hệ thống không thể chịu tải khi quá nhiều sinh viên nộp bài cùng một lúc, vì vậy sinh viên nên nộp bài càng sớm càng tốt. Sinh viên sẽ tự chịu rủi ro nếu nộp bài sát hạn chót (trong

vòng 1 tiếng cho đến hạn chót). Khi quá thời hạn nộp bài, hệ thống sẽ đóng nên sinh viên sẽ không thể nộp nữa. Bài nộp qua các phương thức khác đều không được chấp nhận.

6 Một số quy định khác

- Sinh viên phải tự mình hoàn thành bài tập lớn này và phải ngăn không cho người khác đánh cắp kết quả của mình. Nếu không, sinh viên sẽ bị xử lý theo quy định của trường vì gian lận.
- Mọi quyết định của giảng viên phụ trách bài tập lớn là quyết định cuối cùng.
- Sinh viên không được cung cấp testcase sau khi chấm bài, sinh viên sẽ được cung cấp phân bố điểm của BTL.
- Nội dung Bài tập lớn sẽ được Harmony với một câu hỏi trong bài Kiểm tra Cuối kỳ với nội dung tương tự.

7 Harmony cho Bài tập lớn

Bài kiểm tra cuối kì của môn học sẽ có một số câu hỏi Harmony với nội dung của BTL. Giả sử điểm BTL mà sinh viên đạt được là **a** (theo thang điểm 10), tổng điểm các câu hỏi Harmony **b** (theo thang điểm 5). Gọi **x** là điểm của BTL sau khi Harmony, cũng là điểm BTL cuối cùng của sinh viên. Các câu hỏi cuối kì sẽ được Harmony với 50% điểm của BTL theo công thức sau:

- Nếu $a = 0$ hoặc $b = 0$ thì $x = 0$
- Nếu a và b đều khác 0 thì

$$x = \frac{a}{2} + HARM(\frac{a}{2}, b)$$

Trong đó:

$$HARM(x, y) = \frac{2xy}{x + y}$$

Sinh viên phải giải quyết BTL bằng khả năng của chính mình. Nếu sinh viên gian lận trong BTL, sinh viên sẽ không thể trả lời câu hỏi Harmony và nhận điểm 0 cho BTL.

Sinh viên **phải** chú ý làm câu hỏi Harmony trong bài kiểm tra cuối kỳ. Các trường hợp không làm sẽ tính là 0 điểm cho BTL, và bị không đạt cho môn học. **Không chấp nhận giải thích và không có ngoại lệ.**

8 Gian lận

Bài tập lớn phải được sinh viên TỰ LÀM. Sinh viên sẽ bị coi là gian lận nếu:

- Có sự giống nhau bất thường giữa mã nguồn của các bài nộp. Trong trường hợp này, TẤT CẢ các bài nộp đều bị coi là gian lận. Do vậy sinh viên phải bảo vệ mã nguồn bài tập lớn của mình.
- Bài của sinh viên bị sinh viên khác nộp lên.
- Sinh viên không hiểu mã nguồn do chính mình viết, trừ những phần mã được cung cấp sẵn trong chương trình khởi tạo. Sinh viên có thể tham khảo từ bất kỳ nguồn tài liệu nào, tuy nhiên phải đảm bảo rằng mình hiểu rõ ý nghĩa của tất cả những dòng lệnh mà mình viết. Trong trường hợp không hiểu rõ mã nguồn của nơi mình tham khảo, sinh viên được đặc biệt cảnh báo là KHÔNG ĐƯỢC sử dụng mã nguồn này; thay vào đó nên sử dụng những gì đã được học để viết chương trình.
- Nộp nhầm bài của sinh viên khác trên tài khoản cá nhân của mình.
- Sử dụng mã nguồn từ các công cụ có khả năng tạo ra mã nguồn mà không hiểu ý nghĩa.

Trong trường hợp bị kết luận là gian lận, sinh viên sẽ bị điểm 0 cho toàn bộ môn học (không chỉ bài tập lớn).

KHÔNG CHẤP NHẬN BẤT KỲ GIẢI THÍCH NÀO VÀ KHÔNG CÓ BẤT KỲ NGOẠI LỆ NÀO!

Sau mỗi bài tập lớn được nộp, sẽ có một số sinh viên được gọi phỏng vấn ngẫu nhiên để chứng minh rằng bài tập lớn vừa được nộp là do chính mình làm.

9 Thay đổi so với phiên bản trước

...