TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

BÁO CÁO TỔNG KẾT

ĐỀ TÀI NGHIÊN CỨU KHOA HỌC CỦA SINH VIÊN
NĂM 2020

XÂY DỰNG GAME TÔM HÙM

Sinh viên thực hiện

LÊ QUANG DUY Lớp: CNTT1-K59 Khoa: Công nghệ thông tin

TẠ QUANG HUY Lớp: CNTT1-K59 Khoa: Công nghệ thông tin

Người hướng dẫn: ThS.PHẠM XUÂN TÍCH

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

BÁO CÁO TỔNG KẾT

ĐỀ TÀI NGHIÊN CỨU KHOA HỌC CỦA SINH VIÊN
NĂM 2020

XÂY DỰNG GAME TÔM HÙM

Sinh viên thực hiện

LÊ QUANG DUY	Nam, Nữ: Nam	Dân tộc: Kinh
Lớp: CNTT1-K59	Khoa: CNTT	Năm thứ: 2/4
TẠ QUANG HUY	Nam, Nữ: Nam	Dân tộc: Kinh
Lớp: CNTT1-K59	Khoa: CNTT	Năm thứ: 2/4

Ngành học: Công nghệ thông tin

Người hướng dẫn: ThS.PHẠM XUÂN TÍCH

MỤC LỤC

DANH MỤC HÌNH VẼ, SƠ DỒ:

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

THÔNG TIN KẾT QUẢ NGHIÊN CỨU CỦA ĐỀ TÀI

1. Thông tin chung:

- Tên đề tài: Xây dựng game Tôm Hùm
- Sinh viên thực hiện: Lê Quang Duy, Tạ Quang Huy
- Người hướng dẫn: ThS.PHẠM XUÂN TÍCH

2. Mục tiêu đề tài:

- Phát triển trò chơi Tồm Hùm thể loại chơi offline trên máy tính, thao tác và luật chơi giống với phiên bản truyền thống.

3. Tính mới và sáng tạo:

- Hiện tại chưa có một phiên bản trò chơi Tôm Hùm Việt Nam trên máy tính, vì thế, trò chơi Tôm Hùm offline trên máy tính là hoàn toàn mới, có thể dễ dàng, linh động trong việc chơi trò chơi chỉ cần có máy tính với không gian vừa đủ.

4. Kết quả nghiên cứu:

- Trò chơi Tôm Hùm có thể chạy trên mọi loại hệ điều hành có hỗ trợ JVM (Java Virual Machine), đã thử nghiệm trò chơi trên hai hệ điều hành phổ biến là Window và Linux, trò chơi hoạt động ổn định, dễ dàng thao tác.

5. Đóng góp về mặt kinh tế - xã hội, giáo dục và đào tạo, an ninh, quốc phòng và khả năng áp dụng của đề tài:

- Trò chơi Tôm Hùm trên máy tính ngoài việc đóng góp vào mặt giải trí cho người dùng, còn tái hiện và duy trì lại một trò chơi dân gian truyền thống, vốn có khả năng bị quên lãng trong thế giới công nghệ hiện đại bằng các hoạt động giải trí dễ dàng tiếp cận.

6. Công bố khoa học của sinh viên từ kết quả nghiên cứu của đề tài *(ghi rõ họ tên tác giả, nhan đề và các yếu tố về xuất bản nếu có)* hoặc nhận xét, đánh giá của cơ sở đã áp dụng các kết quả nghiên cứu *(nếu có)*:

Ngày tháng năm

Sinh viên chịu trách nhiệm chính thực hiện đề tài

(ký, họ và tên)

Nhận xét của người hướng dẫn về những đóng góp khoa học của sinh viên thực hiện đề tài *(phần này do người hướng dẫn ghi):*

Ngày tháng năm
Người hướng dẫn
(ký, họ và tên)

I. Mở đầu:

1. Tổng quan tình hình hình nghiên cứu thuộc lĩnh vực đề tài:

- Các phiên bản trò chơi dân gian được chuyển thể sang phiên bản điện tử ngày càng nhiều và phổ biến, đáp ứng xu thế số hóa trò chơi truyền thống, để mọi người có thể dễ dàng tiếp cận, sử dụng cho mục đích giải trí ngày càng tăng cao.
- Cũng bên cạnh đó, với sự phát triển của thể thao điện tử, người dùng có xu hướng giải trí ngay tại chỗ với chiếc máy tính của mình, các trò chơi truyền thống ngày càng trở nên khó để tổ chức vì tính bắt buộc yêu cầu về không gian, số người chơi, dụng cụ chơi..v.v, vì vậy, số hóa trò chơi dân gian là một giải pháp thiết thực.
- Trong các trò chơi dân gian đã được số hóa thành công, vẫn còn thiếu một trò chơi đơn giản nhưng quen thuộc với tuổi thơ nhiều người, đó là Cờ Tôm Hùm, hay còn gọi là Cờ Hùm Ăn Tôm, cũng bởi vì phong cách luật chơi đơn giản, dễ tiếp cận, thời gian chơi cũng linh hoạt, không quá dài cho những lần giải trí.Cờ Tôm Hùm cũng như mọi loại cờ khác, yêu cầu phải có chiến lược và chiến thuật khi chơi, giúp người chơi nâng cao khả năng xử lý tư duy trong những trận đánh.

2. Lý do lựa chọn đề tài:

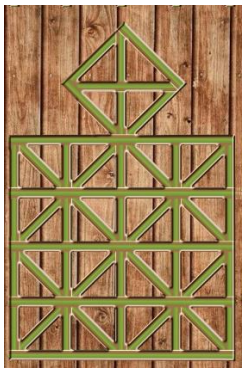
- Trỏ chơi Tôm Hùm phổ biến với tuổi thơ của nhiều người.
- Hiện chưa xuất hiện phiên bản nào trên các thiết bị điện tử.
- Trò chơi lành mạnh, giúp người chơi rèn luyện tư duy, chiến thu

3. Mục tiêu đề tài:

- Phát triển trò chơi Tôm Hùm phiên bản 2 người chơi truyền thống và phiên bản 1 người chơi (chơi với máy), giữ nguyên lối chơi, luật chơi và thiết kế truyền thống, cụ thể như sau:

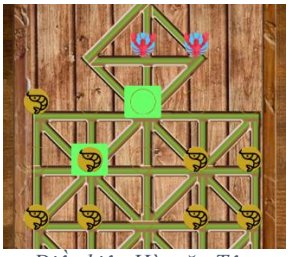
3.1. Luật chơi:

- Trong trò chơi cờ Hùm có 2 bên cụ thể là bên chơi Hùm và bên Tôm. Bên Hùm gồm có 3 quân Hùm, bên Tôm gồm có 16 quân Tôm và 1 Tôm Tướng, tổng cộng 17 quân.
- Thiết kế bàn cờ như sau: Coi bàn cờ là một ma trận kích thước 7x5 (7 hàng, 5 cột).



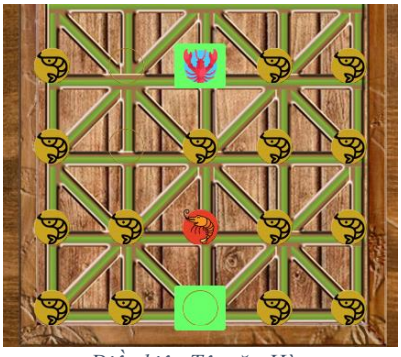
Bàn cờ

- Hùm đi 1 bước mỗi lần, được phép ăn Tôm khi ở giữa Hùm và Tôm là một ô trống:



Điều kiện Hùm ăn Tôm

- Tôm cũng chỉ được đi 1 bước mỗi lần, chỉ có Tôm Tướng được phép ăn Hùm khi và chỉ khi giữa Tôm Tướng và Hùm và một Tôm Tốt, như ví dụ dưới:



Điều kiện Tôm ăn Hùm

- Trò chơi kết thúc khi 1 trong 2 bên hết quân trước, bên thắng là bên còn quân.

3.2. Các tính năng cần có:

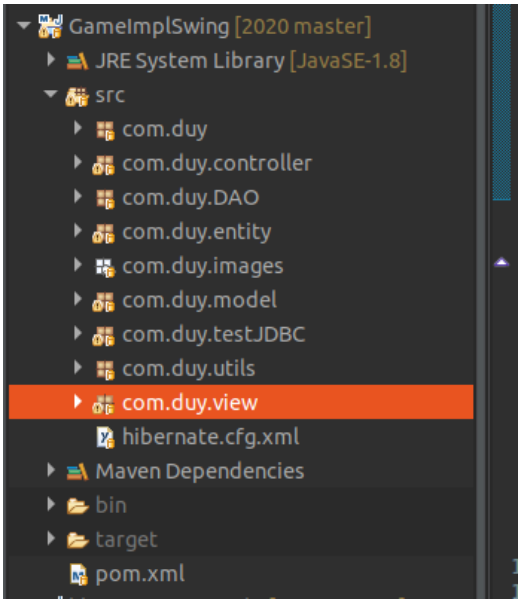
- Với luật chơi như trên, đặt ra yêu cầu trò chơi phải có một số tính năng nhất định và giao diện thân thiện với người dùng, thuật tiện cho việc di chuyển các quân và quan sát.
- Một số tính năng khiến trò chơi trở nên hấp dẫn hơn như tự động gợi ý các nước đi có thể trong mỗi lượt đánh, có thể trở về bước đi trước đó (undo/redo), tạo màn chơi mới hay chơi lại màn chơi cũ, lưu lại màn chơi.
- Không quên tính năng xem lại luật chơi, chọn quân chơi và chọn chế độ chơi.

4. Cách tiếp cận, phương pháp nghiên cứu:

- Trò chơi được lập trình trên ngôn ngữ Java, thiết kế theo hướng đối tượng, thư viện đồ hòa Swing, kết nối cơ sở dữ liệu MySQL bằng công nghệ Hibernate.

4.1. Thiết kế hệ thống:

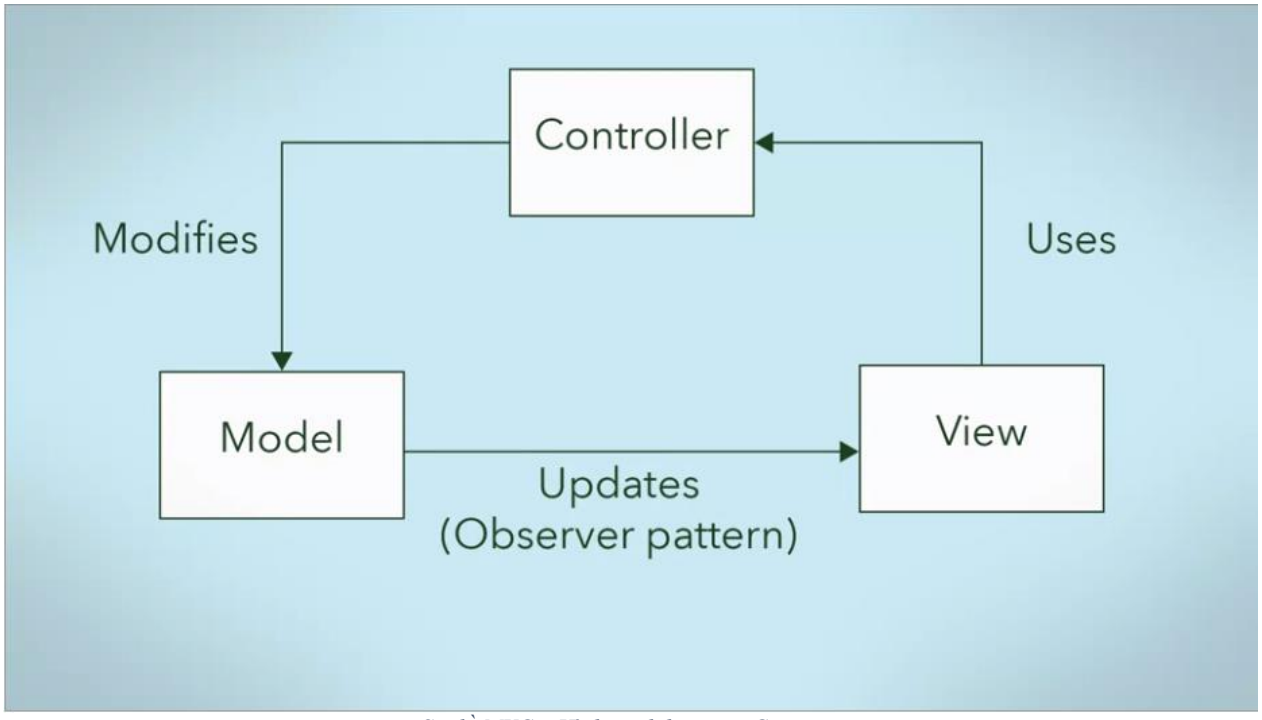
- Trò chơi Tôm Hùm phiên bản offline được thiết kế theo mô hình lập trình hướng đối tượng (Object Oriented Programming - OOP), để thuận lợi cho phương pháp bảo trì, mở rộng, sửa chữa sau này.
- Hệ thống được kết hợp từ nhiều class phân chia thành từng tầng và nhiệm vụ khác nhau.



Cây thư mục chương trình

- Hệ thống áp dụng một số Design Pattern để dễ dàng quản lý các class phối hợp với nhau như MVC Pattern, Singleton Pattern.

- MVC Pattern được cài đặt theo sơ đồ dưới đây:



Sơ đồ MVC – Hình minh họa trên Coursera

Những gì người dùng (client) thao tác là lớp View, phần hiển thị, bất cứ khi nào có thay đổi được diễn ra trên lớp View, View sẽ chuyển cho Controller xử lý, Controller có nhiệm

vụ tiếp nhận yêu cầu và trả lại yêu cầu cho Model để thay đổi dữ liệu, bất cứ khi nào Model thay đổii dữ liệu, sẽ cập nhật lại hiển thị trên lớp View.

Để MVC làm được điều đó, ta cài đặt class Observable, có các phương thức như thêm một Observer (add()), thông báo có sự thay đổi (notifier()) và xóa bỏ sự ràng buộc (reset()):

```
1.  public class Observable {
2.      private ArrayList<Observer> observers = new ArrayList<>();
3.
4.      public void add(Observer observer) {
5.          observers.add(observer);
6.      }
7.
8.      public void notifier() {
9.          for (Observer o:observers) {
10.             o.update(this);
11.         }
12.     }
13.
14.     public void reset() {
15.         observers = new ArrayList<>();
16.     }
17. }
```

Code Observable

ElementsManager sẽ kế thừa những phương thức trên, khi dữ dữ liệu bị thay đổi như di chuyển quân, mở một màn chơi mới, ta sẽ gọi phương thức thừa kề notifier() của lớp Observable để thông báo:

```
1.  public class ElementsManager extends Observable{
2.
3.      public void move(Point x, Point y) {
4.          elements.move(x, y);
5.          Element[][] map = elements.getMap();
6.
7.          notifier();
8.      }
9.
10.     public void setMap(Element[][] map) {
11.         elements.setMap(map);
12.         notifier();
13.     }
14.
15. }
```

Một phần class ElementsManager

BoardView sẽ Implement Observer, như thế mỗi khi có sự thay đổi từ notifier(), tất cả các View implement của Observer sẽ được cập nhật lại bằng cách ghi đè hàm update() , quá trình diễn ra tuần hoàn như sơ đồ MVC bên trên.

```
1.  public interface Observer {
2.      public void update(Observable o);
3.  }
```

Class Observer

```
1.  import com.duy.controller.GameController;
2.  import com.duy.utils.Constants;
3.  import com.duy.utils.Observable;
4.  import com.duy.utils.Observer;
5.
6.  public class BoardView extends JFrame implements MouseListener, Observer {
7.      @Override
8.      public void update(Observable o) {
9.          playPanel.update();
10.     }
11. }
```

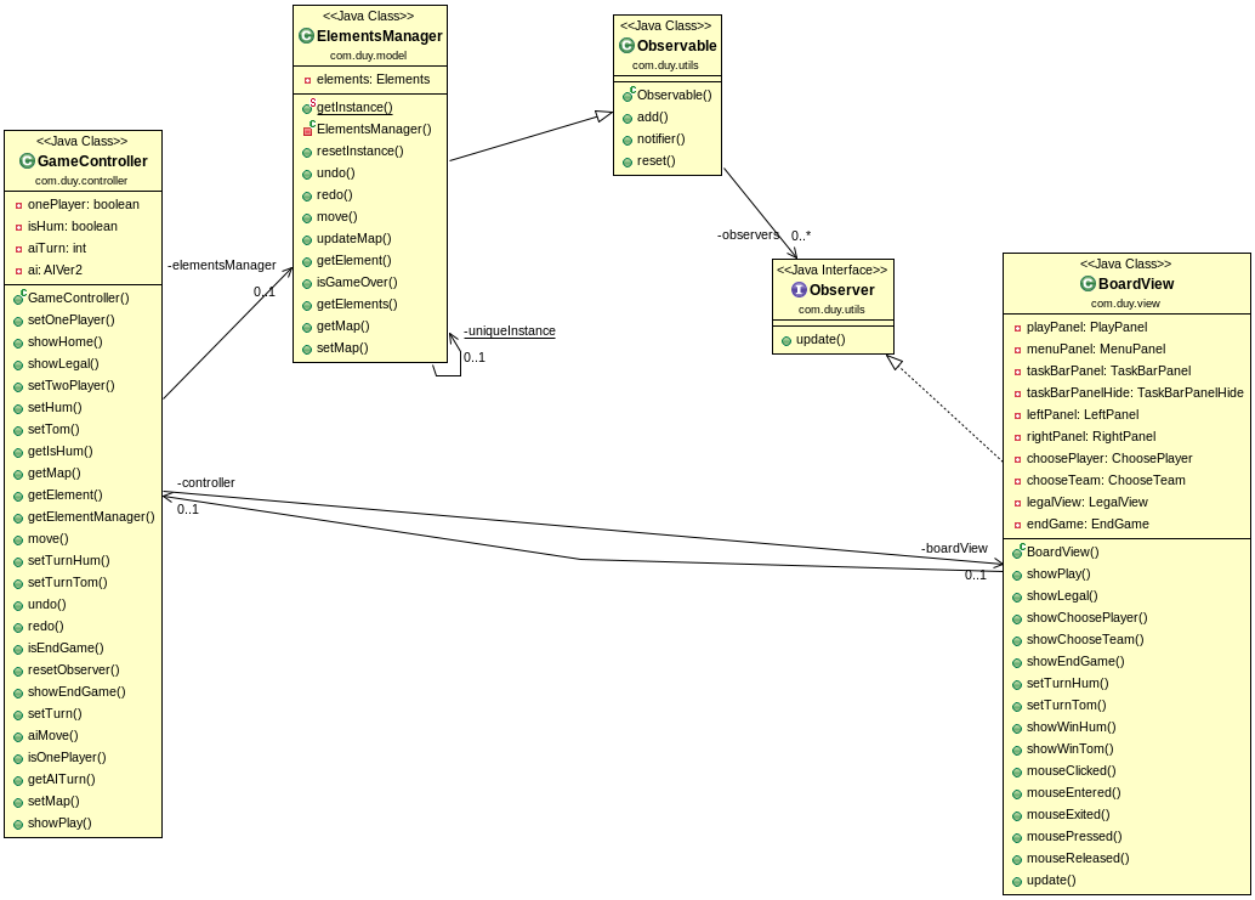
Một phần class BoardView

- Để tránh khởi tạo nhầm, hay khởi tạo lại class dữ liệu động model (bởi mỗi thời điểm chương trình chạy chỉ nên có một class model duy nhất được khởi tạo trong mỗi lượt chơi), vì thế Singleton Pattern giúp ngăn chặn việc gọi hàm khởi tạo trực tiếp từ bên ngoài, và cho phép khởi tạo lại khi cần thiết, ví dụ khi tạo ra một màn chơi mới.

```
1.  public class ElementsManager extends Observable{
2.      private static ElementsManager uniqueInstance= null;
3.      private Elements elements;
4.
5.      public static ElementsManager getInstance() {
6.          if (uniqueInstance==null) {
7.              uniqueInstance = new ElementsManager();
8.          }
9.          return uniqueInstance;
10.     }
11.
12.     private ElementsManager() {
13.         elements = new Elements();
14.     }
15.
16.     public void resetInstance() {
17.         uniqueInstance = null;
18.     }
19. }
```

Cài đặt Singleton Patternt

- Mô hình MVC sẽ thể hiện trong trò chơi Tôm Hùm như sau: ElementManager chính là dữ liệu động (Model), BoardView là phần hiển thị với người dùng (View), còn Controller đứng trung gian ở giữa có nhiệm vụ điều khiển BoardView và Model.



Mô hình MVC trong trò chơi

- Phía bên trong Model , gồm 1 interface chung là Element, đại diện cho tất cả các loại quân cờ, với mỗi loại quân cờ có các luật chơi, phương thức di chuyển, ăn quân và màu sắc khác nhau vì vậy có các lớp tương ứng như Hum, Tom, BTom.

```
1.  public interface Element {
2.      public List<Element> movesPossible(Element[][] boardData);
3.      public Point corr();
4.      public void setCorr(Point x);
5.  }
```

Interfect Element

- Ví dụ về 1 class implements Element: class Hum sẽ hoàn thiện các phương thức còn dang dở của Element, quan trọng nhất là phương thức movePossible(), trả về tất cả các nước đi của quân cờ đang xét có thể thực hiện, mỗi quân cờ đều có những quy luật đi khác nhau, đối với mỗi quân sẽ sét 8 hướng đi, nếu hướng đi nào có thể sẽ thêm vào List kết quả, hảm movePossible() trả về một List các kết quả đi:

```
1.  public class Hum implements Element {
2.
3.      private Point x;
4.      private URL url = getClass().getResource("/com/duy/images/");
5.
6.      public Hum(Point x) {
7.          this.x = x;
8.      }
9.
10.     @Override
11.     public List<Element> movesPossible(Element[][] boardData) {
12.         if (( i-1>=0  && (i>=3 || j==2)) && boardData[i-1][j] instanceof Empty) {
13.             moves.add(boardData[i-1][j]);
14.             if (i-2>=0 && (boardData[i-2][j] instanceof Tom || boardData[i-2][j]instanceof BTom)) {
15.                 moves.add(boardData[i-2][j]);
16.             }
17.         }
18.
19.         if ((i+1<=6 && (i>=2 || j==2)) && boardData[i+1][j] instanceof Empty) {
20.             moves.add(boardData[i+1][j]);
21.             if (i+2<=6 && (boardData[i+2][j] instanceof Tom || boardData[i+2][j]instanceof BTom)) {
22.                 moves.add(boardData[i+2][j]);
23.             }
24.         }
25.
26.         if ((j-1>=0 && (j==2 || i>=2 || j==3)) && boardData[i][j-1] instanceof Empty) {
27.             System.out.println("okok");
28.             moves.add(boardData[i][j-1]);
29.             if (j-2>=0 && (boardData[i][j-2] instanceof Tom || boardData[i][j-2]instanceof BTom)) {
30.                 moves.add(boardData[i][j-2]);
31.             }
32.         }
33.
34.         if ((j+1<=4 && (j==2 || i>=2 || j==1) ) && boardData[i][j+1] instanceof Empty) {
35.             moves.add(boardData[i][j+1]);
36.             if (j+2<=4 && (boardData[i][j+2] instanceof Tom||boardData[i][j+2]instanceof BTom)) {
37.                 moves.add(boardData[i][j+2]);
38.             }
39.         }
40.
41.         if (i+1<=6 && j+1 <=4  && (i-j==4||i-j==2||i-j==0||i-j==-
    2 && i!=1) && boardData[i+1][j+1] instanceof Empty) {
42.             moves.add(boardData[i+1][j+1]);
```

```
43.             if (i+2<=6 && j+2 <=4 && (boardData[i+2][j+2] instanceof Tom || boardData[i+2][j+2]instanceof BTom)) {
44.                 moves.add(boardData[i+2][j+2]);
45.             }
46.         }
47.
48.         if (i-1>=0 && j-1 >=0 &&  (i-j==4||i-j==2||i-j==0||i-j==-2) && boardData[i-1][j-1] instanceof Empty) {
49.             moves.add(boardData[i-1][j-1]);
50.             if (i-2>=0 && j-2 >=0 && (boardData[i-2][j-2] instanceof Tom||boardData[i-2][j-2]instanceof BTom)) {
51.                 moves.add(boardData[i-2][j-2]);
52.             }
53.         }
54.
55.         if (i-1>=0 && j+1<=4 && (i+j==2|| i+j==4||i+j==6||i+j==8) && boardData[i-1][j+1] instanceof Empty) {
56.             moves.add(boardData[i-1][j+1]);
57.             if (i-2>=0 && j+2<=4 && (boardData[i-2][j+2] instanceof Tom || boardData[i-2][j+2]instanceof BTom)) {
58.                 moves.add(boardData[i-2][j+2]);
59.             }
60.         }
61.
62.         if (i+1<=6 && j-1>=0 && ((i+j==2 && i!=1)|| i+j==4||i+j==6||i+j==8) && boardData[i+1][j-
    1] instanceof Empty) {
63.             moves.add(boardData[i+1][j-1]);
64.             if (i+2<=6 && j-2>=0 && (boardData[i+2][j-2] instanceof Tom || boardData[i+2][j-2]instanceof BTom)) {
65.                 moves.add(boardData[i+2][j-2]);
66.             }
67.         }
68.
69.
70.         return moves;
71.     }
72.
73.     @Override
74.     public Point corr() {
75.         return x;
76.     }
77.
78.     public void setCorr(Point x) {
79.         this.x = x;
80.     }
81. }
```

Class Hum implements Element

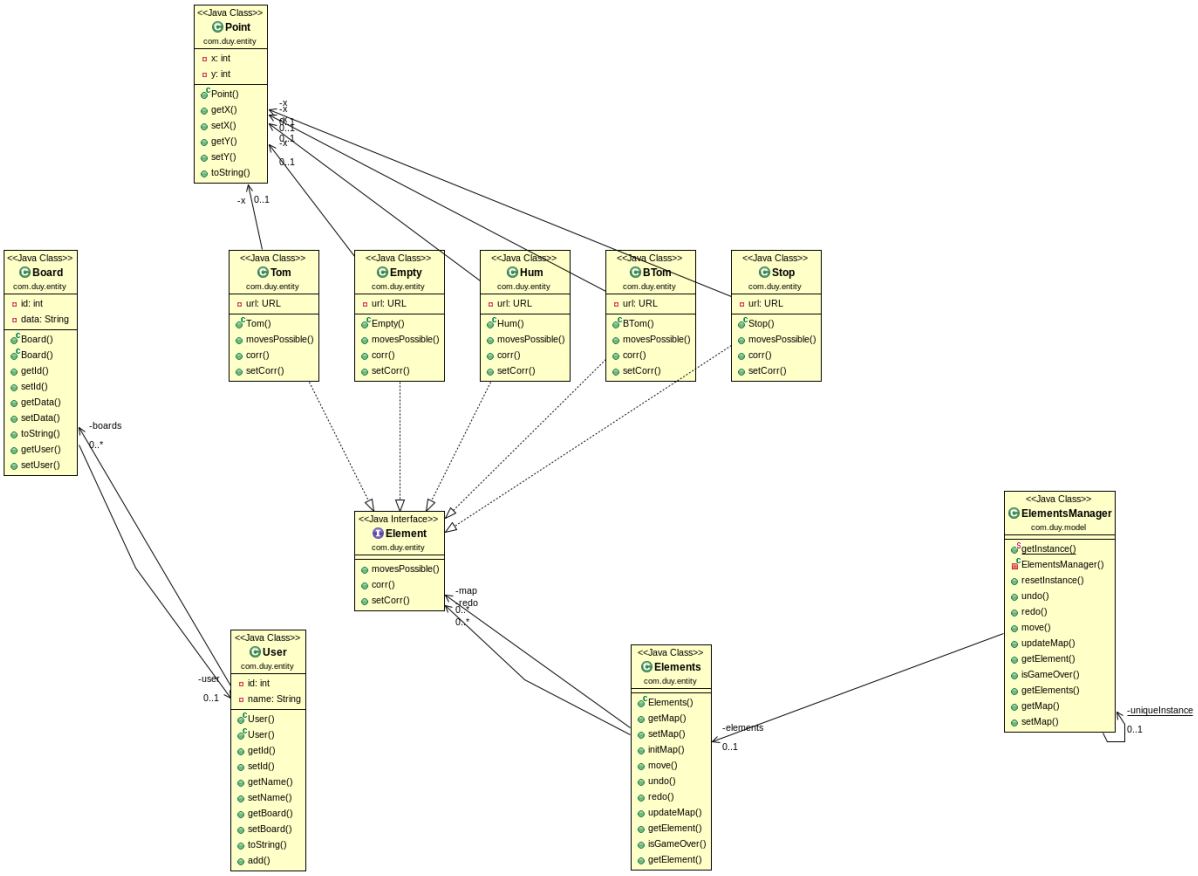
Một bàn cờ gồm các Element hợp thành với ma trận 7x5, vì vậy, Elements là một lớp chứa dữ liệu cả một bàn cờ tại một thời điểm nào đó. ElementManager có nhiệm vụ quản lý các Elements tại các thời điểm khác nhau, phục vụ cho thao tác như undo,redo..v..v

```
1.  public class Elements {
2.
3.      private Stack<Element[][]> map = new Stack<>();
4.      private Stack<Element[][]> redo = new Stack<>();
5.
6.      public Elements() {
```

```
7.          map.push(initMap());
8.      }
9.
10.     public Element[][] getMap() {
11.         return map.peek();
12.     }
13.
14.     public void setMap(Element[][] m) {
15.         map.push(m);
16.     }
17. }
```

Class Elements

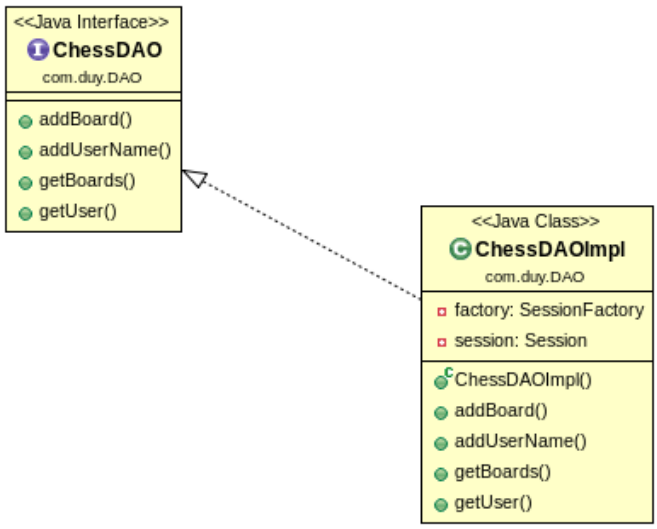
Như vậy tổng quát lại, ta có sơ đồ Model sẽ trông như thế này:



Sơ đồ tầng Model

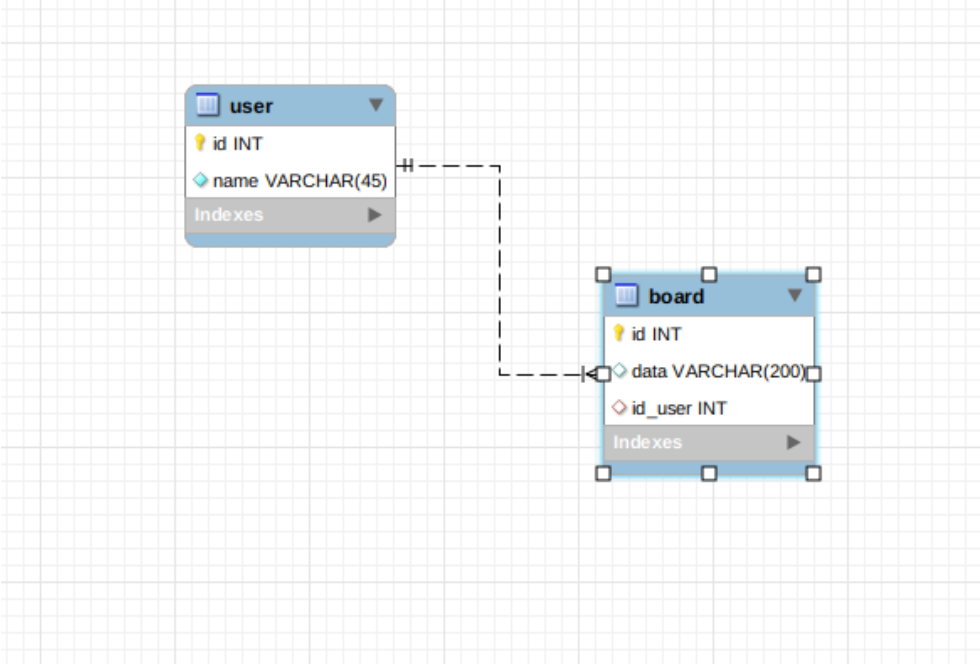
- Hai class Board và User được phát triển sau cùng để quản lý dữ liệu trò chơi, class User quản lý các người chơi khác nhau trên một máy tính, Board là các bàn cờ, User-Board được thiết kế cơ sở dữ liệu theo mô hình one-to-many, mỗi người chơi sẽ được phép lưu lịa nhiều bàn cờ khác nhau.

Dưới đây là mô hình tầng DAO giao tiếp, kiểm soát trực tiếp cơ sở dữ liệu, ChessDAO được định nghĩa là một interface, vì thế, mỗi lần muốn truy cập cơ sở dữ liệu chỉ cần gọi đến phương thức này.



Sơ đồ tầng DAO

Thiết kế Database gồm 2 bảng



Sơ đồ cơ sở dữ liệu

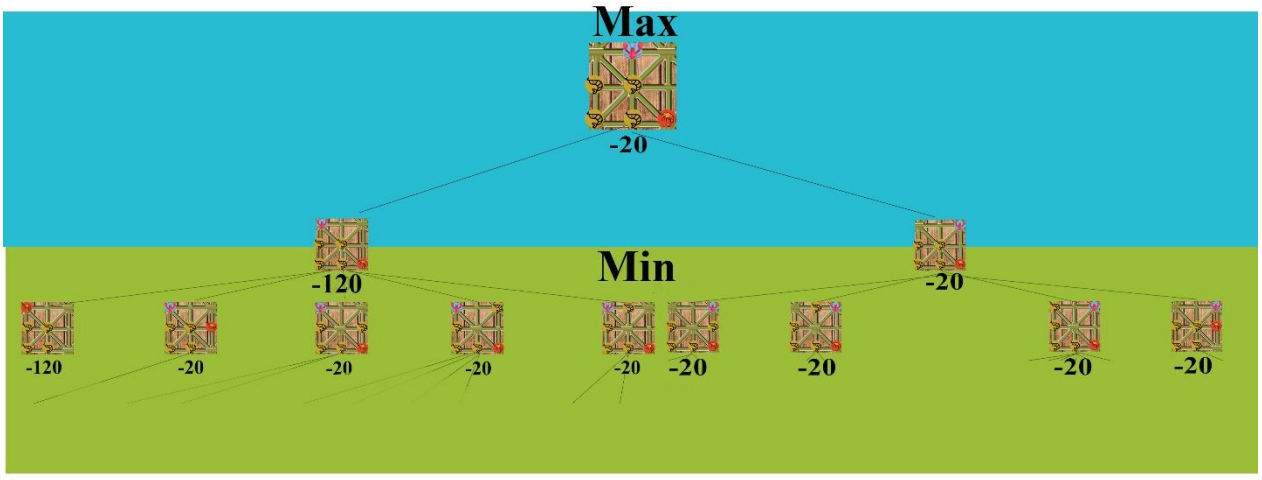
- Dưới đây là 2 class User và Board tham chiếu đến 2 bảng của cơ sở dữ liệu bên trên bằng công nghệ Hibernate:

```
1.  @Entity
2.  @Table(name="board")
3.  public class Board {
4.      @Id
5.      @GeneratedValue(strategy = GenerationType.IDENTITY)
6.      @Column(name="id")
7.      private int id;
8.
9.      @Column(name="data")
10.     private String data;
11.
12.     @ManyToOne(fetch = FetchType.LAZY,cascade= {CascadeType.DETACH,CascadeType.MERGE,CascadeType.P
    ERSIST,CascadeType.REFRESH})
13.     @JoinColumn(name="id_user")
14.     private User user;
15. }
```

```
1.  @Entity
2.  @Table(name="user")
3.  public class User {
4.      @Id
5.      @GeneratedValue(strategy = GenerationType.IDENTITY)
6.      @Column(name="id")
7.      private int id;
8.
9.      @Column(name="name")
10.     private String name;
11.
12.     @OneToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
13.     @JoinColumn(name="id_user")
14.     private List<Board> boards;
15.
16.     public User() {}
17. }
```

Hai class User và Board tham chiếu cơ sở dữ liệu

- Hệ thống đánh tự động sử dụng thuật toán Minimax để tìm kiếm nước đi tối ưu:
+ Minimax là một thuật toán tìm kiếm thường được sử dụng trong các trò chơi đối kháng để tính toán tối đa hóa nước đi tốt nhất của người chơi, với trò chơi Cờ Hùm, bản chất là một trò chơi đối kháng, giả dụ người máy là phe Hùm, với mỗi nước đi có thể của Hùm, thuật toán cần tìm nước đi tốt nhất cho Hùm qua các nước đi ấy, bằng cách tìm kiếm tất cả các nước đi của quân Tôm, bên Tôm lại cố gắng tối thiểu hóa cơ hội chiến thắng của bên Hùm như sơ đồ dưới đây:



Mô phỏng thuật toán Minimax

+ Với bàn cờ hiện tại đang đến lượt Hùm đánh (tức là lượt của máy), Hùm có 2 khả năng đi như sơ đồ trên, với 2 khả năng đi của Hùm sẽ có 9 khả năng đi của Tôm, bên Tôm sẽ đi nước đi tốt nhất của mình có giá trị là bằng cách tìm bàn cờ có giá trị đi càng nhỏ càng tốt, bên Hùm lại muốn giá trị càng lớn càng tốt. Nói cách khác, thuật toán Minimax giả sử 2 người chơi có cùng chiến lược chơi như nhau, cùng tối đa hóa nước đi của mình và tối thiểu hóa nước đi của đối thủ thông qua hàm đánh giá nước đi. Áp dụng nguyên lý trò chơi có tổng bằng 0, dưới đây là hàm đánh giá, gán điểm cho các quân cờ để thuật toán có thể tìm kiếm bước đi tối ưu:

```
1.  int evaluate(String board[][]) {
2.          int ans = 0;
3.          for (int i = 0; i < 7; i++) {
4.              for (int j = 0; j < 5; j++) {
5.                  if (board[i][j].equals("###") || board[i][j].equals("XXX")) {
6.                      ans+ = 0;
7.                  } else if (board[i][j].equals("Tom")) {
8.                      ans + = -15;
9.                  } else if (board[i][j].equals("BTom")) {
10.                     ans += -60;
```

```
                    } else if (board[i][j].equals("Hum")) {
                        ans += 100;
                    }
                }
            }
            return ans;
        }
```

Hàm đánh giá bàn cờ hiện tại

+ Thuật toán Minimax tìm nước đi tốt nhất cho bên máy bằng cách xét tất cả các nước đi với độ sâu mặc định bằng cách gọi đệ quy đến khi đạt độ sâu thì tổng hợp lại:

```
public int minimax(int depth, boolean isMax) {
        if (depth == 0) {
            return -evalution(elementsManager.getMap());
        }

        List<Element> games = e.movesPossible(elementsManager.getMap(),isMax);
        if (isMax) {
            int bestMove = -9999;
            for (Element game:games) {                    elementsManager.move(e.corr(), game.corr());
                bestMove = Math.max(bestMove, minimax(depth-1,e,!isMax));
                elementsManager.undo();
            }
            return bestMove;
        }
        else {
            int bestMove = 9999;
            for (Element game:games) {
                elementsManager.move(e.corr(), game.corr());
                bestMove = Math.min(bestMove, minimax(depth-1,e,!isMax));
                elementsManager.undo();
            }
            return bestMove;

        }
}

```

Mã giả Thuật toán Minimax

5. Đối tượng và phạm vi nghiên cứu:

- Đối tượng là phát triển game đầy đủ tính năng cơ bản, phạm vi nghiên cứu là mô hình hướng đối tượng, phát triển giao diện và thuật toán Minimax.

II. Kết quả nghiên cứu và phân tích (bàn luận) kết quả:

Dưới đây là những hình ảnh của game Tôm Hùm sử dụng Java và Mysql, toàn bộ sourcecode và chương trình được để ở địa chỉ:

https://github.com/Iamthankyou/cohumdangianmore/

Hoặc: https://bit.ly/cohumdangian

Kết quả nghiên cứu được mô tả trò chơi Tôm Hùm được miêu tả ở dưới đây:

1. Màn hình hiển thị chọn chế độ chơi:

- Màn hình hiển thị chế độ chơi bao gồm thanh menu About (thông tin phiên bản trò chơi), Help (trợ giúp), Open(Mở game cũ) và 3 button chính chọn chế độ chơi là 1 người chơi, 2 người chơi và xem luật chơi.

Màn hình mở đầu, chọn chế độ chơi

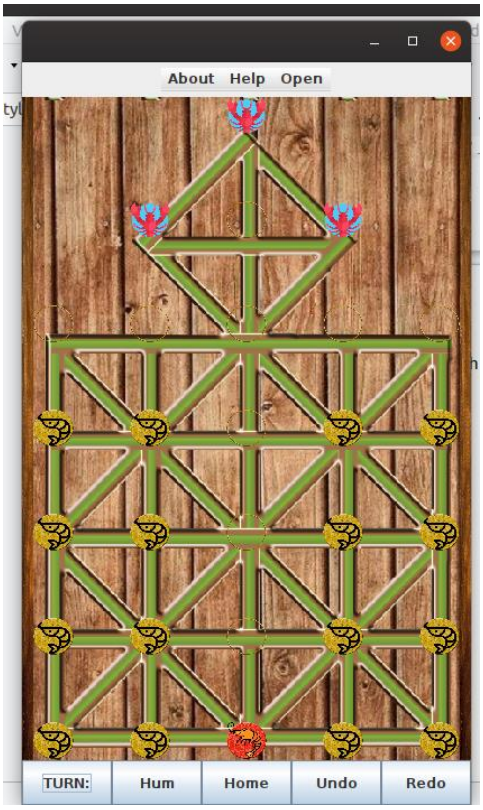
2. Màn hình hiển thị chọn quân chơi:

- Sau khi chọn chế độ chơi sẽ chọn tiếp đến quân chơi là phe Hùm hoặc phe Tôm.

Màn hình chọn quân chơi

3. Màn hình khi bắt đầu trò chơi:

- Đây là màn hình chính khi bắt đầu trò chơi, chia thành hai khu vực, phía bên trên là của phe Hùm và phía bên dưới là của phe Tôm, bên dưới bàn cờ là thanh trợ giúp như hiển thị hiện tại đang là lượt chơi của phe nào (TURN), trở về trang chủ (Home), và tính năng Undo/Redo.



Màn hình chính khi bắt đầu chơi

4. Màn hình kết thúc trò chơi:

- Khi một trong hai bên dành chiến thắng, màn hình kết thúc trỏ chơi sẽ hiện ra, có thể chọn chơi lại hoặc thoát trò chơi.

Màn hình kết thúc trò chơi

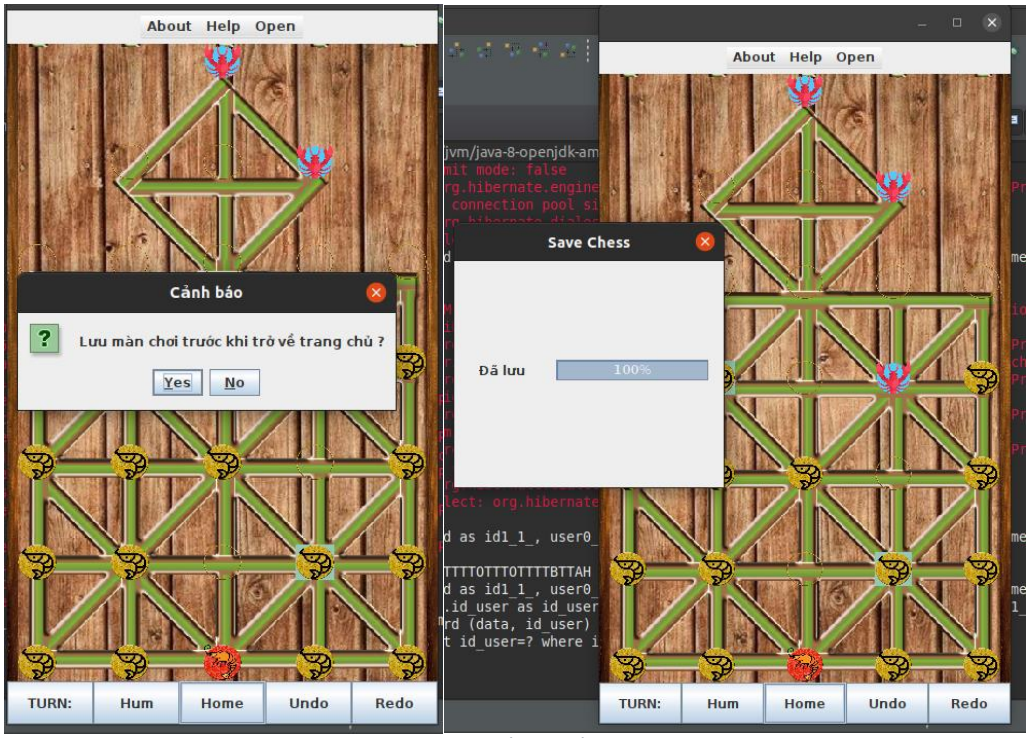
5. Màn hình hiển thị luật chơi:

- Như đã nói về luật chơi ở phần Mục đích bên trên, màn hình luật chơi luôn xuất hiện để người chơi có thể xem ở bất cứ lúc nào của của chương trình.

Màn hình hiển thị luật chơi

6. Màn hình khi chọn màn chơi mới:

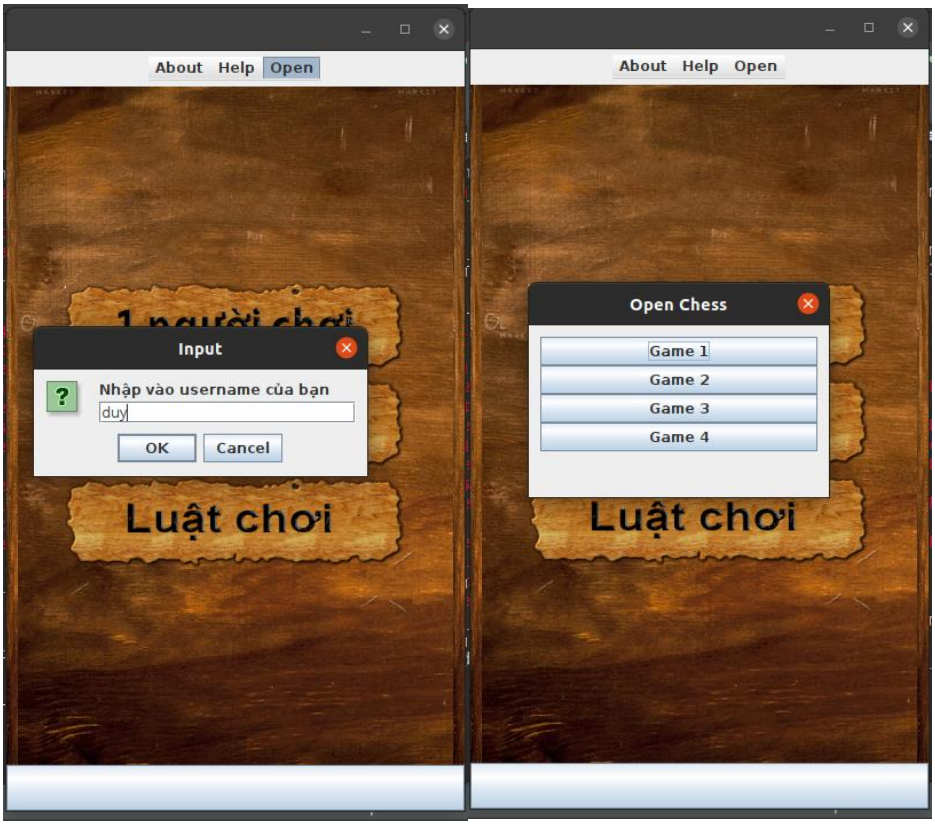
- Khi nhấn vào trở về trang chủ (Home) mà đang ở bàn cờ chơi dở, trò chơi sẽ hỏi có lưu game hay trở lại màn chính.



Màn hình khi nhấn vào về trang chủ

7. Màn hình mở lại game đã lưu khi nhập id người chơi:

- Khi nhấn vào Open trên thanh menu, trò chơi sẽ cho phép mở lại game cũ của người chơi bằng cách nhập vào Username, do cơ sở dữ liệu của trò chơi được lưu trên Internet nên người dùng hoàn toàn có thể chơi lại game cũ của mình trên mọi game cũ khác nhau chỉ cần có chương trình trò chơi.



Màn hình mở lại trò chơi cũ

III. Kết luận và kiến nghị:

a. Phần kết luận:

Trò chơi Tôm Hùm đã có những chức năng cơ bản của một trò chơi dành cho phiên bản máy tính, giao diện thân thiện với người sử dụng bằng cách hiển thị màu sắc, bố cục, hay tối ưu responsive.

Trò chơi có một số tính năng hữu ích cho người sử dụng như gợi ý bước đi, undo/redo, lưu lại màn chơi cho nhiều người chơi khác nhau, quá trình xử lý dữ liệu của trò chơi nhanh chóng và do được thiết kế cẩn thận ngay từ đầu nên trò chơi dễ dàng bảo trì, nâng cấp, mở rộng các tính năng.

Thông qua việc xây dựng trò chơi Tôm Hùm, nhóm nắm được những nền tảng cơ bản của lập trình hướng đối tượng, phát triển giao diện trên Java, liên kết dữ liệu trò chơi với cơ sở dữ liệu, những thuật toán để xác định nước đi có thể, và thuật toán Minimax để tìm kiếm nước đi tốt nhất cho tính năng chơi với máy. Những kỹ năng cần thiết phục vụ công việc xây dựng trò chơi như làm việc nhóm, vẽ sơ đồ mô phỏng, làm việc với một cấu trúc phức tạp với nhiều class liên quan tới nhau.

b. Phần kiến nghị:

Bên cạnh những ưu điểm kể trên, vẫn còn một số vấn đề với Trò chơi Tôm Hùm như chế độ chơi với máy chưa thực sự tốt, không hỗ trợ đa nền tảng, không có chế độ chơi online, vì thế, trò chơi vẫn cần phải phát triển tiếp để hướng tới một phiên bản Tôm Hùm ngày càng hoàn thiện hơn.

IV. Tài liệu tham khảo:

https://hibernate.org/orm/documentation/
https://www.coursera.org/learn/design-patterns
https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/