**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA CÔNG NGHỆ PHẦN MỀM**

z

**BÁO CÁO ĐỒ ÁN CUỐI KỲ**

**MÔN: NGÔN NGỮ LẬP TRÌNH JAVA**

**ĐỀ TÀI: GAME XẾP GẠCH**

**GIẢNG VIÊN HƯỚNG DẪN**

* Thầy Huỳnh Ngọc Tín
* Thầy Đỗ Văn Tiến

**SINH VIÊN THỰC HIỆN**

Bùi Đình Lộc thọ 13520844

*Bình Dương, ngày 1 tháng 6 năm 2015*

**PHỤ LỤC**

[**I. GIỚI THIỆU** 3](#_Toc420668956)

[**1.game xếp gạch** 3](#_Toc420668957)

[**2. Các chức năng chính** 4](#_Toc420668958)

[**3. Phân chia công việc** 5](#_Toc420668959)

[**II. THIẾT KẾ** 6](#_Toc420668990)

[**1.** **Sơ đồ Game** 6](#_Toc420668991)

[**2.** **Luồng xử lý** 8](#_Toc420668992)

[**3.** **Độ khó** 8](#_Toc420668993)

[**4.** **Thiết kế Class** 8](#_Toc420668994)

[**5.** **Kĩ thuật** 12](#_Toc420668995)

[**5.1 Cài đặt GameFlow** 12](#_Toc420668996)

[**5.2 AbstractFactory** 13](#_Toc420668997)

[**5.3 Kiểm tra va chạm** 15](#_Toc420668998)

[**5.4 Lưu Điểm** 15](#_Toc420668999)

[**III. GIAO DIỆN** 16](#_Toc420669000)

[**1.** **Sơ đồ màn hình** 16](#_Toc420669001)

[**2.** **Màn hình Menu Game** 16](#_Toc420669002)

[**3.** **Màn hình GamePlay** 17](#_Toc420669003)

[**4.** **Màn hình HighScore** 19](#_Toc420669004)

[**5.** **Màn hình About** 20](#_Toc420669005)

[**IV. TỔNG KẾT** 20](#_Toc420669006)

[**1.** **Cài đặt thử nghiệm** 20](#_Toc420669007)

[**2.** **Đánh giá** 20](#_Toc420669008)

[**3.** **Cấu hình đề nghị** 21](#_Toc420669009)

[**4.** **Website và tài liệu tham khảo** 21](#_Toc420669010)

# **I. GIỚI THIỆU**

## **1.game xếp gạch**

Tetris ([tiếng Nga](http://vi.wikipedia.org/wiki/Ti%E1%BA%BFng_Nga): Тетрис) là một [trò chơi điện tử](http://vi.wikipedia.org/wiki/Tr%C3%B2_ch%C6%A1i_%C4%91i%E1%BB%87n_t%E1%BB%AD) đầu tiên được thiết kế và phát triển bởi [Alexey Pajitnov](http://vi.wikipedia.org/w/index.php?title=Alexey_Pajitnov&action=edit&redlink=1). Nó được tạo ra vào ngày 6 tháng 6 năm 1984, trong lúc ông đang làm việc tại [Trung tâm Tính toán Dorodnicyn](http://vi.wikipedia.org/w/index.php?title=Trung_t%C3%A2m_T%C3%ADnh_to%C3%A1n_Dorodnicyn&action=edit&redlink=1) của [Viện hàn lâm Khoa học Liên Xô](http://vi.wikipedia.org/w/index.php?title=Vi%E1%BB%87n_h%C3%A0n_l%C3%A2m_Khoa_h%E1%BB%8Dc_Nga&action=edit&redlink=1) tại [Moskva](http://vi.wikipedia.org/wiki/Moskva). Ông lấy tên của trò chơi từ tiền tố "[tetra-](http://vi.wikipedia.org/w/index.php?title=Tetra-&action=edit&redlink=1) của [tiếng Hy Lạp](http://vi.wikipedia.org/wiki/Ti%E1%BA%BFng_Hy_L%E1%BA%A1p), có nghĩa là "bốn" (mỗi bộ phần trong trò chơi, gọi là [Tetromino](http://vi.wikipedia.org/w/index.php?title=Tetromino&action=edit&redlink=1), có bốn phần) và [quần vợt](http://vi.wikipedia.org/wiki/Qu%E1%BA%A7n_v%E1%BB%A3t) (tennis), trò thể thao Pajitnov thích nhất.

Trò chơi (hay một biến thể của nó) được bán cho hầu hết mọi máy trò chơi điện tử và hệ điều hành máy tính, cũng như trong các [máy tính đồ họa](http://vi.wikipedia.org/w/index.php?title=M%C3%A1y_t%C3%ADnh_%C4%91%E1%BB%93_h%E1%BB%8Da&action=edit&redlink=1), [điện thoại di động](http://vi.wikipedia.org/wiki/%C4%90i%E1%BB%87n_tho%E1%BA%A1i_di_%C4%91%E1%BB%99ng), máy nghe nhạc di động, [PDA](http://vi.wikipedia.org/wiki/Thi%E1%BA%BFt_b%E1%BB%8B_k%E1%BB%B9_thu%E1%BA%ADt_s%E1%BB%91_h%E1%BB%97_tr%E1%BB%A3_c%C3%A1_nh%C3%A2n), và nhiều thứ khác. Tuy nhiều biến thể của Tetris được bán trên thị trường vào thập niên 1980, trò chơi [cầm tay](http://vi.wikipedia.org/w/index.php?title=Tetris_(Game_Boy)&action=edit&redlink=1) của [Game Boy](http://vi.wikipedia.org/w/index.php?title=Game_Boy&action=edit&redlink=1), ra mắt năm 1989 đã biến trò chơi thành một trong những trò chơi thịnh hành nhất. Số thứ 100 của tạp chí [Electronic Gaming Monthly](http://vi.wikipedia.org/w/index.php?title=Electronic_Gaming_Monthly&action=edit&redlink=1) gọi Tetris là "Trò chơi Vĩ đại nhất trong Mọi thời đại". Nó đã bán được hơn 70 triệu phiên bản.

## **2. Các chức năng chính**

- Game có 3 mức độ chơi khác nhau đó là Easy, Normal và Difficult giúp cho người chơi giải trí, rèn luyện thêm kĩ năng tư duy, tưởng tượng hình ảnh và tính phản xạ nhanh.

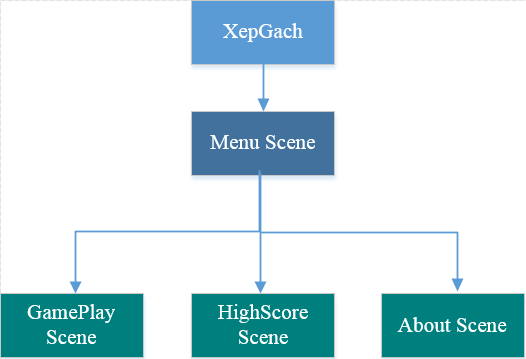
- Game cho phép lưu điểm người chơi và sẽ lưu điểm 10 người có điểm cao nhất.

## **3. Phân chia công việc**

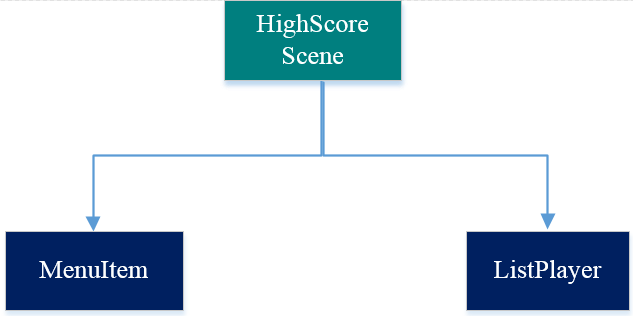
|  |  |
| --- | --- |
| **Thời gian** | **Hoạt động** |
| 16/05/2015 | Nhận được danh sách đồ án tham khảo và suy nghĩ xem nên chọn đồ án nào. |
| 17/05/2015 | Bàn bạc với bạn bè và quyết định làm đồ án với 1 thành viên và chọn đồ án là Game Xếp Gạch. |
| 18/05/2015 | Tạo Project netbeans và sau đó xem xét nên thiết kế game như thế nào. |
| 19/05/2015 | Tìm hiểu về luật chơi, các hình trong game và xem về độ khó xác định xem có những màn hình game nào. |
| 20/05/2015 | Bắt đầu code, xây dựng các Class cơ sở trước như GBlock, Vector, Rect, (GBlock ban đầu được đặt là Block nhưng bị trùng với class bên java nên phải đổi tên) và cài đặt xong GameFlow |
| 21/05/2015 | Thiết kế và bắt đầu viết class Shape và các class kế thừa của nó sau đó vẽ thử lên màn hình và fix một số bugs |
| 22/05/2015 | Bắt đầu xử lý về bàn phím và chuột sau đó cho phép các hình di chuyển nhưng chưa xét va chạm gặp một số lỗi trong việc cài đặt ma trận nhưng đã sửa được lúc cuối ngày |
| 23/05/2015 | Viết các hàm xử lý va chạm và gặp lỗi về logic |
| 24/05/2015 | Tiếp tục fix bugs về va chạm sai với 1 số trường hợp nhưng chưa rõ nguyên nhân chỗ nào |
| 25/05/2015 | Fix được bugs và bắt đầu vào việc viết các Scene sau khi tham khảo mô hình XNA và DirectX |
| 26/05/2015 | Viết Scene và bắt đầu xây dựng các Scene |
| 27/05/2015 | Tạo class MenuItem và lúc này thêm cái MenuItem vào game như Replay, Back, Pause, Resume sau đó thiết kế UI game theo style Flat Build thử ra file jar và chạy thử thấy game chiếm dung lượng ram khá cao (200 – 300 MB) và thấy rằng có quá nhiều rác được tạo ra trong chương trình nên gọi hàm System.gc() để dọn rác 1 cách thủ công và dung lượng được giảm xuống còn khoảng 30 – 60MB RAM |
| 28/05/2015 | Bắt đầu thiết kế các sơ đồ game và viết báo cáo |
| 29/05/2015 | Hoàn thành báo cáo. |
| 30/05/2015 | Hoàn thành đồ án. |

# **II. THIẾT KẾ**

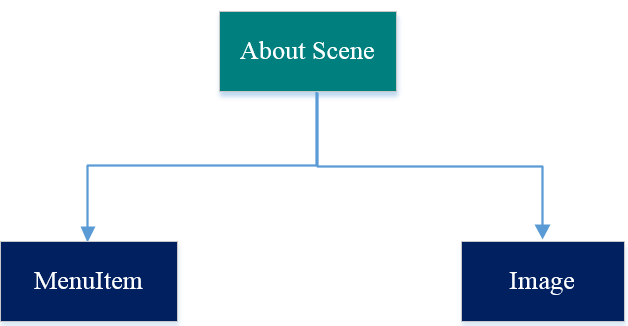
## **Sơ đồ Game**

* 1. **Màn hình chính**

**- Khối Xếp Gạch**: đây là khối nền cho việc chạy các Scene, khối này hoạt động đầu tiên khi game được mở lên.  
 **- Khối Menu Scene:** đây là màn hình đầu tiên thấy được khi vào game được chạy trên nền của khối XepGach, khối này sẽ chứa các khối con là GamePlay Scene, HighScore Scene, About Scene

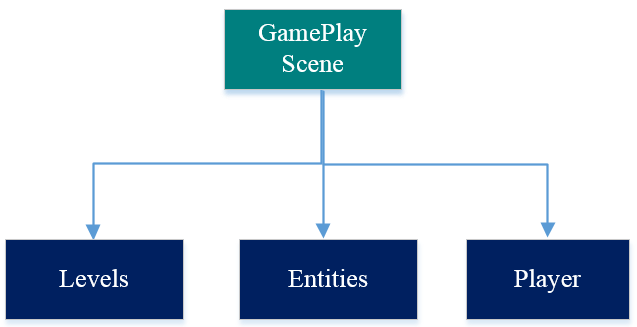
* 1. **HighScore Scene**

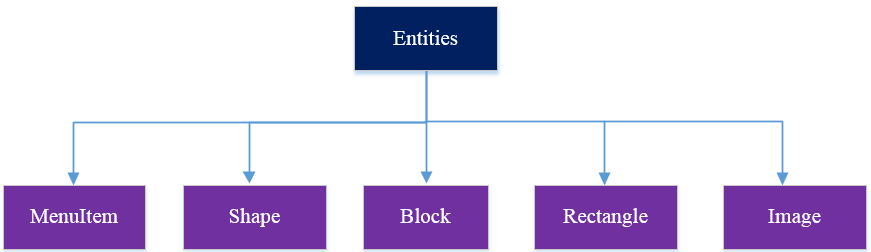
**-** là khối con của MenuScene chứa các thành phần đó là MenuItem và ListPlayer  
- **MenuItem**: các item trong Scene (như Back,…)  
- **ListPlauyer**: danh sách các player có điểm cao

* 1. **About Scene**

**-** là khối con của MenuScene chứa các thành phần đó là MenuItem và Image  
- **MenuItem**: các item trong Scene (như Back,…)  
- **Image**: chứa hình ảnh tác giả và thông tin của tác giả

* 1. **Màn hình GamePlay**





**-** là khối con của MenuScene chứa các thành phần đó là MenuItem và Image  
- **Levels**: độ khó của game  
- **Player**: thông tin người chơi (tên, điểm, …)  
- **Entities**: các thực thể của game

**- MenuItem:** các item trong Scene

**- Shape:** các khối hình trong game

**- Block:** các khối hình nhỏ cấu thành 1 Shape

**- Rectangle:** hình chữ nhật trong game

**- Image:** các hình ảnh hiển thị trong game

## **Luồng xử lý**

- Mô hình được tham khảo từ mô hình của XNA, DirectX. khi game được mở lên thì sẽ khởi tạo những thứ cần thiết cho việc hoạt động game, sau đó một Thread mới sẽ được tạo ra ở Thread này sẽ là một vòng While giúp cho việc Update và Draw được thực hiện liên tục và luân phiên, ngoài ra các sự kiện của Keyboard và Mouse có thể can thiệp vào luồng bất cứ lúc nào.



- **Khối Open Game**: đây là lúc game được mở lên đối tượng game sẽ được khởi tạo.

- **Khối Init**: ở đây sẽ khởi tạo các thành phần cần thiết cho việc chạy game và bắt đầu tạo 1 Thread mới và bắt đầu cho game hoạt động với FPS là 60 frames / giây.

- **Khối Update**: sau khi game được chạy thì sẽ update Game cứ mỗi frame thì hàm Update sẽ được gọi một lần và xử lý ở đây trước khi được hiển thị lên màn hình.

- **Khối Draw**: có nhiệm vụ là vẽ các đối tượng lên màn hình sau khi được xử lý ở khối Update.

- **Khối Keyboard, Mouse Events**: đây là khối có 1 luồng riêng với game có thể can thiệp vào bất cứ khi nào nhận được sự kiện từ bàn phím hay chuột.

## **Độ khó**

- Có 3 mức độ khó cho Game đó là: Easy, Normal và Difficult và điểm tính là 1 điểm cho 1 hàng ngang được ăn. Game kết thúc khi các khối hình chạm phía trên màn hình game.

* + **Easy**: tốc độ di chuyển của Shape chậm, có hình mô phỏng ở dưới điểm chạm của Shape.
  + **Normal**: tốc độ di chuyển nhanh hơn không có hình mô phỏng ở dưới.
  + **Difficult**: tốc độ di chuyển ban đầu nhanh và nhanh dần lên không có hình mô phỏng ở dưới.

## **Thiết kế Class**

|  |  |  |
| --- | --- | --- |
| **STT** | **Tên Class** | **Mô tả** |
| **Class cơ bản** | | |
| **1** | MainGame | Kế thừa JFrame và là class nội của class XepGach, Gọi class XepGach, truyền các Keyboard và Mouse Event cho game. Hàm này được gọi trong hàm static của class XepGach. Các hàm:  + **MainGame():** Constructor sẽ gọi hàm initGUI().  + **void initGUI():** tạo class XepGach, add các Listener cho JFrame.  + **void runGame():** gọi hàm runGame() của class XepGach cho phép chạy game. |
| **2** | XepGach | Kế thừa jPanel, implements các Interface ActionListener, Runnable. Là nơi game được chạy với kiến trúc đã được thiết kế. gồm các hàm:  + **void runGame():** tạo mới 1 Thread và chạy Thread này.  + **void init():** khởi tạo và load các hình ảnh nền.  + **void run():** là hàm Override được gọi khi Thread start.  Hàm này tạo 1 vòng While(true) cho phép tính toán liên tục thời gian sao cho số frame đạt được là 60 frames / giây và gọi hàm update (1 giây sẽ gọi hàm update 60 lần).  + **void update(float):** hàm này sẽ gọi hàm update của Scene hiện tại.  + **void onKeyPressed(KeyEvent e):** nhận sự kiện của bàn phím từ JFrame MainGame sau đó truyền đến hàm xử lý của Scene.  + **void mouseMotion(MouseStates, Vector):** MouseStates là một enum của class XepGach ghi các trạng thái của Mouse, nhận sự kiện của chuột từ JFrame MainGame sau đó truyền đến hàm xử lý của Scene.  + **void doDrawing(Graphics):** hàm này sẽ gọi hàm doDraw của Scene hiện tại. |
| **3** | GBlock | Là 1 khối vuông nhỏ dùng để ghép lại thành Shape |
| **4** | MenuItem | Là 1 item hiển thị hình ảnh lên có thể click vào được |
| **5** | Player | Chứa thông tin về Player |
| **6** | Rect | Mô phỏng 1 hình chữ nhật |
| **7** | Vector | Mô phỏng vector trong toán học |
| **Class về Scene** | | |
| **8** | Scene | Là một abstract class, chứa các hàm dùng để chạy trong game. Một số phương thức chính  - **void** **update(float):** là hàm update trong game 1 lần gọi tương ứng với 1 frame.  - **void** **doDraw(Graphics):** hàm vẽ, các đối hiển thị trên Scene sẽ được vẽ ở đây.  - **void** **loadContent():** hàm load các content cho Scene được gọi khi khởi tạo Scene.  **- mouseMotion(MouseStates, Vector):** hàm nhận và xử lý các Event của chuột  - **void** **keyBoardHandle(KeyEvent):** hàm nhận và xử lý các sự kiện của bàn phím  **- String getSceneName():** hàm trả về tên của Scene |
| **9** | AboutScene | Kế thừa từ lớp Scene, hiển thị thông tin và hình ảnh của tác giả. |
| **10** | GamePlayScene | Kế thừa từ lớp Scene, với input truyền vào Constructor là GameLevel cho biết độ khó của Scene. Ngoài các phương thức kế thừa từ lớp Scene còn có thêm 1 số phương thức khác hỗ trợ cho việc sử lý game. |
| **11** | HighscoreScene | Kế thừa từ lớp Scene, hiển thị điểm số của 10 người chơi có điểm cao nhất và sắp xếp theo thứ tự giảm dần từ trên xuống. |
| **12** | MenuScene | Kế thừa từ lớp Scene, màn hình đầu tiên được hiển thị khi vào game, cho phép chọn các Scene. |
| **Class về Shape** | | |
| **13** | Shape | Là abstract class. Cho phép định nghĩa 1 hình trong game, trong game này có 7 hình. Bên trong có chứa 1 ma trận dùng cho việc vẽ các khối GBlock tạo thành 1 hình. Một số hàm cơ bản: + **void doDraw(Graphics):** dùng để vẽ hình lên màn hình  + **void initialized():**hàm khởi tạo các biến, thành phần cần thiết cho Shape  + **Rect getBound():** trả về vùng hình chữ nhật xung quanh của Shape  **+ void setTimeMove(float):** set thời gian di chuyển xuống của Shape cái này sẽ phụ thuộc vào Level của game  + **setRotationValue(in):** cài đặt giá trị quay hình, chạy từ tới 3 thì ta quay 1 lần là 90° nên chỉ có tối đa 4 hình khác nhau với 1 Shape  + **void setPosition(Vector pos):** set vị trí của Shape  + **List<GBlock> getListBlock():** hàm trả về list các GBlock thuộc Shape |
| **14** | IShape | Kế thừa từ Shape, có hình dạng  C:\Users\DELL\Desktop\Bao cao Java\IShape.png |
| **15** | LLeftShape | Kế thừa từ Shape, có hình dạng  C:\Users\DELL\Desktop\Bao cao Java\LLeftShape.png |
| **16** | LRightShape | Kế thừa từ Shape, có hình dạng  C:\Users\DELL\Desktop\Bao cao Java\LRightShape.png |
| **17** | RectShape | Kế thừa từ Shape, có hình dạng  C:\Users\DELL\Desktop\Bao cao Java\RectShape.png |
| **18** | WShape | Kế thừa từ Shape, có hình dạng  C:\Users\DELL\Desktop\Bao cao Java\WShape.png |
| **19** | ZReftShape | Kế thừa từ Shape, có hình dạng  C:\Users\DELL\Desktop\Bao cao Java\ZShapeLeft.png |
| **20** | ZRightShape | Kế thừa từ Shape, có hình dạng  C:\Users\DELL\Desktop\Bao cao Java\ZShapeRight.png |

## **Kĩ thuật**

### **5.1 Cài đặt GameFlow**

- Từ việc tham khảo và làm việc với kiến trúc của XNA và DirectX nên đã lựa chọn làm GameFlow theo mô hình của 2 Engines này.

**- Các biến cần thiết cho việc cài đặt:**

* + - * ***public Thread m\_thread***

- Dùng để tạo 1 Thread mới và chạy game

* + - * ***private long m\_lastTime***- Dùng để ghi lại lần thời gian lần cuối được gọi của vòng While theo đồng hồ hệ thống
      * ***private long m\_deltaTime***

- Dùng để xác định thời gian chênh lệch giữa 2 Frame. VD: 60 frames / giây thì m\_deltaTime = 1 / 60 (giây)

* + - * ***private static final float SECOND\_PER\_FRAME = 1.0f / 60***

- Hằng dùng xác định số giây chuẩn nếu Game chạy đúng 60 frames / giây

**- Vòng đời game sẽ được cài đặt ở phương thức run**

+ với lần đầu thì m\_lastTime sẽ bằng 0 ta sẽ lấy thời gian hiện tại của hệ thống bằng hàm (Calendar.getInstance()).getTimeInMillis() hàm này trả về thời gian hệ thống ở Mili giây, sau khi lấy được thời gian hiện tại (m\_currentTime) ta tính toán khoảng thời gian chênh lệch giữa 2 lần gọi của While ta được m\_deltaTime sau đó gọi hàm update truyền vào tham số là thời gian chênh lệch giữa 2 Frame và lúc này gán lại m\_lastTime bằng với m\_currentTime. Và cho Thread Sleep 1 khoảng thời gian là số SECOND\_PER\_FRAME để game có FPS trong khoảng 50 – 60 frames/s

public void run() {

m\_lastTime = 0;

while(true){

if(IsClosed)

break;

long m\_currentTime =(Calendar.getInstance()).getTimeInMillis();

m\_deltaTime = m\_currentTime - m\_lastTime;

update((float)m\_deltaTime / 1000);

m\_lastTime = m\_currentTime;

try {

Thread.sleep((long)(SECOND\_PER\_FRAME \* 1000)); //sleep

} catch (InterruptedException ex) {

}

}

}

**- ở hàm update sau khi xử lý sẽ gọi hàm repaint() hàm này là của JPanel cho phép vẽ lại màn hình và hàm để xử lý đó là doDrawing**

### **5.2 AbstractFactory**

#### **1. Class Shape**

- Sử dụng **AbstractFactory** cho các Shape. Class Shape là một abstract class có các phương thức và thuộc tính cần thiết để tạo, tùy chỉnh và vẽ một mình.

- **Các thuộc tính:**

* + - ***protected int m\_rotation;***Thuộc tính này cho biết rằng hình đang quay ở đâu như khi nhấp phím qua lại thì hình sẽ xoay và chạy từ 0 đến 3 tùy hình vì Shape chỉ quay góc 90° một lần.
    - ***protected Vector m\_position;***

Thuộc tính này cho biết tọa độ của Shape trên màn hình, điểm mốc là góc trái trên.

* + - ***protected List<GBlock> m\_listBlocks***Thuộc tính này là một list các GBlock là từng ô nhỏ cấu tạo thành 1 Shape.
    - ***protected float m\_timeMove***Thuộc tính này là thời gian di chuyển xuống của shape, nó phụ thuộc vào độ khó của game
    - ***protected float m\_saveTimeMove***

Thuộc tính này dùng để lưu thời gian di chuyển xuống của shape. Vì sau khi đến thời gian ***timeMove*** sẽ về 0 và cần set lại

* + - ***protected Rect m\_rectBound***

Thuộc tính này dùng để xác định đường viền chữ nhật bao quanh hình

* + - ***protected boolean m\_allowMove***

Thuộc tính này cho phép Shape có được di chuyển nữa hay không, như khi va chạm vào GBlock ở dưới thì Shape phải dừng lại

* + - ***protected int[][] m\_arrayValue***

Thuộc tính này lấy về một mảng giá trị của Shape, giá tị = 1 là chỗ đó có GBlock còn = 0 thì là ô trống. VD:

***m\_arrayValue = new int[][]***

***{***

***{0, 1},***

***{0, 1},***

***{1, 1}***

***};***

Đây là mảng của LLeftShape

#### **2. Class Scene**

- Sử dụng **AbstractFactory** cho các Scene. Class Scene là một abstract class có các phương thức và thuộc tính cần thiết tạo 1 màn game với các phương thức vẽ và xử lý. Class Scene như là 1 template với khung xương cho các Scene khác làm chuẩn.

- Thuộc tính:   
 ***protected String m\_sceneName***: thuộc tính này là tên của Scene

- Các hàm:

* + - ***public void doDraw(Graphics g)***

- Hàm này được gọi trong hàm doDrawing của class XepGach

* + - ***public void update(float dt)***

- Hàm này được gọi trong hàm update của class XepGach

* + - ***public void loadContent()***  
      - Hàm này được gọi ở hàm khởi tạo của Scene có nhiệm vụ là load các dữ liệu cần thiết cho Scene như hình ảnh, âm thanh,….
    - ***public void mouseMotion(MouseStates, Vector)***- Hàm này được gọi ở hàm mouseMotion của class XepGach dùng để nhận và xử lý sự kiện chuột
    - ***public void keyBoardHandle(KeyEvent e)***- Hàm này được gọi ở hàm onKeyPressed của class XepGach dùng để nhận và xử lý sự kiện bàn phím

### **5.3 Kiểm tra va chạm**

- Kiếm tra va chạm là phần không thể thiếu nếu có tương tác giữa các đối tượng trong game. Trong game xếp gạch sẽ có các kiểu kiểm tra va chạm như sau:

* + - ***Va chạm với 1 trong những khối GBlock ở dưới hoặc 2 bên.***
* Hàm **boolean** **checkCollidedUnder** trả về biến boolean cho biết nếu mà phía dưới có va chạm với GBlock nào đó thì trả về true còn không là false.   
  Game có lưu 2 ma trận, 1 ma trận là mảng tương ứng với số dòng / cột tối đa của game (***m\_arrayValue***) và 1 ma trận là của Shape chỗ nào có giá trị bằng 1 là có GBlock. Ta sẽ dựa vào tọa độ hiện tại của Shape sẽ tương ứng với vùng nào trong ma trận Game từ đó so sánh với ma trận của Shape nếu thấy giá trị của 2 ma trận tại vị trí đó cùng bằng 1 thì lúc đó có va chạm xảy ra.
* Hàm **boolean** **checkCollideLeftRightTop(CollitionState state)** trả về biến boolean phụ thuộc vào State để kiểm tra bên trái, phải, trên để cho phép di chuyển hay xoay hình không. Cũng như việc kiểm tra bên dưới nhưng hàm này chỉ gọi khi xử lý bàn phím. Từ vị tri của Shape ta tìm được vị trí tương ứng trong tra trận Game (nếu là bên trái thì tọa độ cột sẽ giảm 1 đơn vị, bên phải thì tọa độ cột tăng một đơn vị, bên trên thì tọa độ hàng giảm 1 đơn vị), sau khi xác định được vị trí trong ma trận Game thì ta đem so sánh với ma trận tương ứng của Shape nếu cùng bằng 1 thì tức là có va chạm.
  + - ***Va chạm với thành màn hình***

- kiểm tra xem tọa độ của hình có vượt quá chiều dài hay chiều rộng của màn hình không rồi mới cho phép tăng tọa độ, di chuyển hay xoay hình.

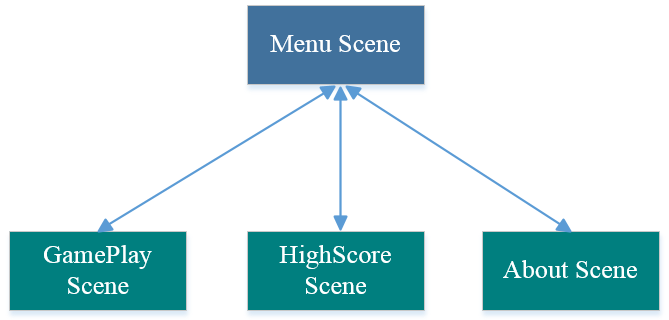
### **5.4 Lưu Điểm**

- Sử dụng **FileReader** và **BufferedReader** để đọc dữ liệu đã lưu (file \*.sav) sẽ lấy thông tin các Player lên và sắp xếp theo thứ tự điểm giảm dần.

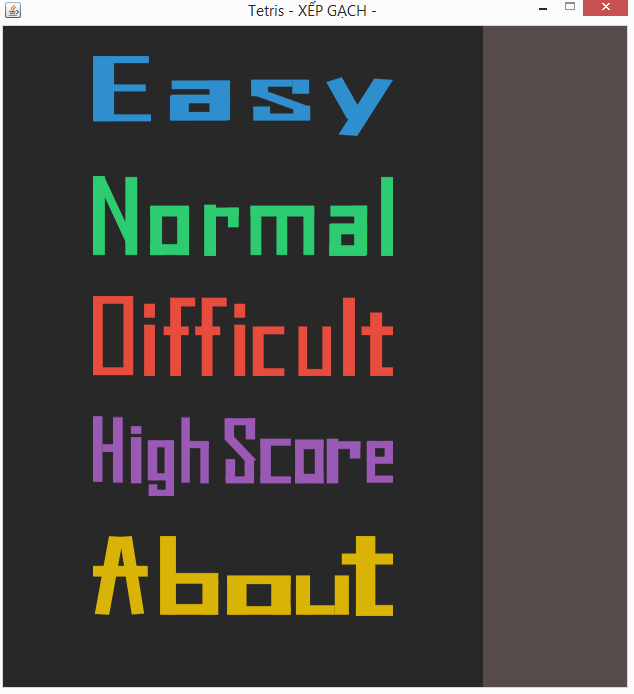
- Sử dụng **FileOutputStream** và **PrintWriter** để ghi dữ liệu ra file và chỉ khi khi số Player được lưu nhỏ hơn 10 người và nếu lớn hơn 10 người thì sẽ thay thế vị trí của người có điểm số thấp nhất

# **III. GIAO DIỆN**

## **Sơ đồ màn hình**

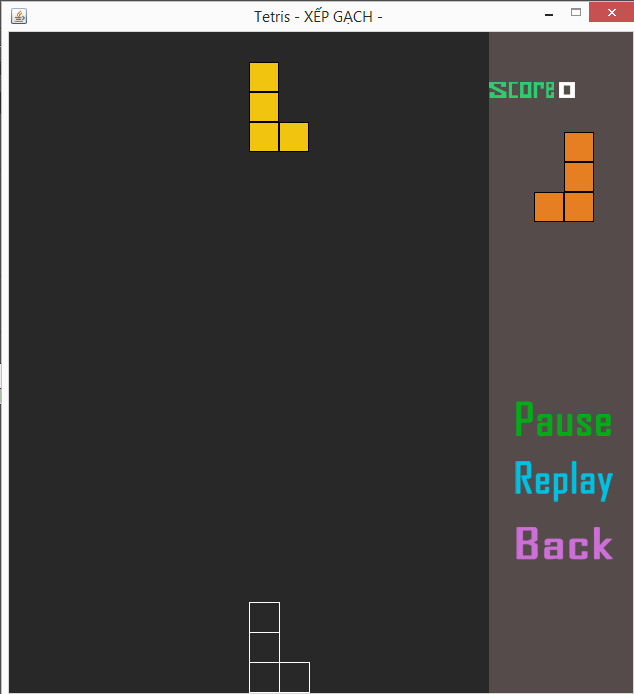


## **Màn hình Menu Game**

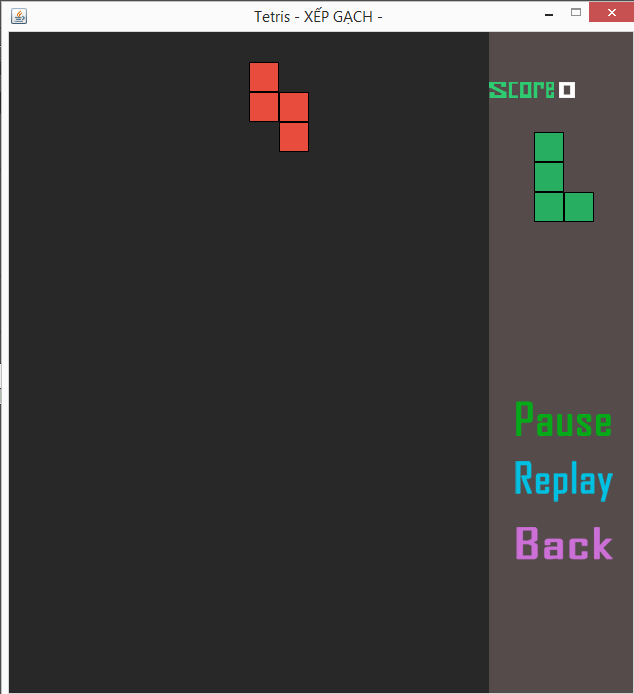


## **Màn hình GamePlay**

**- Easy:**

****

**- Normal và Difficult:**

****

## **Màn hình HighScore**



## **Màn hình About**



# **IV. TỔNG KẾT**

## **Cài đặt thử nghiệm**

**Thử nghiệm thứ nhất:**  
- Cấu hình máy thử nghiệm:

* + - * CPU: Core i5 2.6Ghz (4 CPUs)
      * RAM: 4G
      * VGA: Geforce 630
      * OS: Windows 8 x64

- Kết quả:

* + - * FPS: Duy trì ở mức 58 - 60
      * CPU: Khoảng 45 - 55%
      * RAM khoảng: 40 MB – 60 MB

## **Đánh giá**

- Game sử dụng ổn định duy trì mức FPS 58 – 60, không yêu cầu card màn hình rời và yêu cầu hệ thống phải có cài đặt jdk, và các phần mềm cần thiết khác, khả năng xử lý mức trung bình chưa cao, sử dụng CPU khá nhiều tuy nhiên chiếm bộ nhớ ram khá ít và chưa thấy xuất hiện lỗi.

## **Cấu hình đề nghị**

* + 1. Phần cứng:
       - CPU core dual 1Ghz
       - RAM: 1G
       - VGA: onboard
    2. Phần mềm:
       - OS: Windows 7 trở lên
       - Cài đặt JDK

## **Website và tài liệu tham khảo**

- **Website**: Wikipedia.org, stackoverflow.com, google.com

- **Sách**: C++ programming for Game Module I, C++ programming for Game Module II Copyright 2005 e-Institute, Inc. All rights reserved, **Editor:** Susan Nguyen, **Cover Design:** Adam Hoult