

ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

Họ và tên: Nguyễn Khánh Nguyên

MSSV: 16520846

ĐỒ ÁN LẬP TRÌNH TRỰC QUAN

Giáo viên hướng dẫn: Huỳnh Hồ Thị Mộng Trinh

Phụ lục

Chương 1: Tổng quan đề tài	3
Chương 2: Cơ sở lý thuyết	4
1. Công nghệ sử dụng	4
2. Kiến trúc	4
2.1 Ưu điểm của mô hình này.....	5
Chương 3: Thiết kế xử lý (Hiện thực).....	6
1. Sơ đồ UML	6
1.1 Model	6
1.2 View	7
1.3 Controller	7
2. Thiết kế giao diện.....	8
2.1 Tổng thể giao diện hệ thống	8
2.2 Giao diện Menu chính.....	8
2.3 Giao diện cài đặt	9
2.4 Giao diện about.....	10
3. Sơ lược thiết kế xử lý	10
3.1 Entry point	11
3.2 StateMachine	12
3.3 Scene.....	13
3.4 Renderer	14
4. Thiết kế game.....	15
4.1 Độ khó	15
4.2 Block.....	15
4.3 Map	16
4.4 Thiết kế Map	18
4.5 Kích cỡ banh.....	21
4.6 Độ mạnh banh.....	22
4.7 Powerup.....	22
4.8 Cách tính điểm	23

4.9 Một vài chi tiết khác trong game	24
Chương 4: Cài đặt và cách chơi	25
1. Cài đặt	25
2. Cách chơi.....	25
3. Link.....	25
Chương 5: Tổng kết	26
1. Hạn chế	26
2. Tài liệu tham khảo.....	26

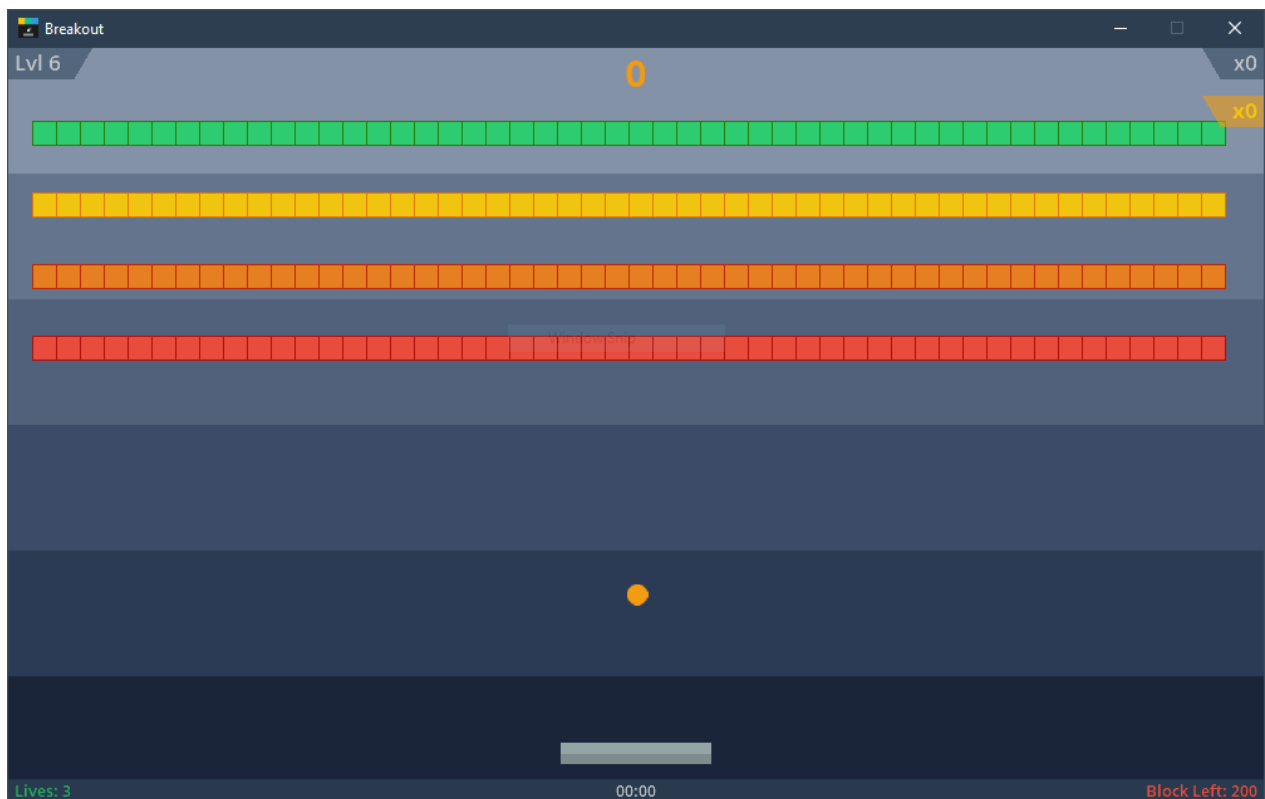
Chương 1: Tổng quan đề tài

Breakout là một game real-time viết bằng framework Monogame. Đây là [video game kinh điển](#) của ngành công nghiệp game khi mà phiên bản đầu tiên được phát hành vào tận lúc 1976.

Trong game, một dãy các khối gạch được xếp thành nhiều lớp, một trái banh sẽ di chuyển trong phạm vi màn hình, khi banh chạm khối gạch, hitpoint của nó bị giảm xuống và banh sẽ nảy ngược lại. Người chơi bị mất một mạng khi trái banh chạm dưới đáy màn hình. Để ngăn chặn việc này, người chơi phải sử dụng một thanh ngang có thể di chuyển trái phải ở gần dưới đáy màn hình để nảy trái banh lại và tiếp tục chơi

Nếu một khối gạch bị vỡ. nó có thể sinh ra một package chứa powerup. Khi powerup chạm phải banh hoặc thanh ngang, thì nó sẽ chỉnh sửa thuộc tính của banh hoặc thanh ngang tùy vào loại powerup nào

Khi mất hết cả 3 mạng game sẽ kết thúc. Người chơi có thể chọn chơi lại hoặc thoát ra ngoài menu.



Chương 2: Cơ sở lý thuyết

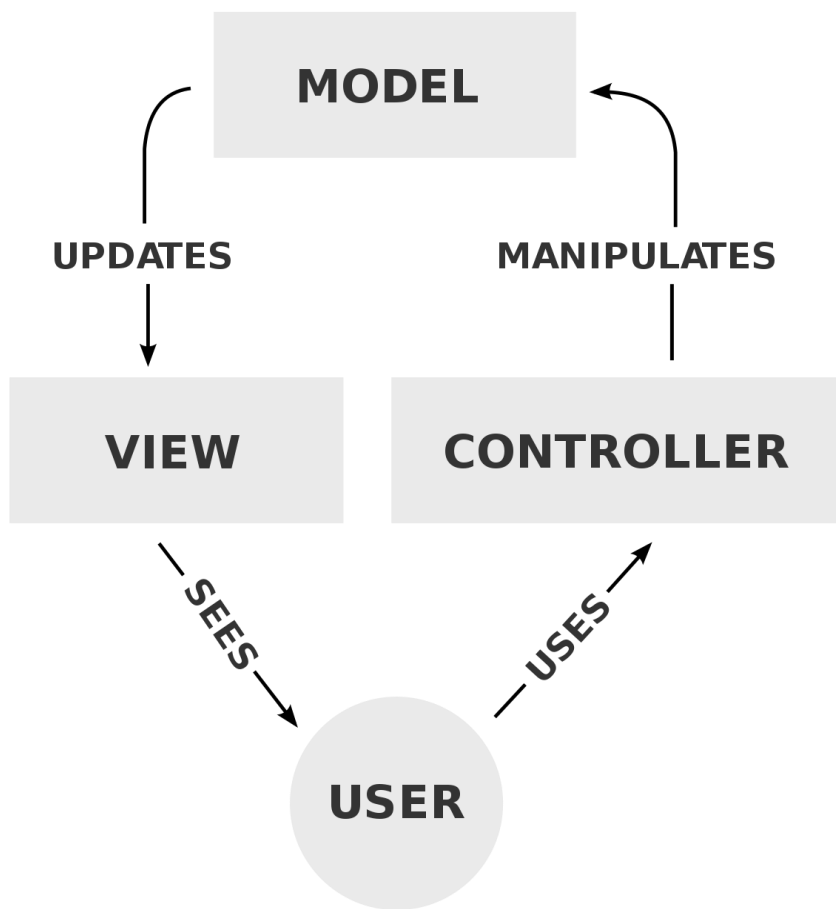
1. Công nghệ sử dụng

- Ngôn ngữ lập trình: C#
- Build tool: Visual Studio 2015
- Framework: [Monogame](#)
- Thư viện sử dụng:
 - [Newtonsoft.Json](#)

2. Kiến trúc

Game được viết theo mô hình [Model-View-Controller](#) (MVC), gồm 3 phần chính sau:

- *Model*: Tất cả mọi luật lệ trong game được xây dựng trong phần **model**, ngoài ra nó còn chứa trạng thái hiện tại của mọi vật thể trong game (như banh, khối gạch, điểm số, background,...). Có thể tưởng tượng nó như là một mô hình mô phỏng thế giới game, ngoại trừ việc không thể nhận input từ người dùng hay xuất dữ liệu ra màn hình
- *View*: Đây là phần xử lý giao diện xuất thông tin ra cho người dùng, trong trường hợp ứng dụng game, phần **view** sẽ render ra màn hình cho người dùng thấy được những gì đang diễn ra trong game. Nó sử dụng phần **model** để biết cần phải vẽ gì, ngay tại đâu và vào lúc nào. Đây là nhiệm vụ duy nhất của phần **view**
- *Controller*: Phần này xử dụng input của người dùng để thao túng phần **model**. Đầu tiên, nó sẽ kiểm tra input đầu vào người dùng, sau đó truy vấn phần view để biết xem đối tượng nào trên màn hình đang được click chuột, và sử dụng thông tin này để ra lệnh cho phần **model** làm phần việc của mình. Nhiều **controller** có thể được sử dụng cùng một lúc, ví dụ như bàn phím, joypad hay một con AI (bot)



2.1 Ưu điểm của mô hình này

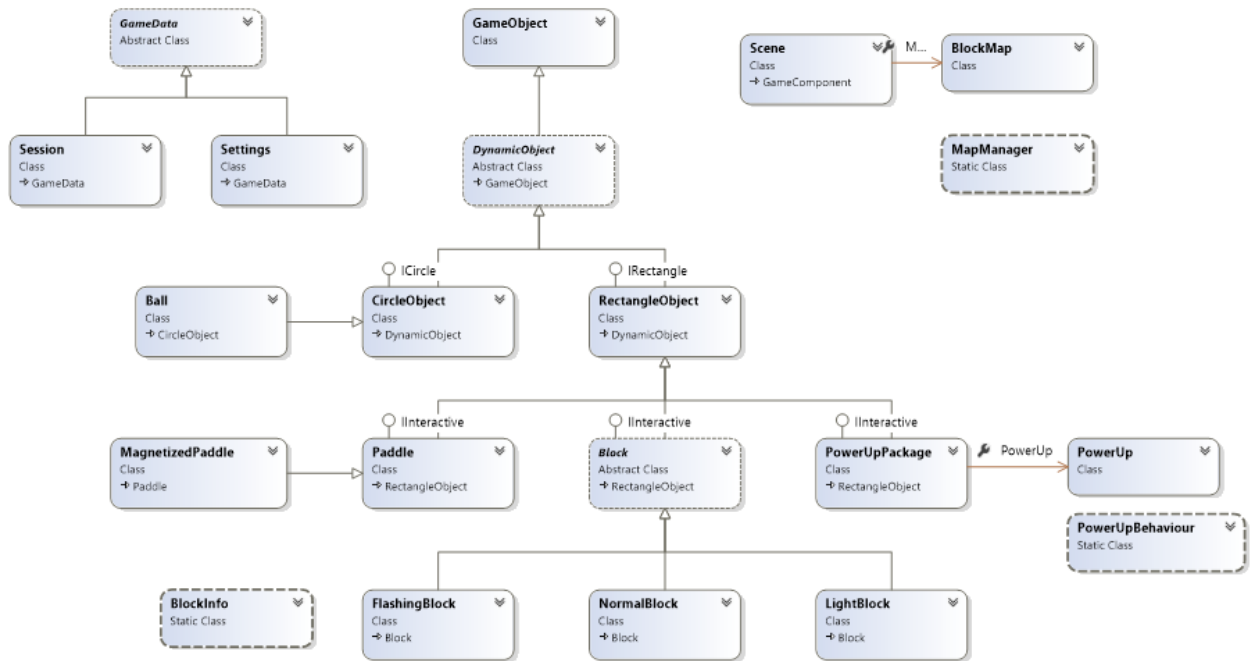
- Phần logic của game gói gọn trong phần model
- Thay đổi phần view sẽ không ảnh hưởng đến logic game
- Có thể hỗ trợ nhiều controller khác nhau

Nhìn chung, việc phân cách thành nhiều phần với nhau và xử lý riêng giúp việc quản lý, debug code dễ dàng hơn. Khi chỉnh sửa một phần, các phần khác sẽ ít bị ảnh hưởng. Nó cũng cho phép nhiều người cùng làm một lúc khi mà mỗi phần trong kiến trúc ít có sự phụ thuộc lẫn nhau

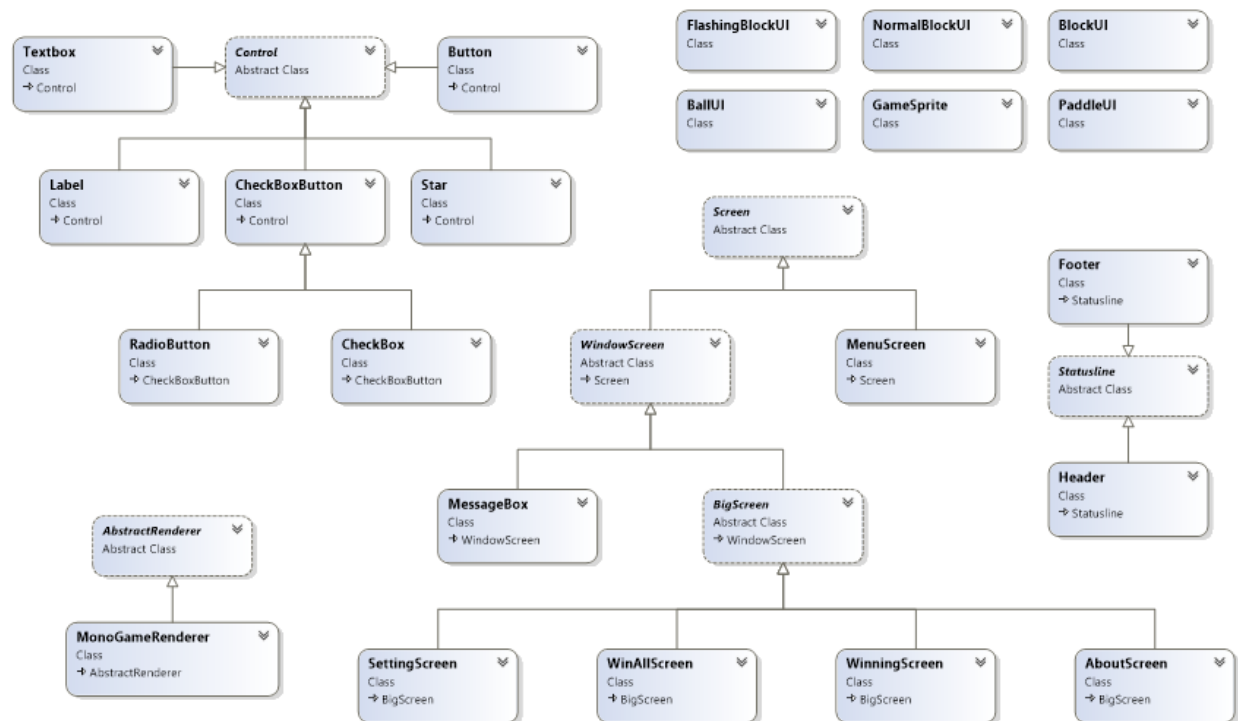
Chương 3: Thiết kế xử lý (Hiện thực)

1. Sơ đồ UML

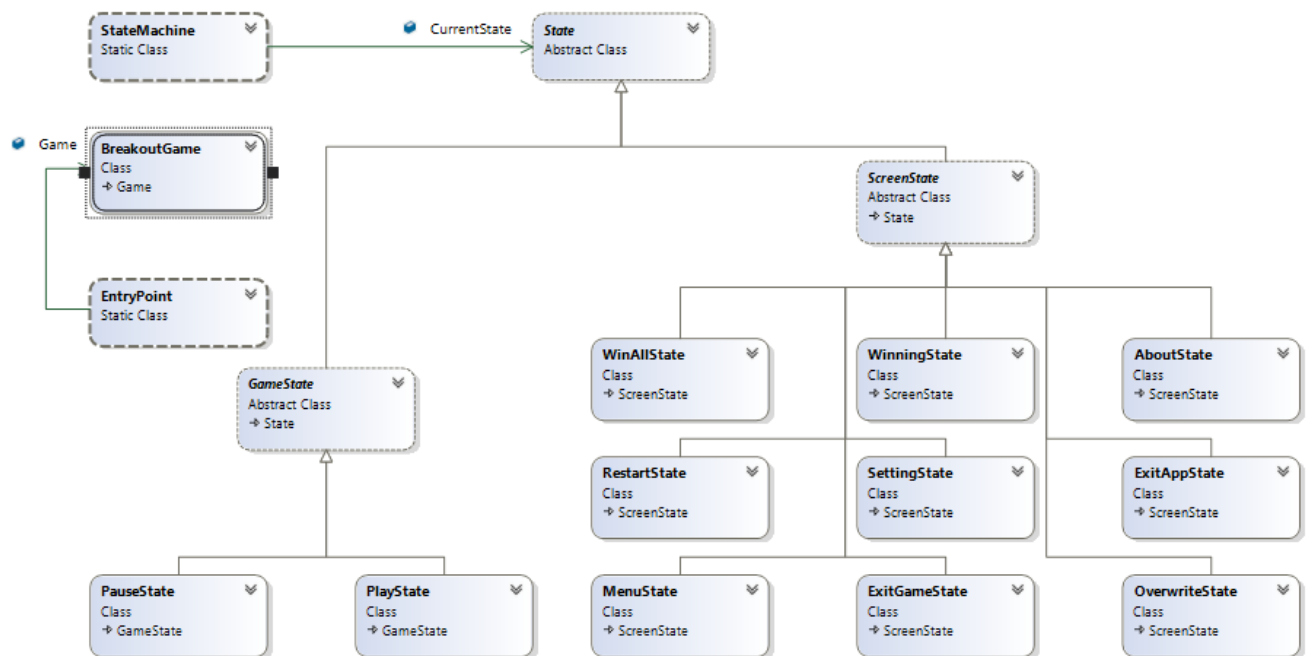
1.1 Model



1.2 View

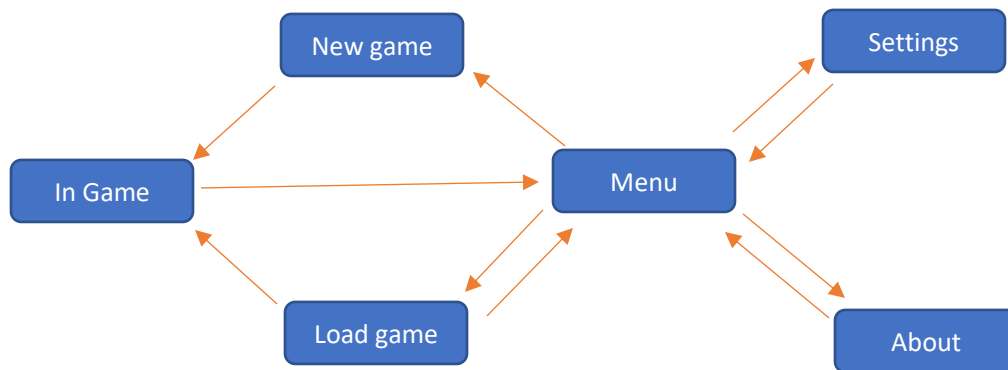


1.3 Controller

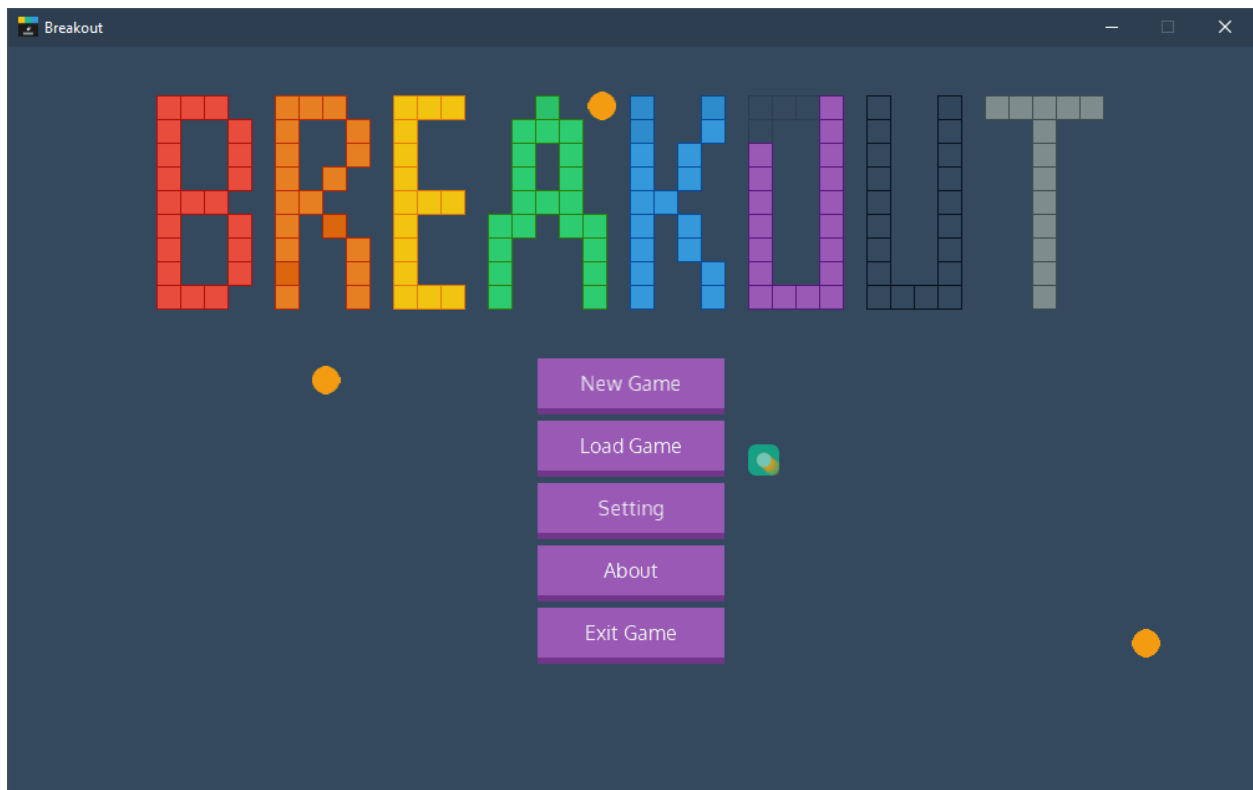


2. Thiết kế giao diện

2.1 Tổng thể giao diện hệ thống



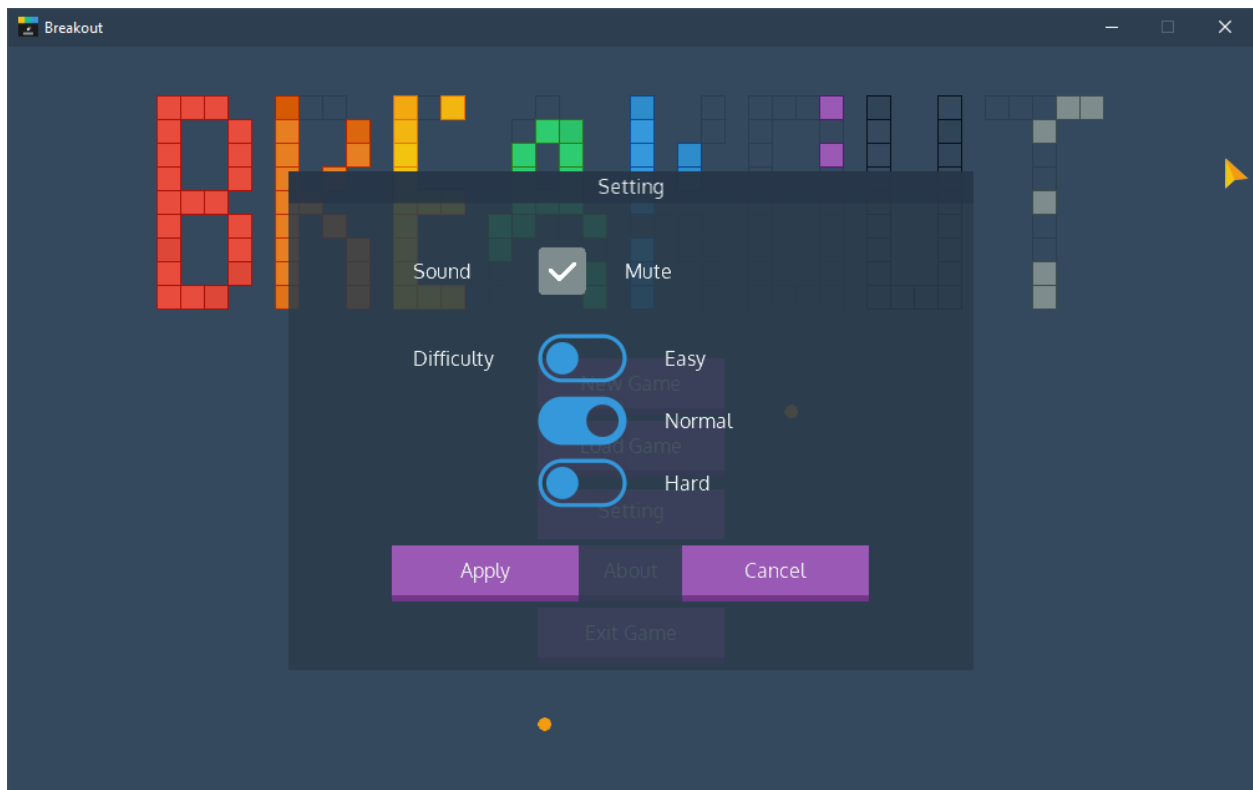
2.2 Giao diện Menu chính



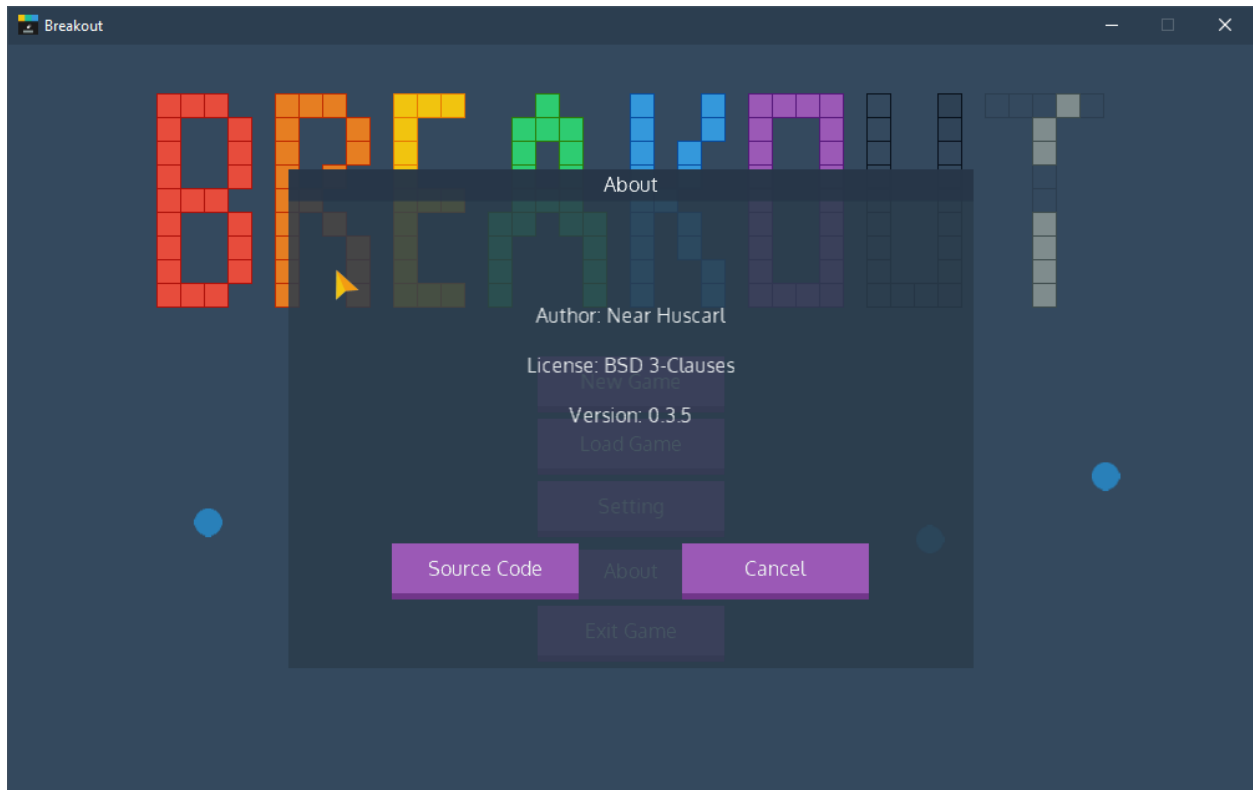
Thành phần giao diện: Một danh sách các button để đưa người dùng đến với các giao diện khác.

Hiệu ứng thêm: Phía sau các button gần trên màn hình là tên của game tạo bởi các khối gạch với nhiều màu khác nhau được lưu dưới dạng [BlockMap](#). Mỗi lần load màn hình menu. Từ 1 đến 3 trái banh được sinh ra và để bắn phá các khối gạch. Mọi thứ chỉ khác trong game ở chỗ không có thanh ngang với điểm số

2.3 Giao diện cài đặt



2.4 Giao diện about

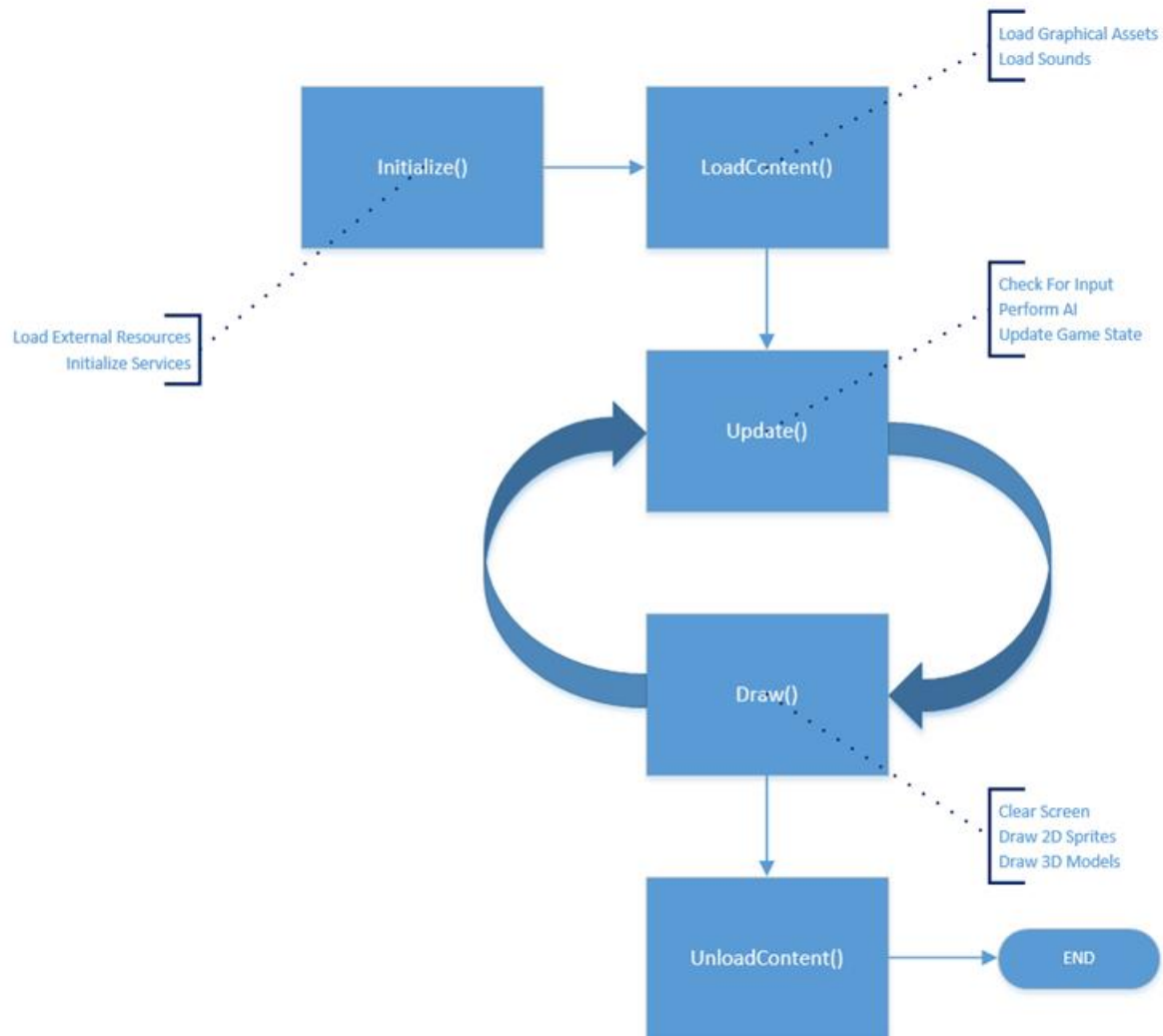


3. Sơ lược thiết kế xử lý

Monogame là một framework. Nhưng nó chỉ cung cấp cho người sử dụng những phương tiện cần thiết đủ để người dùng bắt đầu mà không áp đặt một quy trình cụ thể nào trong việc viết game như Unity. Do đó người dùng có thể thoải mái viết game theo ý mình, việc kiểm soát của người dùng lớn hơn, nhưng đi kèm với đó là việc phải tự mình viết lại rất nhiều thứ. Đó là ưu điểm cũng như hạn chế của framework này

Ở mức độ cơ bản nhất, monogame cung cấp cho người dùng 3 phương thức chính để có thể bắt đầu tạo một game. Ba phương thức này tạo nên một kiến trúc trong game thường biết đến với tên gọi *game loop*

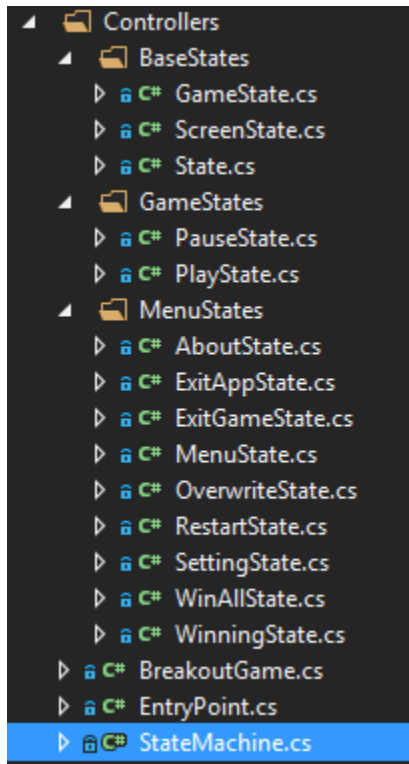
- `Initialize()`: Phương thức này chỉ được gọi một lần duy nhất lúc bắt đầu chạy game để load các thông tin ban đầu cần thiết
 - `UpdateContent()`: Được gọi bởi `Initialize()`. Dùng để load các asset trong game như `Texture2D`, `SpriteFont`,...
- `Update()`: Dùng để update tất cả các thông tin trong game
- `Draw()`: Được gọi sau khi thông tin đã được update, dùng để vẽ ra màn hình



3.1 Entry point

Khi bắt đầu chạy ứng dụng, [EntryPoint](#) là điểm xuất phát đầu tiên khởi tạo lớp [Breakout](#) chính chứa ba phương thức đã kể trên. Trong phần khởi tạo của mình, Lớp Breakout sẽ khởi tạo và nạp tiếp 3 lớp ứng với 3 phần chính trong mô hình MVC

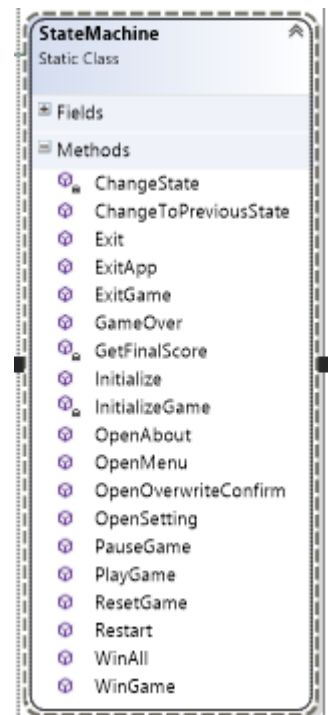
3.2 StateMachine



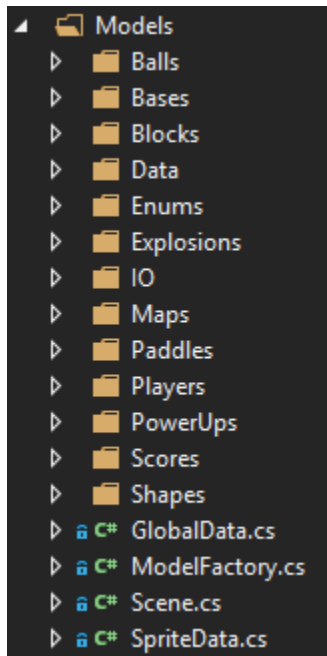
[StateMachine](#) có nhiệm vụ quản lý các [State](#) khác nhau trong game. Mỗi state là một trạng thái game tại một thời điểm. Ví dụ: SettingState là lúc người dùng mở cửa sổ cài đặt, WinningState là lúc người dùng chiến thắng, một màn hình hiện ra để thống kê điểm,...

Có thể hiểu mỗi một state là một giai đoạn khác nhau trong quá trình người dùng tương tác với game. Khi khởi động game thì vào MenuState, bắt đầu chơi game thì vào PlayState, khi xác nhận thoát ra khỏi game thì vào ExitGameState. State sẽ nhận input từ người dùng, xử lý một tình huống cụ thể khác nhau rồi báo lại cho phần model hoặc/và thông báo chuyển sang state khác

Việc chuyển sang State khác được thực hiện trong StateMachine. Nó có nhiệm vụ hỗ trợ chuyển tiếp giữa các state khác nhau trong quá trình chuyển đổi



3.3 Scene



Có thể hiểu [Scene](#) là phần xử lý logic chính trong game. Nó chứa tất cả các thực thể trong game người dùng chơi và tương tác như banh, khối gạch, powerup... Việc update Scene được thực hiện chủ yếu ở [PlayState](#) trong phương thức Update(). Tất cả các State trong game đều phải định nghĩa phương thức này để có thể được gọi lại trong hàm Update() chính bên ngoài của game loop

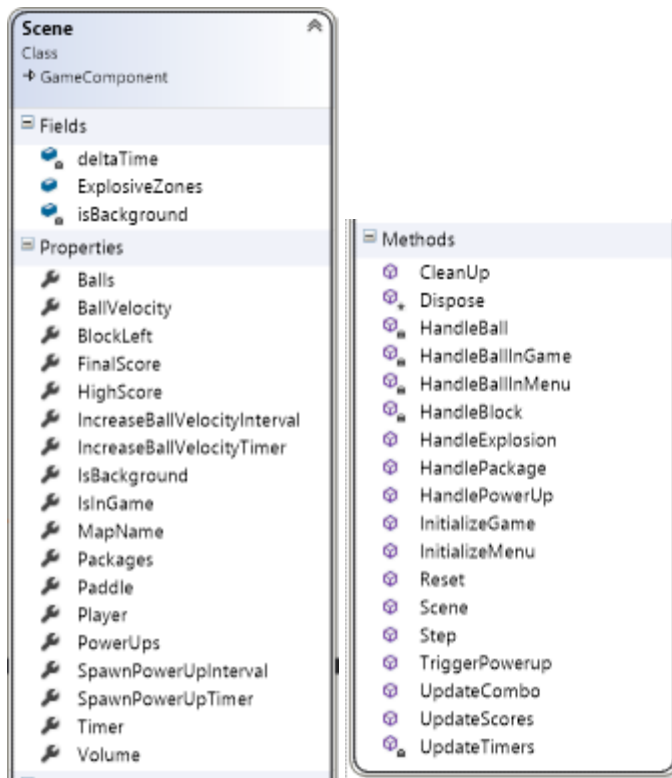
```
protected override void Update(GameTime gameTime)
{
    Elapsed = (float)gameTime.ElapsedGameTime.TotalSeconds;

    StateMachine.CurrentState.Update();

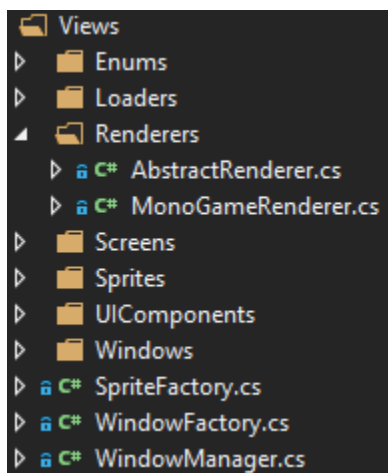
    base.Update(gameTime);
}
```

Mọi quy định trong game khi chơi được miêu tả trong lớp Scene như thời gian để spawn powerup. Nếu mất mạng thì phải làm gì...

Ngoài ra, Scene còn lưu trữ mọi trạng thái của tất cả các vật thể trong game như vị trí hiện tại của các khối gạch với hitpoint là bao nhiêu, vị trí banh và hướng đi, tốc độ thanh ngang, thời gian tính từ lúc bắt đầu, các thống kê hiện tại,... Scene cần khoảng thời gian giữa lần update trước (được gọi là delta time) để update tiếp các đối tượng trong game hay cập nhật timer



3.4 Renderer



[MonoGameRenderer](#) là phần chịu trách nhiệm vẽ ra những gì có trong Scene sử dụng các phương thức do MonoGame cung cấp. Nó được kế thừa từ lớp abstract Renderer vì phần view có thể được tái sử dụng lại ở một hình thức khác như game chơi trong commandline hay webgame chơi trong trình duyệt.

Renderer có 2 phương thức chính là DrawMenu() và DrawGame(). Được gọi bởi các State khác nhau tùy thuộc vào nhiệm vụ của từng State. Trong mỗi State, tương tự như Update(), cần phải có một phương thức Draw() chung để vẽ lại những gì có trong game thông qua Renderer. Hàm Draw() này cũng sẽ được gọi tiếp từ Draw() bên ngoài lớp Game chính.

```
protected override void Draw(GameTime gameTime)
{
    // GraphicsDevice.Clear(Color.CornflowerBlue);

    spriteBatch.Begin();
    stateMachine.CurrentState.Draw(renderer);
    spriteBatch.End();

    base.Draw(gameTime);
}
```

4. Thiết kế game

4.1 Độ khó

Game có 3 độ khó khác nhau. Độ khó có thể tùy chỉnh trong phần cài đặt ngoài menu. Mặc định độ khó game là trung bình

- Dễ: Thanh ngang dài nhất
- Trung bình: Thanh ngang có độ dài vừa
- Khó: Tăng vận tốc trung bình của banh, thanh ngang ngắn nhất

4.2 Block

Có tất cả 3 loại khối gạch. Mỗi loại có 10 màu khác nhau. Các khối gạch trong game đều có các thuộc tính khác nhau tùy thuộc vào loại gạch và màu tương ứng được miêu tả trong [BlockInfo](#)

- [Normal Block](#): Đây là khối gạch có bộ màu sáng. Màu của khối gạch sẽ càng đậm nếu hitpoint càng thấp. Loại này có nhiều hitpoint nhất trong cả 3 loại gạch



- [Light Block](#): Có ít hitpoint hơn, thường đủ để phá vỡ sau mỗi lần chạm banh, màu đậm hơn khối gạch thường



- [Flashing Block](#): Khác với 2 loại gạch đầu, loại gạch này khi bị vỡ sẽ nổ. Tất cả các khối gạch giáp với nó đều bị trừ hitpoint. Loại gạch này nhấp nháy giữa 2 đầu màu là 2 tông đậm nhạt của 2 loại gạch trên

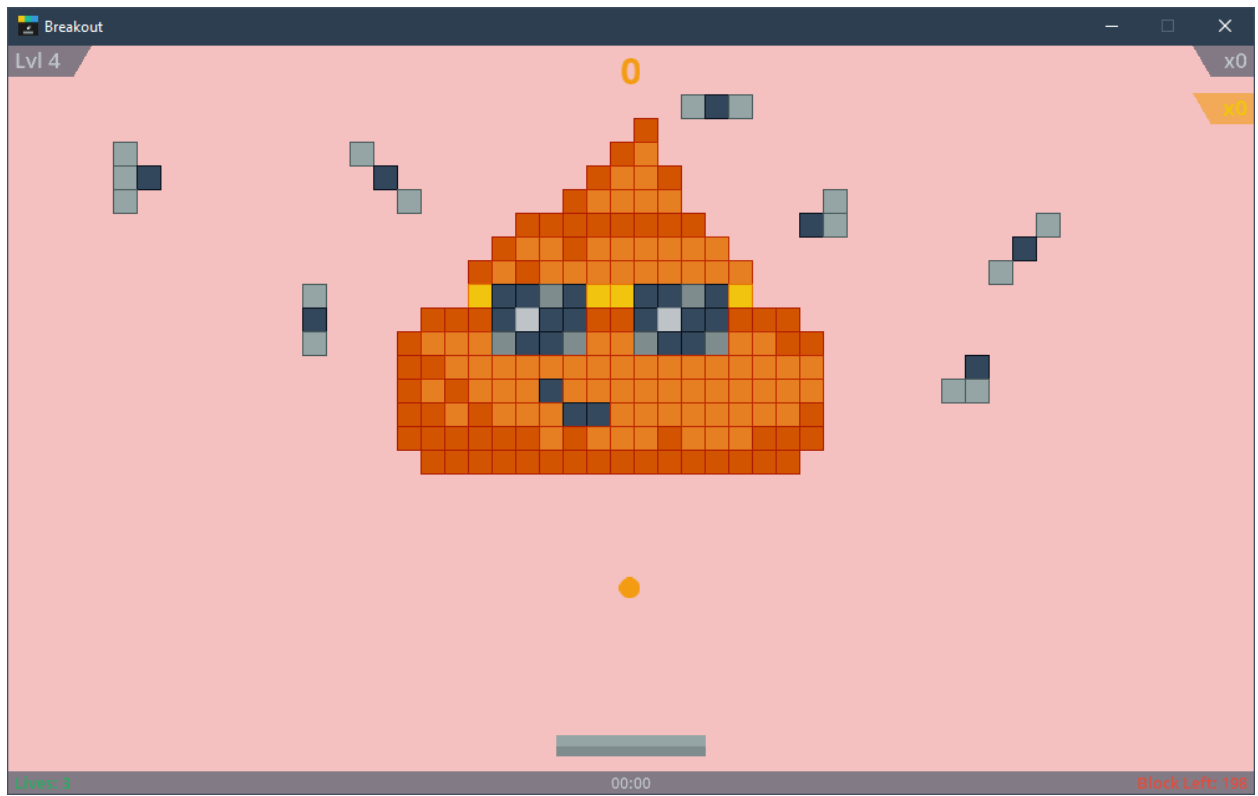
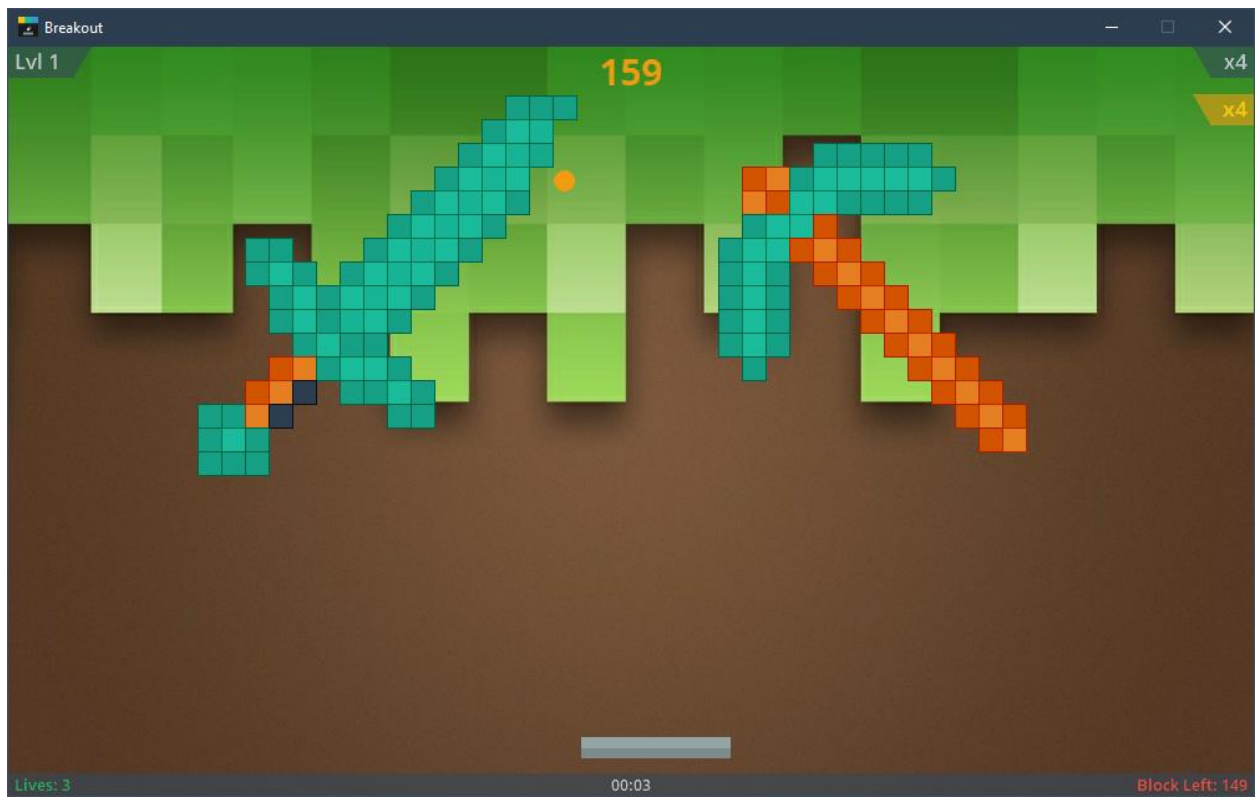
Mỗi loại gạch khi vỡ có thể sinh ra powerup. Tùy vào loại màu khác nhau mà xác suất powerup nào sẽ sinh ra nhiều hơn cũng khác nhau. Ví dụ như gạch đỏ thường sinh ra powerup tăng hitpoint của banh, gạch vàng là powerup tăng tốc độ banh. Tất cả các thông tin trên đều được lưu trong [BlockInfo](#)

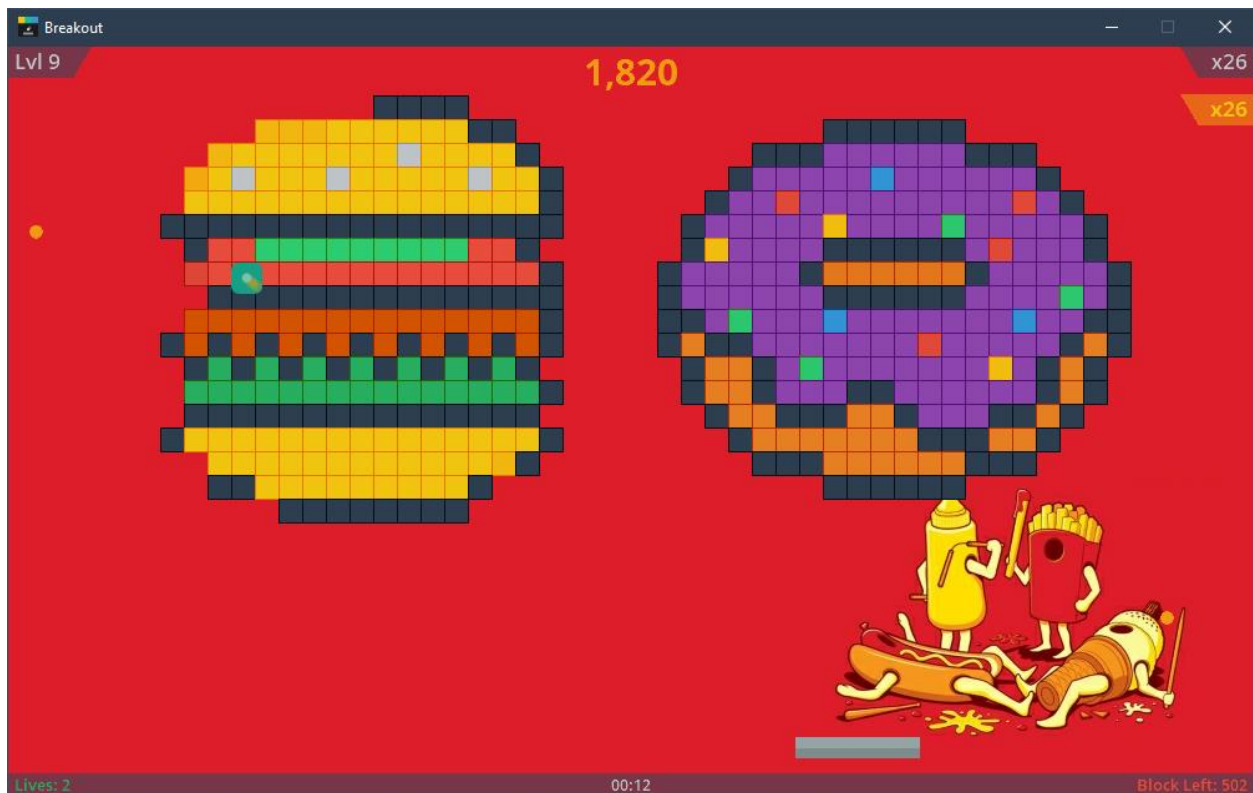
4.3 Map

Một map trong game là một danh sách các gạch khác nhau được sắp xếp theo một trật tự nào đó

Trong game có 10 loại map khác nhau. Được sắp xếp theo thứ tự tổng hitpoint tất cả các block trong map từ ít nhất đến nhiều nhất. Ứng với mỗi map là một background khác nhau liên quan đến map đó

Screenshot một vài map trong game:

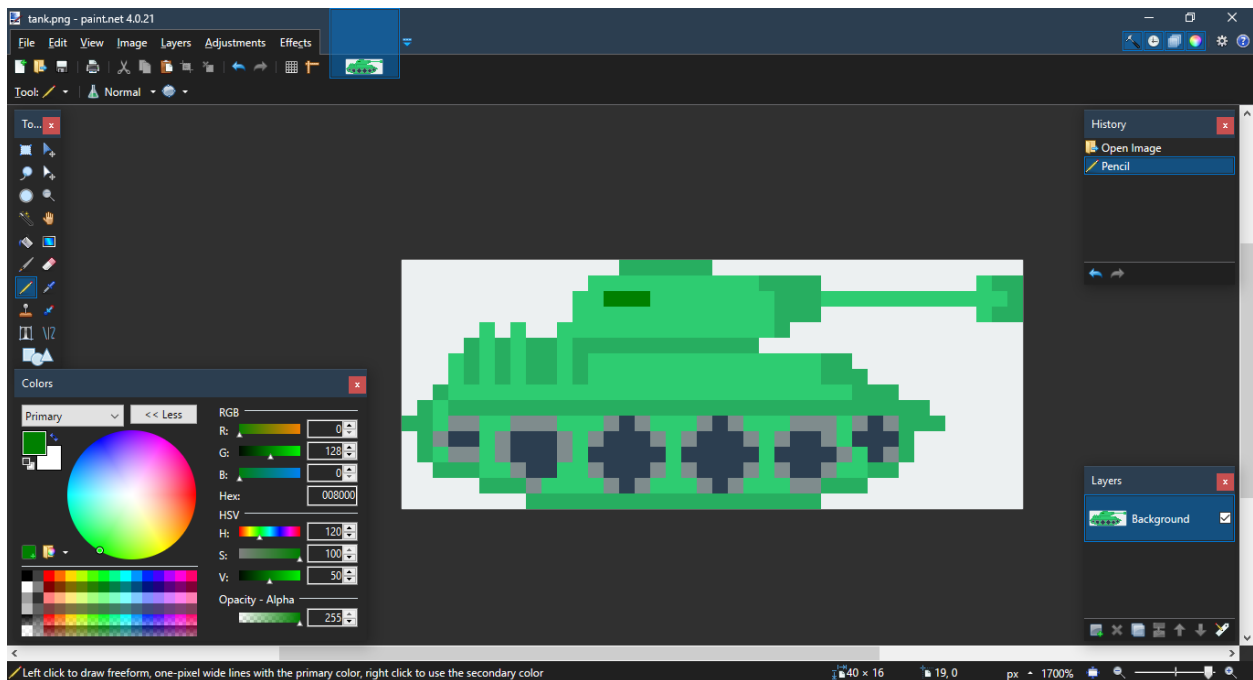




4.4 Thiết kế Map

Việc bố trí các khối gạch trong map bằng danh sách 2 chiều trong C# nếu làm thủ công sẽ [rất mất thời gian](#). Nên sau này, việc tạo ra một map mới sẽ được thực hiện trong một phần mềm chỉnh sửa ảnh chứ không phải viết ngay trong code nữa. Với thay đổi này, thời gian trung bình để tạo map mới giảm xuống tầm 20 phút/map

Đầu tiên, ta sử dụng một phần mềm chỉnh sửa ảnh như [paint.NET](#). Tạo một file ảnh (png hay jpg) mới với kích thước 45 x 18. Kích thước này là vừa đủ để các khối bố trí với nhau một cách hợp lý, không quá gần thanh ngang hay đường viền màn hình



Mỗi một pixel trong hình sẽ đại diện cho một khối gạch. Màu của pixel **phải** nằm trong bộ màu được quy định sẵn. Nếu sử dụng paint.NET, có thể dùng color picker để lấy màu từ [palette này](#). Nếu muốn để trống ở vị trí nào thì phải để màu với độ alpha = 0. (như #00000000 hay #ffffff00)



Flashing



Normal



Light

Khi đã tạo map xong, ta sử dụng [một đoạn script](#) nhỏ để “dịch” từ ảnh đã tạo với thông tin là một dãy các pixel sang file json. Sau đó ta sẽ dùng Monogame Pipeline để đọc các file json này và chuyển sang dạng binary như là một dạng asset trong game

Lưu ý: để chạy được đoạn script này, tất cả các file ảnh được tạo ra phải bỏ trong thư mục maps/. Thư mục này với file script phải được đặt chung một đường dẫn:

```
Windows PowerShell
PS E:\Program Files\Project\Breakout\tool> ls
Directory: E:\Program Files\Project\Breakout\tool
Mode                LastWriteTime         Length Name
----                -
d-----          7/8/2018 1:54 AM            tool > maps
-a-----          7/6/2018 2:50 AM          14959 palette.png
-a-----          7/8/2018 12:58 AM           4471 pixel2char.py

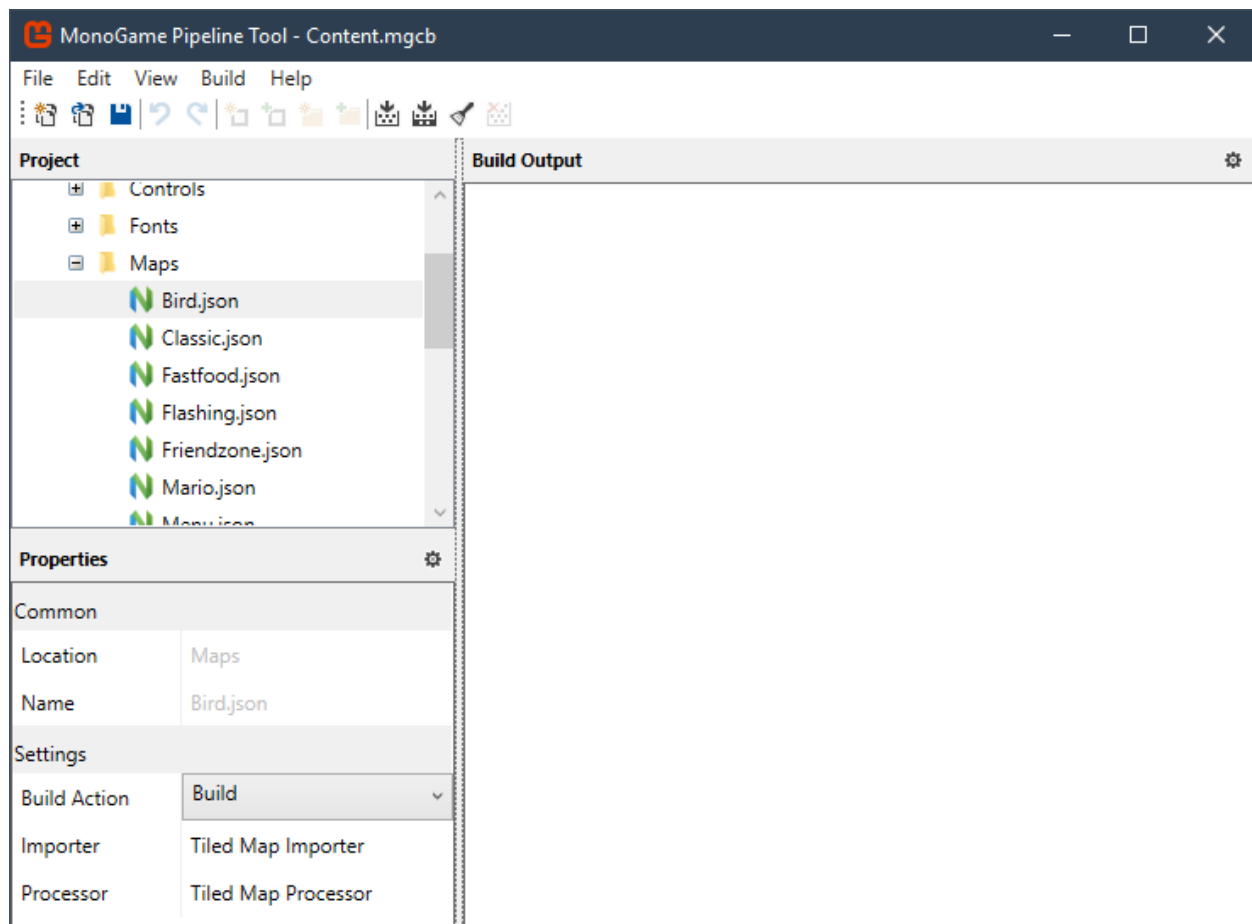
PS E:\Program Files\Project\Breakout\tool> ls maps
Directory: E:\Program Files\Project\Breakout\tool\maps
Mode                LastWriteTime         Length Name
----                -
-a-----          7/6/2018 5:37 AM           442 Bird.png
-a-----          7/7/2018 5:54 PM           217 Classic.png
-a-----          7/6/2018 6:49 AM           535 Fastfood.png
-a-----          7/8/2018 12:55 AM           165 Flashing.png
-a-----          7/6/2018 3:09 AM           380 Friendzone.png
-a-----          7/6/2018 4:38 AM           387 Mario.png
-a-----          7/7/2018 3:49 AM           318 Menu.png
-a-----          5/14/2018 5:27 PM           271 Menu_bg.png
-a-----          7/6/2018 5:56 AM           378 Minecraft.png
-a-----          7/6/2018 4:27 AM           429 Minions.png
-a-----          5/13/2018 4:22 PM           174 Quick.png
-a-----          5/20/2018 6:19 PM           166 Quick2.png
-a-----          7/6/2018 5:03 AM           415 Shit.png
-a-----          7/6/2018 1:53 AM           378 Tank.png
-a-----          5/22/2018 12:53 PM           283 Watermelon.png
-a-----          7/6/2018 6:21 AM           402 Whale.png

PS E:\Program Files\Project\Breakout\tool> python pixel2char.py
pixel2char.py> convert to json map file from Bird.png
pixel2char.py> convert to json map file from Classic.png
pixel2char.py> convert to json map file from Fastfood.png
pixel2char.py> convert to json map file from Flashing.png
pixel2char.py> convert to json map file from Friendzone.png
pixel2char.py> convert to json map file from Mario.png
pixel2char.py> convert to json map file from Menu.png
pixel2char.py> convert to json map file from Minecraft.png
pixel2char.py> convert to json map file from Minions.png
pixel2char.py> convert to json map file from Quick.png
pixel2char.py> convert to json map file from Quick2.png
pixel2char.py> convert to json map file from Shit.png
pixel2char.py> convert to json map file from Tank.png
pixel2char.py> convert to json map file from Watermelon.png
pixel2char.py> convert to json map file from Whale.png
PS E:\Program Files\Project\Breakout\tool>
```

Bên cạnh việc phải có Monogame. Để có thể sử dụng tool này, trên máy cần phải có:

- Python 3
- Pillow (pip install pillow): thư viện xử lý ảnh của python

Sau khi có được các file json. Ta có thể add nó vào asset game bằng công cụ Pipeline của Mono



Chú ý: Ở mỗi lần build project Breakout.Core. Đã có một sự kiện build event sẽ tự động chạy đoạn script kể trên và di chuyển hết tất cả các file json mới tạo được vào thư mục Maps/. Nên nếu muốn chỉnh sửa map, thì sau khi chỉnh lại trong hình, vào Build -> Rebuild, nó sẽ tự động cập nhật lại map trong game

4.5 Kích cỡ banh

Banh có 3 loại kích cỡ khác nhau: Nhỏ, trung bình và lớn. Kích cỡ banh có thể tăng lên hoặc giảm đi do powerup. Tốc độ của banh tỉ lệ nghịch với kích cỡ: Banh càng lớn thì vận tốc càng nhỏ đi

4.6 Độ mạnh banh












Giống với kích cỡ banh. Mỗi banh có 3 độ mạnh khác nhau, phân biệt bởi màu của từng banh:

-  Yếu
-  Trung bình
-  Mạnh

Banh càng mạnh thì càng lấy nhiều hitpoint từ gạch khi cả 2 chạm nhau. Độ mạnh của banh cũng có thể thay đổi thông qua powerup

4.7 Powerup

Hiện tại ở phiên bản 0.3.6. Trong game có 11 loại powerup khác nhau. Mỗi powerup có chỉnh sửa các thuộc tính của banh hay thanh ngang, mà dựa vào đó, người dùng có thể tận dụng vào lợi thế của mình ở những thời điểm thích hợp

-  Nhân đôi số banh hiện có trên màn hình
-  Nhân ba số banh hiện có trên màn hình
-  Tăng vận tốc của tất cả các banh trên màn hình lên một nấc
-  Giảm vận tốc của tất cả các banh trên màn hình xuống một nấc
-  Tăng sức mạnh của tất cả các banh trên màn hình lên một nấc
-  Giảm sức mạnh của tất cả các banh trên màn hình xuống một nấc
-  Tăng kích cỡ mạnh của tất cả các banh trên màn hình lên một nấc
-  Giảm kích cỡ mạnh của tất cả các banh trên màn hình xuống một nấc
-  Tăng độ dài thanh ngang
-  Giảm độ dài thanh ngang
-  Biến thanh ngang thành nam châm. Khi banh chạm vào thanh ngang sẽ bị dính lại cho đến khi người dùng bấm phím thả (space)

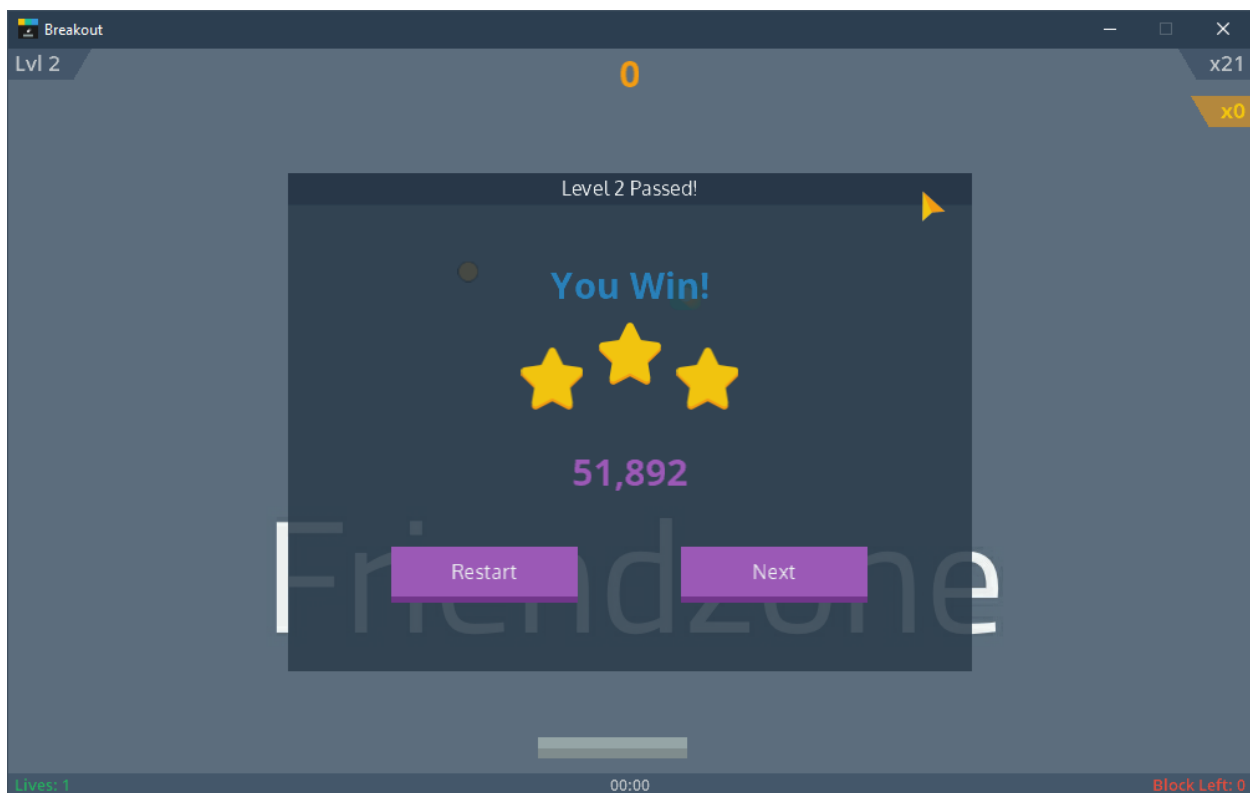
Mỗi powerup được sinh ra ngẫu nhiên bởi các khối gạch khi bị phá vỡ. Khi sinh ra, nó sẽ đi xuống dưới đáy màn hình và biến mất. Để có thể sử dụng powerup. Thanh ngang hoặc một trong số các bánh trên màn hình phải chạm vào nó. Thời gian tác dụng của mỗi loại powerup là 20 giây

4.8 Cách tính điểm

Tổng điểm mỗi vòng = điểm sàn + điểm thưởng thời gian + điểm combo

- Điểm sàn: Mỗi một lần chạm bánh vào gạch. Điểm sẽ tăng thêm $\text{hitpoint mất} * 7$
- Điểm thưởng thời gian: Thời gian sau mỗi màn càng thấp thì điểm càng cao
- Điểm combo: combo là số lần bánh chạm khối gạch liên tiếp mà không bị mất mạng nào. Combo trong một màn càng cao thì điểm combo càng cao

Tùy vào điểm cao hay thấp mà người chơi nhận được bao nhiêu sao



4.9 Một vài chi tiết khác trong game

- Sau mỗi 30s tốc độ banh trung bình sẽ tăng lên một ít. Điều đó sẽ khiến cho màn chơi càng ngày càng khó nếu người chơi để càng lâu
- Sau mỗi 15s. Một powerup ngẫu nhiên sẽ được sinh ra từ phía trên màn hình rơi xuống. Lý do là vì khi trên màn hình chỉ còn một vài khối gạch, xác suất banh chạm vào đúng các khối gạch còn lại đó sẽ giảm đi đáng kể, nên phải có powerup để hỗ trợ người chơi
- Vận tốc banh sẽ bằng vận tốc trước khi đập thanh ngang cộng với vận tốc tương đối của thanh ngang. Sau đó vận tốc này của banh sẽ quay dần trở lại vận tốc gốc (vận tốc trung bình)
- Mỗi loại khối gạch khác nhau sẽ có xu hướng sinh ra một hoặc hai loại powerup hơn phần còn lại. Và powerup đó sẽ có màu trùng với màu khối gạch đó
- Powerup sau khi sinh ra từ block phải chờ khoảng một khoảng thời gian (0.5s) trước khi có thể sử dụng khi chạm vào banh. Lý do là khi banh phá vỡ khối gạch sinh ra powerup, do powerup spawn ở ngay vị trí khối gạch vỡ thì khả năng cao là banh cũng sẽ “đón” luôn phần powerup đó nên cần để timeout

Chương 4: Cài đặt và cách chơi

1. Cài đặt

- Download Breakout.zip trên [github](#)
- Giải nén
- Chơi

2. Cách chơi



3. Link

- Source code: <https://github.com/NearHuscarl/Breakout>
- Demo video: <https://www.dropbox.com/s/15lypif1s7rerdg/Demo.mp4?dl=0>

Chương 5: Tổng kết

1. Hạn chế

Do hạn chế về mặt thời gian, nên đồ án phải kết thúc sớm, xét về tính năng, đồ án mới hoàn thành được khoảng 70% ước tính ban đầu cộng thêm việc các tính năng cuối được thêm vào trong giai đoạn cuối cùng khá gấp nên còn vài thứ chưa được hiệu chỉnh và có nguy cơ phát sinh ra lỗi mới. Sau đây là những phần hạn chế cho đến thời điểm bây giờ:

- Kiến trúc: MVC thích hợp hơn cho những game turn-based nên việc ứng dụng kiến trúc này vào một game real-time như game này có một vài chỗ còn khập khểnh và thiếu hiệu quả
- Thiếu giao diện High Score của người chơi sau khi kết thúc game
- Thiết kế ngưỡng điểm để cho bao nhiêu sao còn sơ xài và chưa được test đầy đủ hoàn toàn
- Chưa test kỹ càng độ khó của game và liên hệ của nó với điểm số nhận được trong game
- Màn hình game giới hạn trong kích cỡ 1000 x 600 do code được viết không tính đến tình huống người dùng thay đổi kích cỡ màn hình
- Lỗi tắt cả hiệu ứng âm thanh chơi đồng loạt sau khi unpause game
- Các phương thức dò va chạm trong CircleObject được copy từ RectangleObject mà chưa được viết lại. Do đó mỗi trái banh về bản chất là hình vuông cho dù có sprite là hình tròn

2. Tài liệu tham khảo

https://www.youtube.com/watch?v=r5dM0_J7KuY&list=PLV27bZtgVIJqoeHrQg6Mt_S1-Fvq_zzGZ

<http://www.koonsolo.com/news/model-view-controller-for-games/>

<https://stackoverflow.com/a/555570/9449426>

<https://codeincomplete.com/posts/javascript-breakout/>

http://xbox.create.msdn.com/en-US/education/catalog/sample/game_state_management

<https://stackoverflow.com/questions/25100029/how-can-i-update-my-gui-score-one-point-at-a-time>