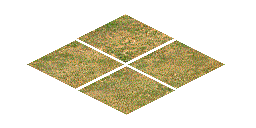
**Việc vẽ ra màn hình nên trình bày trong phần mô tả về XNA, ở đây măc định như đã viết về cách thức XNA vẽ các component thế nào rồi (Game.Components property, GameComponent class, DrawableComponent class, SpriteBatch class, Draw() …)**

1. Xây dựng bản đồ vùng chiến sự.

* Vấn đề :

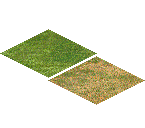
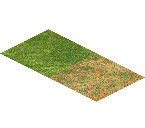
Bản đồ vùng chiến sự là nền để sắp đặt và bố trí các Sprite lên đó, bao gồm Terrain, Resource center, Unit và Structrure. Bản đồ có kích thước lớn hơn nhiều so với vùng nhìn của cửa sổ (Viewport) cho phép người dùng kéo vùng nhìn của cửa sổ trên khắp bản đồ để quan sát cục diện trận chiến trong Game, đồng thời bản đồ cần phải có tính linh động cao trong việc thay đổi kích thước. **Vậy, vấn đề ở đây là xây dựng một bản đồ chiến sự lớn, có khả năng tùy biến về kích thước**.Có hai giải pháp được nêu ra trong việc xây dựng bản đồ vùng chiến sự.

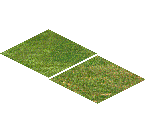
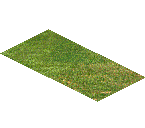
* Giải pháp:
  + Giải pháp thứ nhất là Load một ảnh thật lớn và vẽ ra như một bản đồ chiến sự trên màn hình Game. Nhưng khuyết điểm của giải pháp này là XNA framework chỉ cho phép load hình ảnh có kích thước tối đa là **2024 x 2024 px**, như vậy nếu load một ảnh thật lớn cũng chỉ có thể load được kích thước 2024px cho mỗi cạnh, điều này đã làm giảm tính tùy biến về kích thước của bản đồ chiến sự. Tuy vậy nếu chấp nhận giải pháp này có thể làm giảm ảnh hưởng của khuyết điểm trên bằng cách vẽ hình đã load với kích thước khác nhau, tuy nhiên cách này sẽ làm giảm chất lượng của ảnh được vẽ, thử tưởng tượng ta có một ảnh kích thước 128 x 128px thử resize nó thành 256 x 256 px, hiển nhiên hình ảnh sẽ không còn đẹp nữa. Cần thiết phải có một giải pháp tối ưu hơn.
  + Giải pháp thứ hai được nêu ra ở đây, thử xem bản đồ vùng chiến sự là một tập hợp nhiều ảnh nền nhỏ ghép lại(được gọi là các **Cell**), kích thước bản đồ tùy thuộc vào số lượng các mảnh ghép vì thế, khi tăng số lượng các mảnh ghép, ta được một bản đồ có kích thước to hơn. Việc vẽ bản đồ được thực hiện bằng phép vẽ nối tiếp các mãnh ghép trên Viewport.



Như vậy, các bước thực hiện cho giải pháp này là

* + - Xác định tính đúng đắn của các mãnh ghép(sự chuyển tiếp giữa các vùng đất trên bản đồ).

* + - Từ đó xây dựng được ma trận các mảnh ghép.
    - Xác định vị trí của mỗi mãnh ghép bản đồ, sao cho khi vẽ ra, các mãnh ghép nối tiếp nhau.
    - Vẽ mảnh ghép: thực hiện thao tác vẽ chỉ cho các mãnh ghép nằm trong vùng Viewport và không thực hiện vẽ các mãnh ghép nằm ngoài cùng Viewport vì điều này là không cần thiết.

1. Xác định cách vẽ bản đồ vùng chiến sự với ma trận Cell, sao cho chỉ giới hạn vẽ trong vùng Viewport nhằm làm giảm chi phí.

* Vấn đề:

Giả sử đã có ma trận các Cell với tính đúng đắn và vị trí vẽ thích hợp vào bản đồ, bây giờ phải vẽ các Cell. Ta không thể lặp toàn bộ ma trận và vẽ ra tất cả các Cell trong ma trận, điều này là không cần thiết, vì các Cell không nằm trong Viewport không cần phải được vẽ ra. Không những thế, chi phí để thực hiện vòng lặp trên ma trận Cell sẽ rất lớn, vì nếu bản đồ chiến sự có kích thước rất lớn thì đồng nghĩa với việc số lượng Cell cần lặp và vẽ trên bản đồ chiến sự là rất nhiều. **Như vậy, vấn đề ở đây là làm sao giới hạn lại những Cell nằm trong Viewport để vẽ ra hay nói cách khác là tìm ra index của các Cell trong ma trận sao cho chúng đảm bảo Cell nằm trong Viewport.**

* Giảipháp:
  + Để giải quyết vấn đề trên ta chia thành hai trường hợp
    - Với bản đồ có các Cell dạng hình chữ nhật hoặc hình vuông

*i1 i2*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| *j1* | ***A*** |  |  | ***B*** |  |
|  |  |  |  |  |  |
| *j2* | ***C*** |  |  | ***D*** |  |
|  |  |  |  |  |  |

Giả sử có vùng bản đồ chiến sự như trên, bao gồm nhiều Cell hình vuông ghép lại. Viewport là hình chữ ABCD.

i,j là index của ma trận các Cell

Xác định lần lượt các đỉnh A,B,C,D nằm ở Cell nào trong ma trận. A(i1,j1) B(i2,j1) C(i1,j2) D(i2,j2) như vậy các Cell cần vẽ có **i1<=i<= i2, j1<=j<=j2** (việc xác định D là không cần thiết, với A,B,C đã đủ xác định được i1,i2,j1,j2)

* + - Với bản đồ có các Cell dạng hình thoi

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | j1 | i1 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | **A** |  |  | **B** |  |  |  |
| j2 |  |  |  |  |  |  |  |  | i2 |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | **C** |  |  | **D** |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Bản đồ là một tập các Cell dạng hình thoi ghép lại.

Thực hiện tương tự như dạng bản đồ hình vuông nhưng cách xác định i và j khác do cách biểu diễn hệ trục là khác

Tóm lại giải pháp chủ yếu dựa trên việc xác định index giới hạn trong ma trận Cell để chọn ra các Cell nằm trong Viewport và vẽ.

1. Thể hiện bản đồ vào vùng Viewport

* Vấn đề:

Sau khi xác định được các Cell cần vẽ, việc tiếp theo phảo làm là thể hiện chúng vào Viewport.

* Giải pháp.:
  + Giải pháp được thực hiện
    - Lớp Map được cài đặt là một lớp kế thừa từ lớp DrawableGameComponents của XNA framework, DrawableGameComponents là lớp tương tự lớp GameComponents nhưng sẽ thực hiện chức năng vẽ vào màn hình bằng cách override Draw(). Lớp này chứa ma trận các Cell.
    - Cài đặt hàm Draw() của Map chỉ vẽ các Cell được giới hạn bởi index đã xác định trước.
    - Add đối tượng Map vào Game.Components và vẽ ra các Cell trong giới hạn index đã xác định trước.
    - Tùy vào loại Map mà thiết kế các lớp kế thừa cụ thể.

1. Thể hiện các thành phần của Game và vẽ hình ảnh các thành phần này.

* Vấn đề:

Các thành phần trong Game bao gồm Unit(quân lính), Structure(công trình), Terrain(địa vật), Resource Center(mỏ tài nguyên)... Sau khi đã vẽ từ ma trận Cell ra màn hình một khung cảnh nền cho bản đồ chiến sự, các thành này sẽ được vẽ lên màn hình để thể hiện trực quan cho người chơi cục diện chiến sự.

* Giải pháp:
  + Giải pháp như sau:
    - Đầu tiên, cài đặt lớp Sprite kế thừa từ lớp DrawableGameComponents của XNA framework, DrawableGameComponents là lớp tương tự lớp GameComponents nhưng sẽ thực hiện chức năng vẽ vào màn hình bằng cách override Draw(). Các thành phần của Game đều kế thừa từ lớp Sprite và tùy vào từng thành phần cụ thể, sẽ được cài đặt thành các lớp Unit, Structure … nhưng chung quy lại, hàm Draw() của các thành phần này đều override trên hàm Draw() của Sprite. Để thực hiện vẽ, các thành phần được add vào thuộc tính Components của đối tượng Game dưới dạng Sprite và vẽ ra theo thứ tự trước sau như khi add vào.

+Sprite(kế thừa DrawableGameComponents): cài đặt lại về Draw()

+Unit,Structure… : kế thừa từ Sprite

+Add Unit,Structure… vào Components của lớp Game dưới dạng Sprite

* + - Vấn đề đặt ra ở đây là làm sao vẽ các thành phần này tối ưu nhất(nghĩa là không tốn nhiều tài nguyên phần cứng quá nhiều). Cụ thể, khi được vẽ ra màn hình, hiển nhiên các thành phần đã nằm trong Components của đối tượng Game sẽ được vẽ hết ra ngoài thông qua cài đặt hàm override Draw() của lớp Sprite. Nhưng vấn đề nãy sinh là đâu cần phải vẽ hết các thành phần nếu chúng không cần thiết phải thể hiện, đó là các thành phần không nằm trong Viewport sẽ không cần thiết phải vẽ. Như thế trong cài đặt hàm Draw() của Sprite, cần thiết phải kiểm tra Sprite có nằm trong Viewport hay không.
    - Để thực hiện điều này, thực hiện như sau, xem toàn bộ bản đồ là một hình chữ nhật lớn dù là loại hình thoi hoặc hình vuông. Gốc tọa độ nằm ở gốc trái trên của bản đồ, trục tọa độ là cạnh trên (Ox) và cạnh trái (Oy), ta được một hệ trục tọa độ toàn cục. Vị trí của góc trái trên Viewport chỉ là một điểm nằm trên hệ trục đó, nhưng Game component trong XNA frame work lại xem hệ trục tọa độ khi vẽ luôn có gốc là góc trái trên Viewport, trục tọa độ là hai cạnh trên và trái, như vậy Viewport là một hệ trục tọa độ nhỏ trong hệ trục tọa độ toàn cục. Lớp Sprite được cài đặt thuộc tính Position, thể hiện vị trí của đối tượng Sprite trong hệ trục tọa độ toàn cục, để kiểm tra có vẽ đối tượng Sprite nào đó ra màn hình Viewport không, ta chỉ cầsn xét trong hệ tọa độ Viewport, Position của Sprite có nằm trong vùng Viewport hay không.

O(0,0)

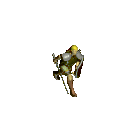
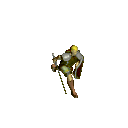
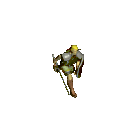
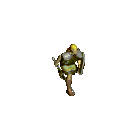
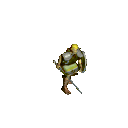
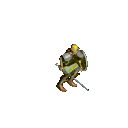
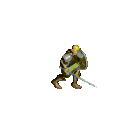
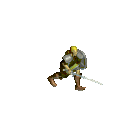
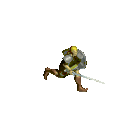
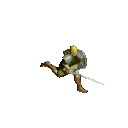
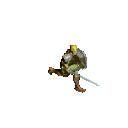
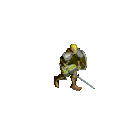
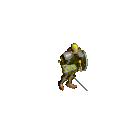
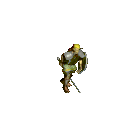
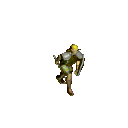
I(x,y) hoặc (O’(0,0))

* + - Hiển nhiên thuộc tính Position phải đảm bảo cho Sprite không được nằm ngoài vùng lát nền của bản đồ dạng hình thoi.

1. Thể hiện trạng thái hoạt động của các Sprite trong Game.

* Vấn đề:

Mỗi Sprite trong Game có một tập những trạng thái hoạt động nhất định bao gồm những hoạt động cơ bản sau : IDLE, WALK, ATTACK, DIE… được gọi tắt là trạng thái. Mỗi trạng thái, cùng với hướng hoạt động của trạng thái đó, bao gồm một loạt các hình ảnh chuyển đổi liên tiếp, mô phỏng từng bước hoạt động của trạng thái đó. Hình ảnh sau là 15 ảnh mô tả một Sprite cụ thể là một quân lính đang di chuyển(WALK)



**Như vậy, để một tả một hoạt động, cần phải liên tục chuyển đổi hình ảnh theo thứ tự cho trước theo một khoảng thời gian trì hoãn giữa mỗi lần chuyển đổi**.

* Giảipháp:
  + Giải pháp thứ nhất là lưu tập hình trên thành một Frame hình(một ảnh có chiều rộng lớn và trên đó là các hình mô tả chuyển động đã xếp thứ tự). Bao gồm 2 cách sau:
    - Tất cả hình của tất cả các trạng thái nằm trong một frame hình.
    - Mỗi trạng thái có một frame hình.

Sự chuyển đổi liên tiếp được thực hiện bằng cách liên tục khoanh vùng hình ảnh trên frame và vẽ ra màn hình. Cách làm như thế có ưu điểm là chỉ cần load Frame một lần, rồi chuyển khoanh vùng liên tục trên Frame hình và vẽ ra, nhưng khuyết điểm của nó là phải xây dựng được Frame hình phù hợp, đồng thời, xác định khoanh vùng phải chính xác và đủ nhanh, mặt khác, XNA chỉ có thể load hình ảnh với size 2024x2024 px, vì thế, nếu số lượng hình ảnh trên một Frame là quá nhiều có lẽ sẽ làm chiều rộng của Frame vượt ngưỡng.

* + Giải pháp thứ 2 được sử dụng là lưu các hình ảnh một cách rời rạc, và mô tả đường dẫn đến mỗi hình trong file XML sao cho mỗi tập hình cho một trạng thái được lưu cùng chỗ. Từ file XML, mỗi lẫn muốn thể hiện trạng thái của Sprite, load toàn bộ tập hình vào mảng và việc chuyển đổi sẽ thực hiện bằng cách chuyển index của mảng. Khuyết điểm của giải pháp này là thời gian xử lý hơi chậm nếu tập hình cho một trạng thái quá nhiều, nhưng ưu điểm của nó là cho phép chuyển đổi hình nhanh bằng index sau khi đã load đủ tập hình vào mảng vì không phải thực hiện tính toán khoanh vùng như Frame, mặt khác sẽ giảm thiểu rủi ro phải load một ảnh vượt quá ngưỡng kích thước cho phép.

Cả hai giải pháp trên đều thực hiện chuyển đổi hình ảnh theo cách thức khác nhau nhưng đều có điểm chung là cách làm này dựa trên cách thức làm phim hoạt hình, sau một khoảng thời gian ngắn, vài mili second, mới thực hiện chuyển hình một lần để thể hiện hoạt động. Khoảng thời gian ngắn đó là delay time để trì hoản giữa hai lần chuyển hình kế tiếp.

1. Quản lý Structure và Unit trên sao cho truy cập thật nhanh.

* Vấn đề:

Các Sprite trên vùng chiến sự có số lượng rất nhiều, và chia thành nhiều loại bao gồm quân lính(Unit), công trình(Structure)… mặt khác, với các Sprite là Unit, Structure, chúng thuộc sở hữu của một Player cụ thể. Vấn đề đặt ra là làm cách nào để quản lý và truy xuất các Sprite theo từng loại nhanh và dễ dàng.

* Giải pháp
  + Giải pháp thứ nhất : duyệt trong sách Components của đối tượng Game mà XNA cung cấp và kiểm tra xem Components lấy được là Sprite cụ thể loại gì, thuộc sở hữu của Player nào... Cách này hoàn toàn không khả thi với một danh sách Components quá nhiều phần tử.
  + Giải pháp thứ hai là
    - Trên toàn bộ Game, chia các Sprite thành các danh sách con mà các phần tử là một loại Sprite muốn quản lý
* Danh sách các Unit hiện có trong vùng chiến sự
* Mảng các Structure hiện có trên BattleField
* Thực hiện vòng lặp trên hai danh sách này để tìm Sprite cần thiết
  + - Đối với mỗi Player lưu hai danh sách bao gồm danh sách các unit và danh sách các Structure
    - Trong Structure, lưu danh sách các Unit do nó sinh ra.
    - Mỗi Unit có thuộc tính là lớp Structure cho biết nó do Structure nào sinh ra.
    - Mỗi Unit và Structure có một thuộc tính là lớp Player cho biết nó thuộc Player nào.

Nhờ vào các mối quan hệ này, việc truy cập thông tin các Sprite trên map sẽ thực hiện nhanh chóng hơn.

1. Với Unit có khả năng tấn công, nhiệm vụ phát hiện và tấn công đối phương là nhiệm vụ chủ yếu.Trong suốt quá trinh tồn tại, Unit sẽ chết nếu sinh lực đã cạn do bị tấn công.

* Vấn đề

Một Unit có khả năng tấn công sẽ tiêu diệt đối phương khi đối phương tiến lại gần hoặc do chính player chủ động điều khiển Unit đánh quân địch. Unit cần phải biết khi nào có kẻ địch ở gần và thu hẹp khoảng cách nếu Unit có ý định tấn công. Unit kiểm tra khoảng cách giữa Unit và địch có nằm trong phạm vi cho phép Unit ra đòn tấn công. Dùng sức mạnh của mình, Unit sẽ làm tiêu hao sinh lực và tiêu diệt địch. Trong qua trình tấn công, Unit cần xác định đối tượng mình tấn công có còn trong phạm vi tấn công hay không(đối phương có thể đã bỏ chạy) để có hành động phù hợp(không tấn công nữa, chuyển sang trang thái IDLE hoặc WALK để đuổi theo). Trong trường hợp đối phương đã hết sinh lực, Unit ngừng tấn công đối phương, nhưng trong trường Unit hết sinh lực, Unit sẽ chuyển sang trang thái DIE. **Vấn đề phải giải quyết là xử lý toàn bộ quá trình phát hiện, tấn công và chết của một Unit**.

* Giải pháp:
  + Giải pháp đưa ra như sau
    - Các Unit có khả năng tấn công có thuộc tính mô tả phạm vi tấn công và phạm vi phát hiện, được thể hiện bằng khoảng cách từ Unit tới đối phương và phạm vi tấn công lớn hơn phạm vi phát hiện, bất kì đối phương nào rơi vào phạm vi phát hiện sẽ bị Unit áp sát và rút ngắn khoảng cách, khi khoảng cách nằm trong phạm vi tấn công Unit sẽ ra đòn tấn công lên đối phương để tiêu hao sinh lực địch.
    - Unit cần phải biết đối tượng hiện tại bị nó tấn công để kiểm tra rằng
      * Đội tượng đang bị Unit tấn công có còn sinh lực không, nếu đã hết, Unit sẽ ngừng tấn công và chờ đợi một đối tượng tấn công khác
      * Nếu đối tượng bị tấn công nếu đã bỏ chạy ra ngoài phạm vi tấn công, Unit sẽ ngừng tấn công và đuổi theo, nếu đối tượng bị tấn công nằm ngoài phạm vi phát hiện, Unit dừng tấn công và đứng yên hoặc chờ lệnh từ Player.

Đối tượng bị tấn công này được lưu thành thuộc tính **WhoIHit** trong lớp Unit. Mọi thao tác kiểm tra như trên(kiểm tra sinh lực, kiểm tra còn nằm trong phạm vi), được thực hiện thông qua nó.

* + - Trong trường hợp Unit đã cạn kiệt sinh lực, DIE là trạng thái Unit sẽ phải nhận lấy và được dispose ra khỏi chương trình.
    - Tất cả các hoạt động của Unit đều được thể hiện qua trạng thái cùng với tập hình ảnh thể hiện thao tác đi kèm đã được nêu trong “**Thể hiện trạng thái hoạt động của các Sprite trong Game”**

1. Structure hoặc Unit đã hết health 🡪 không còn tồn tại trên map 🡪 dispose
2. Producer Unit
   * Xác định vị trí resource center để lấy

**Giải quyết:**

* + - * Duyệt mảng các resource center trên bản đồ
        + Phát hiện ra resource center mà nó có thể khai thác
        + Khai thác resoure center này nếu

Resource mà nó đang mang trên người số lượng chưa max

Nó hiện ko có mang resource nào

Resource center mà nó muốn khai thác khác resource mà nó mang trong người

* + - * Kiểm tra khi nó di chuyển, nó đã chạy ra ngoài phạm vi của resource center để có thể khai thác resource chưa. Nếu chưa ra khỏi phạm vi 🡪 khai thác tiếp, nếu đã ra khỏi phạm vi 🡪 đứng yên
  + Xác định vị trí nhà chính để cung cấp resource
  + Tự động bỏ chạy khi bị tấn công hoặc đứng yên để cho đối phương tấn công(power = 0)
    - Chạy hướng nào, xa bao nhiêu

**Giải quyết:** chưa có

* + - (Hoặc tiếp tục khai thác cho đến khi bị đánh chết 🡪 máu phải thật nhiều để chịu đựng cho tới khi người chơi di chuyển nó bỏ chạy)

1. Hiệu ứng âm thanh (sound effect)
   * Có 2 cách tiếp cận
     + Dùng XACT(nói sau)
     + Không dùng XACT 🡪 sẽ dùng class SoundEffect hoặc SoundEffectInstance, cách này đơn giãn hơn dùng XACT

(XACT không hoạt động trên Zune, nên cách 2 sẽ là 1 tùy chọn tốt cho các game trên Zune)