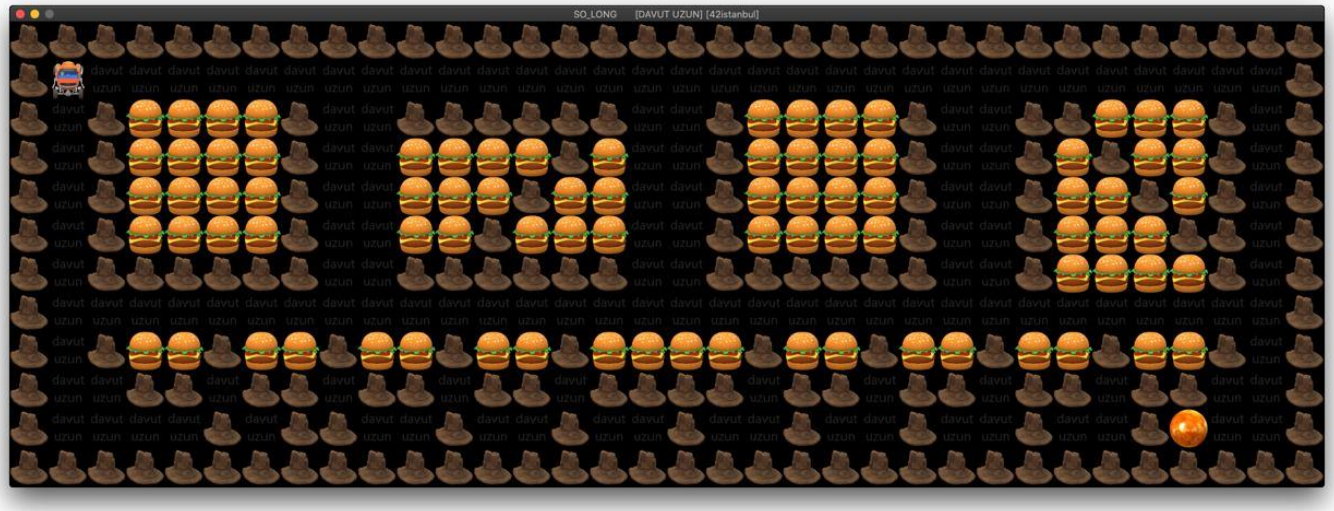
Matristeki İki Hücre Arasında Yol Olup Olmadığını Bulma

(SoLong 2D Oyun projesi için)

(SoLong projesi için haritadaki tüm hedeflere (toplanabirlirlere C ya da E) oyuncunun (P player) ulaşılabilirliği kontrolü algoritması.)



İçindekiler:

Açıklama: Bu dokümanda yer alan bilgiler kişisel deneyimlere çerçevesinde oluşturulmuştur. Sizin tecrübeleriniz farklılık arz edebilir. Notlarda hata ve unutmalar mutlaka vardır. İşletim sistemleri ve sürümlerinden kaynaklı uyumsuzlar olabilir. Tüm bu durumları dikkate alarak bu dokümandan faydalanabilirsiniz.

Kaynakça: 42 İstanbul yazılım okulu akran öğrenimi, https://tr.wikipedia.org/ ve https://www.google.com ile ulaşılan kamuya açık web siteleri.

SoLong Nedir?

Ve tüm balıklar için teşekkürler!

Özet: Bu proje çok küçük bir 2D oyundur. Sizi doku ve hareketli grafiklerle çalıştırmak için hazırlandı. Ve bazı çok temel oynanış elemanları…

Matristeki İki Hücre Arasında Yol Olup Olmadığını Bulma

(So_Long projesi için haritadaki tüm hedeflere (toplanabirlirlere C yada E) oyuncunun (player P) ulaşılabilirliği kontrolü için kullanacağız.)

Öz yineleme (recursion) kullanarak matriste iki hücre arasında yol olup olmadığını bulun:

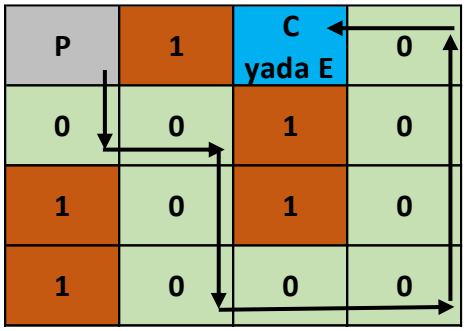
P, C, 0, 1, E ile doldurulmuş verilen **NxN** matrisi. Kaynaktan hedefe, yalnızca boş **(0)** hücrelerden geçen bir yol olup olmadığını bulun. Yukarı, aşağı, sağa ve sola hareket edebilirsiniz.

-
- **P** ifadesi hücrenin değeri, başlangıç (kaynak) player anlamına gelir.
- **C** ifadesi hücre(lerin) değeri, hedef(ler) (toplanabilir(ler)) anlamına gelir.
- **0** ifadesi hücre(lerin) değeri, boş hücre(ler) (**P** ile üzerinde de hareket edilebilen yolları) anlamına gelir.
- **1** ifadesi hücre(lerin) değeri, duvar(ları) (sınırları) temsil eder. (**P** ile üzerinde hareket edilemez.)
- **E** ifaddesi hücrenin değeri, hedef, bitiş son toplanabilir alanı, programdan çıkışı temsil eder.

Not: Yalnızca tek bir başlangıç **(P)**, tek bir çıkış hedefi **E** vardır. Hedef(ler) yani **(C)** birden fazla olabilir.

Aşağıdaki tabloda **So_Long** projesindeki haritada kullanılan 5 olası karakter gösterilmiştir.

1	Duvar için (Blok) (Wall) Davut Uzun
p	Başlangıç Konumu (Start)
C	Toplanabilirler, hedef hücre(ler) (cells)
E	Son Hedef, son hücre (cell)
0	Boş hücre(ler) (blank cell) duzun



Buradaki fikir, her matristeki hücrenin kaynak indeksini (P) bulmak ve daha sonra kaynak indeksten matristeki hedefe (C) tekrar tekrar bir yol bulmaktır. Algoritma, hedefe giden son bir yol bulunana kadar tüm yolları tekrar tekrar bulmayı içerir.

Örnek:
Girdi:
M [4][4] { { ‘P’, ‘1’, ‘C’, ‘0’ },
{ ‘0’, ‘0’, ‘1’, ‘0’ },
{ ‘1’, ‘0’, ‘1’, ‘0’ },
{ ‘1’, ‘0’, ‘0’, ‘0’ } }
Çıktı: 1 (TRUE)

Yukarıdaki matris örneği için aşağıda yer alan programı matris değerleri ile çalıştırırsak 1 (TRUE) çıktısını alırız. Çünkü P den C ye boş hücre 0 üzerinden kesintisiz bir yol mevcuttur.

C Program Kodu:

```
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   matris.c                                           :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: duzun <davut@uzun.ist>                     +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2022/11/12 18:24:47 by duzun             #+#    #+#             */
/*   Updated: 2022/11/18 17:57:16 by duzun            ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

// Matriste (4x4 matris) iki hücre arasında yol olup olmadığını bulma.

#include <unistd.h>
#include <string.h>

// Matris boyutunu tanımlama.
#define N 4

// Sınırları kontrol etme foksiyonu. Matris sınırlarının dışına hareket edilirse
```

```c
// fonksiyonu çağıran koşul gerçekleşmez.
int ft_safe(int i, int j, int matrix[][N])
{
    (void) matrix[N][N];
    if (i >= 0 && i < N && j >= 0 && j < N)
        return (1);
    return (0);
}

// Bir kaynaktan (P) (değerine sahip bir hücre)
// bir hedefe (C/E) (toplanabilir(ler) değerine sahip bir hücreye
// giden yol varsa 1 (Evet/True) değerini döndürür.
// Yol bulunamaz ise 0 (Hayır/False) döndürür.
int ft_path(int matrix[][N], int i, int j, int visited[][N])
{
    // Sınırların, duvarların ve hücrenin ziyaret
    // edilip edilmediğinin kontrol edilmesi.
    if (ft_safe(i, j, matrix) && matrix[i][j] != '1' && !visited[i][j])
    {
    // Hücreyi ziyaret et
    // ziyaret edilen hücrenin değerini 1 olarak değiştir.
    visited[i][j] = 1;
        // Hücre gerekli hedef (C) ise, o zaman 1 (true) değerini döndürür.
        if (matrix[i][j] == 'C')
            return (1);
        // Yol yukarı yönde bulunursa 1 (true) döndür.
        if (ft_path(matrix, i - 1, j, visited))
            return (1);
        // Yol sol yönde bulunursa 1 (true) döndür.
        if (ft_path(matrix, i, j - 1, visited))
            return (1);
        // Yol aşağı yönde bulunursa 1 (true) değerini döndürür.
        if (ft_path(matrix, i + 1, j, visited))
            return (1);
        // Yol sağ yönde bulunursa 1 (true) döndür.
        if (ft_path(matrix, i, j + 1, visited))
            return (1);
    }
    // Yol bulunamazsa 0 (false) döndürür.
    return (0);
}

//Yolun var olup olmadığını bulma ve yazdırma yöntemi.
void    ft_path_find(int matrix[][N])
{
    // Zaten ziyaret edilen dizinleri takip etmek için
    // ziyaret edilen diziyi tanımlama.
    int visited[N][N];
    int result;
    int i;
```

```
    int j;

    // visited matrisi 0 ile doldurulur.
    // ilk durumda tüm hücreler ziyaret edilmemiş olarak işaretlemek için
    // 0 ile doldurulur. Takip amaçlı. Ziyaret edilen hücre içinde 1 rakamını
    // kullanacağım.
    memset(visited, 0, sizeof(visited));
    // Yolun var olup olmadığını belirtmek için result bayrağı ile takip et.
    // ilk bayrak değeri 0  olarak ayarlanır: result = 0 (false)
    result = 0;
    i = -1;
    while (++i < N)
    {
        j = -1;
        while (++j < N)
        {
            // Eğer matris[i][j] kaynak (P oyuncu) ise
            // ve hücre henüz ziyaret edilmemiş ise.
            if (matrix[i][j] == 'P' && !visited[i][j])
            {
                // i, j'den başlayarak yolu bulmaya çalış.
                // ft_paht yolu bulup bulmamaya göre 1 yada 0 döndür.
                if (ft_path(matrix, i, j, visited))
                {
                    // Eğer yol varsa 1 (true) döndür.
                    // result değeri 1 e eşitlenir.
                    result = 1;
                    break ;
                }
            }
        }
    }
    if (result)
        // Kaynaktan (P player) hedefe (C yada E toplanabilir) yol varsa
        // 1 True değeri döndürüldüğünde result = 1 olacağından ekrana
        // Evet/True yazdırılır.
        write(1, "Evet/True\n", 11);
    else
        // Yol bulunamaz ise result değeri = 0 olacağından ekrana
        // Hayır/False yazdırılır.
        write(1, "Hayır/False\n", 13);
}

//Yukarıdaki işlevi kontrol etmek için örnek program.
int main(void)
{
    int matrix[N][N] = { { 'P', '1', 'C', '0' },
                         { '0', '0', '1', '0' },
                         { '1', '0', '1', '0' },
                         { '1', '0', '0', '0' } };
```

```
    ft_path_find(matrix);
    return (0);
}
```

Şimdi aşağıdaki programımız ile örnek bir çalışma yapalım ve kodumuzun çalışma mantığını anlamaya çalışalım. Örneğimizde bir başlangıç ‘**P’** ve bir hedefimiz **‘C’** olacak şekilde düzenliyoruz. Burada amaç programın başlangıç ile hedef arasında nasıl bir yol izlediğini ve yol olup olmadığını doğrulama aşamalarını anlamaktır. (So_long projesinde birden fazla yol tekrar eden döngülere sokarak kontrol edeceğiz.)

Örnek 2:

Girdi:

M [4][4] { { ‘1’, ‘1’, ‘1’, ‘1’ },
{ ‘1’, ‘P’, ‘0’, ‘1’ },
{ ‘1’, ‘1‘, ‘C’, ‘1’ },
{ ‘1’, ‘1’, ‘1’, ‘1’ } }

Çıktı: 1 (TRUE)

Program çalışma mantığı:

Değerleri grip programı çalıştırdığımızda program

```
if (matrix[i][j] == 'P' && !visited[i][j])
```

matrix[N][N] İlk Durumu

1	1	1	1
1	P	0	1
1	1	C	1
1	1	1	1

visited[N][N] İlk Durumu

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

result İlk Değeri: 0

i ve j: 0 0

satırına ulaştığında ilk değerler şöyle oluşur.

İlk matris konumunun değeri **‘1’** olduğundan **if** şartı gerçekleşmeyecek ve döngü **j** ve **i** değerlerini artırarak **‘P’** değerini bulana denk yükseltecek. **i** ve **j = 1** olduğunda **matrix[i][j] = ‘P’** olur ve ilk şart gerçekleşir. **!visited[i][j]** şartı da gerçekleşmiş olduğundan **if** bloğu içine geçilir.

```
                if (ft_path(matrix, i, j, visited))
```

Başlangıç **‘P’** konumunu tespit ettik ve bu konumdan **‘C’** konumuna engelsiz bir yol olup olmadığını kontrol etmek için **ft_path()** fonksiyonu **i** ve **j** değerleri ile çağrılır. **ft_path()** fonsiyonundan **1** değeri ile geri dönüş olursa **result = 1** olur ve döngüden **break** ile çıkılır ekrana **Evet\True** yazdırılır. **0** ile dönüş olursa ekrana **Hayır\False** yazdırılır.

Şimdi **ft_paht()** fonksiyonunda ki aşamaları gözden geçirelim. İlk durum aşağıdaki gibidir

matrix[N][N] Değerleri

1	1	1	1
1	P	0	1
1	1	C	1
1	1	1	1

visited[N][N] Değerleri

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

result Değeri: 0

i ve j: 1 1

ft_paht() fonksiyonu ilk **if** bloğu **i** ve **j** değerinin matris sınırları dışına çıkılıp çıkılmadığını kontrol etmek için **ft_safe()** fonsiynunu çağırır. Dönüş **1** değeri ise diğer diğer şart olan **matrix[i][j] != '1'** kontrolu yapılır yani bu **i,j** konumu duvar **‘1’** değilse üçüncü şart olan **!visited[i][j]** kontrol edilir. Tüm şartlar sağlandığı için **if** bloğunun icrası başlar.

```
    visited[i][j] = 1;
```

matrix[N][N] Değerleri

1	1	1	1
1	P	0	1
1	1	C	1
1	1	1	1

visited[N][N] Değerleri

0	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0

result Değeri: 0

i ve j: 1 1

visited[1][1] matris konumu **1** olarak güncellenir. Ardından yeni bir **if** bloğu kontrol çalıştırılır. Sıradaki şart olan **matrix[i][j] == 'C'** gerçekleşmediğinden bu blok atlanarak sonraki if bloğuna geçilir.

```
        if (matrix[i][j] == 'C')
            return (1);
```

Sıradaki **if** bloğu çalıştırılır. Bu blok ile öz yenileme (recursion) ile **ft_path()** bloğu kendi kendini **i -1** ile yeniden çağırmış oldu.

```
        if (ft_path(matrix, i - 1, j, visited))
            return (1);
```

matrix[N][N] Değerleri			
1	1	1	1
1	P	0	1
1	1	C	1
1	1	1	1

visited[N][N] Değerleri			
0	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0

result Değeri
0
i ve j
0 1

matrix[i][j] = ‘1’ olacağından (duvar ile karşılaşma) bu **if** bloğundan geri dönülecek sıradaki diğer **if** bloğu çağrılacak.

```
if (ft_path(matrix, i, j - 1, visited))
            return (1);
```

matrix[N][N] Değerleri			
1	1	1	1
1	P	0	1
1	1	C	1
1	1	1	1

visited[N][N] Değerleri			
0	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0

result Değeri
0
i ve j
1 0

Bu sefer **j** değeri **1** eksiltilerek **ft_paht** fonsiyonu yeniden çağrılır. **matrix[i][j] = ‘1’** olacağından (duvar ile karşılaşma) bu **if** bloğundan geri dönülecek sıradaki diğer **if** bloğu çağrılacak.

```
        if (ft_path(matrix, i + 1, j, visited))
            return (1);
```

matrix[N][N] Değerleri			
1	1	1	1
1	P	0	1
1	1	C	1
1	1	1	1

visited[N][N] Değerleri			
0	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0

result Değeri
0
i ve j
2 1

Bu yeni durumda da **matrix[i][j] = ‘1’** olacağından (duvar ile karşılaşma) bu **if** bloğundan geri dönülecek sıradaki diğer **if** bloğu çağrılacak. Yeni durumda **j + 1** ile çağrılacak **ft_paht** fonksiyonunda **matrix[i][j] = ‘0’** olacağından (açık yol) fonksiyon icra edilir.

```
        if (ft_path(matrix, i, j + 1, visited))
            return (1);
```

matrix[N][N] Değerleri			
1	1	1	1
1	P	0	1
1	1	C	1
1	1	1	1

visited[N][N] Değerleri			
0	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0

result Değeri
0
i ve j
1 2

visited[i][j] = 1; ile **visited** matrisindeki **visited[i][j]** konumunun değeri **1** yapılır. Ardından bu konum **matrix[i][j] == 'C'** ile kontrol edilir ‘**C**’ eşit ise **return 1** ile dönüş yapılır. Konum **‘C’** eşit olmadığından yukarıdaki aşamalar sırayla tekrarlanır.

matrix[N][N] Değerleri			
1	1	1	1
1	P	0	1
1	1	C	1
1	1	1	1

visited[N][N] Değerleri			
0	0	0	0
0	1	1	0
0	0	0	0
0	0	0	0

result Değeri
0
i ve j
1 2

İkinci turda **i = 2** ve **j = 2** olduğunda **visited[2][2]** matris konumu **1** olarak güncellenir ve **matrix[i][j] == 'C'** ile kontrolü yapılır. Matris konumu ‘**C**’ değerine eşit olacağından **ft_paht()** fonsiyonundan **return (1)** ile dönüş yapılarak **ft_path_find** fonksiyonuna dönülür. Sıradaki işlemlere devam edilir.

matrix[N][N] Değerleri			
1	1	1	1
1	P	0	1
1	1	C	1
1	1	1	1

visited[N][N] Değerleri			
0	0	0	0
0	1	1	0
0	0	1	0
0	0	0	0

result Değeri
0
i ve j
2 2

```
                    result = 1;
                    break ;
```

matrix[N][N] Değerleri

1	1	1	1
1	P	0	1
1	1	C	1
1	1	1	1

visited[N][N] Değerleri

0	0	0	0
0	1	1	0
0	0	1	0
0	0	0	0

result Değeri

1		
i ve j		
2	2	

result değeri **1** yapılır ve **ft_paht_find()** fonksiyonundaki döngü **break ;** ile kesilerek döngü dışındaki sıradaki komutlar icra edilir. **result = 1** olduğundan ekrana “**Evet\True”** yazdırılır. **P** başlangıç noktasından **C** hedef noktasına ulaşan engelsiz bir yol olduğunu teyit etmiş olduk.

```
    if (result)
        write(1, "Evet/True\n", 11);
    else
        write(1, "Hayır/False\n", 13);
```

Program çıktısı:

```
┌─duzun@10 ~/Desktop/matris
└─$ ./a.out                                                    1 ↵
Evet/True
```

Artık bu algoritma ve mantığı kullanarak matris içindeki tüm hedeflere ulaşılabilir olup olmadığını bir dış döngü ekleyerek gerçekleştirebiliriz.

So_long projemize matris kontrol algoritmasını adapte etme

- *Bu aşamada bazı değişken adları ve tanımlamalar farklı kullanılmış olabilir. Siz kendi değişken adlarınızla işlemleri yapabilirsiniz*.

So_long.h içinde **struct** tanımlarınıza aşağıdaki değişken tanımlamalarını ekleyiniz. Bu tanımlamaları matriste başlangıçtan hedefe doğru bir yol olup olmadığının kontrolünde kullanacağız. Ayrıca fonksiyon bildirimlerini ekleyin

```
typedef struct s_game
{
    char    **visited;
    char    **matrix;
    int     mat_y;
    int     mat_x;
...
...
void    init_matrix(t_game *games);
void    map_exit_chack(int i, int j, t_game *games);
void    ft_path_find(t_game *games);
int     ft_path(int y, int x, t_game *games);
void    ft_paht_put(t_game *games, int result);
int     ft_safe(int y, int j, t_game *games);
void    ft_visited_clear(t_game *games);
```

Daha sonra **main.c** içinde ilk değerleri atayalım.

```
void    init_param(t_game *games)
{
    games->visited = NULL;
    games->matrix = NULL;
    games->mat_y = 0;
    games->mat_x = 0;
...
```

Harita matrisini oluşturduğunuz yerde aşağıdaki gibi **matrix** ve **visited** matrislerini de oluşturunuz. Bu şekilde hafızada harita boyutu kadar hafıza alanlarını da oluşturmuş oluruz.

```
    ...
    games->map = ft_split(data, '\n');
    ...
    ...
    games->matrix = ft_split(data, '\n');
    games->visited = ft_split(data, '\n');
```

Haritanızın boyutlarını ilgili değişkenlere atayınız.

```
games->mat_y = games->height;
games->mat_x = games->width;
```

init_matrix() fonksiyonu:
init_matrix() fonksiyonu ile bir döngü oluşturarak bir başlangıç **‘P’** den, birden çok hedef **‘C’** ye ulaşmaya çalışalım.
Verilen haritada hedef(ler) **‘C’** ve çıkış **‘E’** ye giden bir yol olup olmadığını kontrol etmeliyiz. Bunun için çıkışa **‘E’** hedefine giden bir yol var mı kontrolünü en sona bırakıyorum. Döngüde **‘E’** tespit ettiğim yere ‘**1’** yani duvar koyuyorum ve konumunu saklıyorum.
Ve ilk hedefi bulmak **‘C’** için döngü içinde kontrole devam ediyorum. Haritada birden çok **‘C’** noktası olabileceğini düşünerek matriste ilk tespit ettiğim **‘C’** hedefinin adını **‘F’** olarak değiştiriyorum. İlk yol aramam **‘P’** den ‘**F’** ye (yada tersi gibi düşünebiliriz **‘F’** den **‘P’** ye) olacak.
Bir yol olduğu tespiti yapıldıktan sonra bu **‘F’** yi tekrar **‘C’** olarak değiştiriyorum. Ardından döngüdeki sıradaki ‘**C’** hedefini aramaya devam ediyorum. **‘C’** hedef noktaları bittiğinde en son olarak **‘E’** çıkış hedefine bir yol aramak için daha önce **‘1’** duvara çevirdiğim konumunu sakladığım bilgilerini kullanarak **‘F’** aranan hedef olarak değiştiriyorum. Bu son **‘E’** çıkış hedefine de ulaşım sağlanıyorsa hazırlanan harita oynanabilir haritadır.
Haritamızın kontrolu bu şekilde tamamlanmış olur.
Program akışı hata ile kesilmez ise init_matrix() fonksiyonu çağrıldığı noktadan devam eder.

```
void    init_matrix(t_game *games)
{
    static int  exit_i;
    static int  exit_j;
    int         i;
    int         j;

    i = -1;
    while (games->matrix[++i])
    {
        j = -1;
        while (games->matrix[i][++j])
        {
            if (games->matrix[i][j] == 'E')
            {
                exit_i = i;
                exit_j = j;
                games->matrix[i][j] = '1';
            }
            if (games->matrix[i][j] == 'C')
            {
                games->matrix[i][j] = 'F';
                ft_path_find(games);
                games->matrix[i][j] = 'C';
            }
        }
    }
    map_exit_chack(exit_i, exit_j, games);
}
```

ft_path_find() fonksiyonu:
Bu fonksiyon çağrıldığında ilk olarak **ft_visited_clear()** fonksiyonunu çağırır ve **visited[i][j]** matrisinin içi **‘0’** ile doldurulur.
Bu fonksiyon kurulan döngü ile **i** ve **j matrix[i][j]** matrisinin içinde **‘P’** başlangıç konumunu arar. Aranan **‘P’** konumu bulunduğunda ve aynı konumlara sahip **visited[i][j]** matrisi **‘1’** ‘e eşit değilse **ft_path()** fonksiyonu çağrılarak **‘P’** den ‘**C’** kesintisiz bir yol var mı kontrolü yapılır.
Dönüş **return (1);** ile dönüş olursa yol var, **return (0);** geri döner ise uygun yol bulunamamış demektir.
Bu fonksiyonda **ft_paht_put()** fonksiyonu çağırır. Bu fonksiyonda dönüş değerine göre ya yol aramasının başarılı olduğunu yazar sıradaki konum için çağrıldığı yere döner, ya da geçersiz (başarısız) bir yol olduğunu yazar ve çıkış fonksiyonunu çağırır.

```
void    ft_path_find(t_game *games)
{
    int result;
    int i;
    int j;

    ft_visited_clear(games);
    result = 0;
    i = -1;
    while (++i <= games->mat_y)
    {
        j = -1;
        while (++j <= games->mat_x)
        {
            if (games->matrix[i][j] == 'P' && games->visited[i][j] != '1')
            {
                if (ft_path(i, j, games))
                {
                    result = 1;
                    break ;
                }
            }
        }
    }
    ft_paht_put(games, result);
}
```

ft_visited_clear() fonksiyonu:
Bu fonksiyon çağrıldığında **visited[i][j]** matrisini **‘0’** ile doldurur.

```
void    ft_visited_clear(t_game *games)
{
    int i;
    int j;

    i = -1;
    while (++i <= games->mat_y)
    {
        j = -1;
        while (++y <= games->mat_x)
        {
```

```
            games->visited[i][j] = '0';
        }
    }
}
```

ft_path() fonksiyonu:
Bu fonksiyon ile öz yenilemeli olarak **‘P’** noktasından **‘F’** noktasına matris içinde kesintisiz bir yol olup olmadığını kontrol ediyoruz. Yukarıda bununla ilgili daha detaylı açıklama yapmıştık.
Bu fonksiyon içinden çağrılan **ft_safe()** matris içinde yol ararken matris sınırlarının dışına taşma olup olmadığını denetlemektedir.
ft_paht() fonksiyonu iki nokta arasında uygun bir yol var ise **return (1);** uygun bir yol bulunamaz ise **return (0);** değerini döndürür.

```
int ft_path(int y, int x, t_game *games)
{
    if (ft_safe(y, x, games) && games->matrix[y][x] != '1' && \
    games->visited[y][x] != '1')
    {
    games->visited[y][x] = '1';
        if (games->matrix[y][x] == 'F')
            return (1);
        if (ft_path(y - 1, x, games))
            return (1);
        if (ft_path(y, x - 1, games))
            return (1);
        if (ft_path(y + 1, x, games))
            return (1);
        if (ft_path(y, x + 1, games))
            return (1);
    }
    return (0);
}
```

ft_safe() fonksiyonu:
Matris içinde yol ararken matrisin sınırlarını kontrol etmek için kullanılır.

```
int ft_safe(int y, int j, t_game *games)
{
    if (y >= 0 && y < games->mat_y && j >= 0 && j < games->mat_x)
        return (1);
    return (0);
}
```

ft_paht_put() fonksiyonu:
Bu fonksiyon **‘P’** den **‘F’ (‘C’)** aranan yol bulunup bulunmadığına gönderilen **result** değerine bakarak **if** koşulundaki işlemleri yapar.

```
void    ft_paht_put(t_game *games, int result)
{
    static int  k;

    if (result)
    {
        k++;
        ft_printf("[%d] ", k);
        write(1, "KONTROL AŞAMALARI / CONTROL STAGES ... BAŞARILI / \
SUCCESSFUL\n", 63);
    }
    else
    {
        exit_error(games, "[X] KONTROL AŞAMASI / CONTROL PHASE : BAŞARISIZ / \
UNSUCCESSFUL\nGeçersiz Harita: Tüm toplanabilir yada çıkışa erişim \
sağlanamıyor\nInvalid Map: All collectible or not accessible to exit", 0);
    }
}
```

map_exit_chack() fonksiyonu:
Bu fonksiyon ile bu fonksiyona kadar tüm ‘**C’** hedef toplanabilirlere doğru engelsiz bir yol olduğu ve en son olarak **‘E’** çıkış noktasına bir yol olup olmadığını denetlemek üzere bu fonksiyon çalıştırılır. Daha önce konumu kaydedilen çıkış noktası **matrix[i][j]** ilgili konumu **‘F’** olarak değiştirilir ve **‘P’** den **‘F’** ye yol olup olmadığını kontrolu için **ft_path_find()** fonksiyonu çalıştırılır.

```
void    map_exit_chack(int i, int j, t_game *games)
{
    games->matrix[i][j] = 'F';
    ft_path_find(games);
}
```

Son.

Başa ön --->