Amiral Battı 2008

Umut BENZER
05-06-7670
http://www.ubenzer.com
Ege Üniversitesi Bilgisayar Mühendisliği
Algoritma ve Programlama
Proje-2

Teslim Tarihi: 04.01.2008

Analiz

Amiral Battı çok geniş bir kitleyi hedef alabilecek, eskiden beri oynanan klasik oyunlardan biridir. Bu projede bir amiral battı oyunu geliştirilmesi istenmektedir.

Bu amiral attı oyununda savaş 12*12'lik bir alanda geçecektir. Oyun klasik amiral battı oyunlarında olduğu gibi karşılıklı atışlar halinde gerçekleşmeyecektir. Onun yerine düşmanın en fazla 10 tane olabilecek en fazla 6 birim uzunluğundaki gemileri en fazla toplam gemi uzunluğunun iki katı kadar atış hakkıyla vurulmaya çalışılacaktır.

Oyuna girildiğinde oyuncu, bilgisayarın kaç düşman gemisi olacağını yazar. Bilgisayar her gemi için uzunluk sorduktan sonra bunları rastgele sıralar ve bundan sonra oyuncu teker teker atışlar yaparak gemileri vurmaya çalışır. Toplam gemi uzunluğunun iki katı kadar atış hakkında tüm gemileri vurabilirse kazanır, vuramazsa kaybeder.

Gemilerin yerleşiminde bazı kurallar olduğu unutulmamalıdır:

1. Gemiler haliyle oyun alanından taşamazlar.
2. Gemiler haliyle birbirleriyle çakışamazlar.
3. Her geminin etrafında (üstünde, sağında, solunda, altında ve tüm çaprazlarda) en az birer birim boşluk olmak zorundadır.

Yanış	Doğru
01000 01000 01110	01000 01000 01011

Tasarım

Programda gözüme iki ciddi sorun çarptı. Birincisi, gemiler nasıl yerleştirilecek? Gemilerde rastgele fonksiyonunu kullanmak iyi bir çözüm fakat ya gemiler takılır kalırsa? 5*5'lik bir oyun alanına rastgele olarak 2 tane 5 birimlik, bir tane 1 birimlik gemi yerleştirdiğinizi düşünün. Ya gemiler aşağıdaki gibi denk gelirse?

```
10010
10010
10010
10010
10010
```

Yandaki şekilde de görebileceğiniz gibi bir karakterlik geminin yerleşebileceği alan kalmadı. Hâlbuki 4. sütundaki gemi 3. sütunda olsaydı, aynı sayıda ve uzunlukta geminin oyun alanına yerleşe imkânı olacaktı. Program ilk iki gemiyi yandaki gibi yerleştirdikten sonra ne kadar uğraşırsa uğraşsın, isterse sonsuz defa dönsün, üçüncü gemiyi yine de yerleştiremez. O halde programa bazı şartlar altında "dur" dememiz gerekebilir.

Yine programla 2 tane 5 birimlik, bir tane 1 birimlik gemiyi 5*5'e yerleştirmeye çalışalım. Program sırasıyla aşağıdaki yerleşimleri yapmaya başlasın.

```
10000  10100  10100
10000  10100  10100
10000  10100  10110
10000  10100  10100
10000  10100  10100
```

Program ilk iki adımda doğru yerleri tuttursa da, son hamlede yaptığı yerleşim yanlış oldu. O halde program tekrar deneyecek. Ve bu sefer yukarıdaki örnekten farklı olarak gerçekten gemiyi yerleştirebileceği yer de var.

Böyle iki farklı durum olabileceğine göre, iki farklı da çözüm üretmek gerekir.

Program bir gemiyi yerleştiremediğinde tekrar denesin. Ama uzun bir süre bu gemiyle uğraştıysa (DENEMELIMIT sabiti kadar) ve hala aynı gemi yerleşmiyorsa büyük bir ihtimalle o geminin diğer gemilerin konumundan dolayı yerleşme imkânı yoktur. O zaman bir adım geriye dönüp diğer gemileri de baştan yerleştirerek tekrar denemek akıllıca olacaktır. Bu şekilde bir süre denenince (en fazla BASTANDENEMELIMIT kadar)eğer kullanıcıdan alınan istek imkânlıysa (ki verilen ödevde imkânlı) programın gemileri yerleştirmeme olasılığı oldukça düşük olacaktır. Yerleştirme olasılığı ve programın neyi kaç defa denediği ile ilgili daha ayrıntılı bilgiler programcı kataloğu kısmında yer almaktadır.

İkinci büyük sorun ise gemi nasıl batacak? Gemileri rastgele yerleştirme esnasında bellekte tutmak çok kolay. Koordinatın dizideki değeri gemi varsa 1, yoksa 0 olsun. Buraya kadar kolay. Gemi isabet aldıysa 2, karavanaysa 3 olsun. Bu da çok kolay. Ama vurulan gemilerin batıp batmadığını denetlemek? O biraz sorun olabilir. Bir defa oyun alanına gemileri yerleştirip gemilerin durumlarını dizinin içinde tutmak o geminin konumlarını bilmemiz anlamına gelmiyor. Ekrana çizdirince bir insan olarak işte gemi şurada diyebilsek de algoritmik olarak düşündüğümüzde herhangi bir geminin (x uzunluğunda yatay ya da dikey) herhangi bir noktası (ortada, köşede) biliniyorsa o geminin tüm elemanlarının koordinatlarını bulmak sanıldığı kadar kolay olmuyor. Bunu yapabilmek için dört yöne doğru da x kadar elemanın gemiye ait olup olmadığına bakmamız ve batıp batmadığını kontrol etmemiz gerekirdi. Üstelik x'i bile bilmiyoruz. Buna aşağıdaki gibi bir çözüm ürettim:

Gemi durumlarını nasıl bir dizide tutuyorsam, gemilerin her bir elemanını gemi durumunu tutan dizinin kaçıncı indisinde tuttuğumun bir dizisini tutayım. Evet, biraz karışık oldu. Aşağıdaki gibi bir oyun alanımız olsun:

```
10100
10100
10100
10100
10001
```

Bunların hepsinin bir dizide aşağıdaki gibi tutulduğunu düşünelim:

```
Gemiler = "1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,0,0,0,1"
```

Tek boyutlu gemiler dizisinde yandaki bilgi yukarıdaki gibi tutuluyor. Burada bir sorun yok. Ateş edilip tutturulduğunda ilgili koordinattaki 1, 2'ye dönüşecek, bunda da sorun yok. Ama gemi batıyor mu? Eğer gemi yatay yerleşmişse gemiye ait bir sonraki 1 ya bir sağda ya 1 solda… Onu bulmak bile ayrı bir sorun. Eğer gemi dikeyse gemiye ait bir sonraki bir dizinin tamamen farklı bir koordinatında.

Bu sorunu kolaylıkla çözebilmek için koordinatların bir indisi tutulur. Aynı oyun alanı için en fazla 5 birim uzunluğunda gemi yaratılabildiğini, en fazla 3 gemi yerleştirme limiti olduğunu ve bu gemilerden ilk en sağdaki geminin, sonra tek birimlik geminin ve en son da ortadaki geminin yerleştirildiğini var sayarsak, programa aşağıdaki gibi bir dizi *daha* yarattırabiliriz.

```
Gemi İndisleri = "0,5,10,15,20,24,-1,-1,-1,-1,2,7,12,17,-1"
```

Görüldüğü üzere ikinci dizi gemi sıralamalı olarak koordinatları tutuyor. Eğer gemi maksimum gemi uzunluğundan küçükse gemi uzunluğuna -1 ile tamamlanıyor. Böylece örneğin kullanıcı 15 indisli 1'e ateş ettiyse, program 15'in 1. gemi olduğunu ikinci dizi sayesinde kolaylıkla anlıyor ve yine ikinci dizi sayesinde hangi indislerin o gemiye ait olduğunu araştırmak zorunda kalmaksızın 0,5,10 ve 24. indisleri kontrol ederek geminin battığını ya da sadece isabet aldığını kolaylıkla anlıyor.

Bu ikisi dışında pek de bir sorun yok. Diğer tüm işlemler bir değişken yaratıp yeri gelene kadar tutup, sonra da hesaplamaya katmaktan daha karışık değil. Yukarıda bahsedilen tüm sorunlar ayrıca programcı kataloğunda da anlatılacak.

Programcı Kataloğu

Programda (birisi main olmak üzere) 10 fonksiyon bulunmaktadır. Programcı kataloğu, programda geçen tüm fonksiyonların ne yaptığını açıklamakta ve girdi-çıktı değerlerini anlatmaktadır. Fonksiyonlar alfabetik sıralanmıştır. Açıklamalardaki girdiler prototipteki sıraya göre açıklanmıştır. Bu katalog programın 03.01.2007 tarihinde tamamlanmış sürümüne aittir ve buradaki kaynak kodları ile güncel kaynak kodları arasında çok ufak değişiklikler olması mümkündür. Kaynak kodunun en güncel sürümünü ve fonksiyon fonksiyon parçalanmamış halini çevrimiçi kaynaklarda bulabilirsiniz: http://www.ubenzer.com/proje2.c

sabitler

```
#include <stdio.h>
#include <stdlib.h>

#define OYUNALANI 12
#define GEMILIMIT 10
#define GEMIGENISLIK 6
#define DENEMELIMIT 30
#define BASTANDENEMELIMIT 90000
```

Bu program sadece ödevde verilen değerlerde çalışacak şekilde değil, çok ufak hamlelerde çok daha değişik limitlerde çalıştırılabilecek şekilde tasarlanmıştır. Programın başında tanımlı birçok sabitin değerleriyle oynanıp yeniden derlenerek tamamen farklı bir oyun elde edilebilir. Bu bölümde bu sabitler, ne iş yaptıkları ve nasıl değiştirilebilecekleri anlatılacaktır.

OYUNALANI: Bu sabit oyun alanının bir ayrıtının uzunluğunu belirler. Ödevde bu değer 12 olarak belirlenmişti. Oyun alanı programda ayrıca x ve y olarak tanımlanmadığı için oyun alanı kare olmak zorundadır. Bu sabit 1'den küçük olmamalıdır. 1'e 1 bir oyun alanında oyun geçemeyeceği için en az 2 olması önerilir. Bu sabit teorik olarak 26'den büyük olabilir. Ancak program harf tipi koordinatları alırken ASCII kodlarını kullandığı için 26'dan sonra kullanıcı giriş yapamayacağından program çalışamaz. (26. yatay eksende atış yapmak için Z koordinatı girilmektedir.)

GEMILIMIT: Bu sabit oyunda en fazla kaç tane gemi olabileceğini belirler. Ödevde bu değer 10 olarak belirtilmiştir. Bu sabit en az 1 olmalıdır.

GEMIGENISLIK: Bu sabit oyun alanındaki her bir gemi için en fazla kaç birim uzunlukta olabileceğini belirler. Bu sabit en az 1 olmalıdır.

> ! *OYUNALANI, GEMILIMIT ve GEMIGENISLIK sabitlerini değiştirirken GEMILIMIT tane GEMIGENISLIK uzunluğundaki geminin OYUNALANI*OYUNALANI alanındaki kareye en az bir tane olası yerleşimi olmasına dikkat edilmelidir. Aksi halde program imkânsızı gerçekleştiremeyeceğinden yerleştirme yapamayacak ve bunu hata olarak belirterek kendini sonlandıracaktır.*

DENEMELIMIT: Bir gemi yerleştirilemediğinde, aynı gemi için farklı farklı denemeler yapılır. Bu denemelerin en fazla kaç tane olacağını bu sabit belirler. Eğer DENEMELIMIT kadar denemede gemi hala yerleştirilemediyse tüm gemiler hafızadan atılarak hepsi baştan yerleştirilmeye çalışılır.

BASTANDENEMELIMIT: Bir gemi DENEMELIMIT kadar denemede yerleştirilemediyse bunun sebebi önceki gemilerin yeni bir geminin yerleşmesine imkân vermeyecek biçimde yerleşmesi olablir. Bu durumda önceki gemiler değişik biçimlerde yerleşirlerse tüm gemilerin yerleşme olasılığı olabilir. Bundan dolayı program BASTANDENEMELIMIT kadar tüm gemileri sıfırdan yerleştirmeye çalışır. Eğer hala yerleştirilemiyorsa (ki düşük bir ihtimaldir) o zaman büyük bir ihtimalle OYUNALANI, GEMILIMIT ve GEMIGENISLIK sabitlerindeki bir hatadan dolayı kullanıcıya yerleştirmesi imkansız olan bir kombinasyon girme imkanı tanınmıştır.

*BASTANDENEMELIMIT VE DENEMELIMIT sabitlerini arttırmak gemi yerleştirme süresini arttırırken imkanlı kombinasyonlarda yerleştirme olasılığını artıracaktır. Programın bu sürümünde kullanılan değerler 12*12'lik bir kareye 10 tane 6 birimlik geminin yerleşmesi için yeterli olmaktadır.*

veri yapıları

Bu kısımda programda sıkça kullanılan ve programın işleyişini anlayabilmek için çok iyi bilinmesi gereken dizilerin ne amaçla kullanıldıkları, içine nasıl bir düzende veri yazıldığı açıklanacaktır. Program içinde çok yaygın olarak kullanılmayan diziler olabilir. Burada açıklaması olmayan bu tip diziler ilgili fonksiyonun içinde açıklanacaktır.

gemiler (bu yapıdaki dizi programın her yerinde aynı isimle kullanılmaktadır)

Bu yapı tüm gemilerin durumlarını koordinat düzlemine göre bellekte tutan OYUNALANI*OYUNALANI elemanlı tek boyutlu bir dizidir. A1 hücresi için dizi indisi 0, A2 için 1… şeklinde gitmektedir. B satırına geçildiğinde indis sayısı yine bir artmaktadır. Her indisteki elemanın alabileceği değerler ve anlamları aşağıdaki gibidir:

0: Bu indisli koordinatta gemi yok ve (henüz) ateş edilmemiş.
1: Bu indisli koordinata gemi var ve henüz ateş edilmemiş.
2: Bu indisli koordinatta gemi var ve isabet almış.
3: Bu indisli koordinatta bir zamanlar gemi varmış, artık batırılmış.
4: Bu indisli koordinata ateş edilmiş, ama gemi olmadığı için karavanaymış.

Aşağıda 4*4'lük ve çıktısı da yanında görülebilen bir oyun alanının dizide nasıl tutulduğu görülebilir.

0I00	0G00
000B	0G0G
0K0B	000G

G harflerinin gemi olduğu düşünülürse ve oyun ekranı da sol köşedeki gibiyse gemiler dizisinin içeriği aşağıdaki gibidir:

```
Gemiler = "0,2,0,0,0,1,0,3,0,4,0,3"
```

gemidurumu (bu yapıdaki dizi programın her yerinde aynı isimle kullanılmaktadır)

Bu yapı tüm gemilerin ait oldukları koordinatların indislerini gemi numaralarına göre bellekte tutan GEMILIMIT*GEMIGENISLIK elemanlı tek boyutlu bir dizidir. Bu dizi herhangi bir koordinatta bulunan bir geminin başka hangi koordinatlarda olduğunu her seferinde bir sürü if döngüsüyle keşfetmeye gerek bırakmadan bulmamızı sağlar. Dizi birinci gemiden başlayarak tüm gemiler için geminin maksimum genişliği kadar indis ayırır. Örneğin bir gemi en fazla 3 birim olabiliyorsa, 0-2 indisleri 1.gemiye, 3-5 arası ikinci gemiye aittir. Geminin gerçekte ne kadar uzunlukta olduğu bu dizi için önemli değildir. Her gemi için dizinin o gemiye ait ilk elemanından başlanarak geminin hangi konumda olduğu (diğer bir deyişler gemiler dizisinin kaçıncı indisinin o gemiye ait olduğu) kaydedilir. Gemi uzunluğu maksimum gemi uzunluğundan küçükse kalan indisler -1 ile doldurulur.

Aşağıdaki 4*4'lük oyun alanında 1'ler gemileri temsil etsin. Rastgele seçilen gemilerden ilk alttaki, sonra üstteki yerleştirildiyse, en fazla gemi yerleştirme limiti 3'se ve bir gemi en fazla 3 elemanlı olabiliyorsa gemi durumu dizisi aşağıdaki gibi olur:

0100
0101
0101

```
Gemi Durumu = "7,11,-1,1,5,9"
```

int atis(int *, int, int, int *);

Gemiler dizisinde kullanıcıdan alınmış ve geçerliliği daha önce denetlenip harften rakama çevrilmiş yatay ve dikey koordinatlarına atış yapılır. Gemilerin yerleşimi gemi durumu dizisindeki gibidir. Fonksiyon ıskalama, isabet ettirme ve gemiyi batırma gibi durumları göz önünde bulundurur ve değişiklikleri gemiler dizisine işler. Fonksiyonun döndürebildiği değerler ve anlamları aşağıdaki gibidir:

0: Karavana
1: İsabetli atış
2: Gemi battı
3: Daha önce ateş ettiği bir yere tekrar ateş etti

Program gemilerin batıp batmadığını kontrol ederken daha önce atılan şekilde denetleme yapmak için gemi durumu dizisinden de yararlanır.

```
int atis(int *gemiler, int yatay, int dikey, int *gemidurumu) {

    int sayac;
    int sayac2;
    int gemino;
    int tamam;

    switch(gemiler[eleman(yatay,dikey,0)]) {
        case 0: // Bosluga ates etti (sanssiz)
            gemiler[eleman(yatay,dikey,0)] = 4;
            return 0;
        case 1: // Gemiyi vurdu! nhahahaha, acaba batti mi?

            gemiler[eleman(yatay,dikey,0)] = 2;

            for (sayac=0;sayac<GEMILIMIT;sayac++) {
                for (sayac2=0;sayac2<GEMIGENISLIK;sayac2++) {
                    if (gemidurumu[sayac*GEMIGENISLIK+sayac2] == eleman(yatay,dikey,0)) {
                        gemino = sayac;
                        break;
                    }
                }
            }

            tamam = 1;
            for (sayac=0;sayac<GEMIGENISLIK;sayac++) {
                if (gemidurumu[GEMIGENISLIK*gemino + sayac] != -1) {
                    if (gemiler[gemidurumu[GEMIGENISLIK*gemino + sayac]] == 1) {
                        tamam = 0;
                        break;
                    }
                }
            }

            if (tamam == 1) {
                for (sayac=0;sayac<GEMIGENISLIK;sayac++) {
                    if (gemidurumu[GEMIGENISLIK*gemino + sayac] != -1) {
                        gemiler[gemidurumu[GEMIGENISLIK*gemino + sayac]] = 3;
                    }
                }
                return 2;
            }

            return 1;
        default:
            return 3;
    }

}
```

void beklet();

Tek amacı kullanıldığında kullanıcıyı bir tuşa basana kadar bekletmektir.

```
void beklet() {
    /* Bu fonksiyonun tek amaci kullaniciyi bir tusa basana kadar
       bekletmek, ve sonra ekrani silmektir.                        */
    printf("\nDevam etmek icin bir tusa basin.");
    getche();
    system ("cls");
}
```

void cheat(int *);

Bu fonksiyon çağırıldığında gemiler dizisindeki bilgileri göz önüne alarak ekrana gemilerin güncel yerleşimini sunar. Bu esnada aslında görülmemesi gereken henüz vurulmamış gemileri de ekrana çizer. Eğer bu fonksiyon oyunun içinde çağırılsaydı hile yapmaya yardı.

Bu fonksiyon sadece oyunun sonunda kullanılmaktadır. Eğer oyuncu kazanamazsa gemilerin nerede olduğunu kullanıcının öğrenmesini sağlar. Ayrıca ödev kontrolü esnasında kullanılmak üzere herhangi bir yerde de çağırılabilir.

```
void cheat (int *gemiler) {

    /* Bu fonksiyon herhangi bir yerde cagrilirsa ekrana gemilerin
       guncel durumunu cizer. Diger bir deyisle hile yapmaya yarayan
       bir fonksiyondur. Odev yaparken cok isime yaradi. Belki odev
       kontrolunde de kolaylik (gemiler nasil yerlestirilmis bakmak
       acisindan) diye silmek yerine burda birakiyorum.              */

    int sayac;
    int sayac2;

    system("cls");

    printf("* ");
    for (sayac=1;sayac<=OYUNALANI;sayac++) {
        printf("%d ",sayac%10);
    }
    printf("\n");

    for (sayac=0;sayac<OYUNALANI;sayac++) {
        printf("%c ",rakam_to_harf(sayac));
        for (sayac2=0;sayac2<OYUNALANI;sayac2++) {
            switch (gemiler[sayac*OYUNALANI+sayac2]) { // Konumdaki geminin durmunu al
                case 0:
                    printf("0 ");
                    break;
                case 1:
                    printf("X ");
                    break;
                case 2:
                    printf("I ");
                    break;
                case 3:
                    printf("B ");
                    break;
                case 4:
                    printf("K "); // Karavana
                    break;
            }
        }
        printf("\n");
    }

    beklet();
}
```

void ciz(int *);

Bu fonksiyon gemiler dizisindeki bilgiye göre güncel gemi durumunu ekrana çizer. Cheat fonksiyonundan farkı, bu fonksiyon oyun esnasında sürekli çağırılır. Oyuncunun görmemesi gereken henüz isabet almamış gemileri (dizide 1 değerindeki elemanları) 0 olarak gösterir.

```
void ciz(int *gemiler) {

    /* Bu fonksiyon gemiler dizisinde yer alan gemi yerlesim
       ve gemi durumu bilgilerini kullanarak guncel oyun ala-
       nini ekrana cizer.                                    */

    int sayac;
    int sayac2;

    system("cls");
    printf("\n* ");
    for (sayac=1;sayac<=OYUNALANI;sayac++) {
        printf("%d ",sayac%10);
    }
    printf("\n");
    // Bitti

    for (sayac=0;sayac<OYUNALANI;sayac++) {
        printf("%c ",rakam_to_harf(sayac)); // Harf ekseninin ekrana yazma
        for (sayac2=0;sayac2<OYUNALANI;sayac2++) {
            switch (gemiler[sayac*OYUNALANI+sayac2]) { // Konumdaki geminin durmunu al
                case 0:
                case 1:
                    printf("0 ");
                    break;
                case 2:
                    printf("I ");
                    break;
                case 3:
                    printf("B ");
                    break;
                case 4:
                    printf("K "); // Karavana
                    break;
            }
        }
        printf("\n");

    }
    printf("\n");

}
```

int eleman(int,int,int);

Satır girdisiyle gelen satır ve sütun girdisiyle gelen sütunun verisinin gemiler dizisinin hangi indisli elamanında tutulması gerektiğini geri gönderir. Eğer girilen satır ve/veya sütun değeri geçersiz ise geriye -1 gönderir. Eğer yatay_dikey 1 olursa satır girdisinden sütun, sütun girdisinden satır okunur. Bu tersten işlem yapma gemileri yerleştirirken yatay ya da dikey yerleştirme kısmında oldukça büyük bir kolaylık sağlamaktadır.

```
int eleman(int satir, int sutun, int yatay_dikey) {

    /* SATIR satÄ±rÄ±na ve SUTUN sutuna ait noktanin bellekte
       hangi numarali dizinde tutuyor oldugunu geri gonderir.
       yatay_diket 1 olursa SATIR ve SUTUN yer degistirir.

       Ayrintili bilgi raporda bulunabilir.                  */

    if (yatay_dikey == 0) {
        if (satir+1>OYUNALANI || sutun+1>OYUNALANI) return -1;
        return satir*OYUNALANI + sutun;
    } else {
        if (sutun+1>OYUNALANI || satir+1>OYUNALANI) return -1;
        return sutun*OYUNALANI + satir;
    }

}
```

int konum_gecerli(int);

Konum değişkeni ile gelmiş sayının gemiler dizisinin bir indisi olup olamayacağını kontrol eder. Bu fonksiyon dizide taşma olmasını engellemek açısından önemlidir.

```
int konum_gecerli (int konum) {

    /* KONUM degiskeninin oyun alanina ait bir koordinat olup
       olmadigini denetler.

       Ayrintili bilgi raporda bulunabilir.                    */

    if (konum < OYUNALANI*OYUNALANI && konum >= 0) return 1;
    return 0;

}
```

int main()

İlk çalışan fonksiyondur. Açılış yazılarının yazılması, gemi sayısının ve uzunluklarının alınması ve denetlenmesi, koordinatları alan ve denetleyip ilgili fonksiyonlara gönderen temel döngüler, oyunun kazanılıp kazanılmadığı gibi en temel işlemler bu fonksiyon içerisinde olmaktadır.
Başka bir deyişle gerektikçe fonksiyon çağıran, bir bilgiyi bir fonksiyona işletip gerekli yeni bilgiyi aldıktan sonra başka bir fonksiyona giden temel bir fonksiyon gibidir. Bu fonksiyonun kaynak kodu oldukça uzun olduğundan iki parçada verilmiştir.

```
int main() {
    /* Acilis yazilari BASLANGICI */
    printf("Umut BENZER\n");
    printf("05-06-7670\n");
    printf("Ege Universitesi Bilgisayar Muhendisligi 1. Sinif\n");
    printf("http://www.ubenzer.com\n");
    printf("Amiral Batti 1.0\n\n");
    /* Acilis yazilari SONU */

    int gemisayi;
    int gemiuzunluk [GEMILIMIT+1];
    int sayac,sayac2;
    int gemiler[OYUNALANI*OYUNALANI];
    int atissayisi;
    int gemidurumu[GEMILIMIT*GEMIGENISLIK];
    int mesaj;
    int batikgemi;
    int secenek;
    int sag_serbest = 1;

    while (sag_serbest != 0) {

        sifirla(gemiler);
        atissayisi = 0;
        gemisayi = 0;
        batikgemi = 0;
        mesaj = -1;

        for (sayac=0;sayac<GEMILIMIT+1;sayac++) {
            gemiuzunluk[sayac] = 0;
        }

        for (sayac=0;sayac<GEMILIMIT*GEMIGENISLIK;sayac++) {
            gemidurumu[sayac] = -1;
        }

        printf("Amiral batti oyununa hosgeldiniz.\n");
        printf("Oyuna baslamadan gemileri yerlestirmek zorundayim.");

        while (gemisayi < 1 || gemisayi > GEMILIMIT) {
            printf("\nEn fazla %d gemi olmak uzere kac gemi yerlestirmek istediginizi giriniz:
",GEMILIMIT);
            scanf("%d",&gemisayi);
        }

        system("cls");
        printf("Tesekkur ediyorum. Simdi bu %d geminin uzunluklarini sizden
isteyecegim.\n",gemisayi);
        printf("Gemilerin uzunluklari 0'dan buyuk %d'den kucuk olmalidir.",GEMIGENISLIK+1);

        for (sayac=0;sayac<gemisayi;sayac++) {

            while(gemiuzunluk[sayac] < 1 || gemiuzunluk[sayac] > GEMIGENISLIK) {
```

```
                printf("\n(%d/%d) %d. geminin uzunlugunu giriniz: ",sayac+1,gemisayi,sayac+1);
                scanf("%d",&gemiuzunluk[sayac]);
            }

        }

        if (yerlestir(gemiuzunluk,gemiler, gemidurumu) == -1) { // Gemiler yerleÅŸtirilemedi.
            return 0;
        }

        sayac = 0;
        while(gemiuzunluk[sayac] != 0) { // Gemiler bitene kadar
            atissayisi += gemiuzunluk[sayac];
            sayac++;
        }

        atissayisi = atissayisi * 2;

        char yatay;
        int dikey;

        for (sayac=1;sayac<=atissayisi;sayac++) {
            ciz(gemiler);
            printf("\nOyun: (%d/%d)\n",sayac,atissayisi);

            switch (mesaj) {
                case 0:
                    printf("Uzgunum, iskaladin.");
                    break;
                case 1:
                    printf("Tebrikler isabetli atis.");
                    break;
                case 2:
                    printf("Cok iyi! Gemi batti!");
                    break;
                case 3:
                    printf("Acemi misiniz? Oraya daha once ates etmistiniz?");
                    break;
            }

            printf("\nGemi %d/%d",batikgemi,gemisayi);

            yatay = -1;
            dikey = -1;
            do {
                printf("\nAtis yapmak istediginiz koordinati giriniz: ");
                scanf("%c%d",&yatay,&dikey);
            } while(dikey < 1 || dikey > OYUNALANI || (yatay - 65) < 0 || (yatay - 64) >
OYUNALANI);

            mesaj = (atis(gemiler,yatay-65,dikey-1,gemidurumu));
            if (mesaj==2) {
                batikgemi++;
            }

            if (batikgemi == gemisayi) {
                system("cls");
                printf("Tebrikler! Kazandiniz!");
                beklet();
                break;
            }
        }

        if (batikgemi != gemisayi) {
            system("cls");
            printf("Uzgunum, kaybettiniz.\n Gemi dizilimi gormek icin bir tusa basin.");
            getche();
            cheat(gemiler);
        }

        do {
            printf ("\nYeniden oynamak ister misiniz? (E-H)");
            secenek = getche();
            if (secenek == 'H' || secenek == 'h') {
                sag_serbest = 0;
                secenek = 'E';
            }
        } while (secenek != 'E' && secenek != 'e');
        system("cls");

    }
    return 0;
}
```

int rakam_to_harf(int);

Rakam isimli değişkenden belirli bir harf alıp buna 65 ekleyerek alfabede bir harfin ASCII koduna dönüştürerek geri gönderir. Bu program 12*12'lik sabit bir oyun tablosu yerine sabitler ile tanımlanmış değiştirilebilir bir oyun alanına sahip olduğu için oyun alanını her satırının başındaki harfler bu fonksiyon ile gerçek zamanlı oluşturulur.

```
int rakam_to_harf (int rakam) {

    /* RAKAM degiskeninde gelen rakamin hangi harfe karsilik gel-
       digini geri gonderir.

       Ayrintili bilgi raporda bulunabilir.                        */

    rakam = rakam + 65;
    if (rakam > 90) {
        rakam = rakam%65;
    }

    return rakam;

}
```

int sifirla(int *)

Gemiler veri yapısında gelen dizinin tüm elemanlarını sıfırlar. Bu fonksiyon kullanıcı ikinci defa oynamayı seçtiğinde ya da rastgele gemiler yerleştirilemeyip de sıfırdan başlanılacağı zaman kullanılır.

```
void sifirla(int *gemiler) {

    int sayac;

    for (sayac=0;sayac<OYUNALANI*OYUNALANI;sayac++) {
        gemiler[sayac] = 0;
    }

}
```

int yerlestir(int *, int *, int *)

Gemi uzunluk dizisi oyuncudan alınan ve her geminin ne kadar uzunlukta olması gerektiğinin kaydının tutulduğu dizidir. Dizi GEMILIMIT + 1 eleman uzunluğundadır. Dizinin son elemanı her zaman 0'dır. Bu aşağıda da görebildiğiniz gibi while döngüsünden kesinlikle çıkmayı sağlar. Dizi her geminin uzunluğunu tutar. Oyun alanında olmayacak gemiler için değer 0'dır.

Yerleştir fonksiyonu gemiler veri yapısındaki diziye ve gemi durumu veri yapısındaki diziye gemi uzunluk dizisindeki bilgiye göre rastgele gemiler yerleştirir. Bu yerleştirme yatay ya da dikey olabilir ve yanlardan, üstten alttan ve çaprazdan gemilerin birbirine değmemesi şartı da burada kontrol edilmektedir.

Bu fonksiyonun kaynak kodu oldukça uzun olduğundan birkaç parçada verilmiştir.

```
int yerlestir(int *gemiuzunluk, int *gemiler, int *gemidurumu) {

    /* Gemileri oyun alanina yerlestirir.

       Ayrintili bilgi raporda bulunabilir.                        */

    srand((unsigned) time(NULL)); // Sistem saatine göre rastgele

    /* Degisken tanimlamalari BASLANGICI */
    int yon;
    int konum_yatay;
    int konum_dikey;
    int gemino;
    int sayac;
    int olmadi;
    int denemesayisi = 0;
    int denemesayisi_fatal = 0;
    /* Degisken tanimlamalari SONU */

    system("cls");
    printf("Gemiler rastgele yerlestirilirken lutfen bekleyin...\n");
    gemino = 0;
```

```
    while(gemiuzunluk[gemino] != 0) {

        denemesayisi++;

        if(denemesayisi>DENEMELIMIT) {
            denemesayisi_fatal++;
            denemesayisi = 0;
            sifirla(gemiler);
            gemino = 0;

            /* Rastgele yerlesme durumunu goruntulemek icin commenti kaldirin */
            // system("cls");
            // printf("Deneme: (%d/%d)\narama devam ediyor...",denemesayisi_fatal *
DENEMELIMIT,DENEMELIMIT*BASTANDENEMELIMIT);

            if(denemesayisi_fatal>BASTANDENEMELIMIT) {
                system("cls");
                printf("Uzgunuz, girdiginiz degerlere ait bir yerlesim yok.\n Devam etmek icin
bir tusa basin.");
                getche();
                return -1;
            }
        }

        yon = rand()%2;
        konum_yatay = rand()%(OYUNALANI);
        konum_dikey = rand()%(OYUNALANI);

        /* 0 = Kuzey-Güney doÄŸrultusu
           1 = DoÄŸu-BatÄ± doÄŸrultusu    */


        olmadi = 0;


        for(sayac=0;sayac<gemiuzunluk[gemino];sayac++) {

            if(konum_gecerli(eleman(konum_yatay+sayac,konum_dikey,yon))) { // Harita disina
tasiyorsa
                if (gemiler[eleman(konum_yatay+sayac,konum_dikey,yon)] != 0) { // Gemiyle
cakisiyosa
                    olmadi = 1;
                }
            } else {
               olmadi = 1;
            }

            if (konum_dikey%OYUNALANI != 0) { //en solda deÄŸilse
                if(konum_gecerli(eleman(konum_yatay+sayac,konum_dikey-1,yon))) {
                    if (gemiler[eleman(konum_yatay+sayac,konum_dikey-1,yon)] != 0) { // Solun-
da gemi var
                        olmadi = 1;
                    }
                }
            }

            if ((konum_dikey+1)%OYUNALANI != 0) { //en saÄŸda deÄŸilse
                if(konum_gecerli((eleman(konum_yatay+sayac,konum_dikey+1,yon)))) {
                    if (gemiler[eleman(konum_yatay+sayac,konum_dikey+1,yon)] != 0) { //
SaÄŸÄ±nda gemi var
                        olmadi = 1;
                    }
                }
            }
        }

        if (konum_dikey%OYUNALANI != 0) { //en solda deÄŸilse
            if(konum_gecerli(eleman(konum_yatay-1,konum_dikey-1,yon))) {
                if (gemiler[eleman(konum_yatay-1,konum_dikey-1,yon)] != 0) { // Sol Ã¼st
Ã§aprazda gemi var
                    olmadi = 1;
                }
            }
        }

        if ((konum_dikey+1)%OYUNALANI != 0) { //en saÄŸda deÄŸilse
            if(konum_gecerli(eleman(konum_yatay-1,konum_dikey+1,yon))) {
                if (gemiler[eleman(konum_yatay-1,konum_dikey+1,yon)] != 0) { // SaÄŸ Ã¼st
Ã§aprazda gemi var
                    olmadi = 1;
                }
            }
        }

        if(konum_gecerli(eleman(konum_yatay-1,konum_dikey,yon))) {
            if (gemiler[eleman(konum_yatay-1,konum_dikey,yon)] != 0) { // Ãœstte gemi var
                olmadi = 1;
```

```
            }
        }

        if(konum_gecerli(eleman(konum_yatay + gemiuzunluk[gemino],konum_dikey,yon))) {
            if (gemiler[eleman(konum_yatay + gemiuzunluk[gemino],konum_dikey,yon)] != 0) { //
AltÄ±nda gemi var
                olmadi = 1;
            }
        }

        if (konum_dikey%OYUNALANI != 0) { //en solda deÄŸilse
            if(konum_gecerli(eleman(konum_yatay + gemiuzunluk[gemino],konum_dikey-1,yon))) {
                if (gemiler[eleman(konum_yatay + gemiuzunluk[gemino],konum_dikey-1,yon)] != 0)
{ // Sol altÄ±nda gemivar
                    olmadi = 1;
                }
            }
        }

        if ((konum_dikey+1)%OYUNALANI != 0) { //en saÄŸda deÄŸilse
            if(konum_gecerli(eleman(konum_yatay + gemiuzunluk[gemino],konum_dikey+1,yon))) {
                if (gemiler[eleman(konum_yatay + gemiuzunluk[gemino],konum_dikey+1,yon)] != 0)
{ // SaÄŸ altÄ±nda gemivar
                    olmadi = 1;
                }
            }
        }


        if (olmadi != 1) { // gemi kurala uygunsa

            for(sayac=0;sayac<gemiuzunluk[gemino];sayac++) { // gemiyi belleÄŸe at
                gemiler[eleman(konum_yatay+sayac,konum_dikey,yon)] = 1;
                gemidurumu[GEMIGENISLIK*gemino + sayac] = ele-
man(konum_yatay+sayac,konum_dikey,yon);
            }

            denemesayisi = 0;
            gemino++;
        }

    }

    // cheat(gemiler); // Gemilerin yerleÅŸimini gÃ¶rmek iÃ§in bu satÄ±rÄ±n commentini
kaldÄ±rÄ±n.
    return 0;
}
```

Kullanıcı Kataloğu

Bir Ubenzer oyunu olan Amiral Battı 2008 oyununu tercih ettiğiniz için teşekkürler. Amiral Battı oyunu kullanım kılavuzu aradığınız bölüme daha kolay erişebilmeniz için başlıklar halinde sınıflandırılmıştır.

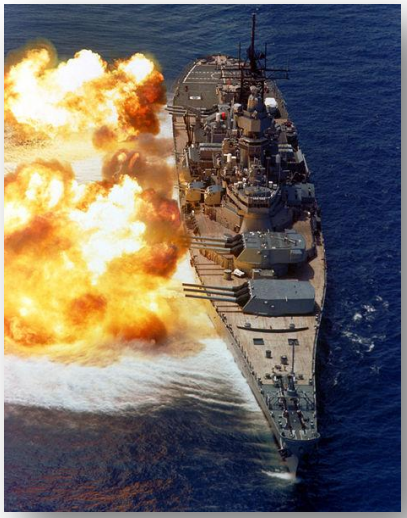
1. Kurulum
2. Kullanım
3. Kısıtlamalar
4. Çevrimiçi yardım

Kurulum

Programın derlenmiş hali Amiral Battı.exe isminde dağıtılmaktadır. Programı çalıştırmak için herhangi bir kurulum söz konusu değildir. Size ulaştırılan Amiral Battı.exe uygulamasına çift tıklayarak programı başlatabilirsiniz.

Programı kaynak koduyla çalıştırmak isteyen kullanıcılarımız ise kendilerine verilen ANSI C# kodunu uygun bir ANSI C# derleyicisinde derleyerek programı açabilirler. Popüler C# derleyicilerinin bir listesini Wikipedia'nın ilgili makalesinde bulabilirsiniz:
http://en.wikipedia.org/wiki/List_of_compilers#C.2FC.2B.2B_compilers (Bu makale İngilizcedir.)

Kullanım



"Geleceği görebilme yeteneğine sahip en, en, en iyi komutanlar bile gemileri batırmakta zorlandılar…"

Amiral Battı çok geniş bir kitleye hitap eden bir şans/zeka oyunudur. Oyunda belirli uzunluklarda ve belirli sayılarda gemiler vardır. Bu gemiler yatay ya da dikey yerleşmiş olabilirler. Sizin amacınız Kaptan C.nin yerleştirdiği bu gemileri atış hakkınız bitmeden bulmak ve imha etmek.

Amiral Battı 2008 alanında 12*12'lik bir oyun alanında savaşıyorsunuz. Oyunu başlattığınızda program size kaç tane gemi yerleştirmek istediğinizi soracak. 1 ile 10 arasında bir sayı yazabilirsiniz. Unutmamalısınız ki, aramanız gereken gemi sayısı arttıkça, hem vurma olasılığınız hem de size verilen atış hakkı arttığı için oyun kolaylaşmaktadır.

Programa kaç tane gemi istediğinizi yazdıktan sonra, oyun size her geminin kaçar birim uzunlukta olmasını istediğinizi tek tek soracak. Her gemi için bu uzunluk 1 ile 6 arasında olabilir.

Oyunumuz sizden bu bilgileri aldıktan sonra Kaptan C. gemileri oyun alanına *kesinlikle bulmayacağınız bir şekilde* yerleştirir.

Oyun alanının genel görüntüsü yandaki gibidir. Dikeydeki sayılar sütunları gösterir. 9'dan sonra gelen 0, 1 ve 2; 10,11 ve 12 anlamına gelmektedir.
Yataydaki harfler ise yataydaki satırın harfini belirtmektedir. Oyun alanındaki harflerin anlamları aşağıdaki gibidir:
O: Henüz atış yapılmamış alan
K: Karavana (atış yapılmış, ama gemi yokmuş)
I: İsabetli (gemi vurulmuş ama batmamış)
B: Batık gemi

Kaçıncı atış hakkında olduğunuzu yandaki kısımda görebilirsiniz: `Oyun: (14/120)` Oyun ekranında görebileceğiniz bu bilgilendirmede 14 kaçıncı atışta olduğunuzu, ikinci kısım 120 ise toplam kaç atış hakkınız olduğunu göstermektedir. Bu satırın bir altında en son yaptığınız atış hakkında bilgi verilir. Onun altındaki `Gemi 1/10` kısmında ise toplam kaç gemiden kaçının batırılmış olduğunun bilgisi verilmektedir. Program atış hakkınız bitene kadar ya da tüm gemileri batırana kadar sizden sürekli koordinat isteyecektir. Koordinatları harfrakam biçiminde yazmalı ve enter tuşuna basmalısınız. Unutmamalısınız ki koordinat girerken küçük harf **kullanmamalısınız**.

```
Oyun: (14/120)
Tebrikler isabetli atis.
Gemi 1/10
Atis yapmak istediginiz koordinati giriniz: A10
```

Atış yapmak istediğiniz koordinatı girdiğinizde oyun alanı güncellenecek ve yeni bir atış yapmanız istenecek. Eğer geçersiz bir konum girmişseniz sizden tekrar bir konum istenecek ama atış hakkınızdan sayılmayacak. Unutmamalısınız ki atış yaptığınız alana tekrar atış yapmanız sizin yaptığınız bir hatadır. Böyle bir durumda bir atış hakkınızı boşuna harcamış olacaksınız.

Eğer atış hakkınız bittiği halde tüm gemileri batırtmışsanız oyunu kaybetmişsiniz demektir. Ekranda yazanları inceler ve bir tuşa basarak ilerlerlerseniz batıramadığınız gemilerin nerelerde gizlendiğini görebilirsiniz. Bu ekranda G harfi isabet ettiremediğiniz gemileri gösterir.

Eğer atış hakkınız bitmeden tüm gemileri batırabilirseniz Kaptan C. sizi tebrik edecek.

Program size yeniden oynamak isteyip istemediğinizi sorduğunda E ya da H harfleriyle cevap verebilirsiniz.

Kolay gelsin. ☺

"Siz de kaybedeceksiniz kapatan… En iyi kaptanlar bile benim gemilerimi batıramadı."
Kaptan C

Kısıtlamalar

Program kullanılırken dikkat edilmesi gereken bazı önemli noktalar mevcuttur. Program bir ayrıtı en az 1 (iki önerilir) en fazla 26 birim olan oyun alanlarında oynanabilir. Oyun alanı kare olmak zorundadır. Oyun alanında en az bir birim uzunluğunda bir gemi olmak zorundadır. Kullanıcı koordinat girerken **büyük harf** kullanmak zorundadır. Küçük harfler geçersiz giriş olarak değerlendirilir. Programa yapılan tüm girdiler istenilen türde olmalıdır. Program aynı türdeki girdilerdeki hataları bulacak ve doğrusunu isteyecek şekilde tasarlanmıştır. Ancak programımızda değişken tipindeki uyuşmazlıktan oluşan hatalara karşı bir ayıklama mevut değildir. Söz gelimi, Gemi sayısı: asfj gibi bir girdi programın çalışmamasına neden olacaktır. Böyle bir şey yapıldığında programı tekrar çalışabilir hale getirmek için programı kapatıp tekrar başlatmalısınız.

Çevrimiçi Yardım

Eğer daha fazlasına ihtiyacınız varsa çevrimiçi yardım alabilirisiniz. Bunun için http://www.ubenzer.com/iletisim adresindeki iletişim formunu kullanarak bize ulaşın.