1. Labirent Tank Oyunu

İki kişi tarafından oynanan, her turda yeni bir labirent içerisinde mücadele edilen ve amacı, ateşlenen mermilerle rakibi vurmak olan bir rekabetçi oyundur. Oyuncular haritanın iki zıt köşesinde doğar. Denetleyiciye bağlı oyun kollarıyla tanklarını hareket ettirerek labirent içinde ilerlerler. Tanklar 4 çeşit harekete sahiptir: İleri gitme, geri gitme, saat yönünde dönme ve saat yönünün tersine dönme. Oyuncular ayrıca rakibi ve kendilerini vurabilen ve labirentin duvarlarından seken mermileri de ateşleyebilirler. Rakibi vurulan oyuncu bir puan kazanır ve belirlenen puana ulaşan oyuncu oyunu kazanmış olur.

2. Labirent Tank Oyunu Teknik Bilgiler

Labirent tank oyunu yazılım ve iki farklı donanım parçasından oluşur. Bu donanım parçaları oyunu çalıştıran STM3210C-EVAL denetleyici kartı ve bu karta bağlı olup üzerlerinde beşer adet buton bulunduran iki adet oyun kolundan oluşur. Oyunun ilerleyişi denetleyici kartına bağlı olan LCD ekrandan takip edilir. Yazılım tarafında ise tankların bilgilerini tutan yapı değişkeni, tankların hareketlerini kontrol eden fonksiyonlar, harita içerisinde ilerleyen mermilerin yapı değişkeni ve fonksiyonları, haritayı oluşturmak için rastgele labirent algoritması ve son olarak da ekrana çizdirme fonksiyonları vardır.

3. Donanım

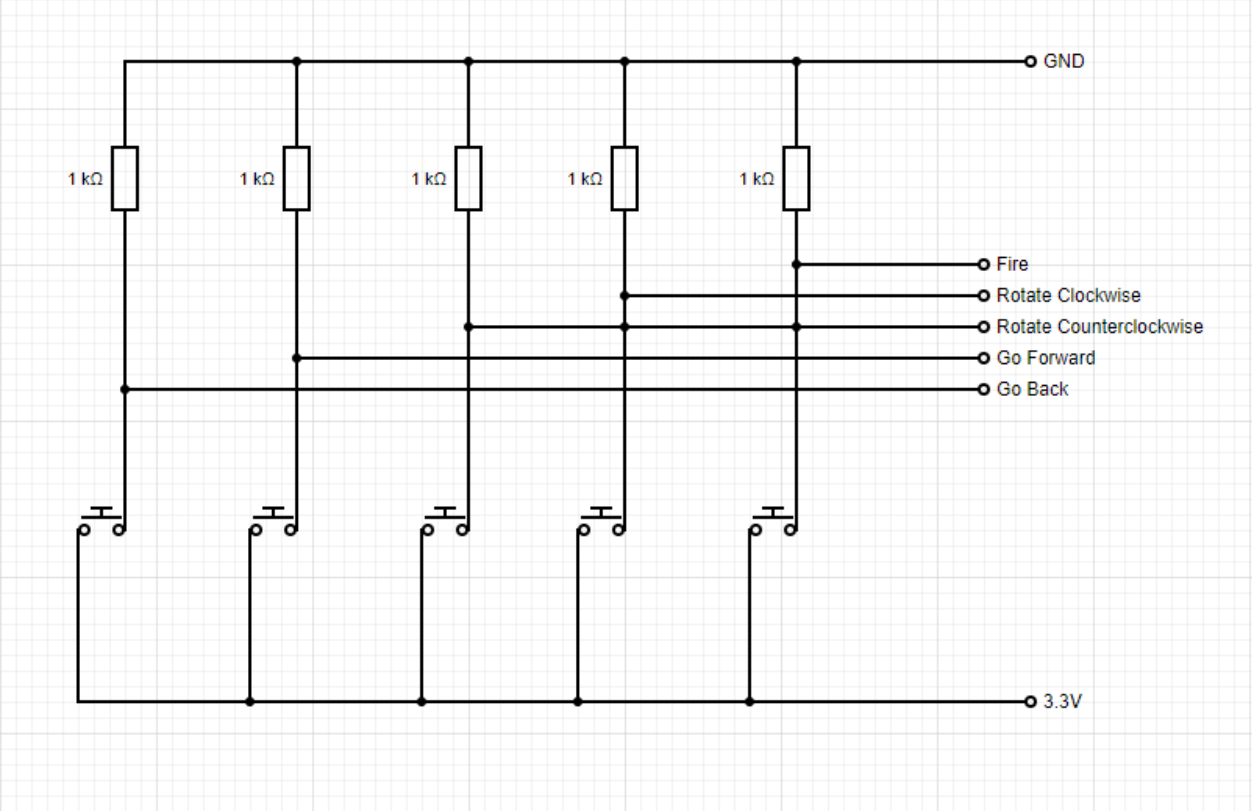
3.1. STM3210C-EVAL Denetleyici Kartı

STM32 Değerlendirme Kiti STM3210C-EVAL, STMicroelectronic'in ARM Cortex-M3 çekirdek tabanlı STM32F107VCT mikrodenetleyici için eksiksiz bir geliştirme platformudur. STM32F107VC MCU içerir.

Kart üzerindeki donanım özellikleri, tüm çevre birimlerini (USB-OTG FS, ethernet, motor kontrolü, CAN, microSD CardTM, akıllı kart, LCD Ekran, USART, ses DAC, MEMS, EEPROM ve daha fazlası) değerlendirmenize ve kendi uygulamalarınızı geliştirmenize yardımcı olur. Uzatma başlıkları, özel uygulamanız için bir daughterboard veya wrapping board bağlamayı kolaylaştırır.

3.2. Oyun Kolu

Breadboard üstüne başlanmış 5 adet buton bulundurur. Bu butonlar denetleyici kartımızın yazılımla ayarlanabilen pinlerine bağlanmıştır. İki adet bulunan bu oyun kolları tank fonksiyonlarının gerçekleştirilmesini sağlar.



4. Yazılım

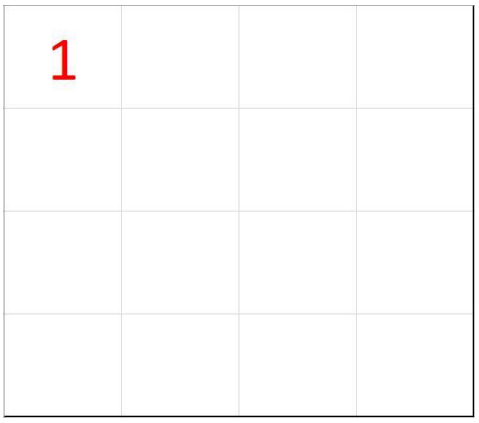
4.1. Harita

Haritamız her yeni skorda yenilenir. Tanklar ve mermilerin geçemeyeceği duvarlarla çevrelenmiştir.

Labirent üretme algoritması için iki boyutlu biz dizi oluşturup her bir bloğuna ilerleme yönümüze bağlı sayılar yerleştirerek bütün blokları sayılarla doldurmaya çalışıyoruz. Bütün bloklar bir değere sahip olduğunda ise değerleri arasındaki fark 1'den büyük olan bloklar arasına duvar çekerek labirenti oluşturuyoruz. Daha sonra bu diziyi oran olarak genişleterek harita boyutuna eşitliyoruz. Böylece düşük boyuttaki bir dizi ile 320x240 boyutunda bir ekrana labirent uyarlanmış olur.

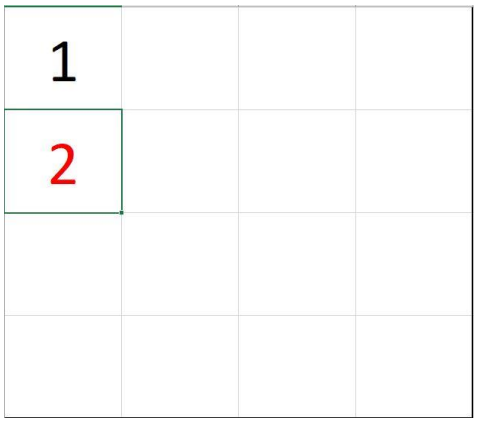
Aşağıdaki 12 görsel labirent üretme algoritmasının işleyişini göstermektedir. Kırmızı ile yazılmış sayılar o an üzerinde durulan bloğu, siyah ile yazılmış sayılar ise daha önce yazılmış blokları belirtir.

İlk adım olarak dizimizin [0,0] elemanına başlangıç değerimiz olan 1 sayısını yazıyoruz. Daha sonra ilerlenebilecek komşu blokları (Üst, Sol, Alt ve Sağ) belirliyoruz. Müsait komşu bloklardan birini rastgele belirliyoruz. Mevcut değerimizin bir fazlasını ilerlediğimiz bloğa yazıyoruz.



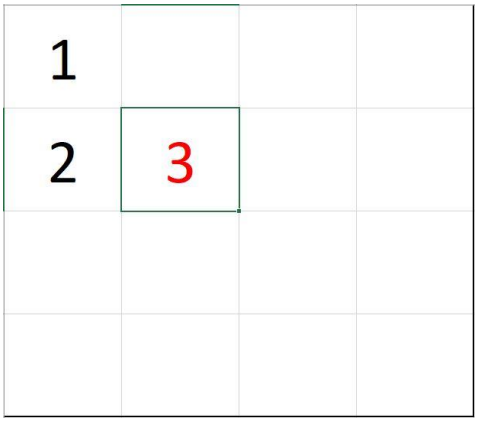
Değer:1

Mevcut Komşuluklar: Alt ve Sağ



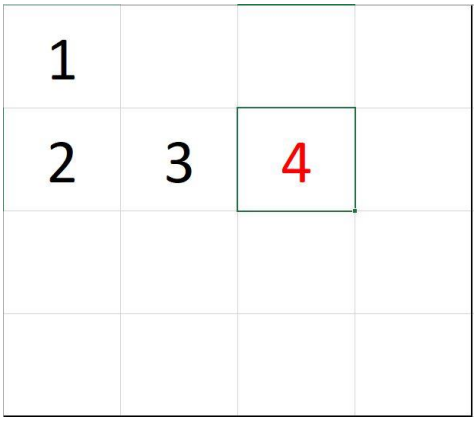
Değer: 2

Mevcut Komşuluklar: Alt ve Sağ



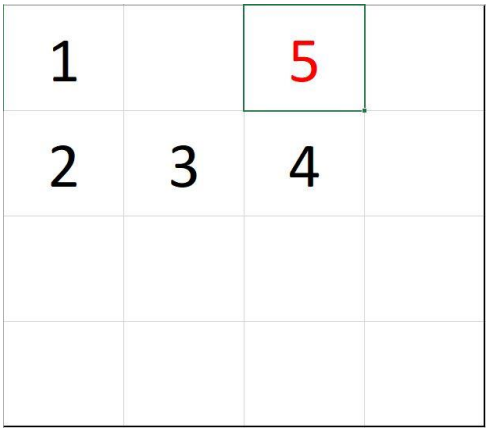
Değer: 3

Mevcut Komşuluklar: Üst, Alt ve Sağ



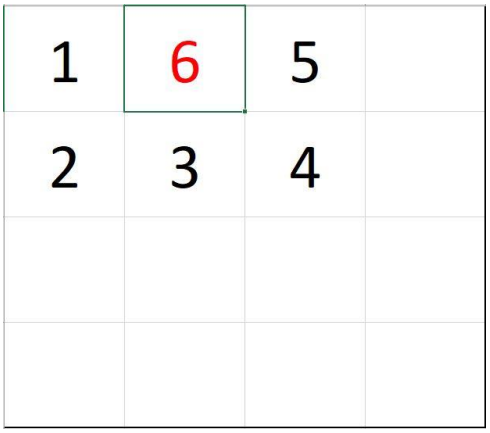
Değer: 4

Mevcut Komşuluklar: Üst, Alt ve Sağ



Değer: 5

Mevcut Komşuluklar: Sol ve Sağ



Değer: 6

Mevcut Komşuluklar: Yok

Son görselimizde mevcut bir komşuluğun olmadığı görülmektedir. Bu durumlarda üzerinde bulunduğumuz sayının bir eksiğine dönüp herhangi bir komşuluk bulana kadar bu işleme devam ederiz.

1	6	5	
2	3	4	

Değer: 5

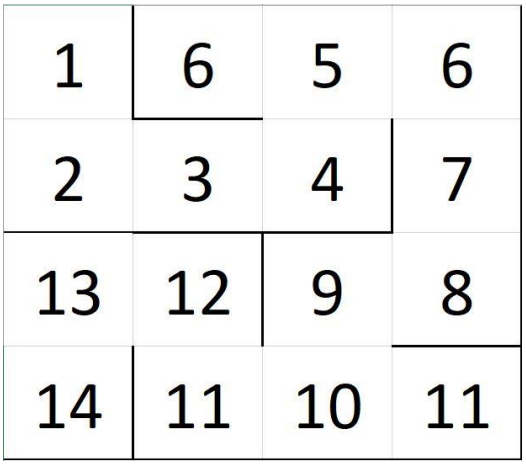
1	6	5	6
2	3	4	

Değer:6

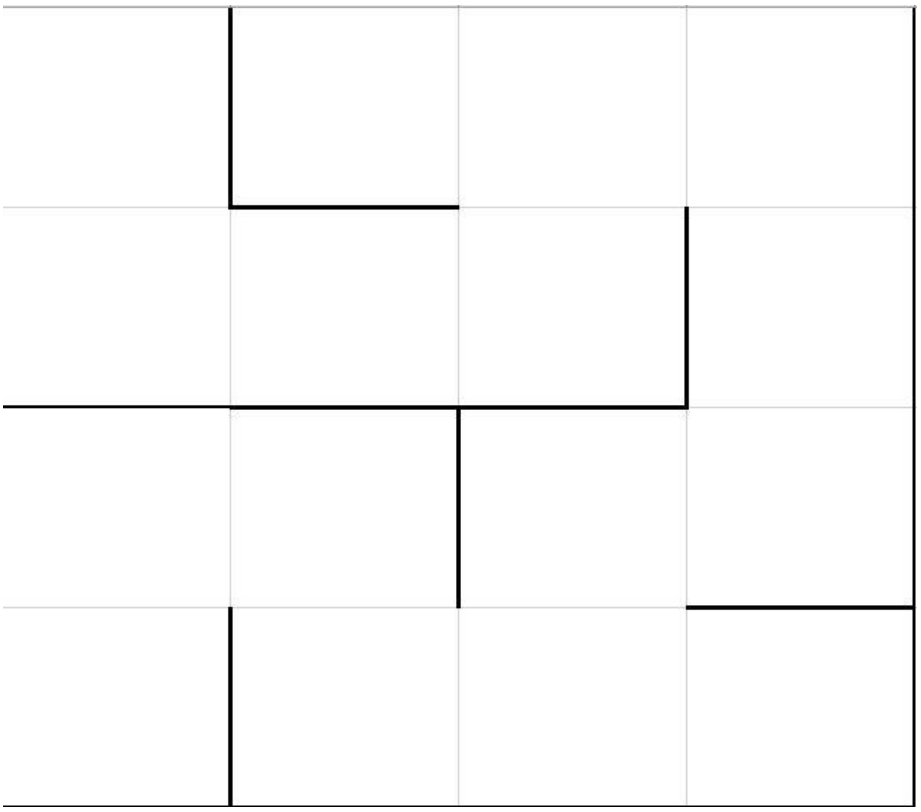
1	6	5	6
2	3	4	7

Bu işlemlere bütün harita dolana kadar devam ederiz.

1	6	5	6
2	3	4	7
13	12	9	8
14	11	10	11



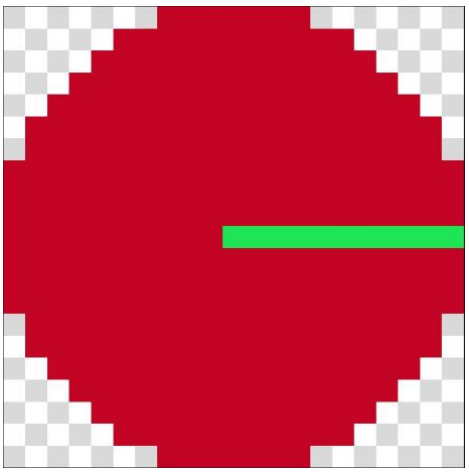
Bütün harita dolduğunda ise aralarındaki fark 1'den büyük olan bütün blokların arasına duvar çekilerek harita oluşturulmuş olur.



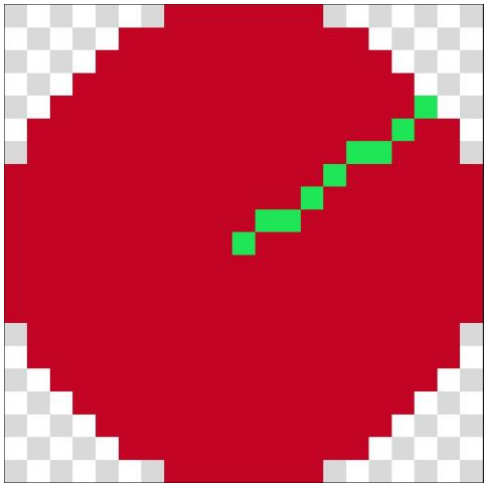
Rastgele oluşturulmuş bir harita

4.2. Tank

Oyunumuzda 360 derece dönebilen ve ileri-geri hareket edebilen 2 adet tank bulunur. Her bir tankın haritadaki konumunu belirten X ve Y koordinatlarıyla beraber bir de mermilerin yönünü belirlemeye yarayan namlu uç nokta koordinatları bulunur. Oyun kolundaki butonlardan aldığımız verilere göre bu hareketler kontrol edilir. Tankların hareketleri duvarlarla sınırlandırılmıştır. Tanklar hareket ederken duvarlardan geçemez.



Tamamen sağ tarafa yönelmiş bir tank



Saat yönünün tersine döndürülmüş bir tank

```c
typedef struct Tank{
	uint8_t ID;
	char name[11];
	uint16_t xPos;
	uint16_t yPos;

	uint8_t radius;

	short xPosBarrel;

	short yPosBarrel;


	bool turnClockwise;
	bool turnCounterClockwise;
	bool goForward;
	bool goBack;
	bool fire;

	uint32_t color;

	GPIO_TypeDef* gpioPort;
	uint16_t pins[6];

	bullet bullets[MAX_BULLET_COUNT];

	uint8_t score;
}Tank;
```

Tanklarımızın xPos ve yPos değişkenleri merkez noktasının haritadaki konumunu belirtir.

xPosBarrel ve yPosBarrel ise namlunun tankın merkezine göre konumunu belirtir.

Tamamen sağ tarafa yönelmiş bir tankta xPosBarrel tankın yarıçap değerine (ör:10) eşitken yPosBarrel 0 değerine eşittir.

Tank döndürülürken xPosBarrel değeri değiştirilir, daha sonra yPosBarrel değeri dik üçgenin kenar uzunluk bağıntısıyla hesaplanır.

Örneğin tamamen sağ tarafa yönelmiş ve yarıçap uzunluğu 10 olan bir tank saat yönünün tersine döndürüldüğünde xPosBarrel değeri azalır. Daha sonra $Yarıçap^2 = x^2 + y^2$ eşitliği kullanılarak yPosBarrel değeri hesaplanır.

```c
void ButtonInput(Tank *tank)
{
	tank->goBack              = HAL_GPIO_ReadPin(tank->gpioPort, tank->pins[0]);
	tank->goForward           = HAL_GPIO_ReadPin(tank->gpioPort, tank->pins[1]);
	tank->turnClockwise       = HAL_GPIO_ReadPin(tank->gpioPort, tank->pins[2]);
	tank->turnCounterClockwise= HAL_GPIO_ReadPin(tank->gpioPort, tank->pins[3]);
	tank->fire                = HAL_GPIO_ReadPin(tank->gpioPort, tank->pins[4]);
}
```

Tankların hareketleri ButtonInput() fonksiyonuyla belirlenir. Parametre olarak aldığı tankın pinlerini kontrol ederek hangi hareketlerin aktif edildiğini kontrol eder.

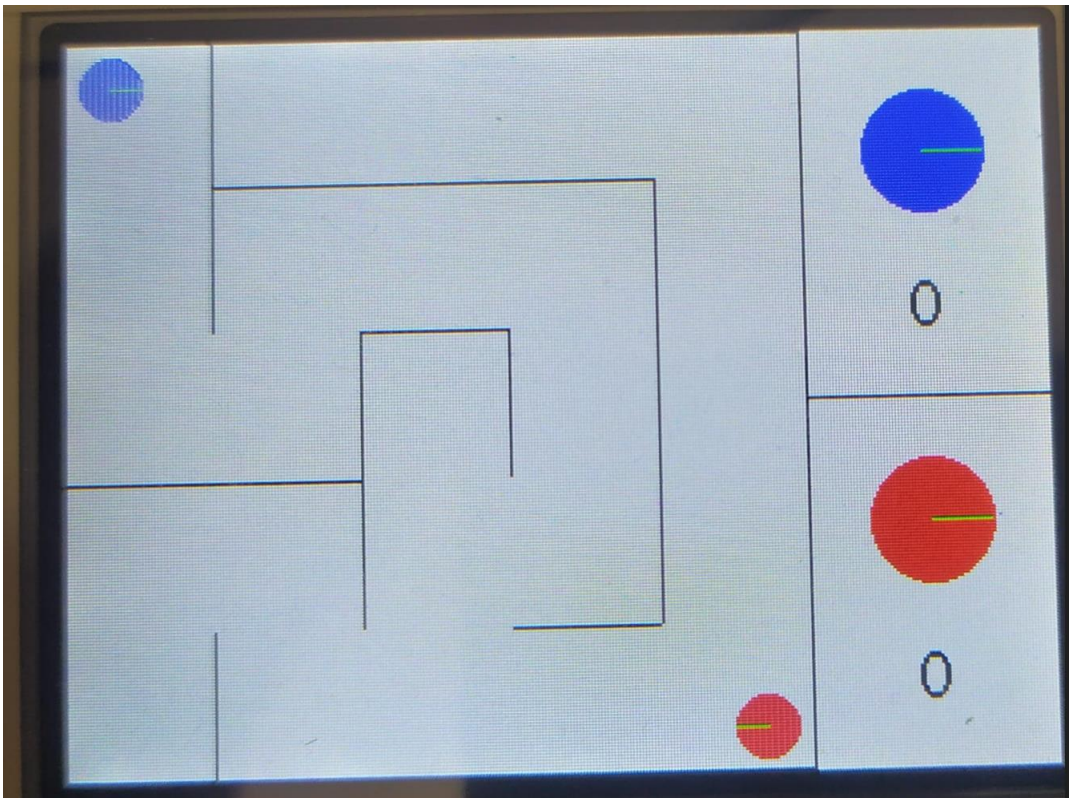
4.3. Mermi

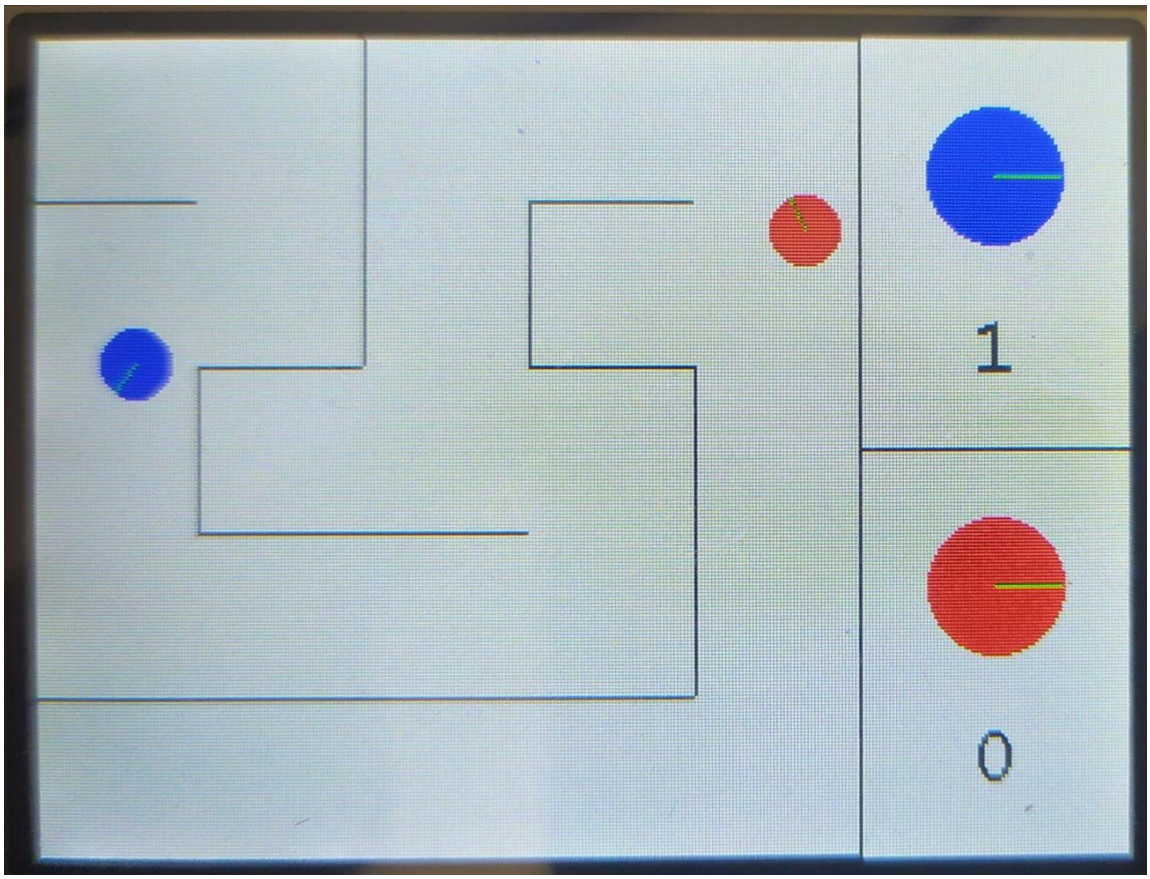
Her bir tankın maksimum sayısı kodla belirlenmiş mermi adeti bulunmaktadır. Mermiler ateşlendiğinde tankın merkez noktasından namlunun uç noktasına olan doğrultuda hareket ederler. Mermiler duvarları aşamayacağı gibi çarptıklarında zıt yöne sekip hareketlerine devam ederler. Mermilerin bir tank ile çarpışması durumunda rakip oyuncu bir puan kazanır.

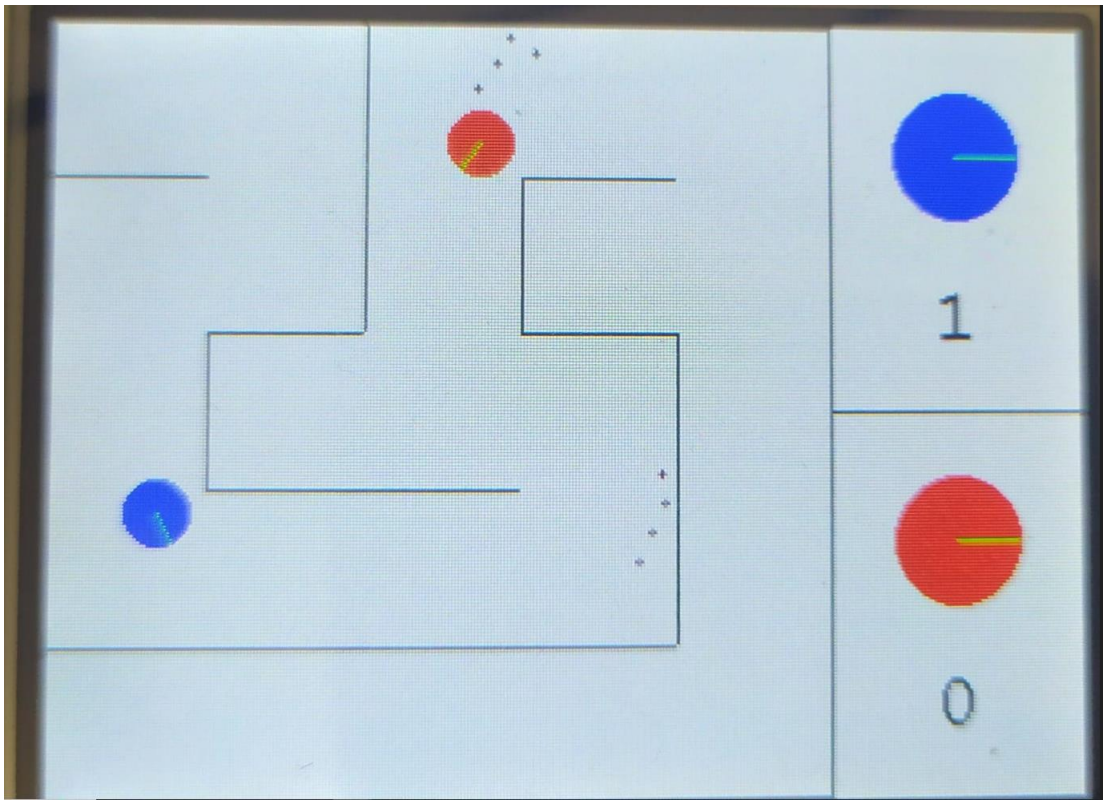
```
typedef struct bullet{
      int xPos;
      int yPos;
      short xVelocity;
      short yVelocity;
      uint8_t lifeTime;
   }bullet;
```

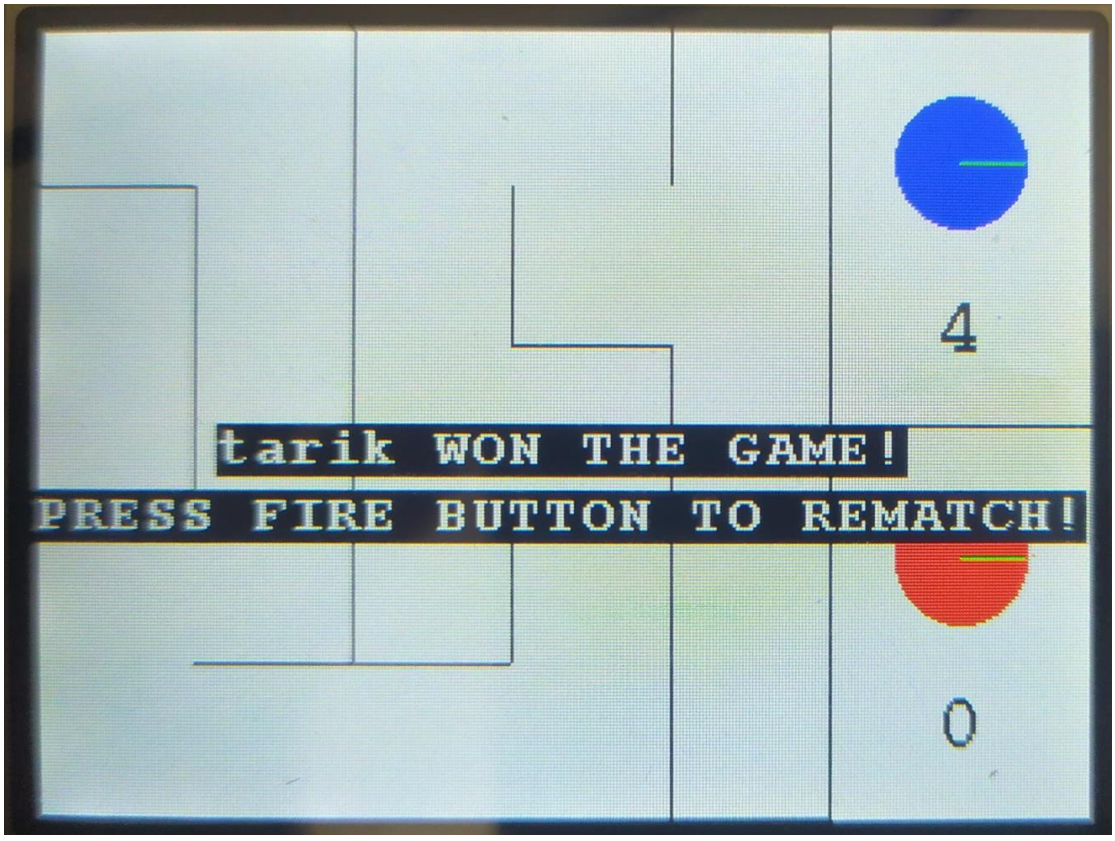
Mermilerin de tanklar gibi haritadaki konumunu belirten xPos ve yPos değerleri vardır. Ayrıca ateşlendiklerinde hareket yönlerini ve hızlarını belirten xVelocity ve yVelocity değerleri de vardır. lifeTime değişkeni ise merminin haritada kalacağı süreyi belirleyen değişkendir.

5. Oyun Görüntüleri




1
0


1
0


4
tarik WON THE GAME!
PRESS FIRE BUTTON TO REMATCH!
0