

Class Diagram

+MaxCell: const int

-ReStart()

-safeValue: const int

-mousePositioni: Point

-GameOver(sender object, e: EventArgs)

Form

-Form1_MouseMove(sender: object, e: MouseEventArgs)

-Form1 MouseDown(sender: object, e: MouseEventArgs)

CircleQueue

- +Front: int
- -direction: Direction
- -boundaries: Rectangle
- +Draw(g: Graphics) +Move(): bool

+Location: Point

+ActivateShot(location: Point)

Shot -moveInterval: const int

- +Count: int

- +Rear: int
- +Size: int
- -arr: Shot
- +ModifySize(size: int)
- +Insert(shot: Shot): bool
- +Reset()
- +GetData(num: int): Shot
- +Push(location: Point): bool
- +Pop(): bool
- +lsFull(): bool
- +IsEmpty(): bool

PlayerShip

- +Interval: const int
- -image: Bitmap

Game

-ReturnFire(attacker: Invader, random: Random): bool

-MaxShot: const int

-lineChange: bool

+EventCall()

+FireShot() +Go()

-invaderMove() -ShotCtrl() -CollideCheck()

-CreateInvaders() +NextWave() +TryAgain()

-MaxWave: const int

-invaderDirectioni: Direction

+GameOverEvent(e: GameOverArgs)

+Draw(g: Graphics, animationCell: int) +MovePlayer(direction: Direction) +GetPlayerLocation(): Point

-prevDirection: Direction

-invaders: List<invader>

-playerShip: PlayerShip -playerShots: CircleQueue invaderShots: CircleQueue

- +Area: Rectangle
- -boundaries: Rrectangle
- +Location: Point
- +Draw(g: Graphics)
- +Move(direction: Direction)

EventArgs GameOverArgs

Invader

- -HorizontalInterval: const int
- -VerticalInterval: const int
- -image: Bitmap
- +Location: Point
- +InvaderType: ShipType
- +Area: Rectangle
- -boundaries: Rectangle
- +Score: int
- +Move(direction: Direction)
- +Draw(g: Graphics, animationCell: int)
- -InvaderImage(animationCell: int)

enum

```
enum InvaderRow // x enum Col // y
enum ShipType
   // name = score
                          Bug = 50,
   Bug = 50,
                                                  First = -140,
   Saucer = 40,
                          Saucer = 110,
                                                  Second = -70,
   Satellite = 30,
                                                  Third = 0,
                          Satellite = 170,
   Spaceship = 20,
                                                  Forth = 70,
                          Spaceship = 230,
   Star = 10
                                                  Fifth = 140
                          Star = 290
```

new Game()

```
public Game(Rectangle boundaries, Random random)
{
    this.boundaries = boundaries;
    this.random = random;

    playerShots = new CircleQueue(MaxShot);
    invaderShots = new CircleQueue(MaxWave);

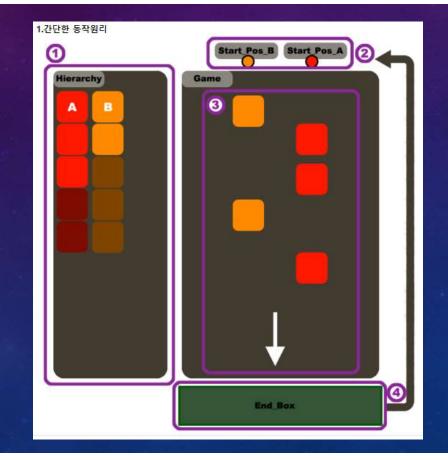
    playerShip = new PlayerShip(new Point(boundaries.Width / 2, boundaries.Height - 90), boundaries);
    invaders = new List<Invader>();
    CreateInvaders();

    // 미사일 미리 생정해서 숨겨두기
    for (int i = 0; i <= MaxShot; i++)
        playerShots.Insert(new Shot(new Point(-100, -100), Direction.Up, boundaries));
    playerShots.Reset(); // rear를 다시 front 위치로

    for (int i = 0; i <= MaxWave + 3; i++)
        invaderShots.Insert(new Shot(new Point(-100, -100), Direction.Down, boundaries));
    invaderShots.Reset();
```



객체를 필요로 할때 풀에 요청을 하고, 반환하고 일련의 작업을 수행하는 패턴.
 많은 수의 인스턴스를 생성할때 혹은 무거운 오브젝트를 매번 인스턴스화 할때 성능 향상을 가져오기도 합니다.
 예를들어, 데이터베이스에 접속하는 여러 객체를 만들때 매번 새로 생성하는 것보단,
 미리 생성된 풀에서 객체를 반환받아오는 것이 더 이득 입니다.



Game.Draw()

```
public void Draw(Graphics g, int animationCell)
{
   playerShip.Draw(g);
   for (int i = 0; i < invaders.Count; i++)
        invaders[i].Draw(g, animationCell);

   for (int i = playerShots.Front; i != playerShots.Rear; i++, i %= playerShots.Size)
        playerShots.GetData(i).Draw(g);

   for (int i = invaderShots.Front; i != invaderShots.Rear; i++, i %= invaderShots.Size)
        invaderShots.GetData(i).Draw(g);

   Twinkle();
}</pre>
```

Game.Go()

```
private void InvaderMove()
   /* ======= 동작 처리 ========= */
   var enemies = from invader in invaders
                 where invader.Location.X <= 10
                 II invader.Location.X >= boundaries.Width - 80
                 select invader;
    if (lineChange)
       lineChange = false;
       invaderDirection = (prevDirection == Direction.Left) ? Direction.Right : Direction.Left;
    // 라인을 바꿔야할 때
    else if (enemies.Count() != 0)
       lineChange = true;
       prevDirection = invaderDirection;
       invaderDirection = Direction.Down;
   for (int i = 0; i < invaders.Count; i++)</pre>
       invaders[i].Move(invaderDirection);
```

Game.Go()

```
invaders. Add(new landars invaders. Add(new landars invaders. Add(new landars invaders. invaders. Add(new landars invaders. invaders. Add(new landars invaders. Add(new landars invaders. Add(new landars invaders. Add(new landars invaders. Add(new landars) invaders. Add(new landars invaders. Add(new landars) invaders. Add(new landars invaders. Add(new landars) invaders.
```

```
invaders.Add(new Invader(ShipType.Bug, new Pointinvaders.Add(new Invader(ShipType.Saucer, new Finvaders.Add(new Invader(ShipType.Saucer, new Finvaders.Add(new Invader(ShipType.Satellite, new Invaders.Add(new Invader(ShipType.Spaceship, new Invader(ShipType.Spaceship)
```

MouseEvent

```
private const int AdjustmentValue = 10;
private Point mousePosition;

private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    mousePosition = e.Location;
}

private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    game.FireShot();
}
```

부족한 점

- Twinkle() 구현
- Invader 생성 시 Factory 패턴 사용
- 하나의 Invader가 여러 번 공격하는 문제
- QueueLoop을 메소드로 구현
- Shot 충돌 시 효과 넣기