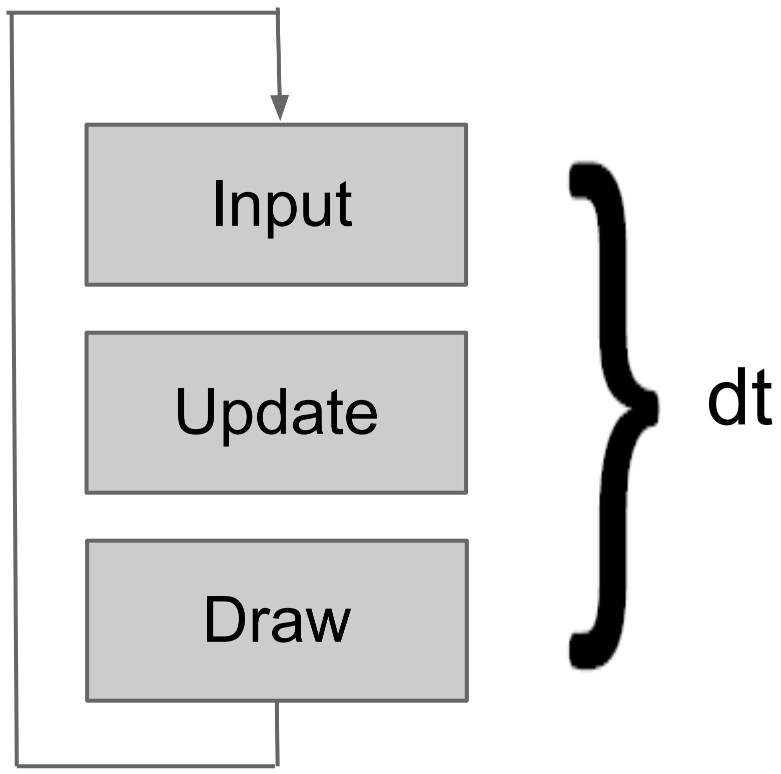
Game Programming

Game Structure



Drawing (Rendering)

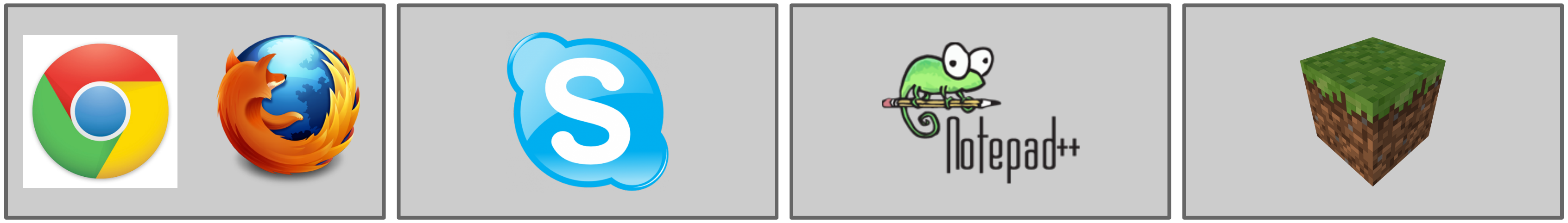
Hardware
Acceleration.

Mainly for 3D.

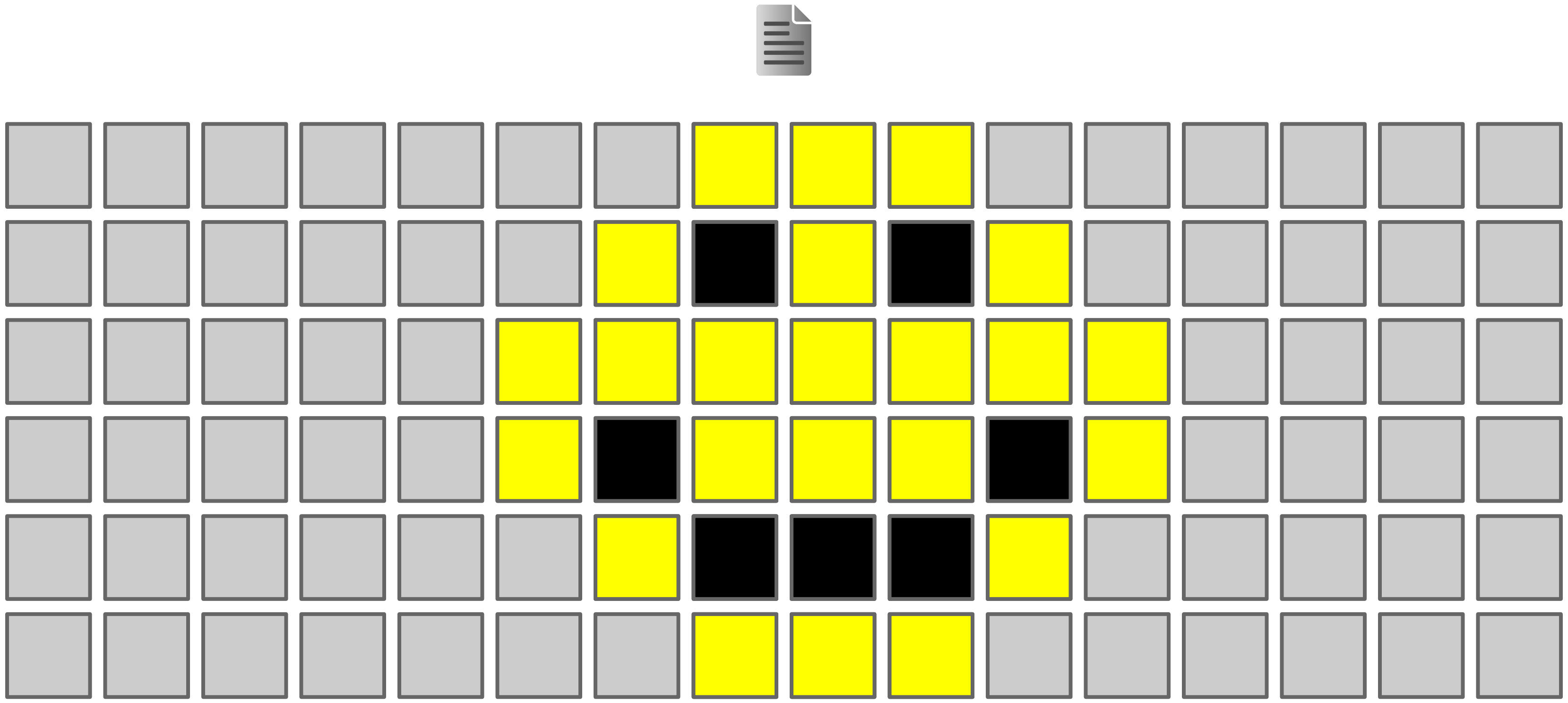
Lots of cores.

Same Program.

CPU



GPU



Programming

Pick a strong language.

- Games need to fast.
- Lots of objects.
- Lots of files.
- Meaning Lots of organization.

Libraries

C# - XNA / SlimDX / OpenTK

C++ - SDL, SFML

Python - PyGame

Java - LWJGL

Free Art Tools

2D

- Gimp
- Paint.NET

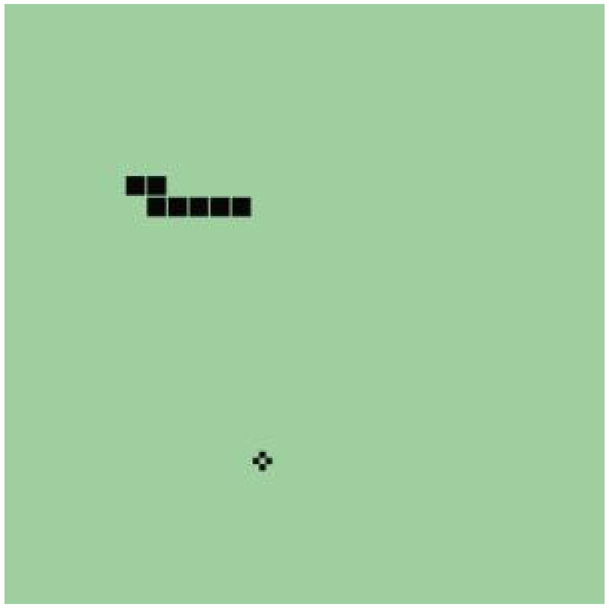
3D

- Blender

Sound

- Audacity
- BXFR

Example: Snake



Core Objects

Snake

- Collection of Segments
- Movement

SnakeSegment

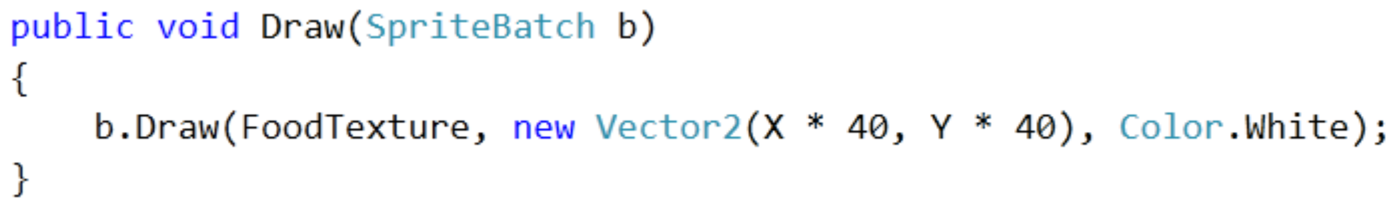
- Drawing

Food

- Moves randomly.

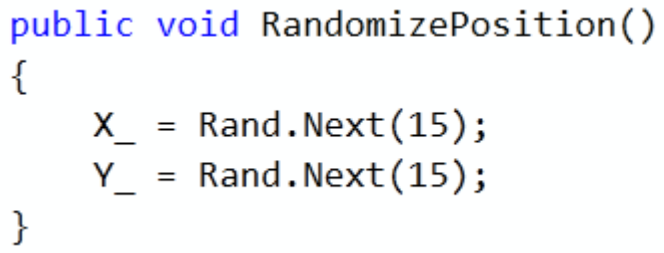
Food

Draw



```
public void Draw(SpriteBatch b)
{
    b.Draw(FoodTexture, new Vector2(X * 40, Y * 40), Color.White);
}
```

Randomize



```
public void RandomizePosition()
{
    X_ = Rand.Next(15);
    Y_ = Rand.Next(15);
}
```

SnakeSegment

Members

```
static SpriteSheet SnakeSheet;
public static void SetSheet(Texture2D texture) { SnakeSheet = new SpriteSheet(texture, 40, 40); }


int X_, Y_;

public int X { get { return X_; } set { X_ = value; } }
public int Y { get { return Y_; } set { Y_ = value; } }


SegmentType Type_;
public SegmentType Type { get { return Type_; } set { Type_ = value; } }

Direction Direction_;
public Direction Dir { get { return Direction_; } set { Direction_ = value; } }

SnakeSegment Next_;
public SnakeSegment Next { get { return Next_; } set { Next_ = value; } }
SnakeSegment Previous_;
public SnakeSegment Previous { get { return Previous_; } set { Previous_ = value; } }
```

```
enum SegmentType
{
    Head,
    Segment,
    Tail
}
```

Snake Movement

Snake segments update from back to front.

Snake Movement

Snake segments update from back to front.

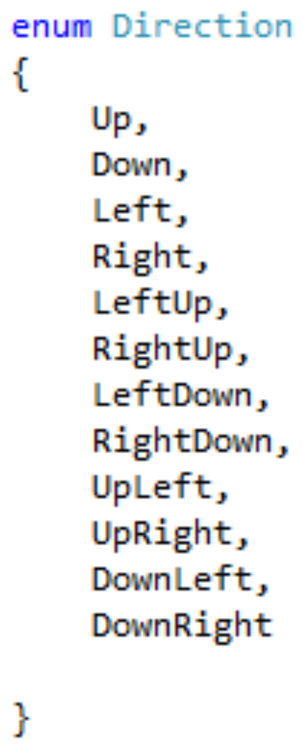
Snake Movement

Snake segments update from back to front.

Snake Movement

Snake segments update from back to front.

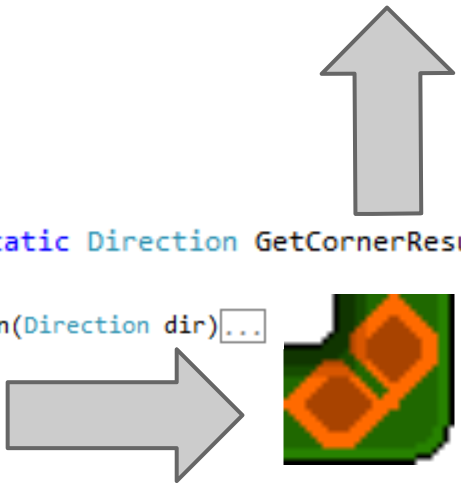
Direction



```
enum Direction
{
    Up,
    Down,
    Left,
    Right,
    LeftUp,
    RightUp,
    LeftDown,
    RightDown,
    UpLeft,
    UpRight,
    DownLeft,
    DownRight

}
```

```
public static Direction GetCornerResult(Direction dir)...
```

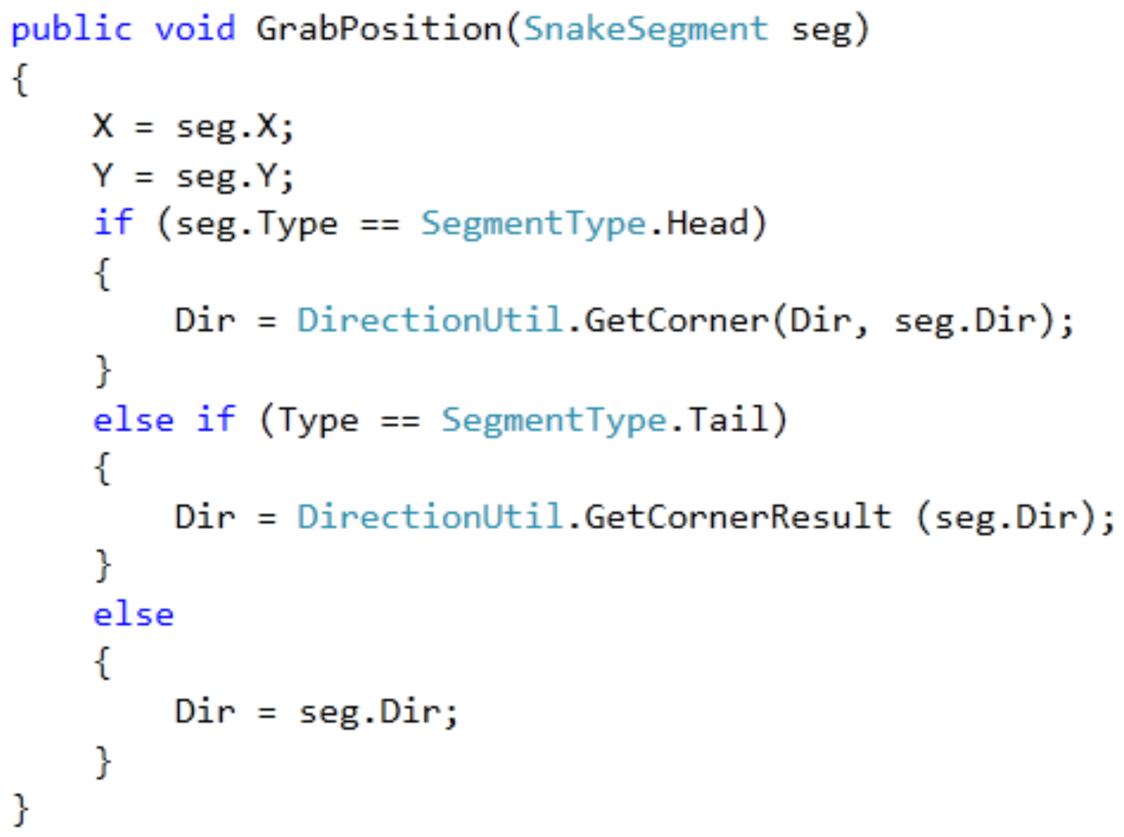
```
public static Direction GetCornerConnection(Direction dir)...
```



RightUp

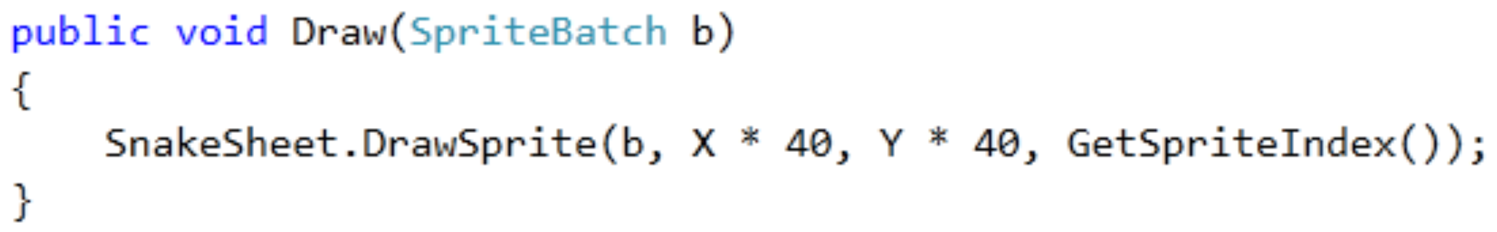
SnakeSegment

GrabPosition



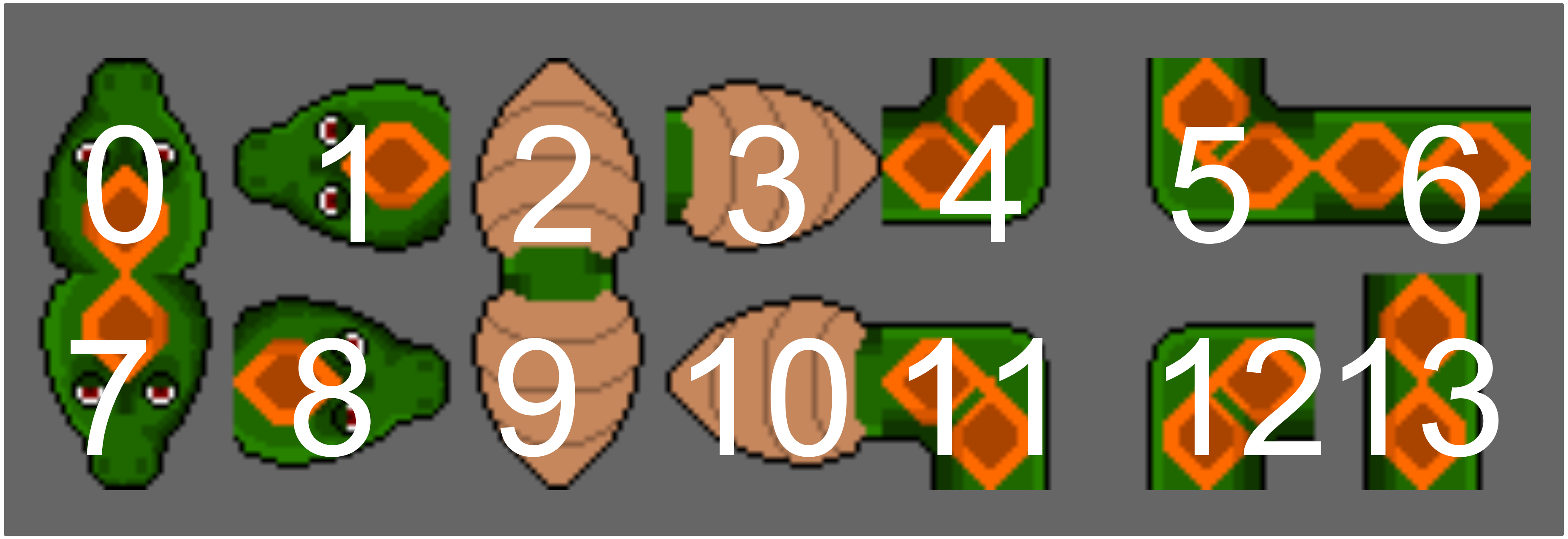
```
public void GrabPosition(SnakeSegment seg)
{
    X = seg.X;
    Y = seg.Y;
    if (seg.Type == SegmentType.Head)
    {
        Dir = DirectionUtil.GetCorner(Dir, seg.Dir);
    }
    else if (Type == SegmentType.Tail)
    {
        Dir = DirectionUtil.GetCornerResult (seg.Dir);
    }
    else
    {
        Dir = seg.Dir;
    }
}
```

SnakeSegment



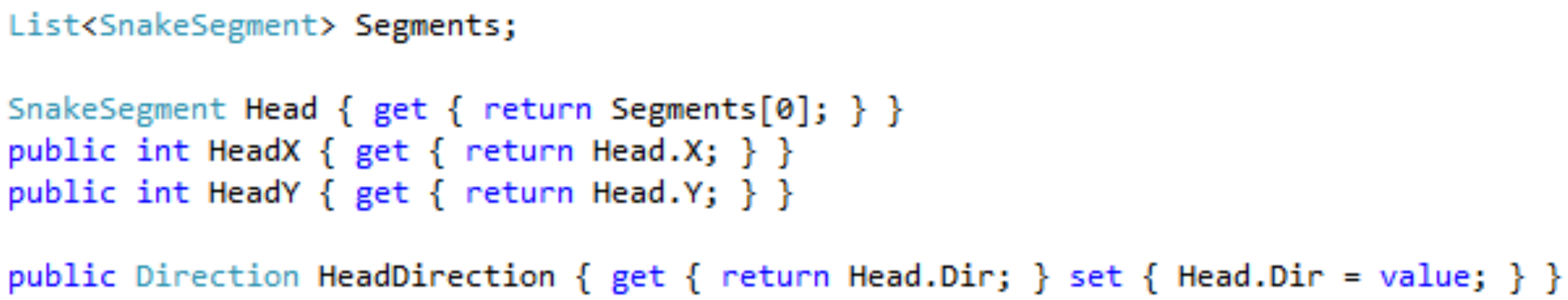
```
public void Draw(SpriteBatch b)
{
    SnakeSheet.DrawSprite(b, X * 40, Y * 40, GetSpriteIndex());
}
```

SpriteSheet



Snake

Members

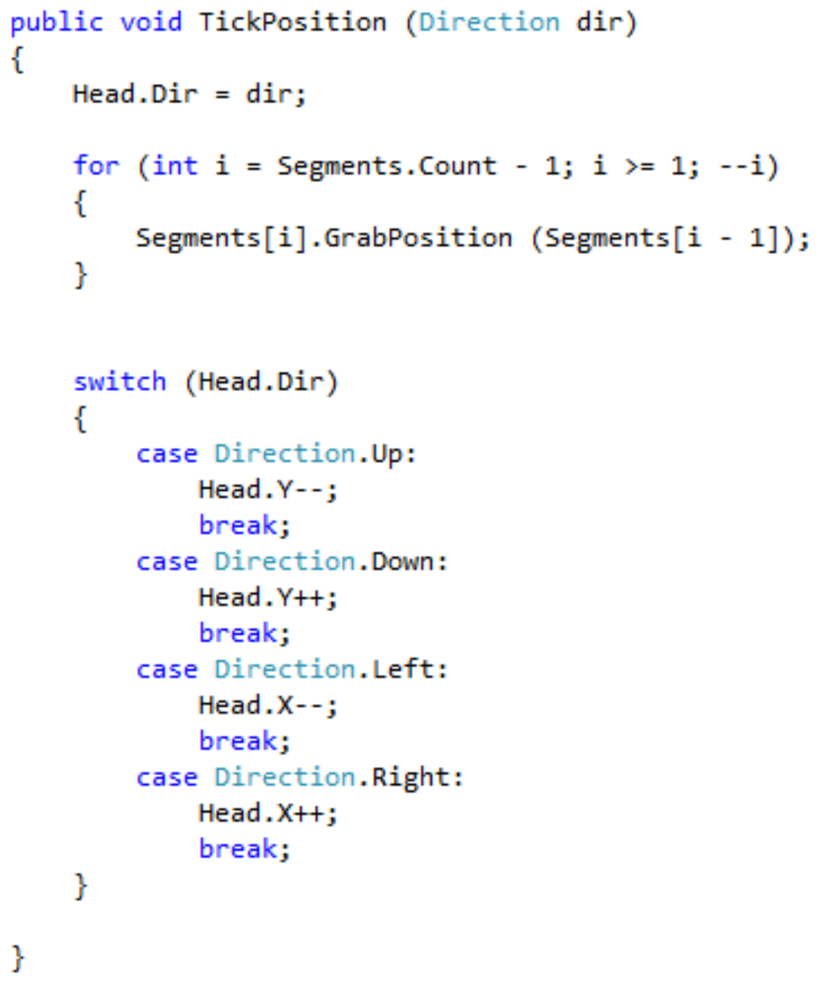


```
List<SnakeSegment> Segments;

SnakeSegment Head { get { return Segments[0]; } }
public int HeadX { get { return Head.X; } }
public int HeadY { get { return Head.Y; } }

public Direction HeadDirection { get { return Head.Dir; } set { Head.Dir = value; } }
```

Snake

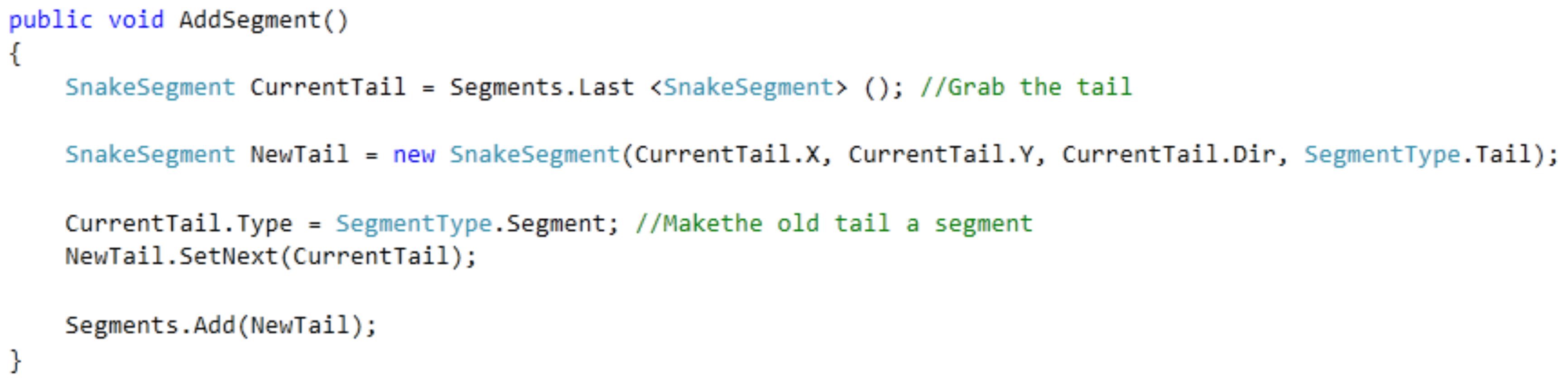
```
public Snake()
{
    Segments = new List<SnakeSegment>();
    Segments.Add(new SnakeSegment(10,10, Direction.Left, SegmentType.Head));
    Segments.Add(new SnakeSegment(11,10, Direction.Left));
    Segments.Add(new SnakeSegment(12,10, Direction.Left, SegmentType.Tail));
    Segments[1].SetNext(Segments[0]);
    Segments[2].SetNext(Segments[1]);
}
```

Snake



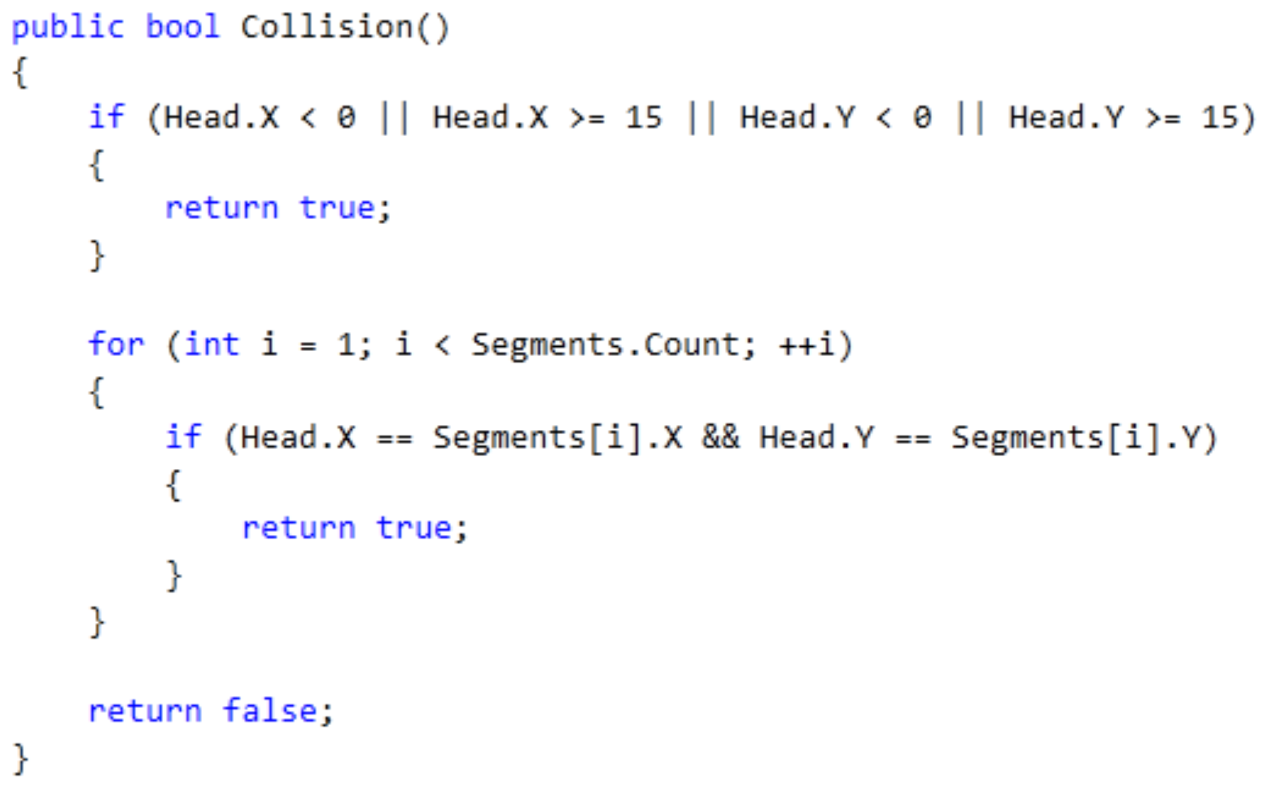
```
public void TickPosition (Direction dir)
{
    Head.Dir = dir;

    for (int i = Segments.Count - 1; i >= 1; --i)
    {
        Segments[i].GrabPosition (Segments[i - 1]);
    }


    switch (Head.Dir)
    {
        case Direction.Up:
            Head.Y--;
            break;
        case Direction.Down:
            Head.Y++;
            break;
        case Direction.Left:
            Head.X--;
            break;
        case Direction.Right:
            Head.X++;
            break;
    }

}
```

Snake



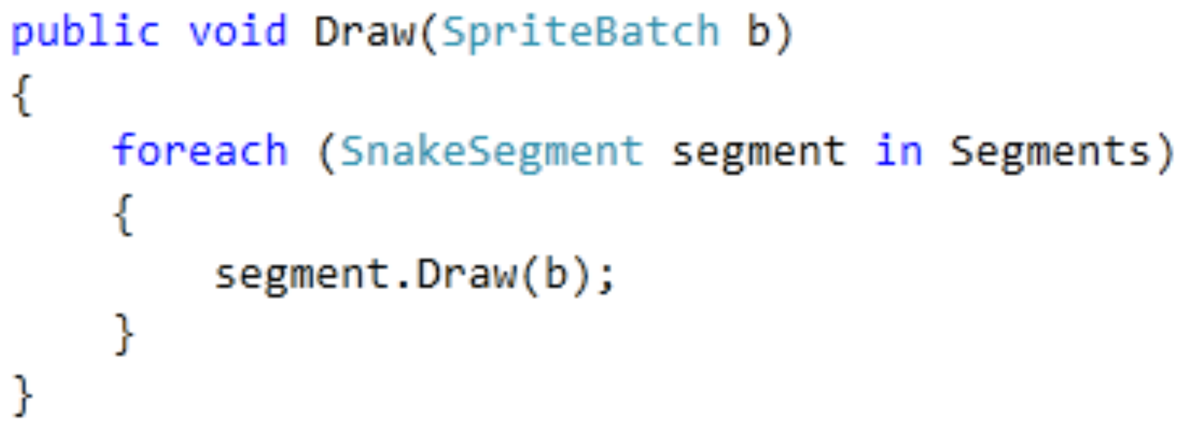
```
public void AddSegment()
{
    SnakeSegment CurrentTail = Segments.Last <SnakeSegment> (); //Grab the tail

    SnakeSegment NewTail = new SnakeSegment(CurrentTail.X, CurrentTail.Y, CurrentTail.Dir, SegmentType.Tail);

    CurrentTail.Type = SegmentType.Segment; //Makethe old tail a segment
    NewTail.SetNext(CurrentTail);

    Segments.Add(NewTail);
}
```

Snake



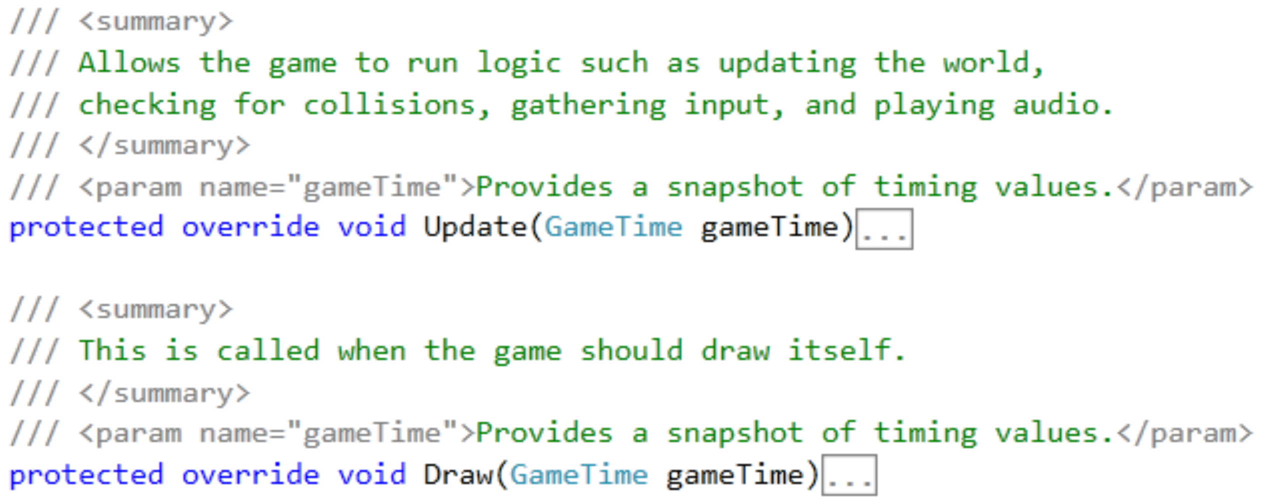
```
public bool Collision()
{
    if (Head.X < 0 || Head.X >= 15 || Head.Y < 0 || Head.Y >= 15)
    {
        return true;
    }

    for (int i = 1; i < Segments.Count; ++i)
    {
        if (Head.X == Segments[i].X && Head.Y == Segments[i].Y)
        {
            return true;
        }
    }

    return false;
}
```

Snake



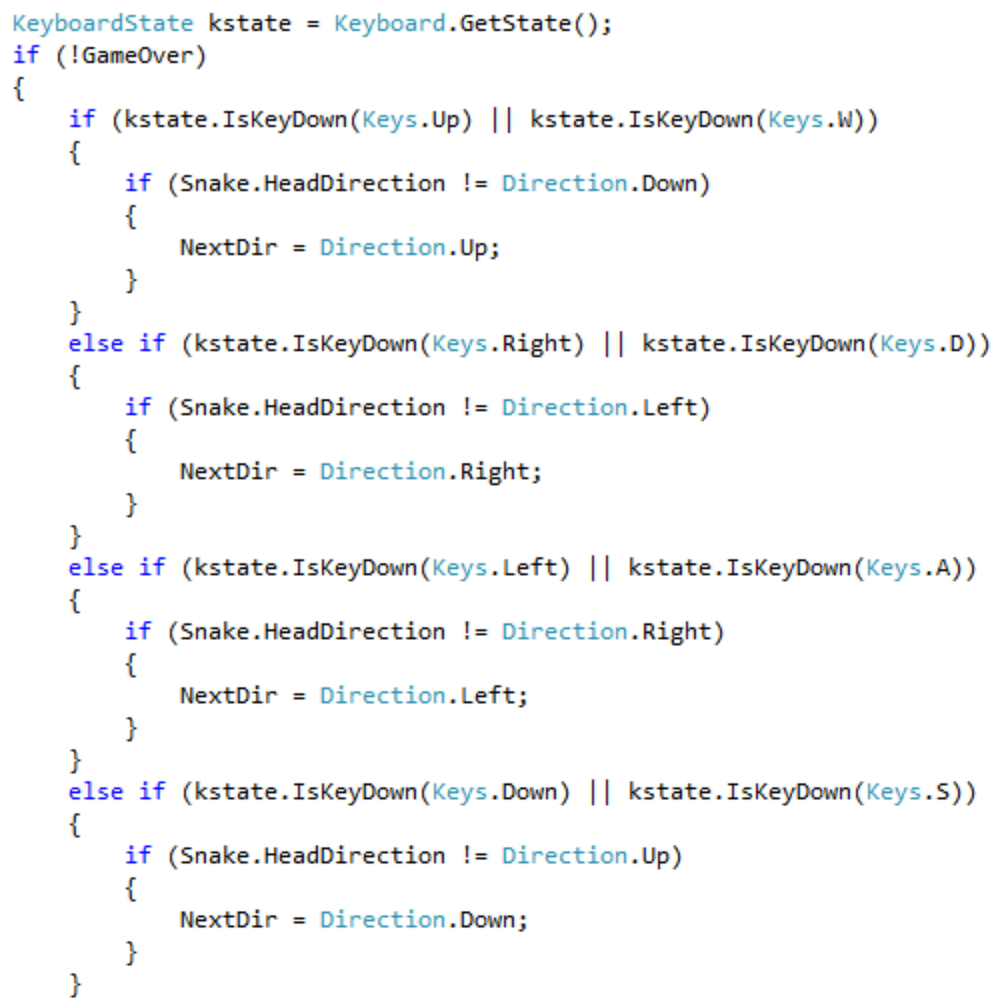
```
public void Draw(SpriteBatch b)
{
    foreach (SnakeSegment segment in Segments)
    {
        segment.Draw(b);
    }
}
```

SnakeGame



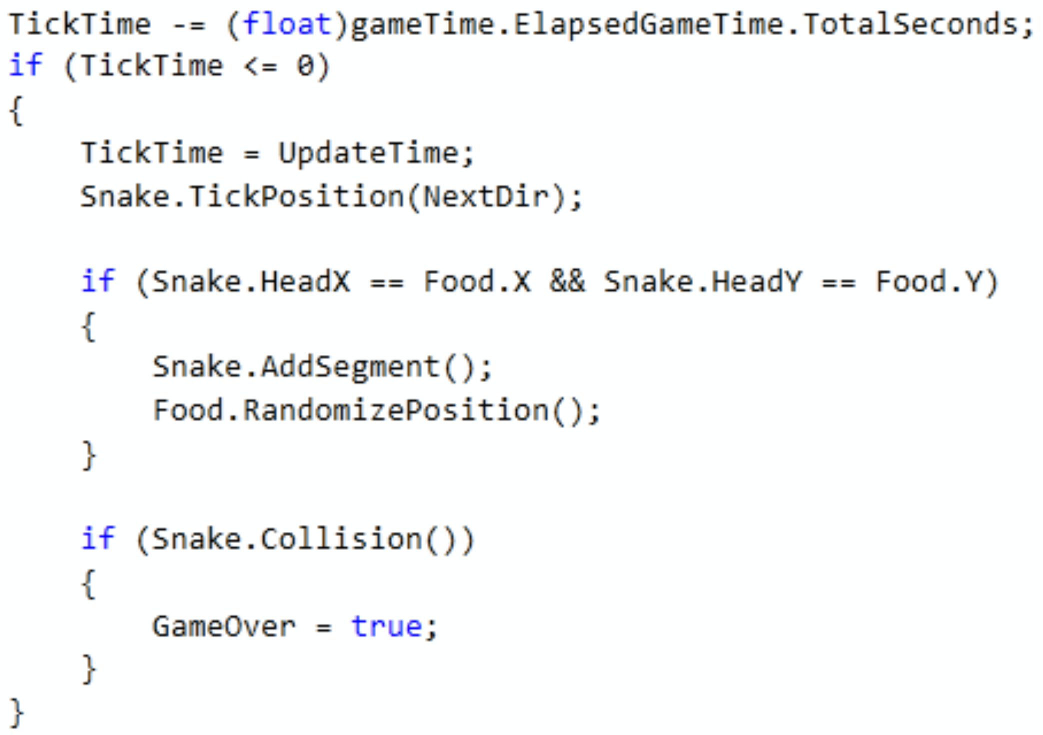
```
/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)...

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)...
```

SnakeGame



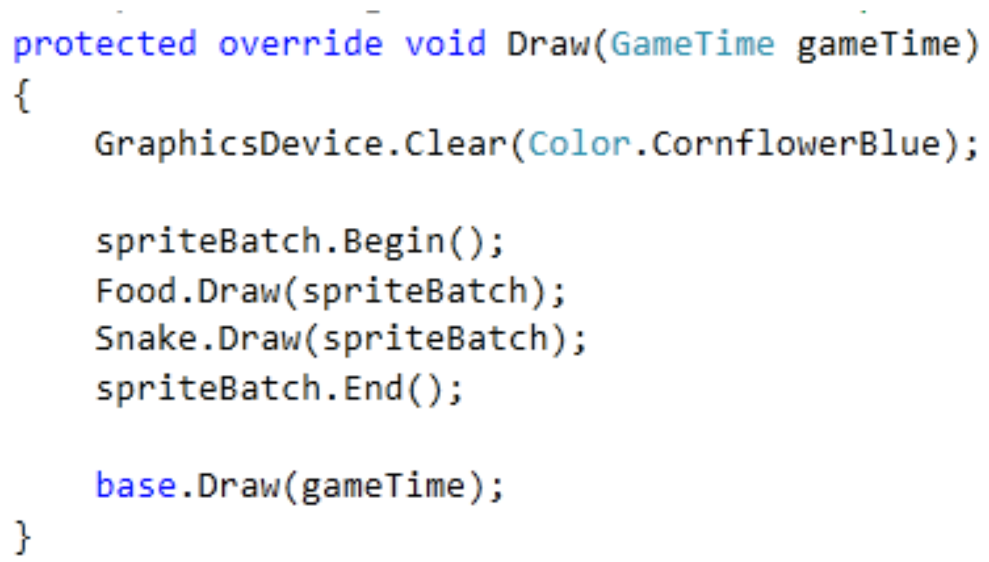
```
KeyboardState kstate = Keyboard.GetState();
if (!GameOver)
{
    if (kstate.IsKeyDown(Keys.Up) || kstate.IsKeyDown(Keys.W))
    {
        if (Snake.HeadDirection != Direction.Down)
        {
            NextDir = Direction.Up;
        }
    }
    else if (kstate.IsKeyDown(Keys.Right) || kstate.IsKeyDown(Keys.D))
    {
        if (Snake.HeadDirection != Direction.Left)
        {
            NextDir = Direction.Right;
        }
    }
    else if (kstate.IsKeyDown(Keys.Left) || kstate.IsKeyDown(Keys.A))
    {
        if (Snake.HeadDirection != Direction.Right)
        {
            NextDir = Direction.Left;
        }
    }
    else if (kstate.IsKeyDown(Keys.Down) || kstate.IsKeyDown(Keys.S))
    {
        if (Snake.HeadDirection != Direction.Up)
        {
            NextDir = Direction.Down;
        }
    }
```

SnakeGame



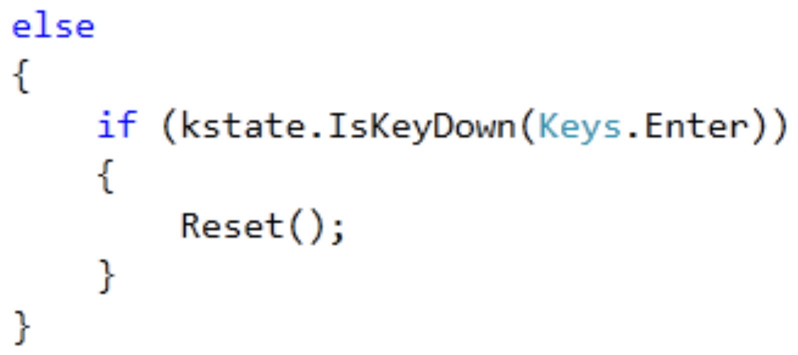
```
TickTime -= (float)gameTime.ElapsedGameTime.TotalSeconds;
if (TickTime <= 0)
{
    TickTime = UpdateTime;
    Snake.TickPosition(NextDir);

    if (Snake.HeadX == Food.X && Snake.HeadY == Food.Y)
    {
        Snake.AddSegment();
        Food.RandomizePosition();
    }

    if (Snake.Collision())
    {
        GameOver = true;
    }
}
```

SnakeGame

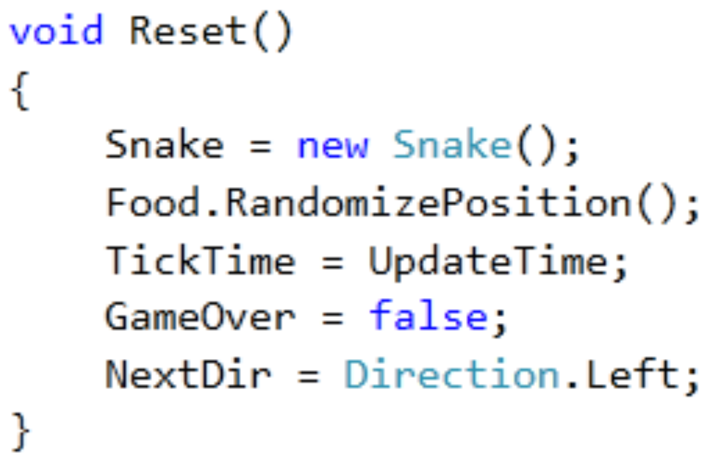


```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    spriteBatch.Begin();
    Food.Draw(spriteBatch);
    Snake.Draw(spriteBatch);
    spriteBatch.End();

    base.Draw(gameTime);
}
```

SnakeGame



```
else
{
    if (kstate.IsKeyDown(Keys.Enter))
    {
        Reset();
    }
}
```



```
void Reset()
{
    Snake = new Snake();
    Food.RandomizePosition();
    TickTime = UpdateTime;
    GameOver = false;
    NextDir = Direction.Left;
}
```