C# spil (del 3) - Space Invaders, sådan!

Artiklen indeholder implementeringen af spillet Space Invaders, og forklarer overordnet ideen bag. Sprite- og Kollisionsklasser fra C# spil (del 2), skal bruges i denne artikel. Kendskab til del 1 & 2 er en klar fordel!

Skrevet den **05. Feb 2009** af **sovsekoder** I kategorien **Programmering / C#** | ★★★☆☆

Indledning
Denne artikel vil præsentere og kommentere koden til et space invaders spil. Kendskab til C# spil (del1 & del2) er en stor fordel.

Sidst i artiklen findes den komplette source-code, og en forklaring til projektets opsætning. Source coden til BlockSprite- og SpriteCollision kan findes i C# spil del 2.

Pga. at det er besværligt at copy/paste koden ind og se spillet i aktion, har jeg gjort det muligt at downloade VS2003 C# projektet her : http://www.svendborggade.dk/download/invaders.zip

Så starter vi :o

Tilstands maskiner
Spillet indeholder 3 tilstandsmaskiner:
1) spil-tilstandsmaskine, styrer spillets tilstande: vent på nyt spil, nyt spil, game over...

2) invader-flytnings tilstandsmaskine.

3) Spiller tilstande: spiller skyder, spiller skyder ikke.

Disse tilstands maskiner sørger for spillets "flow" - at spillet kører korrekt, og overholder de givne regler for "Space Invaders".

Bevægelses mønster for Space Invaders
Jeg vil starte med tilstandsmaskine (2) - den der styrer invaders. Invaders skal styres rigtigt rundt på skærmen, de skal gå fra højre mod venstre indtil den invader længst mod højre rammer kanten. Herefter bevæger de sig et hak ned. Hvorefter de bevæger sig tilbage fra højre mod venstre, indtil den invader længst mod venstre rammer højre kant. Herefter vandrer invaders et hak ned igen. Bevægelsen gentages, men! - når invaders når bunden har spilleren tabt.

For at kunne implementere dette bevægelses mønster, har vi brug for en tilstands maskine, og en go' plan :). Den go'e plan består i at være i stand til at finde ud af, hvornår invaders er nået højre kant, venstre kant og bunden. Dette foretages i metoden FindBorderSprites(), for go' pedagoisk ro og orden er disse invaders farvet rød, blå og gul - så man kan se at algoritmen virker. Vi har nu:

Blå invader => længst til venstre
Gul invader => længst til højre
Rød invader => længst nede.

Igen.. farven er der kun for at understrege algoritmens virkemåde. Algoritmen virker således at, når blå sprite rammer venstre kant skal invaders bevæge sig ned. Når gul sprite rammer højre kant bevæger invaders sig ned. Når rød sprite har nået bunden, har spilleren tabt!

Tilstands maskinen er nu i stand til at flytte invader rundt. Vi har 4 tilstande:

MoveRight,
MoveLeft,
MoveDownLeft,
MoveDownRight

MoveRight og MoveLeft giver mere eller mindre sig selv: disse tilstande flytter invaders til højre og ventre. Det er dog mere underligt at der er to tilstande til at flytte invaders ned, hvorfor er der det?!!?

Grunden til at vi skal bruge to tilstande til at flyte invaders ned er, at vi har brug for en forskellig overgang når invaders er flyttet et hak ned. Hvis vi kom fra højre, og har nået kanten - så går vi ned, og herefter skal vil til VENSTRE!. Hvis vi kom fra venstre og rammer kanten, går vi ned - men skal herefter gå til HØJRE!.

Bevægelsen kan skrives på følgende måde:
MoveRight => ModeDownRight
ModeDownRight => MoveLeft
MoveLEft => MoveDownLEft
MoveDownLeft => MoveRight, og bevægelsen går i ring.

Vi kunne godt ha' implementeret bevægelsen med blot 3 tilstande (kun een tilstand til at bevæge sig ned) - dette ville dog kræve en varibel som skulle sættes ved overgangen fra MoveLeft og MoveRight. Det er en smags-sag hvorvidt man vil benytte den ene eller den anden metode - den ene er ikke mere rigtig (eller forkert) end den anden.

Spiller-tilstandsmaskine
Denne tilstandsmaskine styrer spilleren. Denne tilstandsmaksine har 2 tilstande: Fire, Idle. Disse tilstande tages i brug når der trykkes på musen. Reglerne er således at der kun må være eet skud på skærmen af gangen. Så når der trykkes på musen tjekkes det om spilleren er i Fire-tilstanden, hvis dette er tilfældet ignoreres trykket, ellers skydes der. Spiler tilstanden ryger tilbage i idle, når skuddet når udover spil-rammen.

Spil-tilstandsmaskinen
Denne tilstandsmaksine styrer spillet. Denne tilstandsmaskine benyttes primært i gameloop, der kaldes løbende vha. timeren (artikkel 1). Men tilstandsmaskinen benyttes også i f.eks. MouseDown-metoden, der kaldes når der trykkes på musen.

Grunden til dette er at musse-tryk har flere funktioner. Et musse tryk kan betyde at spilleren skyder, men når man ikke er inde i spillet (dvs. vi venter på at et nyt spil skal starte) så betyder et mussetryk at spillet skal starte. Spil-tilstands maskinen benyttes til at fortolke musse-trykket på den rigtige måde (se koden i MouseDown).

Spil-tilstandsmaskinen har en tilstand, GameOver, der angiver at du har tabt. Når tilstanden sættes = GameOver, så er spillet slut og der vises en besked ("ARGGG - du er død"). Dette udnyttes f.eks. ved at have en CollisionListener der lytter på om invader-skud har ramt spillen. Den eneste kode der er i denne lytter er:

gameState = GameStates.GameOver;

- Hvilket straks sætter tilstanden til GameOver når spilleren rammes af et skud - enkelt!

Spil-tilstandsmaskinen benyttes også i OnPaint -der tegner formen. I dette spil bruges tilstandsmaskinen til at vise de korrekte tekst beskeder som:

"Tryk musen for start.." (start af nyt spil)

"Op med hastigheden!" (næste level)
"ARGG - du er død!" (gameover)

Spil-tilstanden: PlayLevel - spillet er i fuld gang
I Spil-tilstanden, PlayLevel - spilles spillet. I denne tilstand skal invaders opdateres - spilleren skal flyttes, kort sagt her skal vi sørge for at opdatere alt så spillet foregår korrekt.

I følgende kode ses implementeringen af denne tilstand:

```
case GameStates.PlayLevel:
  // Her kæmpes der mod invaders
  // - bevægelse af spiller, samt styring af skud
  // - bevægelse af invaders
  // - bevægelse af missiler
  // - Tjek for kollisioner
  // - Tjek for game over ell. sejr
  PlayerAction();
  MoveInvaders();
  InvaderAttack();
  MoveMissiles();
  collisions.Check();
  CheckForGameOver();
  CheckForPlayerWin();
  break;
```

PlayerAction(): Denne funktion fyrer et missil af, afhængigt af spiller-tilstanden. Herefter flyttes spilleren i retning af musen. Forstået således at spilleren flyttes til højre hvis musen er tilhøjre for spilleren, og omvendt.

MoveInvaders(): Her tages invader-tilstandsmaskinen i brug, og invaders flyttes efter algoritmen beskrevet tidligere.

InvaderAttack(): Her vælges tilfældige invarders til at skyde. Der er et øvre loft på antallet af skud fra invaders (i konstanten: invaderMissileCount), så hvis dette loft er nået skydes der slet ikke før et af missilerne af ude over spil-rammen.

MoveMissiles(): Denne funktion sørger for at flytte missilerne. Der er i denne forbindelse oprettet en klasse, Missile. Dette er gjort for at vise hvordan man kan nedarve fra BlockSprite-klassen. I denne nedarvet klasse findes en metode, AutoMove, der opdaterer missilet. Missilet flyttes op eller ned afhængigt af hvordan det er oprettet (spilleren missiler flyttes op, invader-missiler flyttes ned). I denne nedarvet klasse kan man implementere Draw. På denne måde kan man tegne bitmaps, altså rigtige figurer istedet for bare sorte kasser.

collisions.Check(): her tjekkes for kollisioner, og listeners kaldes hvis der er sprites der har kollideret.

CheckForGameOver(): har spilleren tabt? hvis dette er tilfældet justeres spi-tilstanden med det sammen.

CheckForPlayerWin(): Der tjekkes om spilleren har vundet, hvis dette er tilfældet sættes næste level op (hvor det går en tand hurtigere).

Space invaders
Ideerne omkring implementeringen af space invaders er nu ridset op. Du kan kopiere source koden ind i et windows-form projekt, og se spillet i fuld aktion. Når spillet kører kan man få en fornemmelse af de ting jeg

har gennemgået i artiklen.

Det kan varmt anbefales at kigge koden igennem, start i metoden GameLoop(). Følg metoderne rundt, tegn evt. et diagram over tilstandende i tilstandsmaskinerne - for at få fornemmelsen af logikken. Prøv at skyde den blå/gule/røde invader, og se hvordan en ny invader findes for at opretholde algoritmen til invader-bevægelses mønstret.

Go' fornøjelse :D

Source code
Sådan får du koden op at køre:

Opret et windows form projekt. Filen med formen hedder Form1.cs. Følgende liste viser hvilke filer projektet skal bestå af:

```
* Form1.cs           (DEL3)- selve spillet
* Missile.cs         (DEL3)- missilsprite nedarvet fra BlockSprite
* SpriteCollision.cs (DEL2)- håndtering af sprite kollision
* BlockSprite.cs     (DEL2)- visning af sprites
```

(DEL3) betyder at source-coden findes i denne artikel.
(DEL2) betyder at source-coden er i artiklen C# spil (del 2).

Form1.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;

namespace SpaceInvaders
{
  public class Form1 : System.Windows.Forms.Form
  {
    // konstanter der bruges af andre klasser
    public const int TopBorder = 10;
    public const int LeftBorder = 10;
    public const int RightBorder = 250;
    public const int LowerBorder = 250;

    // Konstanter relateret til invaders
    const int invaderSize = 10;
    const int invaderRows = 5;
    const int invaderColumns = 10;
    const int invaderMissileCount = 3;

    // Størrelsen af spiller spriten
    const int playerSize = 20;

    // Tekstrenge der vises i spillet
    const string newGame = "Tryk musen for start..";
    const string nextLevel = "Op med hastigheden!";
    const string gameOver = "ARGG - du er død!";

    // Spillets Sprite variable
```

```
    SpriteCollision collisions;
    BlockSprite[,] invaders;
    BlockSprite playerSprite;
    Missile playerMissile;
    Missile[] invaderMissiles = new Missile[invaderMissileCount];
    BlockSprite leftMostSprite;
    BlockSprite rightMostSprite;
    BlockSprite lowestSprite;

    // Spil variable
    int score;
    int level;
    string gameText = "";

    // Overordnede Spil-tilstande
    enum GameStates
    {
      InitializeNewGame,
      WaitForNewGame,
      PrepareNextLevel,
      PlayLevel,
      GameOver
    };
    GameStates gameState;

    // invader flytte-tilstande
    enum MoveStates
    {
      Left,
      Right,
      DownLeft,
      DownRight,
    };
    MoveStates moveState;


    // spiller tilstande
    enum PlayerStates
    {
      Idle,
      Fire
    }
    PlayerStates playerState;

    // Variabler relateret til bevægelse af invaders
    int invaderMoveDownCounter;
    int invaderMoveDelay;
    int invaderMoveClock;

    // Bruges til tilfældige tal
    Random rnd;

    private System.Windows.Forms.Timer timer1;
    private System.ComponentModel.IContainer components;
```

```
    public Form1()
    {
      //
      // Required for Windows Form Designer support
      //
      InitializeComponent();

      // Tegn med double-buffering
      SetStyle(ControlStyles.UserPaint, true);
      SetStyle(ControlStyles.AllPaintingInWmPaint, true);
      SetStyle(ControlStyles.DoubleBuffer, true);

      // Initialiser spillet
      InitializeGame();
      rnd = new Random(this.Handle.ToInt32());

      // Set formens størrelse så der er plads til hele spillet
      const int slagX = 100;
      const int slagY = 40;
      this.Size = new Size(RightBorder+LeftBorder+slagX,
TopBorder+LowerBorder+slagY);
    }

    #region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
      this.components = new System.ComponentModel.Container();
      this.timer1 = new System.Windows.Forms.Timer(this.components);
      //
      // timer1
      //
      this.timer1.Enabled = true;
      this.timer1.Interval = 10;
      this.timer1.Tick += new System.EventHandler(this.Gameloop);
      //
      // Form1
      //
      this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
      this.ClientSize = new System.Drawing.Size(456, 350);
      this.Name = "Form1";
      this.Text = "Form1";
      this.MouseDown += new
System.Windows.Forms.MouseEventHandler(this.Form1_MouseDown);
      this.Paint += new
System.Windows.Forms.PaintEventHandler(this.Form1_Paint);

    }
    #endregion

    // Initialiser alt i spillet
    void InitializeGame()
```

```
    {
      InitializeGameVariables();
      InitializeSprites();
    }

    // Initialiser spillets variable
    void InitializeGameVariables()
    {
      // Initialiser spil-tilstands maskinen
      gameState = GameStates.InitializeNewGame;

      // nul-stil score og level
      score = 0;
      level = 0;

      // Initialiser variable der styrer invaders
      invaderMoveDownCounter = 0;
      invaderMoveDelay = 5;
      invaderMoveClock = 0;
    }

    // Initialiser alle sprites i spillet
    void InitializeSprites()
    {
      moveState = MoveStates.Right;
      playerState = PlayerStates.Idle;

      leftMostSprite = null;
      rightMostSprite = null;
      lowestSprite = null;

      collisions = new SpriteCollision();

      // Opret spiller
      playerSprite = new BlockSprite(LeftBorder,
        LowerBorder-playerSize,
        playerSize,
        playerSize);

      // Opret missiler for spiller og invader
      playerMissile = new Missile(0, 0, true);
      playerMissile.Visible = false;
      playerMissile.Color = Color.Blue;

      // Opret invader-missiler
      for(int i=0; i<invaderMissiles.Length; i++)
      {
        invaderMissiles[i] = new Missile(0, 0, false);
        invaderMissiles[i].Speed = 1;
        invaderMissiles[i].Visible = false;
        invaderMissiles[i].Color = Color.Red;
      }

      // Opret SpaceInvaders
      invaders = new BlockSprite[invaderRows,invaderColumns];
```

```
      for(int i=0; i<invaderRows; i++)
      {
        for(int j=0; j<invaderColumns; j++)
        {
          invaders[i,j] = new BlockSprite(LeftBorder+j*(invaderSize*2),
            TopBorder+i*(2*invaderSize),
            invaderSize,
            invaderSize);
        }
      }

      // Sæt kollision lyttere op mellem invaders og missil
      for(int i=0; i<invaderRows; i++)
      {
        for(int j=0; j<invaderColumns; j++)
        {
          if(SpriteExist(invaders[i,j]))
            collisions.AddListener(playerMissile, invaders[i,j], new
CollisionDelegate(PlayerMissileHandler));
        }
      }

      // Sæt kollision lytter om mellem invader missiler og player
      for(int i=0; i<invaderMissiles.Length; i++)
      {
        collisions.AddListener(invaderMissiles[i], playerSprite, new
CollisionDelegate(InvaderMissileHandler));
      }

      // Initialiser grænse sprites (invaders)
      FindBorderSprites();
    }

    // Find de yderste invaders - så invaders ved hvornår de er nået til
kanten
    void FindBorderSprites()
    {
      BlockSprite leftMostSpriteTemp = leftMostSprite;
      BlockSprite rightMostSpriteTemp = rightMostSprite;
      BlockSprite lowestSpriteTemp = lowestSprite;

      for(int i=0; i<invaderRows; i++)
      {
        for(int j=0; j<invaderColumns; j++)
        {
          if(!SpriteExist(invaders[i,j])) continue;
          if(!SpriteExist(leftMostSprite))
          {
            if(!SpriteExist(leftMostSpriteTemp))
              leftMostSpriteTemp = invaders[i,j];
            else if(invaders[i,j].X<leftMostSpriteTemp.X)
              leftMostSpriteTemp = invaders[i,j];
          }

          if(!SpriteExist(rightMostSprite))
```

```
          {
            if(!SpriteExist(rightMostSpriteTemp))
              rightMostSpriteTemp = invaders[i,j];
            else if(invaders[i,j].X>rightMostSpriteTemp.X)
              rightMostSpriteTemp = invaders[i,j];
          }

          if(!SpriteExist(lowestSprite))
          {
            if(!SpriteExist(lowestSpriteTemp))
              lowestSpriteTemp = invaders[i,j];
            else if(invaders[i,j].Y>lowestSpriteTemp.Y)
              lowestSpriteTemp = invaders[i,j];
          }
        }
      }

      // Updater grænse-sprites hvis det er nødvendigt
      if( !SpriteExist(leftMostSprite)) leftMostSprite = leftMostSpriteTemp;
      if( !SpriteExist(rightMostSprite)) rightMostSprite =
rightMostSpriteTemp;
      if( !SpriteExist(lowestSprite)) lowestSprite = lowestSpriteTemp;

      // Sæt farven på grænse sprites (invaders)
      leftMostSprite.Color = Color.Blue;
      rightMostSprite.Color = Color.Yellow;
      lowestSprite.Color = Color.Red;
    }

    // Bevæg invaders
    void MoveInvaders()
    {
      FindBorderSprites();
      if(invaderMoveClock++ < invaderMoveDelay) return;

      invaderMoveClock = 0;
      int dx=0, dy=0;
      switch(moveState)
      {
        case MoveStates.DownLeft:
          dy = 1;
          invaderMoveDownCounter++;
          if(invaderMoveDownCounter==invaderSize)
          {
            invaderMoveDownCounter = 0;
            moveState = MoveStates.Right;
          }
          break;
        case MoveStates.DownRight:
          dy = 1;
          invaderMoveDownCounter++;
          if(invaderMoveDownCounter==invaderSize)
          {
            invaderMoveDownCounter = 0;
            moveState = MoveStates.Left;
```

```
        }
        break;
      case MoveStates.Left:
        dx = -1;
        if(leftMostSprite.X<=LeftBorder)
        {
          dx = 0;
          invaderMoveDownCounter = 0;
          moveState = MoveStates.DownLeft;
        }
        break;
      case MoveStates.Right:
        dx = 1;
        if(rightMostSprite.X+rightMostSprite.Width>=RightBorder)
        {
          dx = 0;
          invaderMoveDownCounter = 0;
          moveState = MoveStates.DownRight;
        }
        break;
    }

    // Flyt invaders
    for(int i=0; i<invaderRows; i++)
    {
      for(int j=0; j<invaderColumns; j++)
      {
        if(SpriteExist(invaders[i,j]))
        {
          invaders[i,j].X += dx;
          invaders[i,j].Y += dy;
        }
      }
    }
  }

  // Se om der findes flere invaders
  bool MoreInvadersAlive()
  {
    return (
      SpriteExist(leftMostSprite) ||
      SpriteExist(rightMostSprite) ||
      SpriteExist(lowestSprite));
  }

  // Check om spilleren har vundet (ikke flere space invaders)
  void CheckForPlayerWin()
  {
    if(MoreInvadersAlive() == false)
    {
      // Du har skudt alle invaders, du vinder dette level!
      gameState = GameStates.PrepareNextLevel;
    }
  }
```

```
    // Tjek om spilleren har tabt fordi invaders har nået bunden
    void CheckForGameOver()
    {
      if(SpriteExist(lowestSprite) && (lowestSprite.Y>=playerSprite.Y))
      {
        gameState = GameStates.GameOver;
      }
    }

    // Håndtering af musse tryk afhængig af gamestate
    private void Form1_MouseDown(object sender,
System.Windows.Forms.MouseEventArgs e)
    {
      switch(gameState)
      {
        case GameStates.PlayLevel:
          playerState = PlayerStates.Fire;
          break;
        case GameStates.WaitForNewGame:
          gameState = GameStates.PlayLevel;
          break;
        case GameStates.PrepareNextLevel:
          gameState = GameStates.PlayLevel;
          break;
        case GameStates.GameOver:
          gameState = GameStates.InitializeNewGame;
          break;
      }
    }

    // Håndtering af tilfældig invader skud
    void InvaderAttack()
    {
      for(int m=0; m<invaderMissiles.Length; m++)
      {
        if(!SpriteExist(invaderMissiles[m]))
        {
          int k = rnd.Next(invaderRows*invaderColumns)+1;
          int i=0;
          int j=0;
          while(k>0)
          {
            i++;
            if(i>=invaderRows)
            {
              i=0;
              j++;
              if(j>=invaderColumns)
              {
                j=0;
              }
            }

            if(SpriteExist(invaders[i,j])) k--;
          }
```

```
        invaderMissiles[m].X = invaders[i,j].X+invaderSize/2;
        invaderMissiles[m].Y = invaders[i,j].Y+invaderSize/2;
        invaderMissiles[m].Visible = true;
      }
    }
  }

  // Håndtering af spillerens aktioner
  void PlayerAction()
  {
    switch(playerState)
    {
      case PlayerStates.Fire:
        if(playerMissile.Visible == false)
        {
          FireTheMissile();
        }
        playerState = PlayerStates.Idle;
        break;
      case PlayerStates.Idle:
        break;
    }
    Point mouseLocation = PointToClient(MousePosition);
    int centerSpriteX = playerSprite.X+playerSprite.Width/2;

    if(mouseLocation.X > centerSpriteX)
    {
      if(playerSprite.X < RightBorder-playerSize)
        playerSprite.X++;
    }
    else if(mouseLocation.X < centerSpriteX)
    {
      if(playerSprite.X > LeftBorder)
        playerSprite.X--;
    }
  }

  // Hoved Gameloop
  private void Gameloop(object sender, System.EventArgs e)
  {
    switch(gameState)
    {
      case GameStates.InitializeNewGame:
        // Initialisering af spillet
        InitializeGame();
        gameState = GameStates.WaitForNewGame;
        break;
      case GameStates.WaitForNewGame:
        // Her ventes på musse tryk i Form1_MouseDown
        break;
      case GameStates.PrepareNextLevel:
        // Forbered det næste level_
        // - Sæt hastigheden op, opdater level værdi
        // - Genindlæs sprites på deres positioner
```

```
        if(invaderMoveDelay>0) invaderMoveDelay--;
        level++;
        InitializeSprites();
        gameState = GameStates.WaitForNewGame;
        break;
      case GameStates.PlayLevel:
        // Her kæmpes der mod invaders
        // - bevægelse af spiller, samt styring af skud
        // - bevægelse af invaders
        // - bevægelse af missiler
        // - Tjek for kollisioner
        // - Tjek for game over ell. sejr
        PlayerAction();
        MoveInvaders();
        InvaderAttack();
        MoveMissiles();
        collisions.Check();
        CheckForGameOver();
        CheckForPlayerWin();
        break;
      case GameStates.GameOver:
        // Her ventes på musse tryk i Form1_MouseDown
        break;
    }

    // Paint screen
    Invalidate(true);
  }

  // Fyr spillerens missil af
  void FireTheMissile()
  {
    // Fyr missil af fra midten af spiller-spriten
    playerMissile.X = playerSprite.X+playerSprite.Width/2;
    playerMissile.Y = playerSprite.Y;
    playerMissile.Visible = true;
  }

  // Flyt alle missiler i spillet
  void MoveMissiles()
  {
    if(SpriteExist(playerMissile))
    {
      playerMissile.AutoMove();
    }
    for(int i=0; i<invaderMissiles.Length; i++)
    {
      if(SpriteExist(invaderMissiles[i]))
      {
        invaderMissiles[i].AutoMove();
      }
    }
  }

  // Tjek om en given sprite eksisterer
```

```
bool SpriteExist(BlockSprite sprite)
{
  if(sprite != null && sprite.Visible == true)
    return true;
  else
    return false;
}

// Håndtering af kollision mellem spiller missil og invaders
public void PlayerMissileHandler(object sender)
{
  Listener l = (Listener) sender;
  playerMissile.Visible = false;
  l.Sprite2.Visible = false;
  l.Remove = true;

  score += 10;
}

// Håndtering af kollision mellem invader missiler og spiller
public void InvaderMissileHandler(object sender)
{
  gameState = GameStates.GameOver;
}

// Tegn grafikken inde i spillet
void DrawGameGraphics(Graphics g)
{
  g.DrawRectangle(new Pen(Color.Black),
    LeftBorder,
    TopBorder,
    RightBorder-LeftBorder,
    LowerBorder-TopBorder);

  g.DrawString(
    string.Format("Score: {0:000000}\n\nLevel: {1:000}",score,level),
    this.Font,
    new SolidBrush(Color.Black),
    RightBorder+10,
    TopBorder);

  if(SpriteExist(playerMissile)) playerMissile.Draw(g);
  if(SpriteExist(playerSprite)) playerSprite.Draw(g);
  for(int i=0; i<invaderMissiles.Length; i++)
  {
    if(SpriteExist(invaderMissiles[i])) invaderMissiles[i].Draw(g);
  }

  if(invaders != null)
  {
    for(int i=0; i<invaderRows; i++)
    {
      for(int j=0; j<invaderColumns; j++)
      {
        if(SpriteExist(invaders[i,j])) invaders[i,j].Draw(g);
```

```
        }
      }
    }
  }

  // Skriv tekst ud centreret i spil-rammen
  void CenterText(Graphics g, string text)
  {
    SizeF textSize = g.MeasureString(text, this.Font);
    g.DrawString(
      text,
      this.Font,
      new SolidBrush(Color.Black),
      (RightBorder-LeftBorder-textSize.Width)/2+LeftBorder,
      TopBorder+(LowerBorder-TopBorder)/2);
  }

  // Tegning af spillet
  private void Form1_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
  {
    Graphics g = e.Graphics;
    DrawGameGraphics(g);

    // Gamestate afhængig grafik
    switch(gameState)
    {
      case GameStates.PrepareNextLevel:
        gameText = nextLevel;
        break;
      case GameStates.InitializeNewGame:
        gameText = newGame;
        break;
      case GameStates.GameOver:
        gameText = newGame;
        CenterText(g, gameOver);
        break;
      case GameStates.WaitForNewGame:
        CenterText(g, gameText);
        break;
    }
  }

  /// <summary>
  /// Clean up any resources being used.
  /// </summary>
  protected override void Dispose( bool disposing )
  {
    if( disposing )
    {
      if (components != null)
      {
        components.Dispose();
      }
    }
```

```
      base.Dispose( disposing );
    }

    [STAThread]
    static void Main()
    {
      Application.Run(new Form1());
    }
  }
}
```

Missile.cs

```
using System;

namespace SpaceInvaders
{
  // Missile: Nedarvet klasse fra BlockSprite, tilføjer 2 properties og 1
metode
  // Properties:
  // - up : angiver om missilen flyver op eller ned
  // - speed : angiver hvor hurtigt missilet flyver.
  // Metode:
  // - AutoMove : Opdaterer missilets position
  public class Missile : BlockSprite
  {
    const int missileWidth = 2;
    const int missileHeight = 8;
    int speed = 7;
    bool up = true;

    // kun til sjov
    double t=0;

    public Missile(int x, int y, bool up) :
base(x,y,missileWidth,missileHeight)
    {
      this.up = up;
    }

    public void AutoMove()
    {
      /*
      // Sjove skud :)
      if(up)
      {
        t += Math.PI/20;
        double dx = 5*Math.Sin(t);
        base.x += (int)(dx);
      }
      */
```

```
      if(up)
      {
        if(base.Y < Form1.TopBorder)
          base.Visible = false;
        else
          base.Y -= speed;
      }
      else
      {
        if(base.Y > Form1.LowerBorder)
          base.Visible = false;
        else
          base.Y += speed;
      }
    }

    public int Speed
    {
      get { return this.speed; }
      set { this.speed = value; }
    }

    public bool Up
    {
      get { return this.up; }
      set { this.up = value; }
    }
  }
}
```

Kommentar af mikze d. 16. Nov 2004 | 1

Cool :)

Kommentar af mmbn d. 11. Nov 2005 | 2

Super artikel, kan næsten ikke vente på del 4 og 5.

Kommentar af sorenbs d. 13. Jul 2005 | 3

Man kan virkeligt få brugt noget tid på det skidt :) Jeg glæder mig til de næste 2 artikler

Kommentar af andr3as d. 30. Nov 2004 | 4

ok

Kommentar af qbus d. 31. Oct 2005 | 5

Alletiders artikel :) Er igang med at lave dette spil: http://parachute.datamatikeronline.dk/ ud fra principperne i din artikel, dog med ændringer selvfølgelig :P

Kommentar af tobiasahlmo d. 24. Jan 2008 | 6

Nice :)

Kommentar af sovsekoder d. 23. Feb 2011 | 7

kig evt. på mit codeplex projekt ang. XNA og platform spil:

http://mrdev.codeplex.com/