Period 5 – Övningsuppgifter A

Objektorienterad spelprogrammering, 15,0 HP

Höstterminen 2015

1. Arvshierarki

Skapa ett nytt projekt i lösningen Övningar och kalla projektet Ö5A_1_3.

a. Använd bollklasserna och Game1 från Ö4A uppgift 6. Deklarera Ball-klassen till abstract.

```
abstract class Ball
{
  ...
```

b. Gör samma på Draw och Update. Om det står nått i metodkroppen ta bort det. Observera att måsvingarna också ska bort:

```
public abstract void Update();

public abstract void Draw(SpriteBatch spriteBatch);
```

Om du nu kör ditt program kommer kompilatorn att klaga på att du inte implementerat de abstrakta metoderna: 'Ball_Bounce' does not implement inherited abstract member 'Ball.Draw()'. . När vi ärver en abstrakt metod krävs det att objektet som ärver den också implementerar metoden.

c. Implementera Draw (och om Update inte redan finns även den) i både Ball_Bounce och Ball_Mouse (det räcker med den enkla draw-metoden här):

```
public override void Draw(SpriteBatch spriteBatch)
{
    spriteBatch.Draw(tex, pos, Color.White);
}
```

Nu bör kompilatorn inte längre klaga och när du kör ditt program bör du ha en studsande boll och en boll som följer musen (samma beteende som i Ö3A uppgift 6 fast nu med en abstrakt klass).

Lägg märke till skillnaderna mellan ”abstrakt” och det vi hade tidigare ”virtual”. Virtual säger att huvudklassen har en implementation som klasser som ärver får lov att skriva över (”override”). Abstrakt säger att huvudklassen helt saknar en implementation av metoden och klasser som ärver *måste* göra en egen implementation.

2. Kollision

a. Ändra i Ball_Bounce så att den har en hastighet på x +1 och y 0 och placera den i botten av fönstret:

```
this.speed = new Vector2(1,0);
```

b. Ändra i Ball_Bounce så att den använder den långa Draw, skapa en variabel `Color color` och använder variabeln för att välja färg (glöm inte att sätta ett defaultvärde för color i konstruktorn):

```
Color color;

...

public override void Draw(SpriteBatch spriteBatch)
 {
     spriteBatch.Draw(tex, pos, null, color, 0, new Vector2(tex.Width / 2.0f,
     tex.Height / 2.0f), 1, SpriteEffects.None, 0);
 }
```

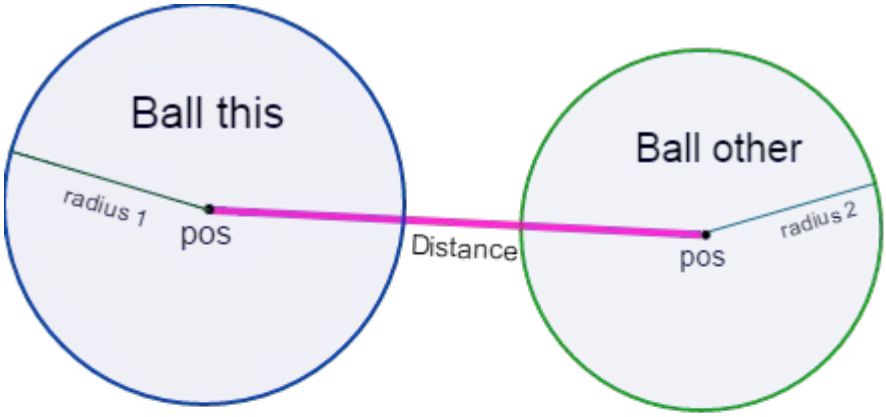
c. Skapa en ny boll som ärver av Ball_Bounce; Ball_Keyboard:

```
class Ball_Keyboard : Ball_Bounce
{
    KeyboardState ks;
    public Ball_Keyboard(Texture2D tex, Vector2 pos, GameWindow window)
        : base(tex, pos, window)
    {
        color = Color.Gainsboro;
    }

    public override void Update()
    {
        ks = Keyboard.GetState();

        if (ks.IsKeyDown(Keys.Right) && (pos.X + (tex.Width / 2.0f)) < bX)
        {
            pos += speed;
        }
        if (ks.IsKeyDown(Keys.Left) && pos.X > 0)
        {
            pos -= speed;
        }
    }
 }
```

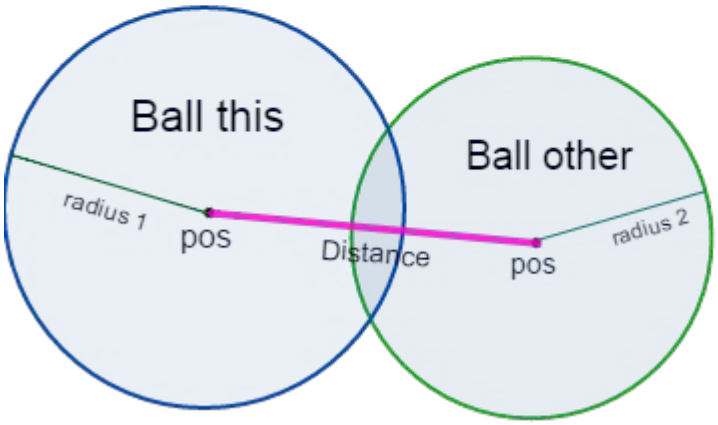
d. Ge Ball_Bounce en separat färg från Ball_Keyboard. Placera även Ball_Keyboard längst ner i fönstret, bredvid Ball_Bounce. Nu bör du ha en boll som åker längst botten av fönstret och studsar vid sidorna och en som du styr med piltangenterna.

e. I Ball, implementera en metod för att detektera cirkelkollision. Lägg till variabeln `float radius;` som en medlem i Ball och intiera den i konstruktorn till halva brädden av texturen.

```
public virtual bool CircleCollision(Ball other)
{
     return Vector2.Distance(pos, other.pos) < (radius + other.radius);
}
```



Längden på Distance är större än radius 1 + radius 2, så ingen kollision.



Längden på Distance är nu mindre än längden på radius 1 + radius 2, så en kollision har skett.

f. Lägg till en abstrakt metod i Ball som hanterar en kollision när CircleCollision-metoden rapporterar att en kollision har inträffat.

```
public abstract void HandleCollision();
```

g. Implementera HandleCollision i alla klasser som ärver av Ball.

i. Ball_Mouse:

```
public override void HandleCollision()
{
    throw new NotImplementedException();
}
```

ii. Ball_Bounce:

```
public override void HandleCollision()
{
    pos -= speed/2;
}
```

iii. Ball_Keyboard:

```
public override void HandleCollision()
{
    if (ks.IsKeyDown(Keys.Right)&&(pos.X+(tex.Width / 2.0f)) < bX)
    {
        pos -= speed / 2;
    }
    if (ks.IsKeyDown(Keys.Left)&&(pos.X - (tex.Width / 2.0f)) > 0)
    {
        pos += speed / 2;
    }
}
```

h. I Game1 efter update-Loopen lägg till kollen av kollisionerna:

```
foreach (Ball ball in balls)
{
    if (!(ball is Ball_Mouse))
        foreach (Ball other in balls)
        {
            if (!(other is Ball_Mouse) && ball != other)
                while (ball.CircleCollision(other))
                {
                    ball.HandleCollision();
                    other.HandleCollision();
                }
        }
}
```

3. Pixelperfekt Kollision

a. I Ball lägg till variabeln `protected Rectangle hitBox;` och i konstruktorn initiera den till `new Rectangle((int)pos.X, (int)pos.Y, tex.Width, tex.Height);`

b. I Update och HandleCollision för Ball_Bounce och Ball_Keyboard, sist i metoden uppdatera rektangelns x till nuvarande pos.X.

c. I Ball implementera en metod PixelCollision:

```
public bool PixelCollision(Ball other)
{

    Color[] dataA = new Color[tex.Width * tex.Height];
    tex.GetData(dataA);
    Color[] dataB = new Color[other.tex.Width * other.tex.Height];
    other.tex.GetData(dataB);

    int top = Math.Max(hitBox.Top, other.hitBox.Top);
    int bottom = Math.Min(hitBox.Bottom, other.hitBox.Bottom);
    int left = Math.Max(hitBox.Left, other.hitBox.Left);
    int right = Math.Min(hitBox.Right, other.hitBox.Right);

    for (int y = top; y < bottom; y++)
    {
        for (int x = left; x < right; x++)
        {
            Color colorA = dataA[(x - hitBox.Left) +
                                 (y - hitBox.Top) * hitBox.Width];
            Color colorB = dataB[(x - other.hitBox.Left) +
                                  (y - other.hitBox.Top) *
                                   other.hitBox.Width];
            if (colorA.A != 0 && colorB.A != 0)
            {
                return true;
            }
        }
    }
    return false;
}
```

d. Uppdatera din kollisions-loop så att den använder sig av den nya metoden:

```
foreach (Ball ball in balls)
 {
     if (!(ball is Ball_Mouse))
         foreach (Ball other in balls)
         {
             if (!(other is Ball_Mouse) && ball != other)
                 if (ball.CircleCollision(other))
                 {
                     while (ball.PixelCollision(other))
                     {
                         ball.HandleCollision();
                         other.HandleCollision();
                     }
                 }
         }
 }
```

e. För att kontrollera att PixelCollision fungerar som den ska, flytta en av bollarnas y-position lite så texturernas genomskinliga pixlar överlappar innan kollisionen inträffar.

4. Scrollande bakgrund (parallax)

Skapa ett nytt projekt i lösningen Övningar och kalla projektet Ö5A_4_5. Sätt det nya projektet som startprojekt.

a. Skapa en klass Background:

```
class Background
{
    List<Vector2> foreground, middleground, background;
    int fgSpacing, mgSpacing, bgSpacing;
    float fgSpeed, mgSpeed, bgSpeed;
    Texture2D[] tex;
    GameWindow window;

    public Background(ContentManager Content, GameWindow window)
    {
        this.tex = new Texture2D[3];
        this.window = window;

        tex[0] = Content.Load<Texture2D>("Ground");
        tex[1] = Content.Load<Texture2D>("Cloud");
        tex[2] = Content.Load<Texture2D>("Cloud");


    }

    public void Update()
    {

    }

    public void Draw(SpriteBatch sb)
    {

    }
}
```

b. Initiera foreground och lägg till några positioner där marken skall ritas:

```
foreground = new List<Vector2>();
fgSpacing = tex[0].Width;
fgSpeed = 0.75f;
for (int i = 0; i < (window.ClientBounds.Width / fgSpacing) + 2; i++)
{
      foreground.Add(new Vector2(i * fgSpacing, window.ClientBounds.Height
      - tex[0].Height));
}
```

c. Lägg till liknande för båda lager av moln:

```
middleground = new List<Vector2>();
mgSpacing = window.ClientBounds.Width / 5;
mgSpeed = 0.5f;
```

```
for (int i = 0; i < (window.ClientBounds.Width / mgSpacing) + 2; i++)
{
      middleground.Add(new Vector2(i * mgSpacing,
      window.ClientBounds.Height - tex[0].Height - tex[1].Height));
}
```

Och:

```
background = new List<Vector2>();
bgSpacing = window.ClientBounds.Width / 3;
bgSpeed = 0.25f;
for (int i = 0; i < (window.ClientBounds.Width / bgSpacing) + 2; i++)
{
      background.Add(new Vector2(i * bgSpacing, window.ClientBounds.Height
      - tex[0].Height - (int)(tex[1].Height * 1.5)));
}
```

d. I Draw rita ut alla positioner med rätt textur:

```
foreach (Vector2 v in background)
{
    sb.Draw(tex[2], v, Color.White);
}

foreach (Vector2 v in middleground)
{
    sb.Draw(tex[1], v, Color.White);
}

foreach (Vector2 v in foreground)
{
    sb.Draw(tex[0], v, Color.White);
}
```

Nu om du i Game1 initierar en Background i LoadContent och sedan kör Draw på den borde du ha en mark med lite moln åvan.

e. I Update för Background lägg till följande för att få marken att röra på sig:

```
for (int i = 0; i < foreground.Count; i++)
{
    foreground[i] = new Vector2(foreground[i].X - fgSpeed,
    foreground[i].Y);
}
```

f. Gör motsvarande för båda lager av moln:

```
for (int i = 0; i < middleground.Count; i++)
{
    middleground[i] = new Vector2(middleground[i].X - mgSpeed,
    middleground[i].Y);
}

for (int i = 0; i < background.Count; i++)
{
    background[i] = new Vector2(background[i].X - bgSpeed,
    background[i].Y);
```

```
}
```

5. Rullande bakgrund

a. I Update för Background lägg till följande för att få marken att loopa runt:

```
for (int i = 0; i < foreground.Count; i++)
{
    foreground[i] = new Vector2(foreground[i].X - fgSpeed,
    foreground[i].Y);
    if (foreground[i].X <= -fgSpacing)
    {
        int j = i - 1;
        if (j < 0)
        {
            j = foreground.Count - 1;
        }
        foreground[i] = new Vector2(foreground[j].X + fgSpacing - 1,
        foreground[i].Y);
    }
}
```

b. Gör motsvarande för båda lager av moln:

```
for (int i = 0; i < middleground.Count; i++)
{
    middleground[i] = new Vector2(middleground[i].X - mgSpeed,
    middleground[i].Y);
    if (middleground[i].X <= -mgSpacing)
    {
        int j = i - 1;
        if (j < 0)
        {
            j = middleground.Count - 1;
        }
        middleground[i] = new Vector2(middleground[j].X + mgSpacing - 1,
        middleground[i].Y);
    }
}

for (int i = 0; i < background.Count; i++)
{
    background[i] = new Vector2(background[i].X - bgSpeed,
    background[i].Y);
    if (background[i].X <= -bgSpacing)
    {
        int j = i - 1;
        if (j < 0)
        {
            j = background.Count - 1;
        }
        background[i] = new Vector2(background[j].X + bgSpacing - 1,
        background[i].Y);
    }
}
```

6. Hoppa

Skapa ett nytt projekt i lösningen Övningar och kalla projektet Ö5A_6_7. Sätt det nya projektet som startprojekt

a. Skapa en abstrakt klass Boll:

```
abstract class Ball
{
    protected Vector2 pos;
    protected Texture2D tex;
    protected Rectangle hitBox;
    public Ball(Texture2D tex, Vector2 pos)
    {
          this.tex = tex;
          this.pos = pos;
          this.hitBox = new Rectangle((int)pos.X, (int)pos.Y, tex.Width,
          tex.Height);
    }

    public abstract void Update();
    public abstract void Draw(SpriteBatch spriteBatch);
```

b. Skapa en ny klass Ball_Jump och implementera de metoder som behövs:

```
class Ball_Jump : Ball
{
    KeyboardState ks;
    GameWindow window;
    Vector2 speed;
    int bY, bX;
    bool isOnGround;

    public Ball_Jump(Texture2D tex, Vector2 pos, GameWindow window)
        : base(tex, pos)
    {
        this.window = window;
        this.speed = new Vector2(0, 0);
        this.bX = window.ClientBounds.Width;
        this.bY = window.ClientBounds.Height;
    }
    public override void Update()
    {
        throw new NotImplementedException();
    }


    public override void Draw(SpriteBatch spriteBatch)
    {
        throw new NotImplementedException();
    }
```

c. Låt bollen kunna gå åt höger och vänster och bli visad:

```
        public override void Update()

        {
            ks = Keyboard.GetState();

            if (ks.IsKeyDown(Keys.Right) &&
               (pos.X + (tex.Width / 2.0f)) < bX)
            {
                speed.X = 1;
            }


            if (ks.IsKeyDown(Keys.Left) &&
               (pos.X - (tex.Width / 2.0f)) > 0)
            {
                speed.X = -1;
            }

            pos += speed;

            hitBox.X = (int)(pos.X >= 0 ? pos.X + 0.5f : pos.X - 0.5f);
            hitBox.Y = (int)(pos.Y >= 0 ? pos.Y + 0.5f : pos.Y - 0.5f);
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(tex, pos, Color.White);
        }
    }
```

d. Implementera hopp funktionen:

```
        public override void Update()

        {
            ks = Keyboard.GetState();


            if (!isOnGround)
                speed.Y += 0.2f;
            speed.X = 0;


            if (ks.IsKeyDown(Keys.Right) &&
               (pos.X + (tex.Width / 2.0f)) < bX)
            {
                speed.X = 1;
            }
            if (ks.IsKeyDown(Keys.Left) &&
               (pos.X - (tex.Width / 2.0f)) > 0)
            {
                speed.X = -1;
            }
            if (ks.IsKeyDown(Keys.Space) && isOnGround)
            {
                speed.Y = -3;
                isOnGround = false;
            }

            pos += speed;
```

```
            hitBox.X = (int)(pos.X >= 0 ? pos.X + 0.5f : pos.X - 0.5f);
            hitBox.Y = (int)(pos.Y >= 0 ? pos.Y + 0.5f : pos.Y - 0.5f);

            if (hitBox.Y + hitBox.Height > bY)
            {
                pos.Y = bY - hitBox.Height;
                isOnGround = true;
                speed.Y = 0;
            }
        }
```

7. **Hoppa mellan plattformar**
   a. Skapa en klass som representerar en plattform.

```
class Plattform
{
    Texture2D tex;
    public Rectangle hitBox;
    public Plattform(Texture2D tex, Rectangle hitBox)
    {
        this.tex = tex;
        this.hitBox = hitBox;
    }

    public void Draw(SpriteBatch spriteBatch)
    {
        spriteBatch.Draw(tex, hitBox, Color.White);
    }
}
```

   Skapa några plattformar som ligger på samma höjd och intill varandra.

   b. Hantera kollision mellan spelare och plattform så att spelaren kan springa och hoppa på plattformarna:

Implementera metoderna HandelCollition och IsCollidning i Ball:

```
        public virtual bool IsColliding(Plattform p)
        {
            return hitBox.Intersects(p.hitBox);
        }

        public virtual void HandleCollision(Plattform other)
        {
            hitBox.Y = other.hitBox.Y - hitBox.Height;
            pos.Y = hitBox.Y;
        }
```

Och i Ball_Jump:

```
        public override void HandleCollision(Plattform p)
        {
            isOnGround = true;
            speed.Y = 0;
            base.HandleCollision(p);
        }
        public override void Update()

        {
            ks = Keyboard.GetState();

            speed.Y += 0.2f;
            speed.X = 0;
```