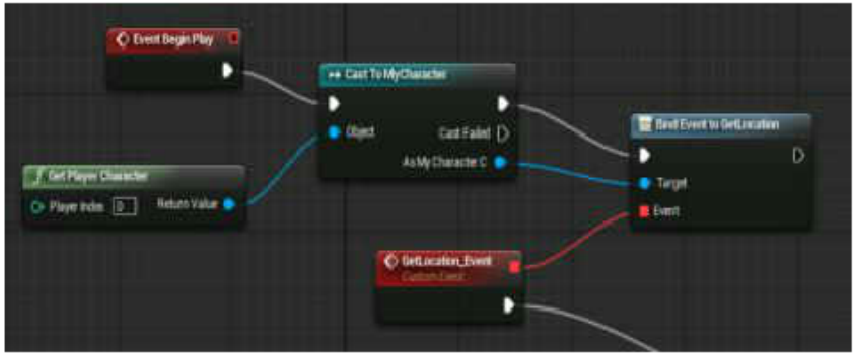
Game programming : Pong

Gaëtan Blaise-Cazalet - Head of programming section

What are we talking about?

~~Visual Scripting~~



Script / Code

```
string sInput;
int iLength, iN;
double dblTemp;
bool again = true;

while (again) {
    iN = -1;
    again = false;
    getline(cin, sInput);
    system("cls");
    stringstream(sInput) >> dblTemp;
    iLength = sInput.length();
    if (iLength < 4) {
        again = true;
        continue;
    } else if (sInput[iLength - 3] != '.') {
        again = true;
        continue;
    } while (++iN < iLength) {
        if (isdigit(sInput[iN])) {
            continue;
        } else if (iN >= (iLength - 3) ) {
```

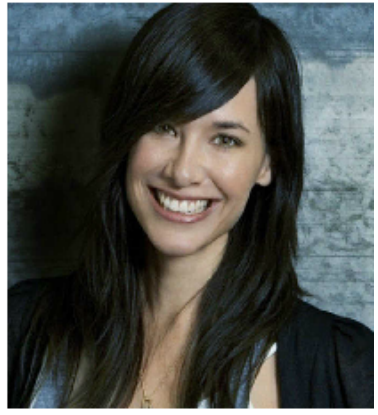
Not so much differences. When you are visual scripting, you are coding :

- In a less efficient way (-)
- Without learning a programming language (+)

Why coding ?

- You make video game. The matter you use, the feeling of your games, is computer code.
- To discuss with him or her

(without being a fool)

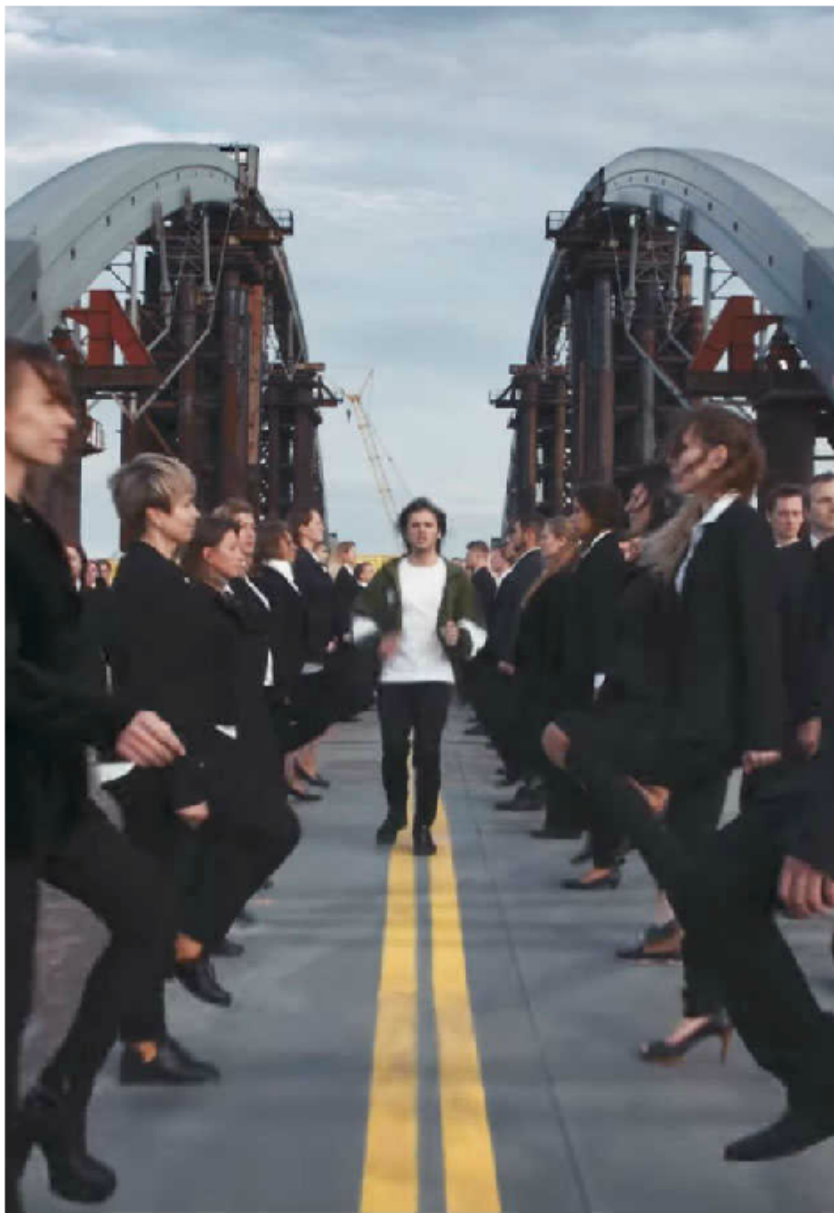


- To prototype your gameplays without depending of anyone

Basics of a video game

```
window {
    load();

    loop {
        inputs();
        update();
        draw();
    };
}
```



Monogame Starter

Create a project

Open visual studio 2015

File / New Project. Under Visual C#, select Monogame, then Monogame Window Project

Choose a folder on your DATA disk (usually D: or I:). Don't chose any folder on C:, it will be randomly discarded.

Name your project Pong and validate.

Base code

- Game1.cs

```
public class Game1 : Game
   {
      GraphicsDeviceManager graphics;
      SpriteBatch spriteBatch;

      public Game1()
      {
         graphics = new GraphicsDeviceManager(this);
         Content.RootDirectory = "Content";
      }

      protected override void Initialize()
      {

         base.Initialize();
      }

      protected override void LoadContent()
      {
         // Create a new SpriteBatch, which can be used to draw textures.
         spriteBatch = new SpriteBatch(GraphicsDevice);

      }

      protected override void UnloadContent()
      {

      }

      protected override void Update(GameTime gameTime)
      {
         if (GamePad.GetState(PlayerIndex.One)...)
            Exit();

         base.Update(gameTime);
      }

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


         base.Draw(gameTime);
      }
   }
```

Press F5 to launch game.

Drawing ball and paddle

Add image in monogame

Create a 32px radius ball

Create a 32 (horizontal) * 128 (vertical) paddle

In Visual Studio (VS), in the solution explorer, deploy the Content folder, right click on Content.mgcb then Open with..., then choose Monogame Pipeline Tool. It will open a small window.

Right click on Content, Add a new folder called "images"

Right click on this folder and Add / Add existing item. Select the two images you created. The prompt adk you if you want to copy or link the files. Choose the copy option.

(The link option can be useful if you share ressources between project.)

Save and Build.

Draw the ball

At the top of the class, add :

```
Texture2D ball;
```

In the LoadContent and Draw functions :

- Game1.cs / LoadContent()

```
protected override void LoadContent()
{
    spriteBatch = new SpriteBatch(GraphicsDevice);
    ball = Content.Load<Texture2D>("images\\ball");
}
```

- Game1.cs / Draw()

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

    spriteBatch.Begin();
    spriteBatch.Draw(ball, new Rectangle(0, 0, 64, 64), Color.White);
    spriteBatch.End();

    base.Draw(gameTime);
}
```

Transparent ball background

A texture is rectangular, so the ball background is drawn. 3 solutions :

- Set the background black. Will work for Pong but not for more complex games
- Set the background transparent in a graphics software. Will work, but has some impact on performances.
- Use a special color for backgrounds, that won't be rendered. By default in monogame, this special color is Magenta (#FF00FF)

Modify your ball image to set a magenta background. On Monogame Pipeline Tool, remove the ball and re-add it. Save and compile.

Now launch the game. The ball's background should be transparent.

Ball rectangle

Spritebatch.Draw need a rectangle to draw the texture on the screen. We will create a rectangle variable for the ball, which will help us to manipulate the ball.

Add at the top of the Game1 class :

```
Rectangle ballRect;
```

Modify LoadContent and Draw :

- Game1.cs / LoadContent()

```
protected override void LoadContent()
{
    ...
    ballRect = new Rectangle(100, 100, 64, 64);
}
```

- Game1.cs / Draw()

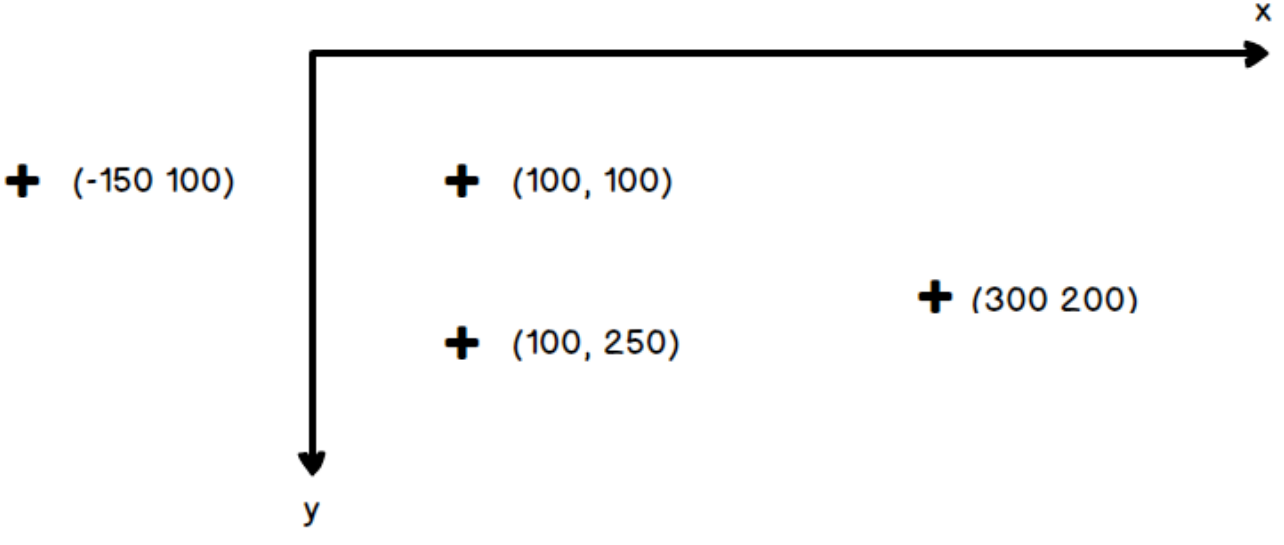
```
protected override void Draw(GameTime gameTime)
{
    ...
    spriteBatch.Draw(ball, ballRect, Color.White);
    ...
}
```

Drawing left paddle

Now, do the same for the left paddle.

Left paddle's rectangle should have 0, 200, 32, 128 as Rectangle initial values.

Cartesian coordinate system in videogames :



Drawing left paddle

Add at the top of the Game1 class :

```
...
    Texture2D leftPaddle;
    Rectangle leftPaddleRect;
```

- Game1.cs / LoadContent()

```
    protected override void LoadContent()
    {
        ...
        leftPaddle = Content.Load<Texture2D>("images/paddle");
        leftPaddleRect = new Rectangle(0, 200, 32, 128);
    }
```

- Game1.cs / Draw()

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

        spriteBatch.Begin();
        spriteBatch.Draw(ball, ballRect, Color.White);
        spriteBatch.Draw(leftPaddle, leftPaddleRect, Color.White);
        spriteBatch.End();

        base.Draw(gameTime);
    }
```

Move the ball and make it bounce

Ball movement

To move the ball, we will use the Update fonction. Because the ball is drawn at the ballRect position, we just have to update the rect position.

- Game1.cs / Update()

```
protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ...
      Exit();

    ballRect.X = ballRect.X + 5;
    ballRect.Y = ballRect.Y + 5;

    base.Update(gameTime);
}
```

Speed variables

It would be easier to change the ball vertical and horizontal spped if it would be in variables.

At the top of Game1 class :

```
...
    int ballSpeedX = 5:
    int ballSpeedY= 5:
```

- Game1.cs / Update()

```
protected override void Update(GameTime gameTime)
{
    ...
    ballRect.X = ballRect.X + ballSpeedX;
    ballRect.Y = ballRect.Y + ballSpeedY
    ...
}
```

Delta time

If our computer would lag or be very quick, the ball would change speed. It would not be fair. Programmers created a technique, called "delta time" to avoid this problem. The delta time is the time between this frame and last frame. If you multiply your speeds by the delta time, your moves will be the same whatever the computer speed.

Be cause delta time, when mesured in seconds, is very little, we have to increase our base speeds.

```
...
    ballSpeedX = 300:
    ballSpeedY= 300:
```

- Game1.cs / Update()

```
protected override void Update(GameTime gameTime)
{
    ...
    double delta = gameTime.ElapsedGameTime.TotalSeconds;
    ballRect.X = ballRect.X + (int)(ballSpeedX * delta);
    ballRect.Y = ballRect.Y + (int)(ballSpeedY * delta);
    ...
}
```

Bounce

Making a ball bounce is basically inversing its speed. We want the ball to bounce when it meet the bottom and the top of the screen. Here is fot the top of the screen.

- Game1.cs / Update()

```
protected override void Update(GameTime gameTime)
{
    ...
    if(ballRect.Y < 0)
    {
        ballRect.Y = 0;
        ballSpeedY = -ballSpeedY;
    }
    ...
}
```

Create the condition for the bottom bounce.

Bounce

- Game1.cs / Update()

```
protected override void Update(GameTime gameTime)
{
    ...
    if(ballRect.Y < 0)
    {
        ballRect.Y = 0;
        ballSpeedY = -ballSpeedY;
    }
    if (ballRect.Y > GraphicsDevice.Viewport.Height - 64)
    {
        ballRect.Y = GraphicsDevice.Viewport.Height - 64;
        ballSpeedY = -ballSpeedY;
    }
    ...
}
```

Moving the left paddle

Move left paddle with arrow

We will detect the keyboard's up and down deys status to move the paddle.

- Game1.cs / Update()

```
protected override void Update(GameTime gameTime)
{
    ...
    KeyboardState ks = Keyboard.GetState();
    if (ks.IsKeyDown(Keys.Down))
    {
        leftPaddleRect.Y += (int)(leftPaddleSpeed * delta);
    }
    if (ks.IsKeyDown(Keys.Up))
    {
        leftPaddleRect.Y -= (int)(leftPaddleSpeed * delta);
    }
    ...
}
```

Limit paddle move

We want to block the paddle so it cannot get out of the screen.

Here is the code for blocking the paddle at the top of the screen. Write the code to block the paddle at the bottom of the screen. You have to take into account the paddle height.

- Game1.cs / Update()

```
protected override void Update(GameTime gameTime)
{
    ...
    if(leftPaddleRect.Y < 0)
    {
        leftPaddleRect.Y = 0;
    }
    ...
}
```

Limit paddle move

- Game1.cs / Update()

```
protected override void Update(GameTime gameTime)
{
    ...
    if(leftPaddleRect.Y < 0)
    {
        leftPaddleRect.Y = 0;
    }
    if (leftPaddleRect.Y > GraphicsDevice.Viewport.Height - 128)
    {
        leftPaddleRect.Y = GraphicsDevice.Viewport.Height - 128;
    }
}
```

Moving the right paddle

Move right paddle ?

We could copy all variables (paddle texture, paddle rect, paddle speed etc.) and create a second paddle at a different position. But it would be code duplication.

We would like to create a code that would serve for both the right and the left paddle. We would create a category Paddle, that will hold texture, rect, move function, input detection, and create two objects of this category for the left and the right paddle.

This "category" is called a class. We will create a paddle class.

~~Moving the right paddle~~

Create the paddle class

The Paddle class

To create the Paddle class :

- Right click on the Pong project (under Solution) in the solution explorer, on the right of Visual Studio.
- Add / New element / Class. Name the class "Paddle", with a capital P.
- In the paddle class, we will add a Load, a Update and a Draw function. We will have a Texture2D texture field and an Rectangle rect field.
- We also need a special function call Paddle, as the class, which is called the "constructor".

- Paddle.cs

```
public class Paddle
{
    Texture2D texture;
    Rectangle rect;
    int speed;

    public Paddle(int x, int y, int speed)
    {

    }

    public void Load(ContentManager content)
    {

    }

    public void Update(GameTime gameTime)
    {

    }

    public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
    {

    }
}
```

The Paddle class

Now, we want to load the data into the Paddle class. We will put in our code what is common to every paddle, and use as argument what is specific to either the left or the right paddle.

- Paddle.cs

```
public Paddle(int x, int y, int speed)
{
    rect = new Rectangle(x, y, 32, 128);
    this.speed = speed;
}

public void Load(ContentManager content)
{
    texture = content.Load<Texture2D>("images/paddle");
}
```

The Paddle constructor will get a x and a y to set the paddle position.

The word this in the constructor means that the speed of the class take the value of the parameter speed. We must use "this" because the parameter and the class field have the same name.

The Load function will load the paddle texture.

The Paddle class

The Update function will contain the paddle related code from the Game1's Update function.

- Paddle.cs

```
public void Update(GameTime gameTime)
{
    KeyboardState ks = Keyboard.GetState();
    double delta = gameTime.ElapsedGameTime.TotalSeconds;

    if (ks.IsKeyDown(Keys.Down))
    {
        rect.Y += (int)(speed * delta);
    }
    if (ks.IsKeyDown(Keys.Up))
    {
        rect.Y -= (int)(speed * delta);
    }
    if (rect.Y < 0)
    {
        rect.Y = 0;
    }
    if (rect.Y > GraphicsDevice.Viewport.Height - 128)
    {
        rect.Y = GraphicsDevice.Viewport.Height - 128;
    }
}
```

There is a problem though : GraphicsDevice.Viewport does not exist in our class. We have to find a way to store the screen height in our Paddle class. The constructor is a good candidate.

- Paddle.cs

```
...
int screenHeight;

public Paddle(int x, int y, int speed, int screenHeight)
{
    ...
    this.screenHeight = screenHeight;
}
```

Replace GraphicsDevice.Viewport.Height by screenHeight in the Update function.

The Paddle class

The Draw function will call the paddle draw instruction from the Main's Draw.

- Paddle.cs

```
public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
    spriteBatch.Draw(texture, rect, Color.White);
}
```

Using the Paddle class

Let's now use the Paddle class to create the left paddle in the Game1 class.

- Game1.cs

```
    Paddle leftPaddle;

    protected override void Initialize()
    {
        leftPaddle = new Paddle(0, 100, 200, GraphicsDevice.Viewport.Height);

        base.Initialize();
    }

    protected override void LoadContent()
    {
        ...
        leftPaddle.Load(Content);
    }

    protected override void Update(GameTime gameTime)
    {
        ...
        leftPaddle.Update(gameTime);

        base.Update(gameTime);
    }

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

        spriteBatch.Begin();
        spriteBatch.Draw(ball, ballRect, Color.White);
        leftPaddle.Draw(gameTime, spriteBatch);
        spriteBatch.End();

        base.Draw(gameTime);
    }
```

Right paddle

Now, let's add the right paddle !

...You're doing it :)

Right paddle

Here is the Game1 code with the second paddle.

- Game1.cs

```
    Paddle leftPaddle;

    protected override void Initialize()
    {
        leftPaddle = new Paddle(0, 100, 200, GraphicsDevice.Viewport.Height);
        rightPaddle = new Paddle(GraphicsDevice.Viewport.Width - 32, 300, 200, GraphicsDevice.Viewport.Height);

        base.Initialize();
    }

    protected override void LoadContent()
    {
        ...
        leftPaddle.Load(Content);
        rightPaddle.Load(Content);
    }

    protected override void Update(GameTime gameTime)
    {
        ...
        leftPaddle.Update(gameTime);
        rightPaddle.Update(gameTime);

        base.Update(gameTime);
    }

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

        spriteBatch.Begin();
        spriteBatch.Draw(ball, ballRect, Color.White);
        leftPaddle.Draw(gameTime, spriteBatch);
        rightPaddle.Draw(gameTime, spriteBatch);
        spriteBatch.End();

        base.Draw(gameTime);
    }
```

See ? No code copy, all reuse. We just change the arguments in the constructor.

Still, we have a bug : when we hit an arrow, both paddle are moving.

Inheritance : an AI paddle

The AIPaddle class

Create a new AIPaddle class. It will "inherit" from the Paddle class, which means this class will have all field and all methods of the Paddle class, but can extend or replace them.

When creating a daughter class, we need at leat to create the constructor. We will also replace the Update function.

- AIPaddle.cs

```
public class AIPaddle : Paddle
{
    public AIPaddle(int x, int y, int speed, int screenHeight) : base(x, y, speed, screenHeight)
    {

    }

    public void Update(GameTime gameTime)
    {
        double delta = gameTime.ElapsedGameTime.TotalSeconds;

    }
}
```

Our AIPaddle constructor will call the Paddle constructor thanks to the base keyword.

Visual Studio will be unhappy with our Update function, saying it hides the Paddle.Update function. It is true, but intended. So we add a "virtual" keyword after the public keyword in the Paddle.Update definition, and an "override" keyword after the public keyword in the AIPaddle definition.

```
In Paddle.cs
    public virtual void Update(GameTime gameTime)

In AIPaddle.cs :
    public override void Update(GameTime gameTime)
```

Now VS is happy.

Using the AIPaddle

To use the AIPaddle, we just have to change the type in the Game1 class, and the constructor call in Initialize.

- Game1.cs

```
Paddle leftPaddle;
AIPaddle rightPaddle;

protected override void Initialize()
{
    leftPaddle = new Paddle(0, 100, 200, GraphicsDevice.Viewport.Height);
    rightPaddle = new AIPaddle(GraphicsDevice.Viewport.Width - 32, 300, 200, GraphicsDevice.Viewport.Height);

    base.Initialize();
}
```

Our AIPaddle does not move anymore.. We should add a real (simple) AI in our AIPaddle Update function. But befor, we shall create the Ball class.

The ball class

The Ball class

Create the class yourself :) Don't forget to update Game1.cs !

The Ball class

Create the class yourself :) Don't forget to update Game1.cs !

- Ball.cs

```
public class Ball
    {
        Texture2D texture;
        Rectangle rect;
        int speedX;
        int speedY;
        int screenHeight;

        public Ball(int x, int y, int speedX, int speedY, int screenHeight)
        {
            rect = new Rectangle(x, y, 64, 64);
            this.speedX = speedX;
            this.speedY = speedY;
            this.screenHeight = screenHeight;
        }

        public void Load(ContentManager content)
        {
            texture = content.Load<Texture2D>("images/ball");
        }

        public void Update(GameTime gameTime)
        {
            double delta = gameTime.ElapsedGameTime.TotalSeconds;
            rect.X = rect.X + (int)(speedX * delta);
            rect.Y = rect.Y + (int)(speedY * delta);

            if (rect.Y < 0)
            {
                rect.Y = 0;
                speedY = -speedY;
            }
            if (rect.Y > screenHeight - 64)
            {
                rect.Y = screenHeight - 64;
                speedY = -speedY;
            }
        }

        public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(texture, rect, Color.White);
        }
    }
```

The Ball class

- Game1.cs

```
Ball ball;
Paddle leftPaddle;
AIPaddle rightPaddle;

public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
}

protected override void Initialize()
{
    ball = new Ball(100, 100, 300, 300, GraphicsDevice.Viewport.Height);
    leftPaddle = new Paddle(0, 100, 200, GraphicsDevice.Viewport.Height);
    rightPaddle = new AIPaddle(GraphicsDevice.Viewport.Width - 32, 300, 200, GraphicsDevice.Viewport.Height);

    base.Initialize();
}

protected override void LoadContent()
{
    spriteBatch = new SpriteBatch(GraphicsDevice);
    ball.Load(Content);
    leftPaddle.Load(Content);
    rightPaddle.Load(Content);
}

protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ...
        Exit();

    ball.Update(gameTime);
    leftPaddle.Update(gameTime);
    rightPaddle.Update(gameTime);

    base.Update(gameTime);
}

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

    spriteBatch.Begin();
    ball.Draw(gameTime, spriteBatch);
    leftPaddle.Draw(gameTime, spriteBatch);
    rightPaddle.Draw(gameTime, spriteBatch);
    spriteBatch.End();

    base.Draw(gameTime);
}
```

The Paddle AI

A simple AI

Pong AI is simple : the AI paddle go up when the ball is above the paddle, and go down when the ball is under the paddle.

We have to define what is "above" and "under" the paddle. We could choose the center of the paddle as a reference point, but it can make the paddle move up and down stangely. It is better to choose the point halfway to to top or half way to the bottom.

But to make this calculation, we must know where is the Ball. We have to pass the ball to the AIPaddle. We could do this in the Update method, but it would not be optimal. We better pass a referece to the ball in the constructor.

- AIPaddle.cs

```
Ball ball;

public AIPaddle(int x, int y, int speed, int screenHeight, Ball ball) : base(x, y, speed, screenHeight)
{
    this.ball = ball;
}
```

A simple AI - Getting the ball info

We also need the X and Y position of the Ball. Those positions are stored in the ball's rectangle, but this rectanle is not accessible. We will create two properties in the ball class to get those coordinates.

- Ball.cs

```
public int X
{
    get
    {
        return rect.X;
    }
}

public int Y
{
    get
    {
        return rect.Y;
    }
}
```

A property allow to make public private variables. It is important the keep most class variables and functions private. But it is sometimes important to allow the access (and sometimes the modification) to some information.

We also need to access the Paddle rect from the AIPaddle. Add the "protected" modifier to the rect and the speed variable. The protected keyword allow the children of a class to access this class' fields.

- Paddle.cs

```
protected Rectangle rect;
protected int speed;
```

A simple AI - Now, the AI !

Let us make the code now. The algorithm is the following :

```
if ball x < paddle x + paddle height / 4
        move paddle up
else if ball x > paddle x + paddle height * 3 / 4
        move paddle down
```

A simple AI - Now, the AI !

Here is the code :

- AIPaddle.cs

```
public override void Update(GameTime gameTime)
{
    double delta = gameTime.ElapsedGameTime.TotalSeconds;

    if(ball.Y < rect.Y + rect.Height / 4)
    {
        rect.Y -= (int)(speed * delta);
    }
    if (ball.Y > rect.Y + rect.Height * 3 / 4)
    {
        rect.Y += (int)(speed * delta);
    }
}
```

Paddle - Ball Collision

Collision detection

We want the ball to bounce when it meet a paddle.

We decide to check the collision in the Game1's Update function, and to create a public Bounce function in the Ball class.

To help up, we will create four properties X, Y, Width and Height in the Paddle class and one property Radius in the Ball class.

Write the bounce condition for each paddle.

Collision detection

- Game1.cs

```
protected override void Update(GameTime gameTime)
{
    ...
    BallPaddleCollision();

    base.Update(gameTime);
}

void BallPaddleCollision()
{
    if (ball.X < leftPaddle.Width)
    {
        if (ball.Y >= leftPaddle.Y && ball.Y <= leftPaddle.Y + leftPaddle.Height)
        {
            // Bounce
        }
    }
    if (ball.X + ball.Radius * 2 > rightPaddle.X)
    {
        if (ball.Y >= rightPaddle.Y && ball.Y <= rightPaddle.Y + rightPaddle.Height)
        {
            // Bounce
        }
    }
}
```

Bounce

We will create a public PaddleBounce function for the ball. It has to inverse the speedX and to set the ball at the right X position after the speed has been inversed.

Bounce

- Ball.cs

```
public void PaddleBounce(int newX)
{
    speedX = -speedX;
    rect.X = newX;
}
```

- Game1.cs

```
void BallPaddleCollision()
{
    if (ball.X < leftPaddle.Width)
    {
        if (ball.Y >= leftPaddle.Y && ball.Y <= leftPaddle.Y + leftPaddle.Height)
        {
            ball.PaddleBounce(leftPaddle.Width);
        }
    }
    if (ball.X + ball.Radius * 2 > rightPaddle.X)
    {
        if (ball.Y >= rightPaddle.Y && ball.Y <= rightPaddle.Y + rightPaddle.Height)
        {
            ball.PaddleBounce(rightPaddle.X - ball.Radius * 2);
        }
    }
}
```

Option - Circle/Rectangle collision

The algorithm we have used checks roughly if the ball rectangle collides the paddles' rects. You can make a better code by considering the ball is a circle and use and Rectangle / Circle collision algorithm.

The implementation is left as an exercice.

Scoring points

Reset game when ball get out of screen

Reset the ball position when it is not stopped by the paddles.

Reset game when ball get out of screen

Reset the ball position when it is not stopped by the paddles.

- Ball.cs

```
public void Reset()
{
    rect.X = 100;
    rect.Y = 100;
}
```

- Game1.cs

```
protected override void Update(GameTime gameTime)
{
    ...
    BallOutCheck();

    base.Update(gameTime);
}

void BallOutCheck()
{
    if(ball.X < -ball.Radius || ball.X > GraphicsDevice.Viewport.Width)
    {
        ball.Reset();
    }
}
```

Polish

Accelerate the ball after each paddle bounce

Increase speedX after each bounce.

Reset speedX when a player scores.

Display the score

Display points on the side of each player

Increase points when a player scores

Game over

Define a maximum number of points

When a player hits this score, stop the game an announce the winner

Detect a key hit to restart the game. Restart the scores.

Start screen

Before the game starts, show a start screen to present your game

Hit a key to start the game

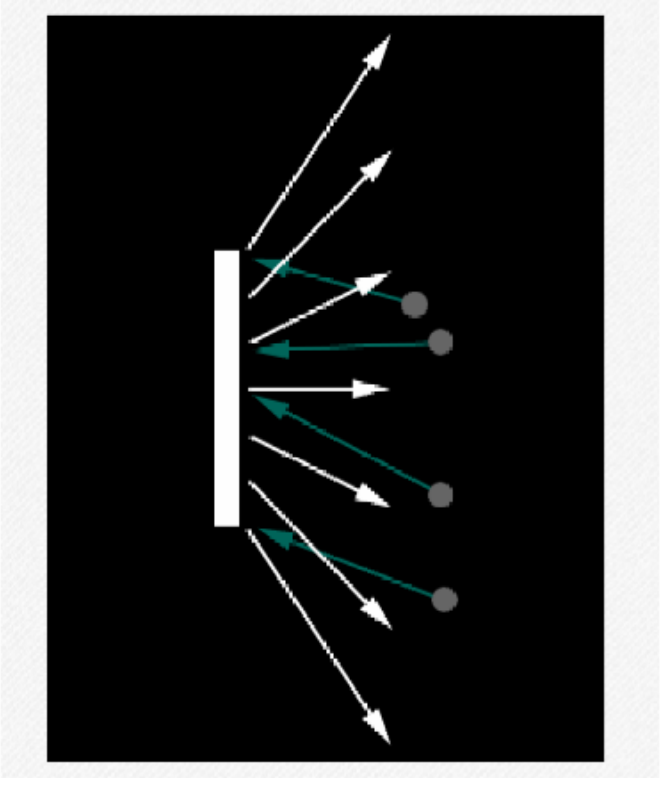
Second player mode

On the start screen, allow to select a mode where the right paddle is controlled by an other player

Use new controls for this player

Bounce control

- When the ball bounces on the paddle, we want its vertical speed to vary in function of the distance to center :



A brick breaker game

Introduction

Brick breaker is a classic game which most known variant is named "Arkanoid". It consists in using a ball to destroy bricks. The ball bounces on walls and bricks, and can quit the screen by the bottom side of the game. The player uses a paddle to keep the ball on screen.

A good start

- Create a black window with a ball bouncing on the edges inside

Paddle

- Reset the game if the ball cross the bottom edge

- Create a mouse-movable paddle class that would make the ball bounce if it collides it

You will find mouse input details here :
http://rbwhitaker.wikidot.com/mouse-input

- Set the paddle vertical position 10% away from the bottom of the screen

A little control

Bouncing angle shall vary with the position of the ball on the paddle (the further on the paddle, the bigger the angle)

- To achive that, mesure the distance between the place where the center of the ball hit the paddle and the center of the paddle

- Ball's horizontal speed will become : maximum horizontal speed of the ball * distance between centers / (paddle horizontal size / 2)

Paddle fix

- When the ball hit the paddle from the side, it may have a strange behaviour

- Increase the paddle thickness to highlight this behaviour (only for tests)

- Fix it by making the paddle affect the ball only if the ball go downward (if speed Y is positive, that is)

Another brick in the wall

- Create a brick class. It will have a rectangle (x, y, width, height), a column id (col), a row id (row), a boolean isDestroyed set to false. It will have a Draw function.

We will create bricks from the top left of the screen :

- We want 14 rows and 10 cols of bricks which dimensions are 80 (width) * 20 (height)

- When you draw a brick, make a visual gap of 2 pixels on the right side and on the bottom of the bricks. To do so, just create a temporary rect in your draw function, and set its size to 2 pixel less than the brick rect. Draw using this temporary rect.

Start by drawing one brick.

A wall of bricks

Create an empty list of Bricks.

Now, we want to create each brick of the wall. To do so, use a for loop to create each row, and inside a for loop to create each col

- Brick instances will be stored in a bricks List<Brick>. They are stored from left to right, then top to bottom.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	...				

- When you want to access a brick from the bricks list and you only have the column and the row of the brick, you can use the following code :

```
int brickIndex = col + BRICK_COLS_NUMBER * row;
Brick brick = bricks[brickIndex];
brick.doSomething();
```

For instance brick on the second row (row 1) and 8th column (col 7) will have index : 7 + 1 * 10 = 17, as we can see on the table.

Brick collision

The easier way to test collision is to go through the brick list and to test if the ball is colliding (rec

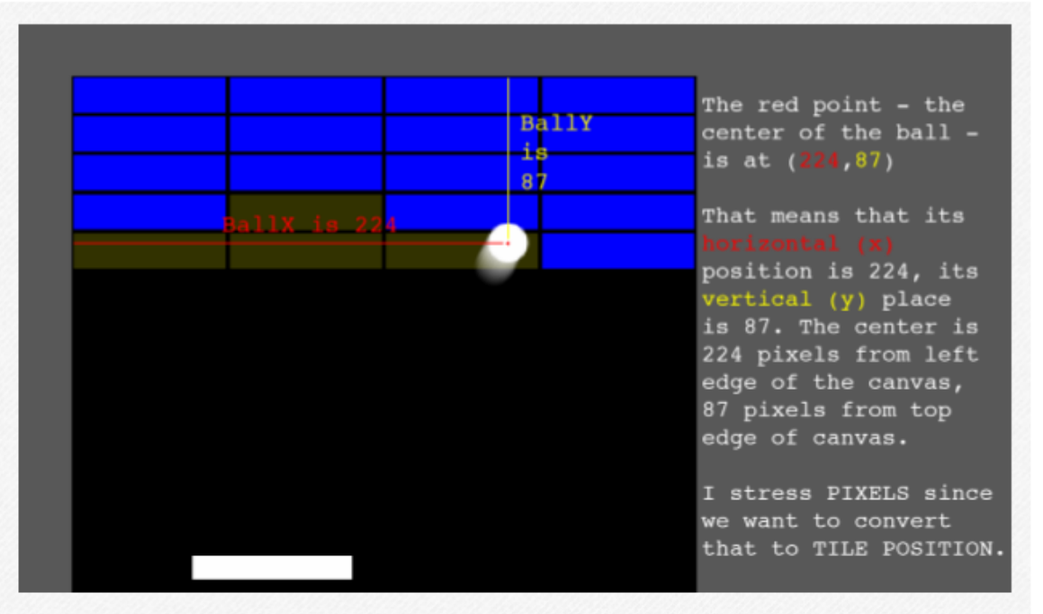
If there is a collision, set the boolean isDestroyed to true. Make the ball bounce.

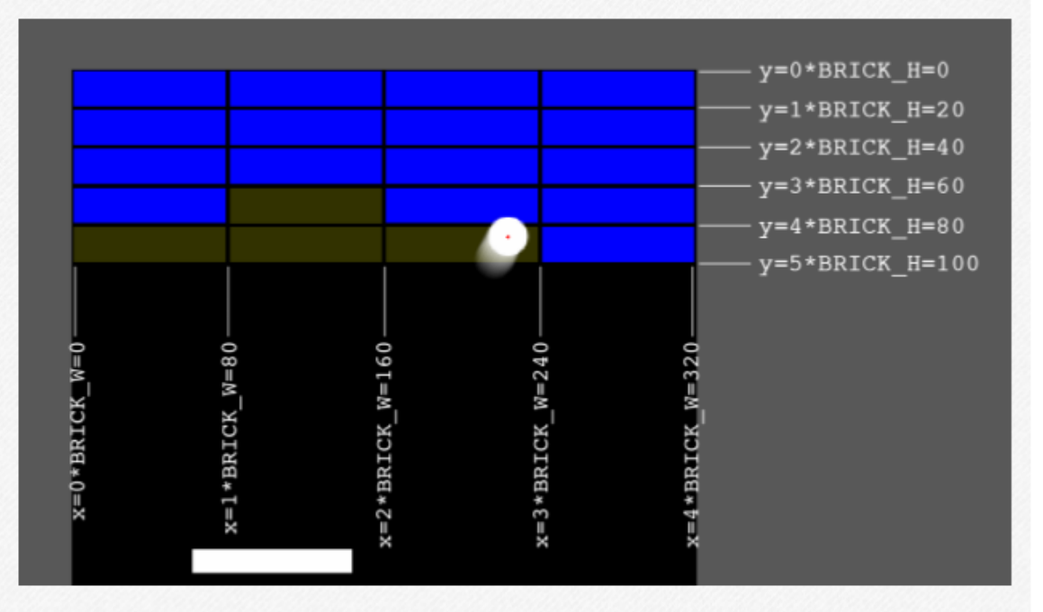
The brick shall not be drawn if isDestroyed is true. The collision test shall not occure anymore (if n

Brick collision - Optimisation (optional)

The re is on optimisation to have a more performant collision code. This part is optinal.

- We rather use a method to know where a point (the ball's center) is on a grid (the grid of bricks).





- Dividing the ball coordinates by the bricks' width or height will give you its position on the grid

- Erase bricks that "are touched" by the ball, without making the ball bounce. You just have to set the boolean isDestroyed to true and to draw your brick only if isDestoyed is false.

- This method is REALLY important, keep it in mind for similar problems

- Now inverse the ball's vertical speed when a brick is hit

End game

- Reset bricks when the last one is gone

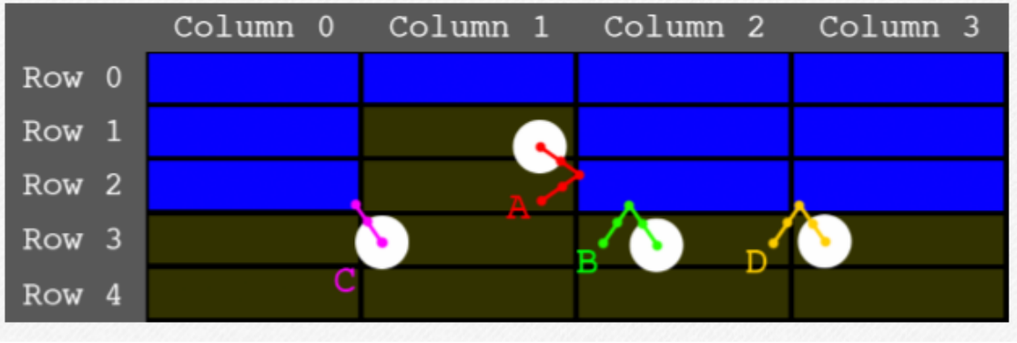
- Efficiency : to know if all the brick are gone, keep a counter of the "living" brick number, instead of checking all the bricks list each frame

- Remove the first three rows

Bonus : better bounces on bricks (optional)

This will help to get a more realistic feeling while playing.

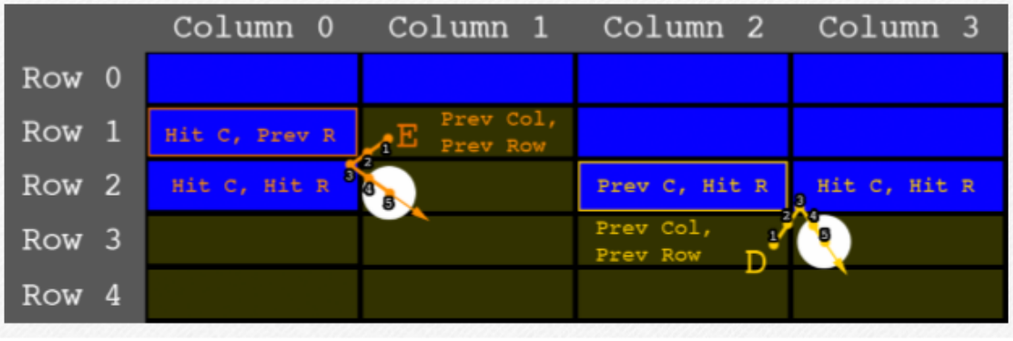
- We would like different bounces directions if the ball hit an other side of a brick



- The key to solve the problem is to check on which part of the grid the ball was on the frame before (using the ball's speed)

- Did it change column (A) ? Did it change row (B) ? Both (C) ?

- D is a special case. There is also case E, D's vertical counterpart :



- We don't want the C scenario to happen in case D or E. The difference with C is we cross a diagonal (because of speed) before the collision

- We handle this case by logging the column or row of the previous frame

- There is a last case, where both adjacent sides are blocked by bricks. It only happens in corners, and we should reverse direction like with C

- Debug by adding a code to move the ball to the mouse position with left click, and change vertical/horizontal speed with right click

Congrats !

Two games ! Not so bad :)