

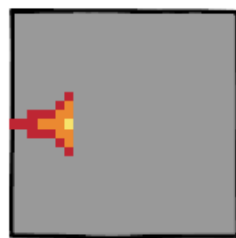
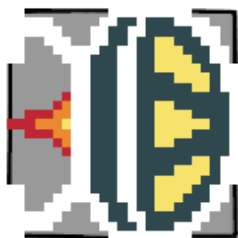
# Lunar Lander



## Building the Lander

# The Lander class

- Create a new Lander class (with a capital L)
- As usual, it will have a :
  - public Lander() constructor function,
  - public void Load(ContentManager content) function
  - public void Update(GameTime gameTime) function
  - public void Draw(GameTime gameTime, SpriteBatch spriteBatch) function
- Create the Lander object in the Game1.cs file and call those functions
- Create a lander image and a lander-fire image with the same format. 25 \* 25



Thus, we will be able to display the propeller fire without complex calculation.

- Add the images in the Monogame Content Manager.

# Displaying the lander

- Our Lander will have a x and a y float members to display it on the screen.
- It will have a float rotation, to make it turn around its axis
- It will have a Texture2D image for the lander itself

Implement this first version of the Lander. Use the following Draw function :

```
public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
    Rectangle rect = new Rectangle((int)x, (int)y, image.Width, image.Height);
    spriteBatch.Draw(image, rect, null, Color.White, rotation, new Vector2(rect.Width / 2, image.Height / 2), SpriteEffects.None, 0);
}
```

We use floats because for a physically-simulated movement, ints would generate strange behaviours.

# Displaying the lander

```
public Lander()
{
    x = 400;
    y = 0;
    rotation = (float)Math.PI * 3 / 2;
}

float x;
float y;
float rotation;
Texture2D image;

public void Load(ContentManager content)
{
    image = content.Load<Texture2D>("lander");
}

public void Update(GameTime gameTime)
{
}

public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
    Rectangle rect = new Rectangle((int)x, (int)y, image.Width, image.Height);
    spriteBatch.Draw(image, rect, null, Color.White, rotation, new Vector2(rect.Width / 2, image.Height / 2), SpriteEffects.None, 0);
}
```

## Remarks:

- The initial rotation orientates our lander to the top.
- The spriteBatch.Draw's 6th argument (new Vector2 ...) set the origin of the Draw to the center of the sprite. It will allow easier rotations.

# Displaying the lander fire

- Add a boolean `isFireOn` member to our lander.
- Add a second `Texture2D` `fireImage` member. Load the image in the `Load` function.
- In the `Update` function, set the boolean `fireOn` to true if the player is pressing the Spacebar.

```
public void Update(GameTime gameTime)
{
    if (Keyboard.GetState().IsKeyDown(Keys.Space))
    {
        ...
    }
}
```

- In the `Draw` function, display the `fireImage` if `fireOn` is true

# Displaying the lander fire

```
...
bool isFireOn;
Texture2D fireImage;

public void Load(ContentManager content)
{
    ...
    fireImage = content.Load<Texture2D>("lander-fire");
}












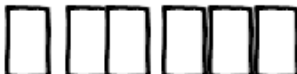

public void Update(GameTime gameTime)
{
    if(Keyboard.GetState().IsKeyDown(Keys.Space))
    {
        isFireOn = true;
    }
    else
    {
        isFireOn = false;
    }
}

public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
    ...
    Rectangle fireRect = new Rectangle((int)x, (int)y, fireImage.Width, fireImage.Height);
    if(isFireOn)
    {
        spriteBatch.Draw(fireImage, fireRect, null, Color.White, rotation, new Vector2(rect.Width / 2, image.Height / 2), SpriteEffects.None, 0);
    }
}
```

# Gravity

- Gravity is an acceleration. Acceleration is a variation of a speed.
- Gravity affect vertical speed. It continuously increase the speed to the bottom of the screen.
- We need a const flat GRAVITY value and a float vx variable.

```
const float GRAVITY = 98.1f;  
float vy;  
  
public void Update(GameTime gameTime)  
{  
    float dt = (float)gameTime.ElapsedGameTime.TotalSeconds;  
    ...  
    vy += GRAVITY * dt;  
    y += vy * dt;  
}
```

	t = 0	t = 1	t = 2	t = 3	t = 4	
Acceleration						(Constant)
Speed						(Linear)
Position						(Quadratic)



# Propeller

Now, when we push the Space bar, we want to create a vertical force, to push the lander up.

- Create a const float PROPULSION that will be the acceleration toward the top of the Lander
- When Space is pushed, add PROPULSION to  $v_y$ .
- Don't forget delta time.
- PROPULSION must be superior to GRAVITY, or it will be impossible to stop the lander.

# Propeller

```
...  
const float PROPULSION = -180f;  
  
public void Update(GameTime gameTime)  
{  
    float dt = (float)gameTime.ElapsedGameTime.TotalSeconds;  
    if(Keyboard.GetState().IsKeyDown(Keys.Space))  
    {  
        isFireOn = true;  
        vy += PROPULSION * dt;  
    }  
    else  
    {  
        isFireOn = false;  
    }  
    ...  
}
```

# Rotation

Now, when we want the Lander to turn around its center when we press right or left arrow.

# Rotation

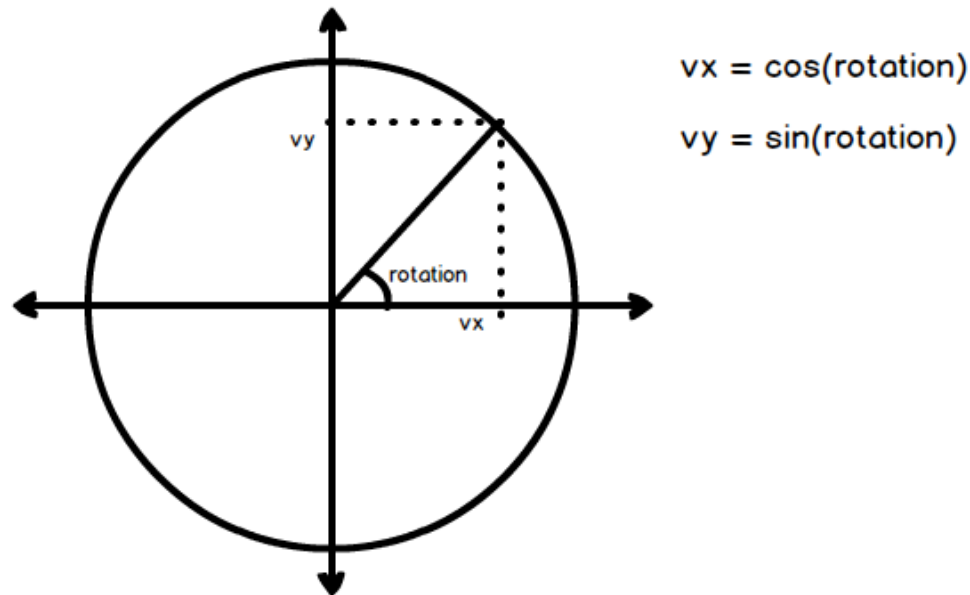
```
...
const float ROTATE_SPEED = 1f;
...

public void Update(GameTime gameTime)
{
    float dt = (float)gameTime.ElapsedGameTime.TotalSeconds;
    if(Keyboard.GetState().IsKeyDown(Keys.Space))
    {
        ...
    }
    else
    {
        ...
    }
    if (Keyboard.GetState().IsKeyDown(Keys.Left))
    {
        rotation -= ROTATE_SPEED * dt;
    }
    if (Keyboard.GetState().IsKeyDown(Keys.Right))
    {
        rotation += ROTATE_SPEED * dt;
    }
    ...
}
```

# Horizontal move

We have a rotation, but the Lander is not propelled in the direction of this rotation. It is because it has no horizontal speed.

The more the angle of the Lander will orientate it to the side, the bigger will become the horizontal speed. We will use trigonometry to find the right proportion of horizontal and vertical speed in function of the angle.



We will let the cosinus and the sinus determine the sign of the  $v_x$  /  $v_y$  value. That's why we will use Absolute value of the PROPULSION constant.

# Horizontal move

```
..
public void Update(GameTime gameTime)
{
    float dt = (float)gameTime.ElapsedGameTime.TotalSeconds;
    if(Keyboard.GetState().IsKeyDown(Keys.Space))
    {
        isFireOn = true;
        vx += (float)Math.Cos(rotation) * Math.Abs(PROPULSION) * dt;
        vy += (float)Math.Sin(rotation) * Math.Abs(PROPULSION) * dt;
    }
    else
    {
        ...
    }
}
```

# A moving Lander

Nice ! We now have a quite-realistic move for our Lunar Lander.

We will now implement the game logic, through a target landing paddle.

Landing Paddle



# A Landing Paddle

Our landing paddle will be a simple small rectangle of 50 \* 25.

- Create the image, import it in the Content Manager
- Create the Paddle class, with a x coordinate, a y coordinate, a Texture2D image, a Load and a Draw function.
- Our paddle will also have a ox and oy members, that will be the offset from the upper left point of the image to the origin. Set ox to image.Width / 2 et oy to 0 in the Load function, once the image loaded :

```
public void Load(ContentManager content)
{
    image = content.Load<Texture2D>("paddle");
    ox = image.Width / 2;
    oy = 0;
}
```

- We will pass those origins in the draw function :

```
public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
    Rectangle rect = new Rectangle(x, y, image.Width, image.Height);
    spriteBatch.Draw(image, rect, null, Color.White, 0, new Vector2(ox, oy), SpriteEffects.None, 0);
}
```

- For now, we will set our Paddle at a hard-coded position. E.g. :

```
public Paddle()
{
    x = 300;
    y = 400;
}
```

- Don't forget to create the Paddle object and Load it in the Game1 class.

# Random paddle position

- Add a Random random member in the Paddle class, a int screenHeight, and int screenWidth.
- Use the Constructor function to set the screenWidth and screenHeight value, and to create the Random instance.
- Use the Load function to set a random x and y value :
  - x shall be between  $\text{image.Width} / 2$  and  $\text{screenWidth} - \text{image.Width} / 2$
  - y shall be between a MINIMUM\_HEIGHT const int and  $\text{screenWidth} - \text{image.Height}$

```
public void Load(ContentManager content)
{
    image = content.Load<Texture2D>("paddle");
    x = random.Next( ..., ... );
    y = random.Next( ..., ... );
}
```

- Launch the game several times : you will see the paddle is set at a different position each time

# Random paddle position

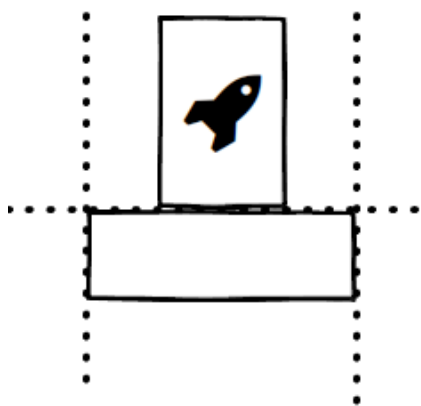
```
...  
int screenWidth;  
int screenHeight;  
  
Random random;  
const int MINIMUM_HEIGHT = 300;  
  
public Paddle(int screenWidth, int screenHeight)  
{  
    this.screenWidth = screenWidth;  
    this.screenHeight = screenHeight;  
    random = new Random();  
}
```

Make the Lander land

# What is landing ? 1/3

When you want to make a gameplay feature, you have to define first :

1. What are the gameplay characteristics of this feature
2. (then) How to translate those characteristics into an algorithm
3. (then and only then) How to structure our code for this algorithm



- The Lander shall be between horizontal bounds of the paddle
- The Lander shall reach the paddle top
- The rotation of the Lander shall be near to zero
- The vertical speed of the lander shall be under a maximum land speed
- The horizontal speed shall be near to zero

## What is landing ? 2/3

- The Lander shall be between horizontal bounds of the paddle. Both sides must be within the paddle width.

} Similar to the pong collision algorithm, but with both ends of the image rect between the start and end of the paddle. If only one end -> crash

- The Lander shall reach the paddle top

} The Lander shall come from the top of the paddle. We will detect it has a positive vertical speed. Else -> crash.

- The rotation of the Lander shall be near to zero

→ We must display the rotation, and define a maximum absolute rotation

- The vertical speed of the lander shall be under a maximum land speed

→ We must display the vertical speed, and define a maximum landing absolute speed

- The horizontal speed shall be near to zero

→ We must display the horizontal speed, and define a maximum landing absolute speed

→ Will be checked with the collision. We need Paddle and Lander Rect property

} We have conditions about the Lander. We will create a boolean property that will tell if the lander is in Landing condition.

} If not, it will count as a crash.

} We need a DisplayText class to display text for the landing result and for the ui.

} We will ask this property when the Lander collides the paddle.

# What is landing ? 3/3

Finally, we will create a outcome int in the main class :

- While Lander has not tried to land, it will be equal to -1
- If the Lander lands well, it will become equal to 0
- If the Lander lands bad, it will become equal to 1

# Landing code 1/2

## Paddle.cs

```
public Rectangle Rect
{
    get
    {
        return new Rectangle(x - ox, y - oy, image.Width, image.Height);
    }
}
```

## Lander.cs

```
const float MAX_VERTICAL_SPEED = 25;
const float MAX_HORIZONTAL_SPEED = 10;
const float MAX_ROTATION = (float)(5 * Math.PI / 180);

public Rectangle Rect
{
    get
    {
        return new Rectangle((int)x - image.Width / 2, (int)y - image.Height / 2, image.Width, image.Height);
    }
}

public bool IsLandingOk
{
    get
    {
        return vy >= 0
            && Math.Abs(vx) < MAX_HORIZONTAL_SPEED
            && Math.Abs(vy) < MAX_VERTICAL_SPEED
            && Math.Abs(rotation + Math.PI / 2) < MAX_ROTATION;
    }
}
```



# Landing code 2/2

Game1.cs

```
int outcome;
...

protected override void Update(GameTime gameTime)
{
    ...
    if (outcome == -1)
    {
        lander.Update(gameTime);
        outcome = LandCheck();
    }

    base.Update(gameTime);
}

int LandCheck()
{
    if (lander.Rect.Intersects(paddle.Rect))
    {
        if (lander.Rect.X >= paddle.Rect.X
            && lander.Rect.X + lander.Rect.Width <= paddle.Rect.X + paddle.Rect.Width
            && lander.IsLandingOk)
        {
            return 0; // Landing ok
        }
        else
        {
            return 1; // Landing ko
        }
    }
    return -1; // No landing
}
```

We need to display the rotation, vertical speed and horizontal speed of the Lander.

Lander UI

# Create a font for a text UI

- Open Monogame Content Manager
- Right click on Content / Add / New item
- Name it "uiFont.spritefont"

# Create UIText class

- Create a new UIText class

UIText.cs

```
class UIText
{
    string label;
    string text;
    int x;
    int y;
    int ox;
    int oy;
    SpriteFont font;

    public string Text
    {
        set
        {
            text = value;
        }
    }

    public void Load(ContentManager content)
    {
        font = content.Load<SpriteFont>("uiFont");
    }

    public UIText(int x, int y, string label, string defaultText)
    {
        this.x = x;
        this.y = y;
        ox = 0;
        oy = 0;
        this.label = label;
        text = defaultText;
    }

    public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
    {
        string labelText = (label == "" ? "" : label + ": ");
        spriteBatch.DrawString(font, labelText + text, new Vector2(x + ox, y + oy), Color.White);
    }
}
```

# Initialize and draw UI text

- Create a new `UIText` class in `Game1.cs`

# Initialize and draw UI text

- Create a new UIText class in Game1.cs

Game1.cs

```
...
UIText rotationUI;
UIText vxUI;
UIText vyUI;
UIText resultUI;

protected override void Initialize()
{
    ...
    rotationUI = new UIText(20, 10, "Rotation", "0");
    vxUI = new UIText(20, 40, "H-Speed", "0");
    vyUI = new UIText(20, 70, "V-Speed", "0");
    resultUI = new UIText(20, 100, "", "");
    base.Initialize();
}

protected override void LoadContent()
{
    ...
    rotationUI.Load(Content);
    vxUI.Load(Content);
    vyUI.Load(Content);
    resultUI.Load(Content);
}

protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.Black);
    spriteBatch.Begin();
    ...
    rotationUI.Draw(gameTime, spriteBatch);
    vxUI.Draw(gameTime, spriteBatch);
    vyUI.Draw(gameTime, spriteBatch);
    resultUI.Draw(gameTime, spriteBatch);

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

# Update UI content

- We need access to Lander's vx, vy and rotation
- Then we can set UIText's text field in Game1's update function.

# Initialize and draw UI text

## Lander.cs

```
public float Rotation
{
    get
    {
        double rad = rotation + Math.PI / 2;
        return (float)Math.Round(rad * 180 / Math.PI, 2);
    }
}

public float Vx
{
    get
    {
        return (float)Math.Round(vx, 2);
    }
}

public float Vy
{
    get
    {
        return (float)Math.Round(vy, 2);
    }
}
```

## Game1.cs

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

    if (outcome == -1)
    {
        lander.Update(gameTime);
        outcome = LandCheck();

        rotationUI.Text = lander.Rotation.ToString();
        vxUI.Text = lander.Vx.ToString();
        vyUI.Text = lander.Vy.ToString();
    }

    base.Update(gameTime);
}
```



# Game result

- We want to change resultUI text in function of the outcome
- `outcome == 0` will display "Success", `outcome == 1` will display "Failure"

# Game result

Game1.cs

```
protected override void Update(GameTime gameTime)
{
    ...
    if (outcome == -1)
    {
        ...
    }
    else if (outcome == 0)
    {
        resultUI.Text = "Success";
    }
    else if (outcome == 1)
    {
        resultUI.Text = "Failure";
    }

    base.Update(gameTime);
}
```

Polish

# Turn the engine off

When the lander has landed, turn off the engine fire

# Restart key

When the player has Landed, restart the game when the player press R

# Title screen

Display title images and text before the game starts. Start the game by pressing space

# Sound

Play sounds :

- When the lander's engine is on
- When the lander lands (one sound for success, another for failure)
- When the lander rotates
- When the game starts from the title screen / restart

Add a music into the game. Restart the music when the game starts / restarts

How to play sounds ?

<http://rbwhitaker.wikidot.com/playing-sound-effects>

# Fuel

Add a new variable in the lander. It will be a fuel variable. This variable decreases while you keep space pressed.

When this variable reaches 0, the engine cannot be turned on anymore



# UI colors

We want rotation, vx and vy UI to be green when their values are proper to a landing.

# Mountains

We will draw lines to figure mountains. The function to draw lines is :

```
DrawLine(spriteBatch, new Vector2(200, 200), new Vector2(100, 50));
```

First Vector2 is the coordinate of the line starting point, second Vector2 is the coordinate of the line ending point.

We will have a point à  $x == 0$ ,  $x == 100$ ,  $x == 200$ ,  $x == 300$  ...,  $x == 800$  (if 800 is the width of the screen).

The y of the point will be chosen as a random value between a MAX\_MOUNTAIN\_HEIGHT value and 500 (if 600 is the height of the screen).

Lines will be drawn between each adjacent point, except if the paddle is between one point and the following point. If so, the line will be drawn between the point and the paddle top left point. If the paddle ends before the next point, a line will be draw between the paddle's top right point and the next point. If the paddle ends after the next point, this point is ignored.

