

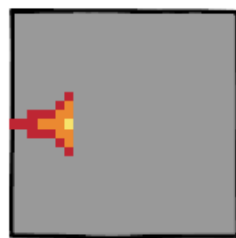
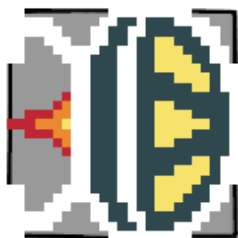
# Lunar Lander



## 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");
}












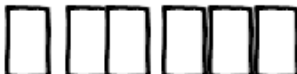

public void Update(GameTime gameTime)
{
    if(Keyboard.GetState().IsKeyDown(Keys.Space))
    {
        isFireOn = true;
    }
    else
    {
        isFireOn = false;
    }
}

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

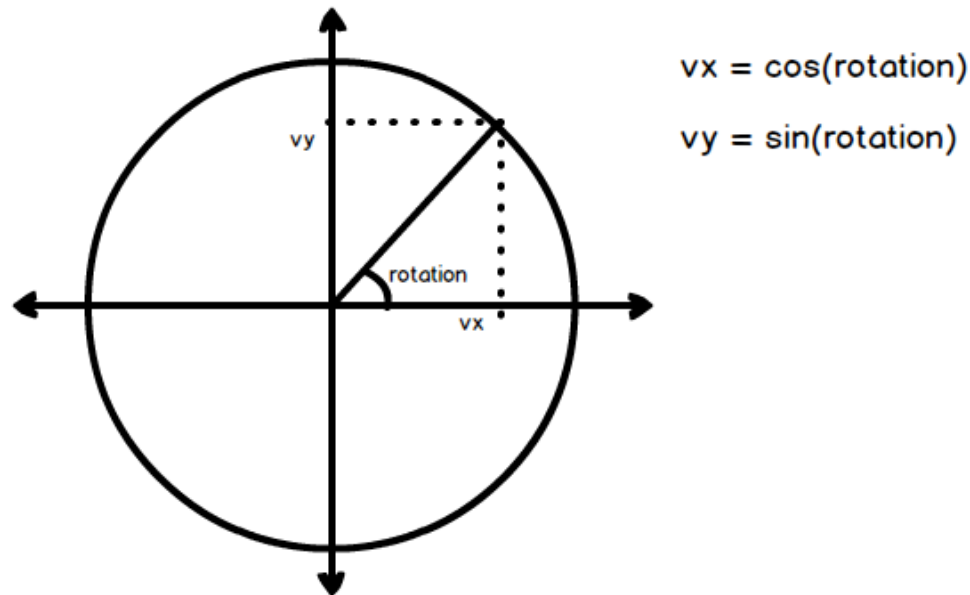
```
...
const float ROTATE_SPEED = 1f;
...

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