Contents

1 Transformation
   1.1 Translations 3
   1.2 Rotations 3
   1.3 Scaling 4

2 Rectangular Collision
   2.1 What Is a Rectangular Collision? 4
   2.2 Detecting Collision or Intersection 4
   2.3 Different Coordinate Systems 5

3 Velocity
   3.1 Direction 6
   3.2 Sprite Direction Using Vector Coordinates (x, y) 6
   3.3 Speed 7

Game Implementation:
Adding movement to the “Condor” sprite

   Step 1: Positioning the “Condor” Sprite 7
   Step 2: Setting Its Vector 8
1 Transformation

1.1 Translations

A translation transformation is applied to an object by repositioning it along a straight-line path from one coordinate location to another.

Facts:
- $t_x$ and $t_y$ are called translation distances along the $x$-axis and the $y$-axis.
- $\overrightarrow{T}(t_x, t_y)$ is called the translation vector.

1.2 Rotations

Rotations rotate a point along a circular path. To specify a rotation transformation we need:

- An angle.
- A pivot point (reference for the rotation).
- A rotation axis. In 2D, the axis is perpendicular to the $x$-$y$ plane. It is the $z$-axis.
- To specify a positive (counterclockwise) and a negative (clockwise) rotation angles.
1.3 Scaling

Scaling alters the size of objects. It requires scaling factors $s_x$ and $s_y$ that determine the change in the $x$-direction and the $y$-direction. Scaling is carried out by multiplying the coordinate values of each vertex of object with the scaling factors.

Example:
- $x' = s_x x$
- $y' = s_y y$

If $s_x = s_y$, we have a uniform scaling; if $s_x \neq s_y$, we have a non-uniform scaling.

2 Rectangular Collision

2.1 What Is a Rectangular Collision?

A rectangular collision is a rectangle relative to each frame. It sets the boundaries of a sprite and is used to check for collisions.

2.2 Detecting Collision or Intersection

Facts:
- The limit of the first bounding box is determined by: $L = 1$, $R = 3$, $T = 3$, and $B = 1$.
- The limit of the second bounding box is determined by: $L' = 2$, $R' = 4$, $T' = 4$, and $B' = 2$.
- Find the following:
  - $\text{Max}(L, L') = \text{Max}(1, 2) = 2$
  - $\text{Min}(R, R') = \text{Min}(3, 4) = 3$
  - $\text{Max}(B, B') = \text{Max}(1, 2) = 2$
  - $\text{Min}(T, T') = \text{Min}(3, 4) = 3$
- The two bounding boxes intersect if:
  - $\text{Max}(\text{Max}(L,L') – \text{Min}(R, R'), \text{Max}(B, B') – \text{Min}(T, T')) \leq 0$. 
\[
\text{Max}(2 - 3, 2 - 3) \leq 0 \Rightarrow \text{Max}(-1, -1) = -1
\]

- Result: The two bounding boxes intersect and form a new rectangle: L=2, R=3, T=3, and B=2.

Facts:

- The limit of the first bounding box is determined by:
  \[L = 1.2, R = 2.8, T = 2.6, \text{and } B = 1.\]

- The limit of the second bounding box is determined by:
  \[L' = 3.6, R' = 5.3, T' = 4.5, \text{and } B' = 3.\]

- Find the following:
  \[\text{Max}(L, L') = \text{Max}(1.2, 3.6) = 3.6\]
  \[\text{Min}(R, R') = \text{Min}(2.8, 5.3) = 2.8\]
  \[\text{Max}(B, B') = \text{Max}(1, 3) = 3\]
  \[\text{Min}(T, T') = \text{Min}(2.6, 4.5) = 2.6\]

- The two bounding boxes intersect if:
  \[\text{Max}(\text{Max}(L, L') - \text{Min}(R, R'), \text{Max}(B, B') - \text{Min}(T, T')) \leq 0.\]
  \[\text{Max}(3.6 - 2.8, 3 - 2.6) \leq 0 \Rightarrow \text{Max}(0.8, 0.4) = 0.8 > 0\]

- Result: The two bounding boxes do not intersect since the result is positive.

### 2.3 Different Coordinate Systems

A screen coordinate system is where top left is (0, 0). For example:
The same equation of the previous section applies with the following modifications:

- Nothing changes regarding the left and the right.
- Since the top has a value less than the bottom:
  - \( \max(B, B') \) will be \( \min(B, B') \), and
  - \( \min(T, T') \) will be \( \max(T, T') \).
- Finally, the equation should be:
  - \( \max(\max(L, L') - \min(R, R'), \min(B, B') - \max(T, T')) \leq 0 \).

### 3 Velocity

The velocity \( V(x, y) \) is the speed and direction describing a moving object.

#### 3.1 Direction

Unit vectors are used to indicate the direction.

**Example:**

\[
uv \left( \frac{x}{l}, \frac{y}{l} \right) \quad \text{where} \quad l = \sqrt{x^2 + y^2}
\]

#### 3.2 Sprite Direction Using Vector Coordinates \((x, y)\)

To find the sprite direction using vector coordinates \((x, y)\):
Get the length \( L \) of the vector \((x, y)\)
\[ L = \sqrt{x^2 + y^2} \]

The unit vector will be:
\[ X = \frac{x}{L} \]
\[ Y = \frac{y}{L} \]

According to a different coordinate system, the unit vector will be:
\[ X = \frac{x}{L} \]
\[ Y = -\frac{y}{L} \]

\[
\begin{array}{|c|c|c|c|}
\hline
X & Y & \text{Forward} & \text{Backward} & \text{Upward} & \text{Downward} \\
\hline
1 & 0 & \uparrow & \leftarrow & \uparrow & \downarrow \\
-1 & 0 & \uparrow & \leftarrow & \uparrow & \downarrow \\
0 & 1 & \uparrow & \uparrow & \uparrow & \downarrow \\
0 & -1 & \uparrow & \uparrow & \uparrow & \downarrow \\
\hline
\end{array}
\]

3.3 Speed

The speed is the magnitude of \( V(x, y) \).

Example:
\[ S = \sqrt{x^2 + y^2} \]

**Game Implementation:**

**Adding movement to the “Condor” sprite**

**Step 1: Positioning the “Condor” Sprite**

- Create a copy of the Condor object with all its properties and values and called it condor.

```csharp
Condor condor = (Condor)Condor.Clone();
```

- Set the sprite position.

```csharp
condor.Position = new Point(m_Random.Next(-100, 740), -150);
```

- Add the sprite to the game.

```csharp
Game.Add(condor);
```

- Type the following code in the ‘StarTrooperGame.cs’ file – in the StarTrooper class under public override void Update() function inside the if (){} statement.

```csharp
Condor condor = (Condor)Condor.Clone();
condor.Position = new Point(m_Random.Next(-100, 740), -150);
Game.Add(condor);
```
**Step 2: Setting Its Vector**

- Save the reference of the existing sprite Trooper in the *StarTrooperGame* class object b.
  
  ```csharp
  Trooper b = StarTrooperGame.Trooper;
  ```

- v is a class Vector2 object with a direction to the Trooper position.
  
  ```csharp
  ```

- Normalize the vector by changing the speed value to 1 while preserving the direction.
  
  ```csharp
  v.Normalize();
  ```

- Assign the new setting to the velocity.
  
  ```csharp
  Velocity = v;
  ```

- The condor will flip its animation relative to the trooper’s position.
  
  ```csharp
  if (v.X >= 0)
    ScaleX = 1;
  else
    ScaleX = -1;
  ```

- Type the following code in the ‘StarTrooperSprite.cs’ file – in the Condor class under public override void Update() function.

```csharp
Trooper b = StarTrooper.Trooper;
v.Normalize();
Velocity = v;
if (v.X >= 0)
  ScaleX = 1;
else
  ScaleX = -1;
```