Lecture 12 | Jumping

CS230
Game
Implementation
Techniques

1. Platformer – Physics

1.1. Jump

- Jumping is used in many games, especially in platform games.
- There are several ways to implement jumping.
- Hard coded elevation: Each elevation height is preset in some data table.
- Linear elevation: Elevation speed is constant both while going up and down.
- Using acceleration: The elevation is controlled by a second degree equation.

- Hard coded elevation (frame based):
  - Array of heights
  - Each array element represents a jump height which will be used for a certain defined number of frames.
  - When the jump is triggered, start by setting the Y position of the sprite to the first element of the "elevation array".
  - Now let's say the traversal counter is 5, which means the sprite's height changes every 5 game loops. After 5 loops, set the height of the sprite to the next element in the "elevation array, and so on until the jump is completed.
  - Special care must be taken in case the sprite lands on a platform higher or lower than the one he jumped from.

- Linear elevation
  - The elevation's speed is constant both while going up and down.
  - When the jump is triggered, set the current jumping mode to "GoingUp"
  - While GoingUp is enabled, increment the Y position of the sprite by a certain value each game loop until it reaches a certain pre-set value.
  - When the elevation reaches that pre-set height limit value, disable GoingUp and enable GoingDown.
  - While GoingDown is enabled, decrement the Y position of the sprite by a certain value each game loop.
  - GoingDown is disabled when the sprite collides back with a platform.

- Using acceleration
  - The elevation is controlled by a second degree equation.
  - This jumping technique imitates "real world jumping" better than the aforementioned 2 techniques.
  - It simulates applying a small upward force to the object as the jump triggered, and then allows the gravity to smoothly stop the sprite from going up and to start pulling it down.
  - The mathematical equation is: $Pos_1 = \frac{1}{2}at^2 + V_0t + Pos_0$
  - Pos_1: Current position of the object.
  - V_0: The velocity during the previous frame.

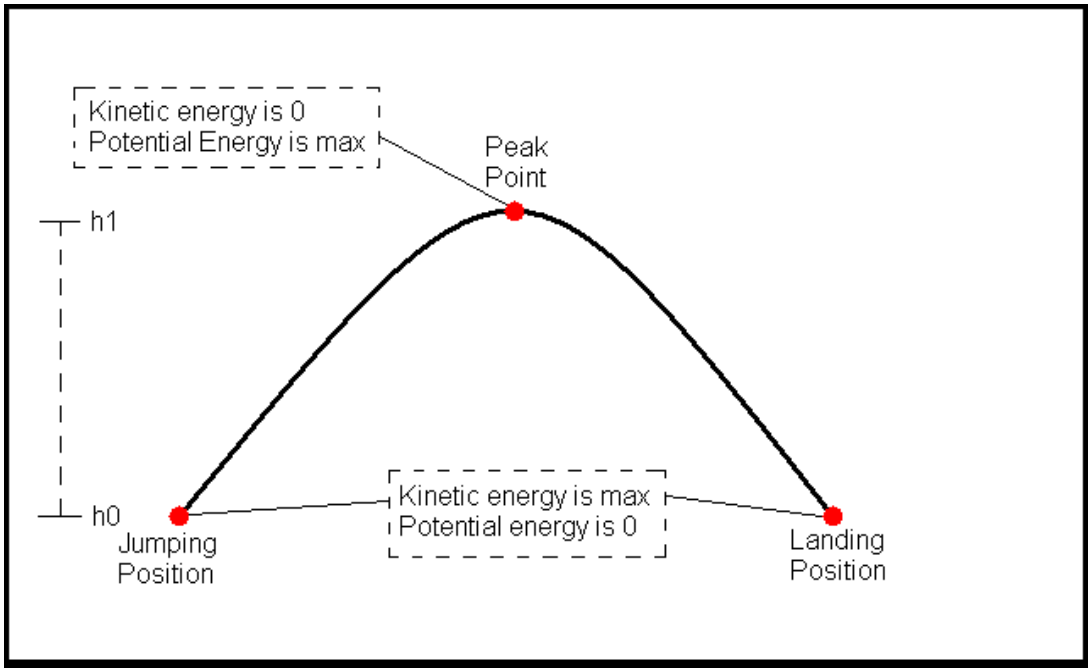
  - Pos_0: Previous position of the object (during the previous frame)
  - t: Time incremental between game loops. In time based games, it represents the frame time. In frame based games, it can be any value.
  - a: Acceleration of the object. In our case it's gravity, which is (0, -g) where g being your wanted gravity force. Note that your gravity value doesn't have to be 9.8 as in real life.
- Since jumping usually affects only the Y positions of game objects, the X component of the acceleration (which is a vector) is 0. Note that this is not a general rule, because some games could be designed in a way that game objects can walk normally on walls for example, and jump horizontally.
- Although the previous equation is correct, we can use a simpler approach in games.
- We can divide the above equation into 2 parts:
  - $V_1 = a*t + V_0$ (1)
  - $Pos_1 = V_1*t + Pos_0$ (2)
- Each time the player presses the jump key, change the Y component of the velocity using equation (1), or you can just set it to any positive value.
- Then update the position of the object using equation (2)
- The object will start moving upward, but the downward acceleration (gravity) will pull it back down, thus creating the jump effect.
  - Example:

```
//Apply gravity
// Gravity downward acceleration is -20
Velocity Y = -20 * Frame Time + Velocity Y

//Activate jump
//Jump upward velocity is 10
if(Jump key is pressed)
 Velocity Y = 10

//Updating the position
Pos1 = Velocity * Time + Pos0
```

- Jumping motion
  - So how to produce a good jumping motion?
  - Will it be too low?
  - Will it be too high?
  - We should be able to control the maximum height that the object should reach when jumping.
  - Since the acceleration is constant (-g, whatever g is), we have control only over the initial upward velocity)
  - The maximum height reached by the game object is directly dependent on the initial velocity (which is set upon triggering the jump button).
  - Remember that energy is conserved during the entire jumping motion. It goes from being fully kinetic (Maximum velocity at ground level), then to fully potential energy (Velocity is null at peak point), then back to fully kinetic (at landing point).



  - The kinetic energy is computed using the mass and the velocity: $\frac{1}{2}mv^2$
  - The potential energy is computed using the mass, the acceleration value and the height: mgh
- The total energy (Kinetic + Potential) is constant throughout the entire jump.
  - $\frac{1}{2}mv_0^2 + mgh_0 = \frac{1}{2}mv_1^2 + mgh_1$
  - 0 represents values at the beginning of the jump
  - 1 represents values at the peak of the jump
- The X component of the velocities and the gravity is equal to 0, it can be removed from the equation:
  - $\frac{1}{2}mv_{y0}^2 + mg_yh_0 = \frac{1}{2}mv_{y1}^2 + mg_yh_1$
- Divide by m
  - $\frac{1}{2}v_{y0}^2 + g_yh_0 = \frac{1}{2}v_{y1}^2 + g_yh_1$
- The velocity at the peak of the jump is 0 on the Y axis, therefore:
  - $\frac{1}{2}v_{y0}^2 + gh_0 = gh_1$
  - $\frac{1}{2}v_{y0}^2 = gh_1 - gh_0$
  - $\frac{1}{2}v_{y0}^2 = g(h_1 - h_0)$
  - $v_{y0} = \sqrt{2g(h_1 - h_0)}$
- Therefore, the initial upward velocity should be set depending on the desired jump peak, which is the difference between the highest and lowest point of the jump.