Lab7 - Gameplay and Game Mechanics

COMP394-001 - Practical Game Programming
Fall 2024
Instructions

Due Date(s):

- Class Work portion: in the end of class(es)
- The challenge portion: by end-of-week's Friday, 11:59PM. (you can try to finish most of the work in class as well).

ClassWork (50%): - Follow the hands-on class work taking relevant **snapshots** in a document named **Lab7_Snapshots_{YourInitials}**.

Substitute all instances of *{YourInitials}* with the,...wait for it..., your initials :)

Challenge (50%): Continue working with the class instructions on your own til you complete the Lab7.

Gameplay and Game Mechanics

Gameplay comes in many different guises, and can mean many different things. Let's implement some recipes and solutions to some of the features that we may have to build and/or use over and over again in different game engines.

0.1 InClass work

In class we implemented a few mechanics of the Platformers as seen in Scott Rogers' Level Up book:

- PlatformMove.cs: Horizontal and Vertical Movement (using Mathf.PingPong)
- PlatformMoveWithSin.cs: Horizontal and Vertical Movement (using Mathf.Sin)
- PlatformExpand.cs: Expand the platform uniformly at a given size (using Mathf.PingPong)

We playtested them with a simple scene with cube as platforms sized up suitably.

The files from the class work:

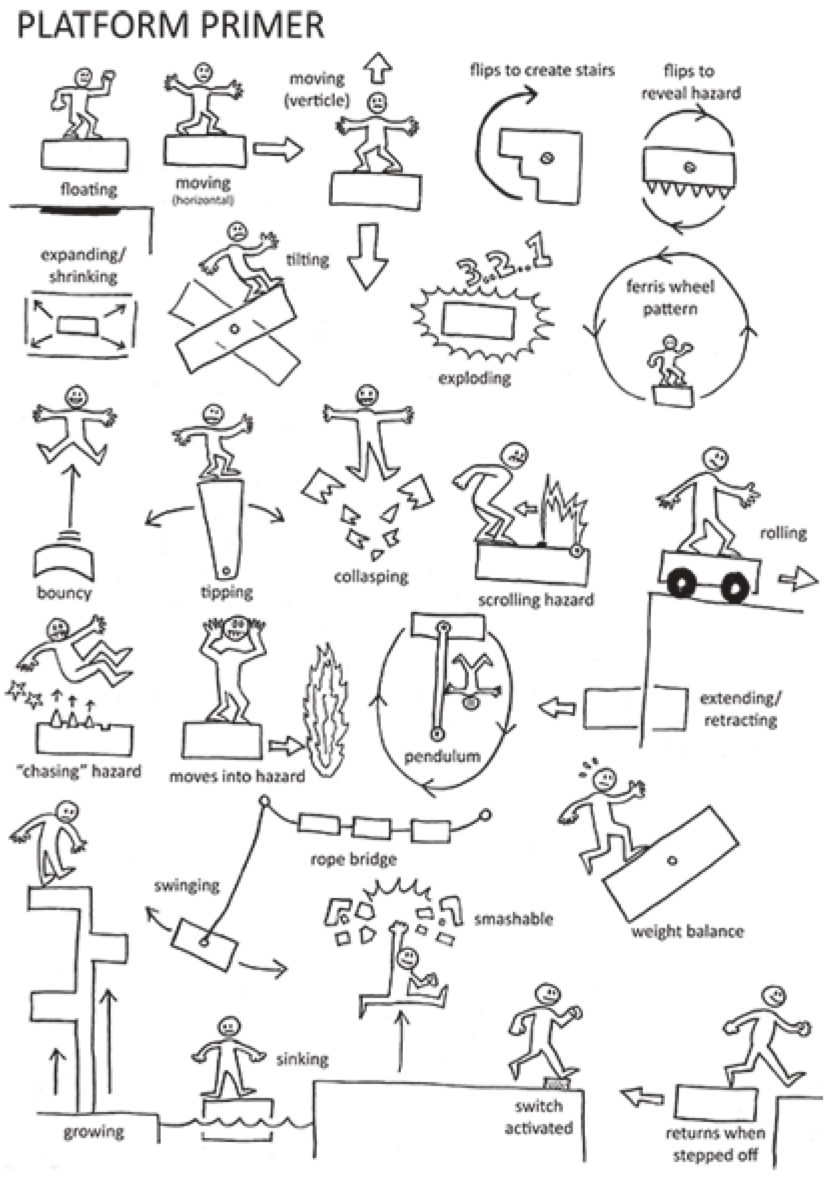
0.0.1 PlatformMove.cs

```
using UnityEngine;
public class PlatformMove : MonoBehaviour
{
  public float speed = 5;                 // m/sec
  public float maxHorizontalDistance = 4; // m
  public float verticalSpeed = 3;         // m/sec
  public float maxVerticalDistance = 5;   // m
  public Vector3 originalPos;
  // Start is called before the first frame update
  void Start()
  {
    originalPos = this.transform.position;
  }
  void FixedUpdate()
  {
    if (maxHorizontalDistance > 0)
    {
      var newX = Mathf.PingPong(Time.fixedTime * speed, maxHorizontalDistance);
       print($"Time.fixedTime={Time.fixedTime}, newX={newX}");
       this.transform.position = originalPos+ new Vector3(newX, 0, 0);
    }
    else if (maxVerticalDistance > 0)
    {
      var newY = Mathf.PingPong(Time.fixedTime * verticalSpeed, maxVerticalDistance);
      this.transform.position = originalPos + new Vector3(0,newY,0);
    }
  }
}
```

0.0.2 PlatformMoveWithSin.cs

```
using UnityEngine;
public class PlatformMoveWithSin : MonoBehaviour
{
  public float speed = 5; // m/sec
  public float maxHorizontalDistance = 4; // m
  public float verticalSpeed = 3; // m/sec
  public float maxVerticalDistance = 5; // m
  public Vector3 originalPos;
  // Start is called before the first frame update
  void Start()
  {
    originalPos = this.transform.position;
  }
  void FixedUpdate()
  {
    if (maxHorizontalDistance > 0)
    {
      var newDeltaPos = Vector3.zero;
      newDeltaPos.x =
maxHorizontalDistance*Mathf.Abs(Mathf.Sin(Time.fixedTime*speed/maxHorizontalDistance*Math
      print($"Time.fixedTime={Time.fixedTime}, newDeltaPos.x={newDeltaPos.x}");
      this.transform.position = originalPos + newDeltaPos;
    }
    else if (maxVerticalDistance > 0)
    {
      //TODO: Implement vertical movement with Mathf.Sin, similarly as above
      // ...
    }
  }
}
```

Challenge was to implement 3-4 more mechanics shown on that picture reproduced hereby:



PlatformerMechanics

Using Randomization in Games

Use the given Excel and .cs files as well as the Class Work to guide you in completing the following:

- Using Excel, simulate N8 Normal Distribution pseudo-random variable by adding 8 uniform distribution random variables (using Excel's RAND() function)
- Create it's corresponding histogram with 10, 20 and 30 bins. (In class we did the cases with averaging 2 and 3 uniform distribution variables and corresponding histograms with 10 bins each).

- Create the 20 and 30 bin histograms for N3 (with 3 RAND() calls).
- Compare N3 with N8 in terms of proximity with normal distribution.
- Answer the following questions:
  - Which number of bins works best, 10, 20 or 30?
  - Did you need to make corrections to start/end points of the bins, to get a better picture? If yes, what changes did you do?

- In Unity, implement two functions with the following signature:

```
void float Normal01(int Count){
  //...
}
```

and

```
void float Normal(float mu, float sigma, int Count){
  //...
}
```

- Note that **count* is the number of Uniformly Distributed random variables used to “simulate” the normal distribution
- Generate 500 points from each function and try to create histograms out of them with 20 bins each.
- Answer the following:
  - Does the resulting distributioes seem like Normal?