Software Studio

軟體設計與實驗

Action System Tutorial

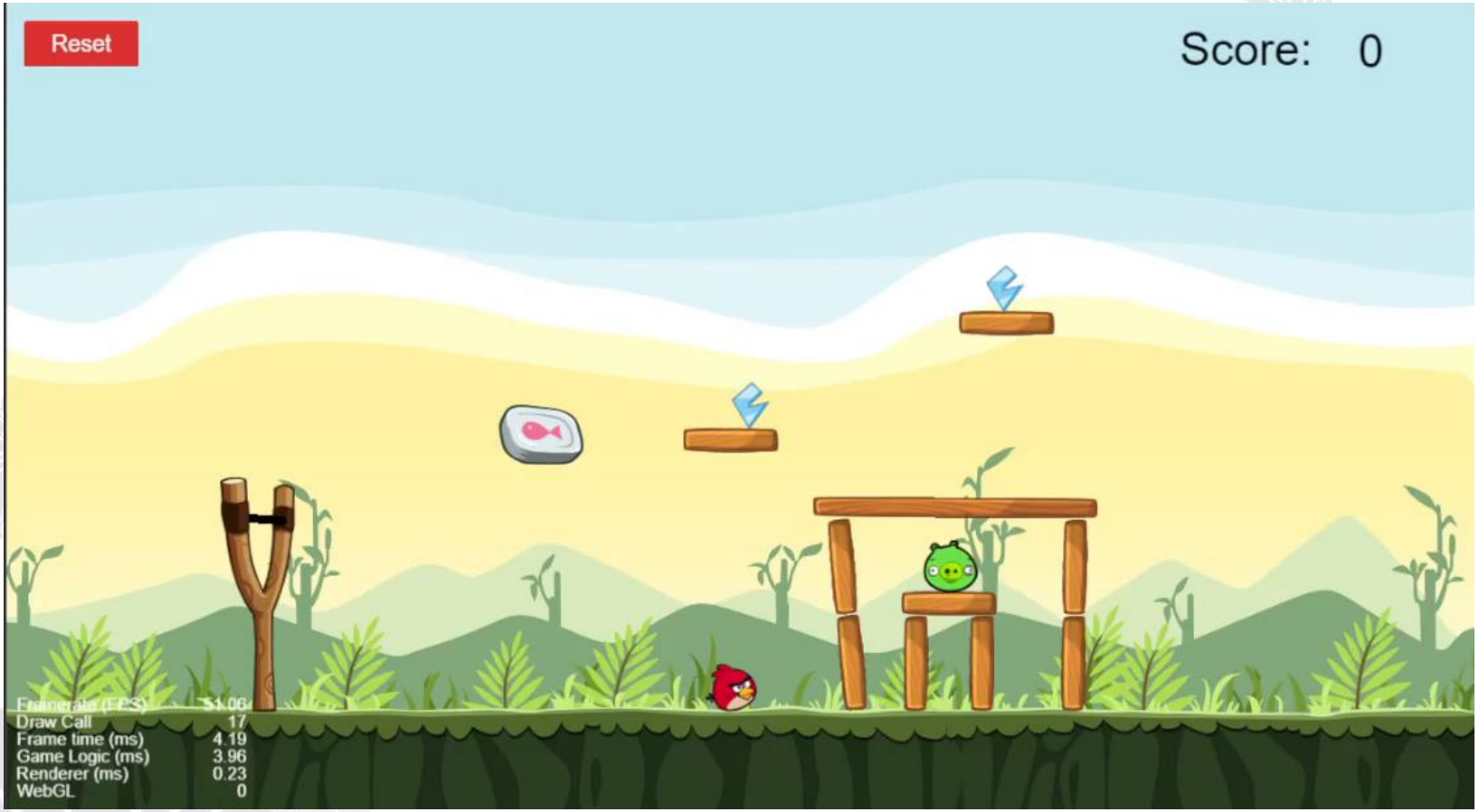
Hung-Kuo Chu

Department of Computer Science

National Tsing Hua University

CS2410

Goal

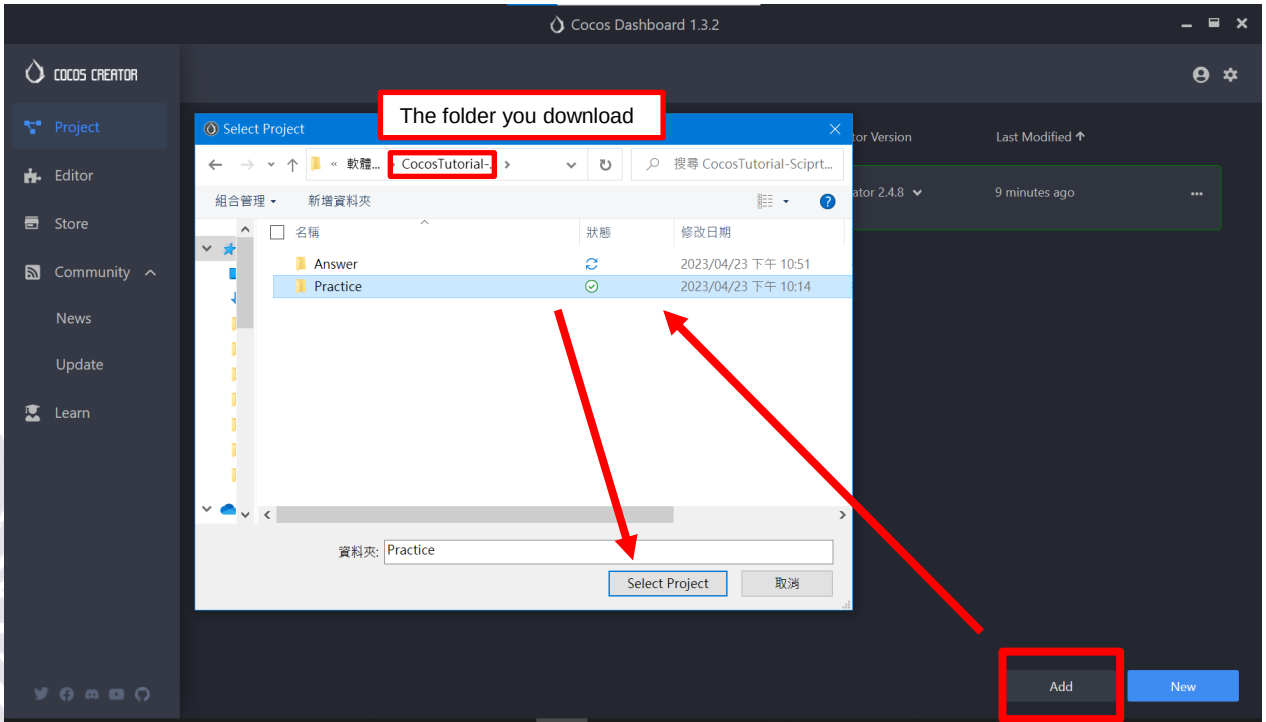

Reset
Score: 0
Draw Call 17
Frame time (ms) 4.19
Game Logic (ms) 3.96
Renderer (ms) 0.23
WebGL 0

Contents

- Create moving platforms and spikes
- Split into two birds

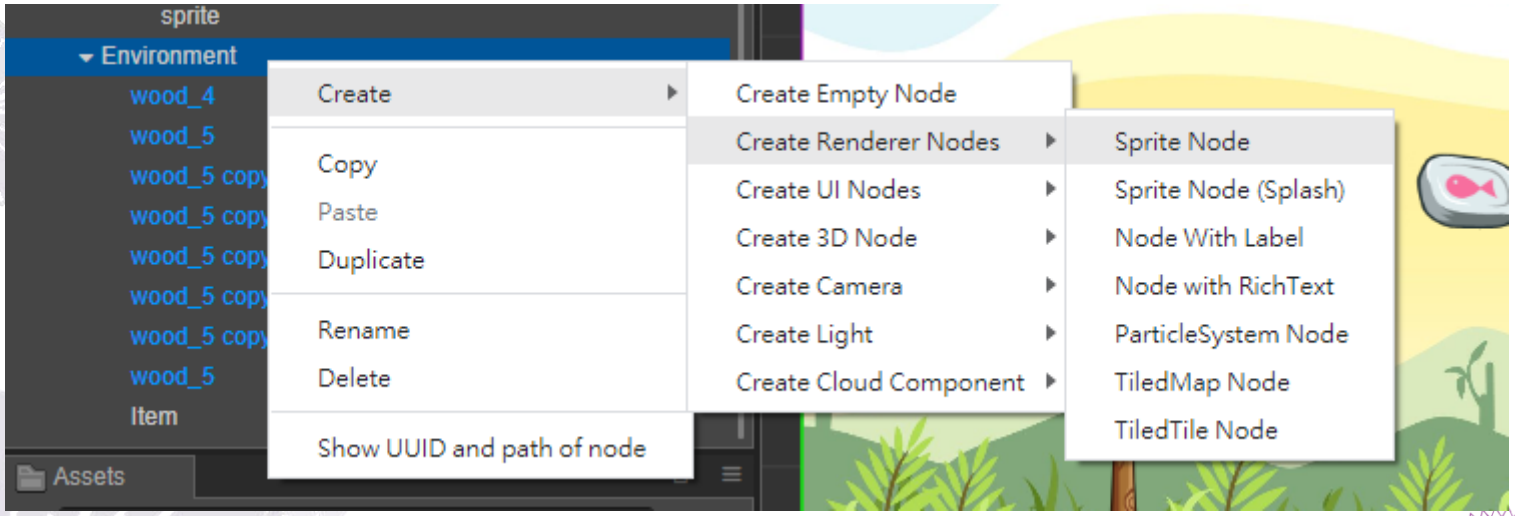
Open the project

- Step1. Download project from eeclass or GoogleDrive and unzip
  - https://shorturl.at/hCHR0
- Step2. Add the Practice folder to Cocos



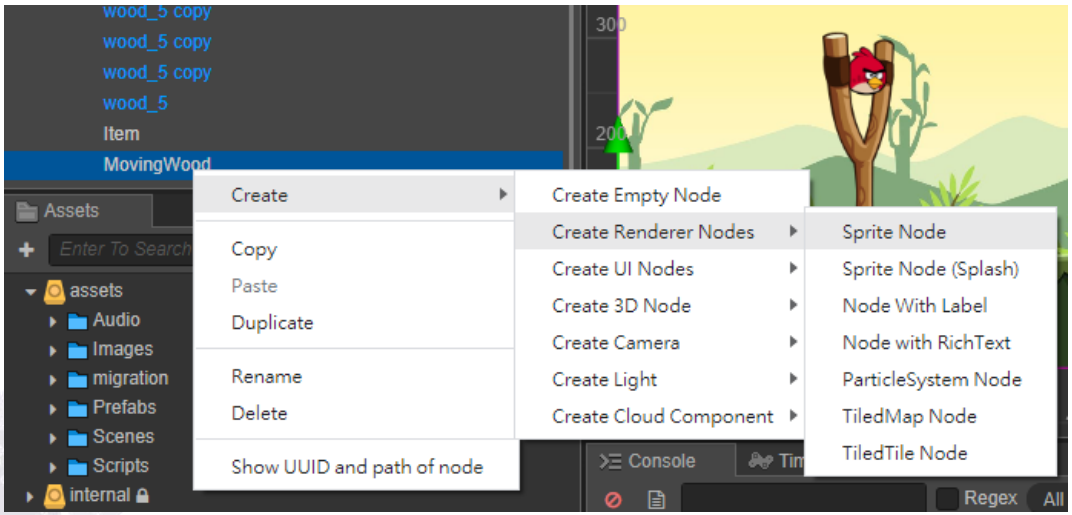
Create platforms and spikes

- Step 1: Create a Sprite node under Environment node as platforms and rename it to “**MovingWood**”
  - Right Click Environment
  - Create/Create Renderer Nodes/Sprite node



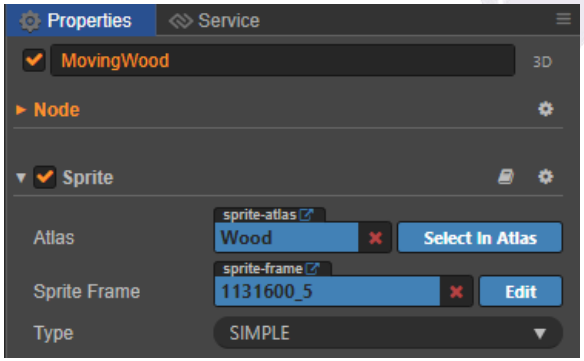
Create platforms and spikes

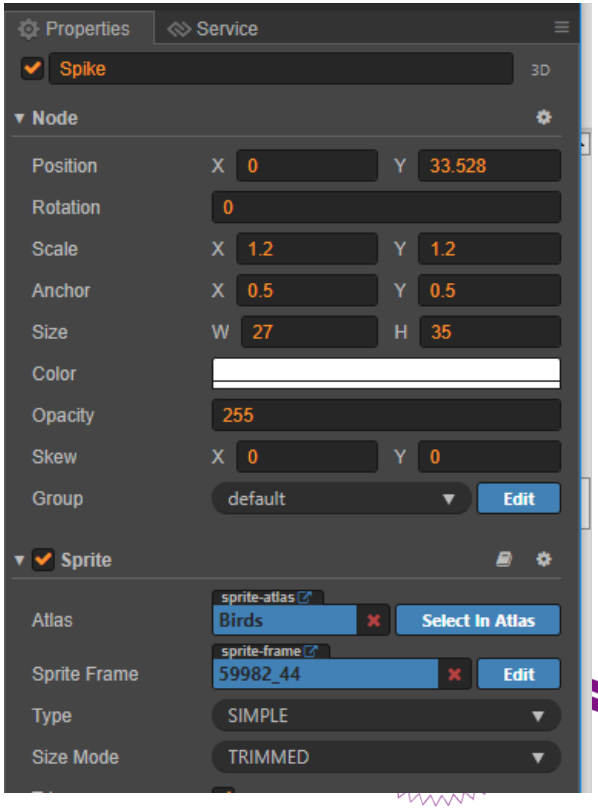
- Step 2: Create a Sprite node under MovingWood node as spike and rename it to “**Spike**”
  - Right Click MovingWood
  - Create/Create Renderer Nodes/Sprite node

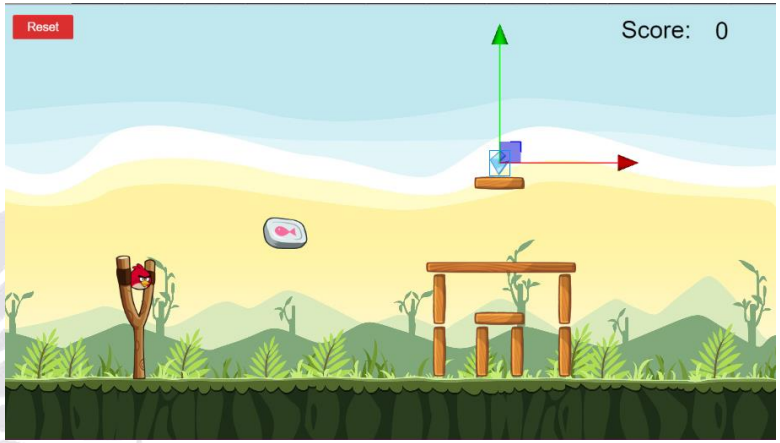


Create platforms and spikes

- Step 3: Find a good image sprite frame
  - assets/Images/
- Step 4: Find a good position and set the scale

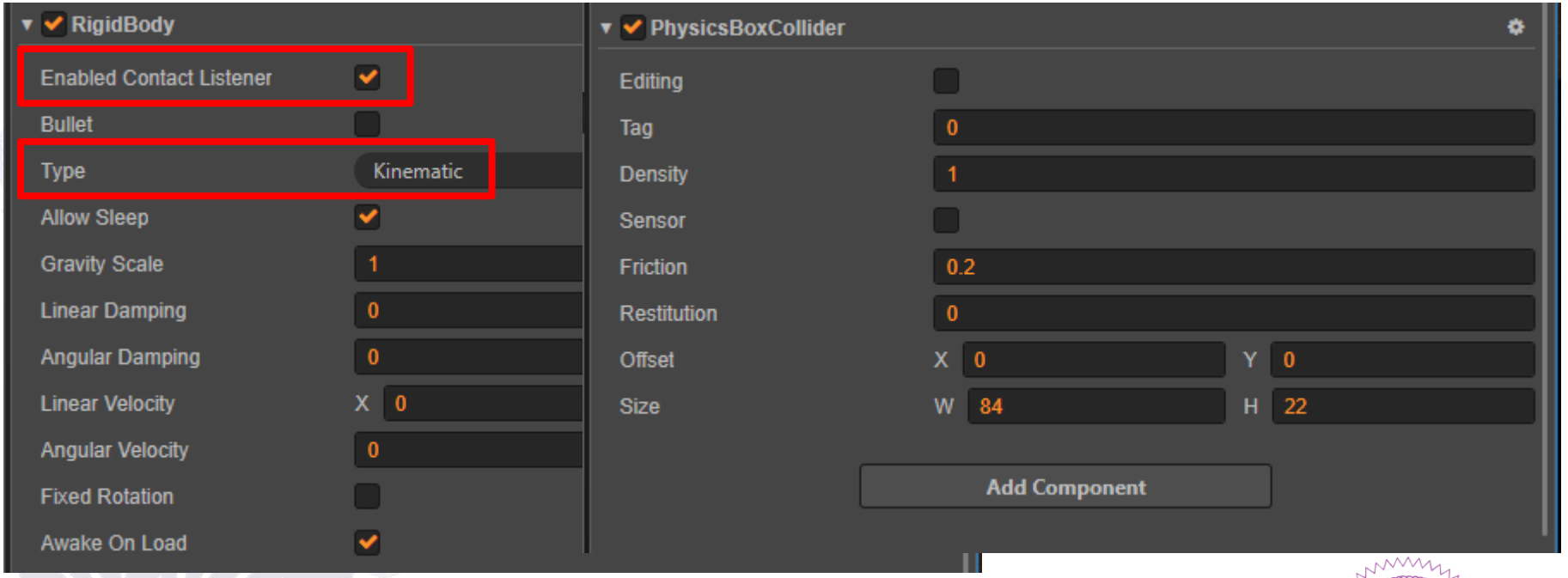






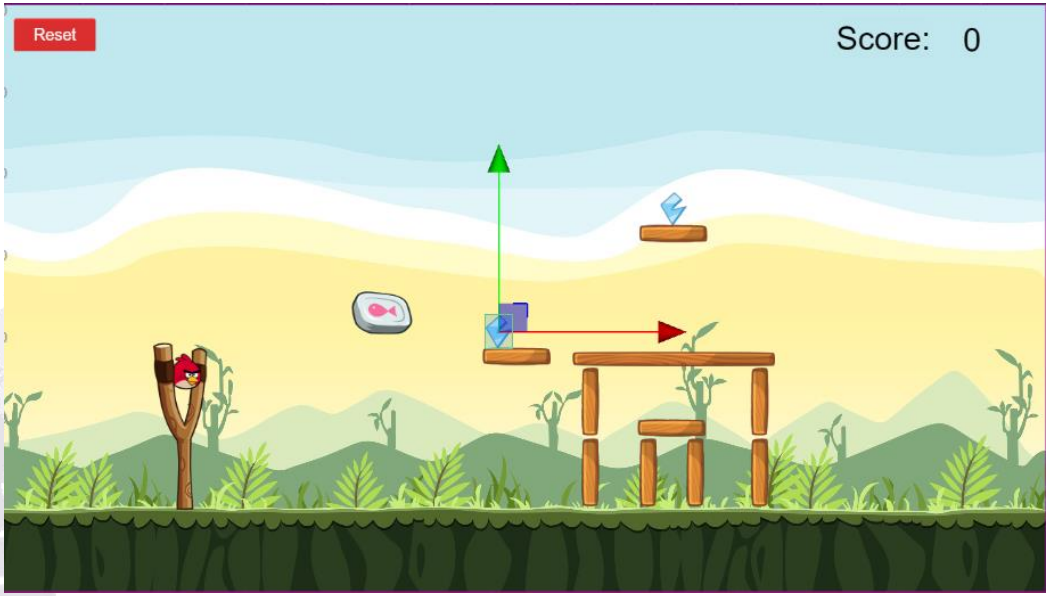
Create platforms and spikes

- Step 5: Set up RigidBody and Collider on both wood & spike
  - Use **Kinematic** type



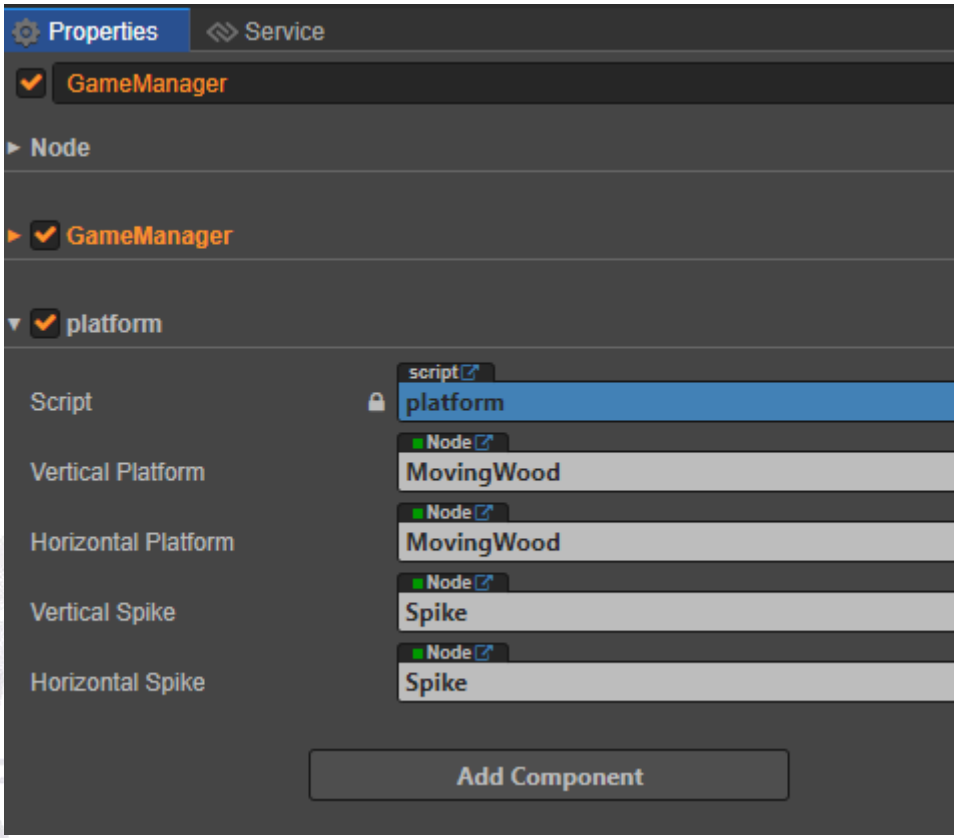
Create platforms and spikes

- Step 6: Copy to create another platform
- Step 7: Find a good position



Create platforms and spikes

- Step 8: Drag them into GameManager node's platform component




Your
turn!
C++
PASCAL

Recap: Basic Action

- Interval Action:
  - Interval action is a gradual change action that is done in a certain time interval.
  - E.g.: **cc.moveBy**, cc.MoveTo, cc.rotateTo
- Free Action:
  - Different from interval actions, free actions run immediately.
  - E.g.: cc.show, cc.hide, cc.removeSelf

Recap: Container Action

- The container action can organize actions in different ways, such as:
  - **Sequential** action
  - **Synchronization** action
  - **Repetitive** action
  - **Repeat forever** action
  - **Speed** action
  - **Combination** action

Recap: Sequential Action

- Sequential action makes a series of child actions run one by one.
- Use **cc.sequence** to create a sequential action.

```
// the action will make the node move back and forth
let action = cc.sequence(cc.moveBy(1, 200, 0), cc.moveBy(1, -200, 0));
this.node.runAction(action);
```

Recap: Repeat Forever Action

- Repeat forever action can make the target action repeat forever until it is stopped manually.
- Use **cc.repeatForever** to create a repeat forever action.

```
// the action will make the node move back and forth and keep repeating
let action = cc.repeatForever(
    cc.sequence(cc.moveBy(1, 200, 0), cc.moveBy(1, -200, 0)) );

this.node.runAction(action);
```

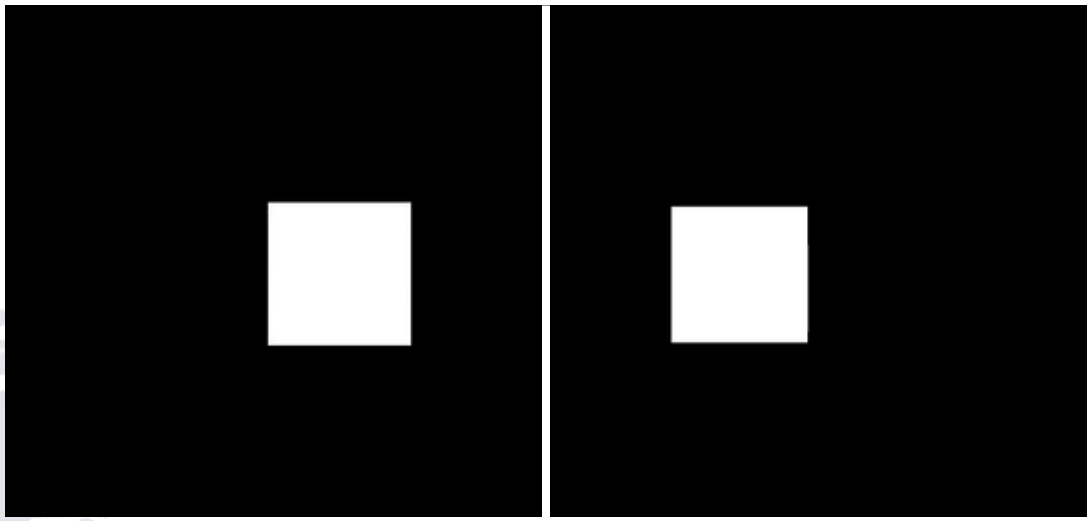
Recap: Slow Motion

- Slow motion is used to alter the time curve of the basic action to give the action fast in/out, ease in or other complicated special effects.
- Slow motion **cannot exist alone**.
- **Only interval actions** support slow motion.

Recap: Slow Motion: Example

- We can modify **scaleTo** action by:

```
let scaleUp = cc.scaleTo(1, 2);
let scaleDown = cc.scaleTo(1, 1);
scaleUp.easing(cc.easeIn(3.0));
this.node.runAction(cc.sequence(scaleUp, scaleDown)).repeatForever();
```



wo/ slow motion w/ slow motion

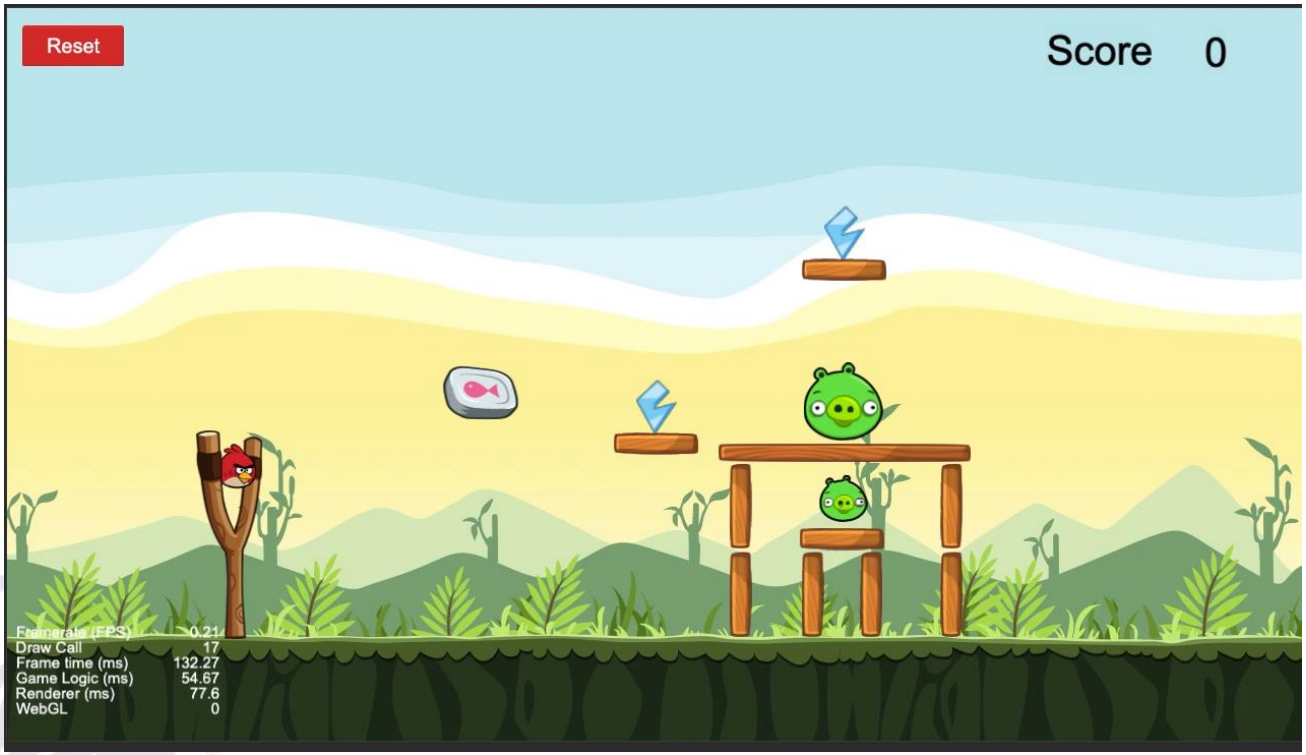
Design action function

- Step 1: In platform.ts, create a function platformMove() (TODO 1.1)
  - moveDir: determine moving direction
  - delayTime: when platform will start to move after game starts
- Step 2: Design moving action sequence
  - Use **cc.moveBy(time , x , y)** to move
  - Use **cc.sequence** to cascade actions

```
43    platformMove(moveDir: string, delayTime: number, platform: cc.Node) {
44        let action: cc.Action;
45        let easeRate: number = 2;
46        var sequence1 = cc.sequence(cc.moveBy(2, 0, 120).easing(cc.easeInOut(easeRate)), cc.moveBy(2, 0, -120).easing(cc.easeInOut(easeRate)));
47        var sequence2 = cc.sequence(cc.moveBy(2, 120, 0).easing(cc.easeInOut(easeRate)), cc.moveBy(2, -120, 0).easing(cc.easeInOut(easeRate)));
48        if (moveDir == "vertical") {
```

Design action function

- Without using easeInOut():



Recap: Scheduler

- Scheduler is a **timer** component for programmers to design time-related functions.
- Compared to javascript timing events, such as **setTimeout** and **setInterval**, scheduler is preferred because it is more powerful, and it combines better with other components in Cocos Creator.

Recap: Scheduler

- The syntax of schedule()

```
let interval = 2; // time interval in the unit of second
let repeat = 3; // time of repetition
let delay = 5; // start delay

// the schedule will execute 3+1 times every 2 seconds after 5 seconds
this.schedule(function() {
    cc.log("Hello world!"); }, interval, repeat, delay);
```

Recap: Schedule Once

- If we only want to execute an event once, we can use **scheduleOnce**.
- For example, the two piece of codes are the same:

```
// the schedule will execute once after 2 seconds
this.schedule(function() { cc.log("Hello world!"); }, 0, 0, 2);

// the schedule will execute once after 2 seconds
this.scheduleOnce(function() { cc.log("Hello world!"); }, 2);
```

Design action function

- Step 3: Use **cc.repeatForever()** to set the action repeated.
- Step 4: Use **scheduleOnce()** to runAction() after delayTime

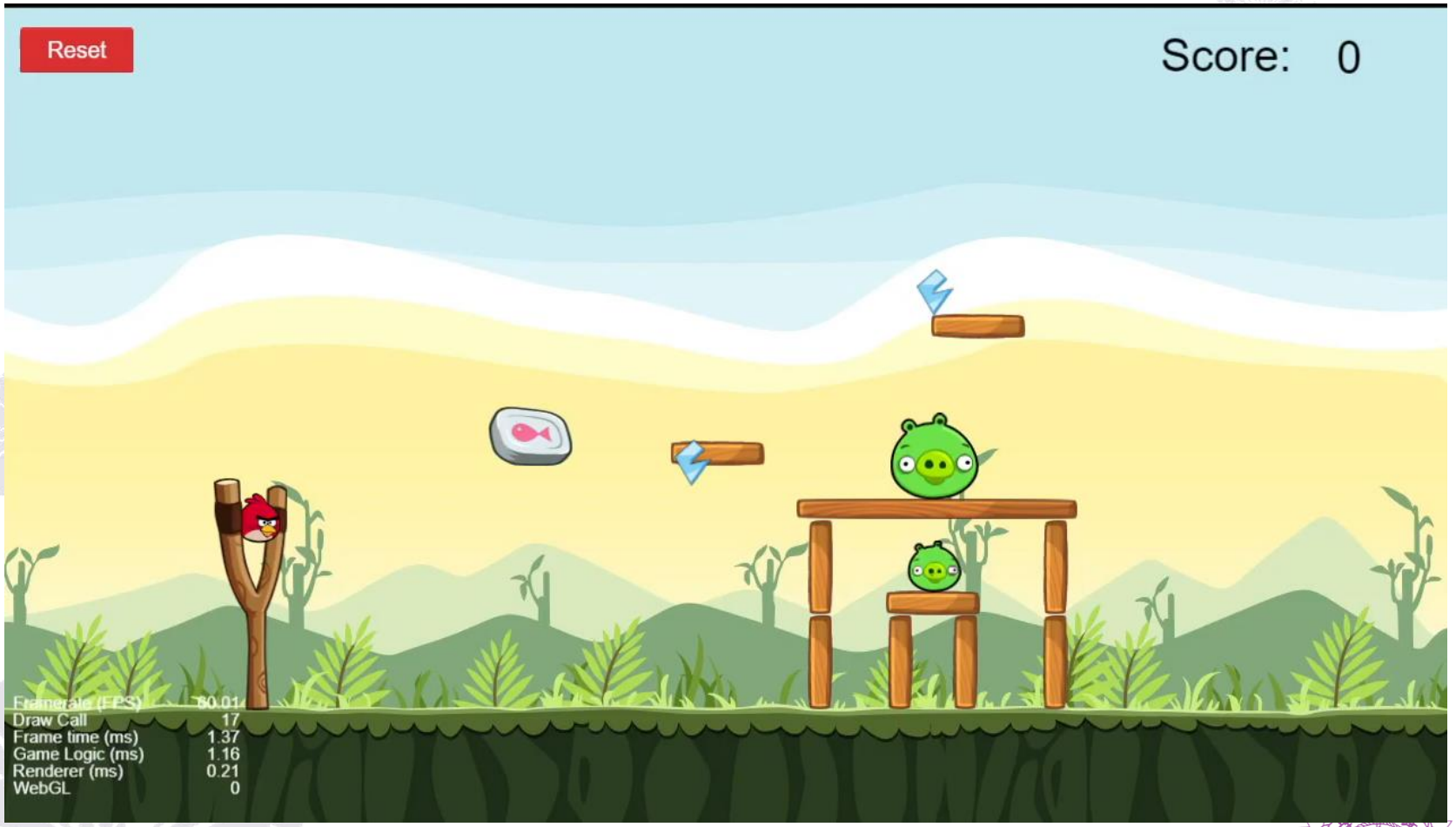
```
48          if (moveDir == "vertical") {
49              action = cc.repeatForever(sequence1);
50          }
51          else {
52              action = cc.repeatForever(sequence2);
53          }
54          this.scheduleOnce(function () {
55              platform.runAction(action);
56          }, delayTime);
57      }
58      // ==============================
```

Design action function

- Step 5: In platform.ts, create start(), call platformMove() (TODO 1.2)
  - Let platforms do the moving action after game start

```
18      // ========== TODO 1.2 ==========
19      // 1. Let platforms continue to move during the game.
20      start() {
21          this.platformMove("vertical", 1, this.Vertical_platform);
22          this.platformMove("horizontal", 1, this.Horizontal_platform);
23      }
24      // ==============================
```

Problem of child node



Handle the Spike Node

- Step 6: In start(), use spikeMove() to make the spike move (TODO 1.3.1)

```
20      start() {
21          this.platformMove("vertical", 1, this.Vertical_platform);
22          this.platformMove("horizontal", 1, this.Horizontal_platform);
23          // ========= TODO 1.3.1 =========
24          // 1. Let the vertical spike move
25          this.spikeMove(1, this.Vertical_spike);
26          // ==============================
27      }
```

  - spikeMove() in platfrom.ts, just like platformMove()

```
64      spikeMove(delayTime: number, spike: cc.Node) {
65          let action: cc.Action;
66          let easeRate: number = 2;
67          var sequence = cc.sequence(cc.moveBy(1, 40, 0).easing(cc.easeInOut(easeRate)), cc.moveBy(1, -40, 0).easing(cc.easeInOut(easeRate)));
68          action = cc.repeatForever(sequence);
69          this.scheduleOnce(function () {
70              spike.runAction(action);
71          }, delayTime);
72      }
```

Handle the Spike Node

- Step 7: Write update(), let the spike node always stay at the initial local position (TODO 1.3.2)
  - So, it will move with its parent correctly

```
32      // ========= TODO 1.3.2 =========
33      // 1. Let the horizontal spike always stay at the initial local position,
34      //    So, it will move with its parent correctly
35      update() {
36          this.Horizontal_spike.setPosition(0, 30);
37      }
38      // ==============================
```

Contact with spikes

- Step 1: In bird.ts, onBeginContact(), reset the bird to initial position when contact with spike (TODO 2.1)

```
251         // ========== TODO 2.1 ==========
252         // 1. If contact with a spike, reset the bird
253         //    to the initial position after 1 second.
254         else if (other.node.name == "Spike") {
255             console.log("hurt");
256             this.scheduleOnce(function () {
257                 this.reset();
258             }, 1);
259         }
260         // ==============================
```

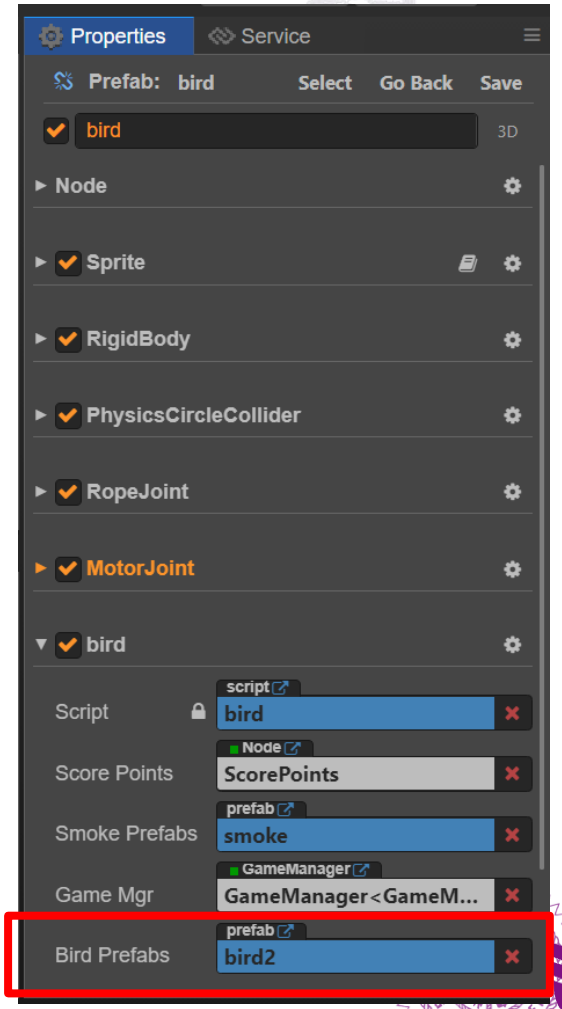

Your
turn!
C++
PASCAL

Split into two birds

- Step 1: In bird.ts, define birdPrefabs (TODO 3.1)

```
34      // ========== TODO 3.1 ==========
35      // 1. Define birdPrefabs
36      @property(cc.Prefab)
37      birdPrefabs: cc.Prefab = null;
38      // ============================
```

- Step 2: Drag bird2 into bird node's bird component as Bird Prefabs
  - assets/prefabs/bird2



Split into two birds

- Step 3: Create a function split() (TODO 3.2)
  - Instantiate another bird
  - Set the bird's position and velocity
  - Add the node under Canvas/Slingshot

```
204     split() {
205         console.log("split");
206         var bird_split = cc.instantiate(this.birdPrefabs);
207         bird_split.setPosition(this.node.position.x, this.node.position.y + 10);
208         bird_split.getComponent(cc.RigidBody).linearVelocity = cc.v2(this.rb.linearVelocity.x, this.rb.linearVelocity.y + 50);
209         cc.find("Canvas/Slingshot").addChild(bird_split);
```

Split into two birds

- Step 4: In dragEnd(), call split() after a few seconds (TODO 3.3)

```
175     dragEnd() {
176         if (!this.draggable) return;
177
178         if (this.node.position.sub(this.startPos).mag() > 10) {
179             this.draggable = false;
180         }
181
182         this.motorJoint.enabled = true;
183
184         this.rb.gravityScale = 1;
185         this.rb.linearVelocity = cc.v2(1, 0);
186
187         // ========== TODO 3.3 ==========
188         // 1. Split to two birds after 0.5 sec.
189         this.scheduleOnce(function () {
190             this.split();
191         }, 0.5);
192         // ==============================
193
194         this.GameMgr.playEffect();
195
196     }
```

Recap: updateScore()

```
updateScore(number) {
    this.score += number;
    this.scorePoints.getComponent(cc.Label).string = this.score.toString();
}
```

bird.ts

- updateScore() is defined under bird component.

Recap: updateScore()

```
onBeginContact(contact, self, other) {
    if (other.tag == 1) { // enemy tag
        console.log("BeginContact")
        console.log(contact.getWorldManifold().points);

        var smoke = cc.instantiate(this.smokePrefabs);
        smoke.setPosition(contact.getWorldManifold().points[0]);

        cc.find("Canvas/Environment").addChild(smoke);
        this.scheduleOnce(function () {
            smoke.destroy();
        }, 1.5);

        this._bird.updateScore(30);
    }
    else if (other.tag == 2) { // game item tag

        console.log("Trigger");

        other.node.destroy();
        this._bird.updateScore(10);
    }
}
```

splitBird.ts

- Want to use updateScore() in splitBird.

Use bird to update score

- Step 5: In splitBird.ts, define bird's component (TODO 3.4)
  - since we need to update the score by the original bird

```
27      // ========== TODO 3.4 ==========
28      // 1. Define bird's component.
29      @property(bird)
30      _bird: bird = null;
31      // ==============================
```

- Step 6: Get the bird's component when start (TODO 3.5)
  - onBeginContact() is like bird.ts

```
38      // ========== TODO 3.5 ==========
39      // 1. Get the bird's component from bird node when instantiated.
40      start() {
41          this._bird = cc.find("Canvas/Slingshot/bird").getComponent(bird);
42      }
43      // ==============================
```


Your
turn!
C++
PASCAL