Mobile device input

Mobile device input

- On mobile devices, the Input class offers access to touchscreen, accelerometer and geographical/location input.
- Access to keyboard on mobile devices is provided via the iOS keyboard.

Multi-touch screen

- The iPhone, iPad and iPod Touch devices are capable of tracking up to five fingers touching the screen simultaneously.
- You can retrieve the status of each finger touching the screen during the last frame by accessing the Input.touches property array.
- Android devices don't have a unified limit on how many fingers they track. Instead, it varies from device to device and can be anything from two-touch on older devices to five fingers on some newer devices.
- Each finger touch is represented by an Input.Touch data structure:

Property:	Description:
fingerId	The unique index for a touch.
position	The screen position of the touch.
deltaPosition	The screen position change since the last frame.
deltaTime	Amount of time that has passed since the last state change.
tapCount	The iPhone/iPad screen is able to distinguish quick finger taps by the user. This counter will let you know how many times the user has tapped the screen without moving a finger to the sides. Android devices do not count number of taps, this field is always 1.
phase	Describes the state of the touch, which can help you determine whether the user has just started to touch screen, just moved their finger or just lifted their finger.
Began	A finger just touched the screen.
Moved	A finger moved on the screen.
Stationary	A finger is touching the screen but hasn't moved since the last frame.
Ended	A finger was lifted from the screen. This is the final phase of a touch.
Canceled	The system cancelled tracking for the touch, as when (for example) the user puts the device to their face or more than five touches happened simultaneously. This is the final phase of a touch.

Here's an example script that shoots a ray whenever the user taps on the screen:

```
using UnityEngine;

public class TouchInput : MonoBehaviour
{
    GameObject particle;

    void Update()
    {
        foreach(Touch touch in Input.touches)
        {
            if (touch.phase == TouchPhase.Began)
            {
                // Construct a ray from the current touch coordinates
                Ray ray = Camera.main.ScreenPointToRay(touch.position);
                if (Physics.Raycast(ray))
                {
                    // Create a particle if hit
                    Instantiate(particle, transform.position, transform.rotation);
                }
            }
        }
    }
}
```

Mouse simulation

- On top of native touch support Unity **iOS** /Android provides a mouse simulation.
- You can use mouse functionality from the standard Input class. Note that iOS/Android devices are designed to support multiple finger touch.
- Using the mouse functionality will support just a single finger touch. Also, finger touch on mobile devices can move from one area to another with no movement between them.
- Mouse simulation on mobile devices will provide movement, so is very different compared to touch input.
- The recommendation is to use the mouse simulation during early development but to use touch input as soon as possible.

Accelerometer

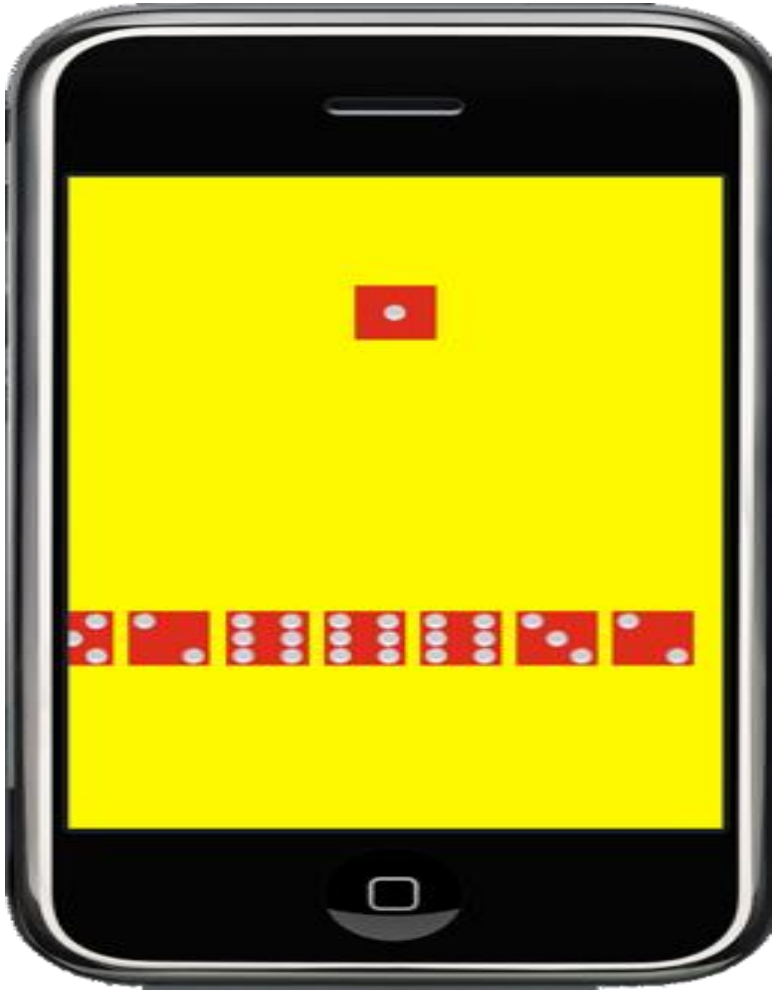
- As the mobile device moves, a built-in accelerometer reports linear acceleration changes along the three primary axes in three-dimensional space.
- Acceleration along each axis is reported directly by the hardware as G-force values.
- A value of 1.0 represents a load of about +1g along a given axis while a value of –1.0 represents –1g. If you hold the device upright (with the home button at the bottom) in front of you, the X axis is positive along the right, the Y axis is positive directly up, and the Z axis is positive pointing toward you.
- You can retrieve the accelerometer value by accessing the Input.acceleration property.

The following is an example script which will move an object using the accelerometer:

```
using UnityEngine;

public class Accelerometer : MonoBehaviour
{
     float speed = 10.0f;

     void Update()
     {
          Vector3 dir = Vector3.zero;
          // we assume that the device is held parallel to the ground
          // and the Home button is in the right hand

          // remap the device acceleration axis to game coordinates:
          // 1) XY plane of the device is mapped onto XZ plane
          // 2) rotated 90 degrees around Y axis

          dir.x = -Input.acceleration.y;
          dir.z = Input.acceleration.x;

          // clamp acceleration vector to the unit sphere
          if (dir.sqrMagnitude > 1)
               dir.Normalize();

          // Make it move 10 meters per second instead of 10 meters per frame...
          dir *= Time.deltaTime;

          // Move object
          transform.Translate(dir * speed);
     }
}
```

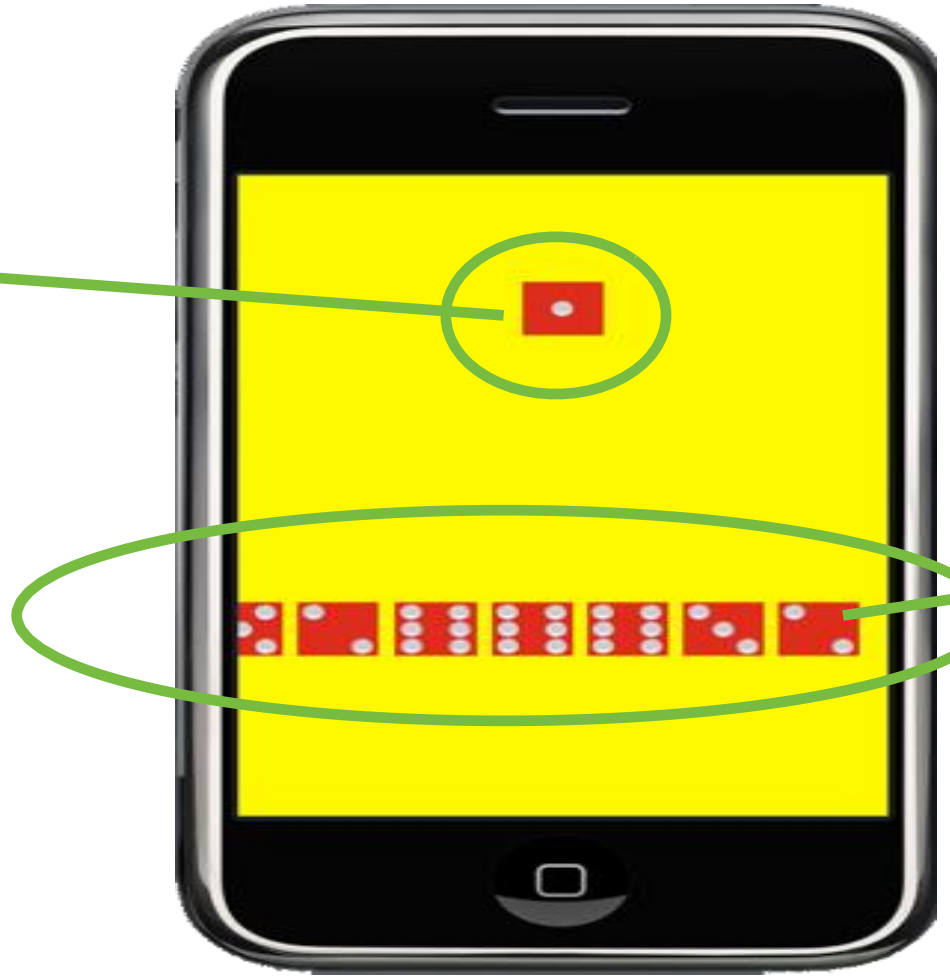
Block Game

- Teach you about:
  - Sprites
  - Actions
  - Collision detection



Game Elements

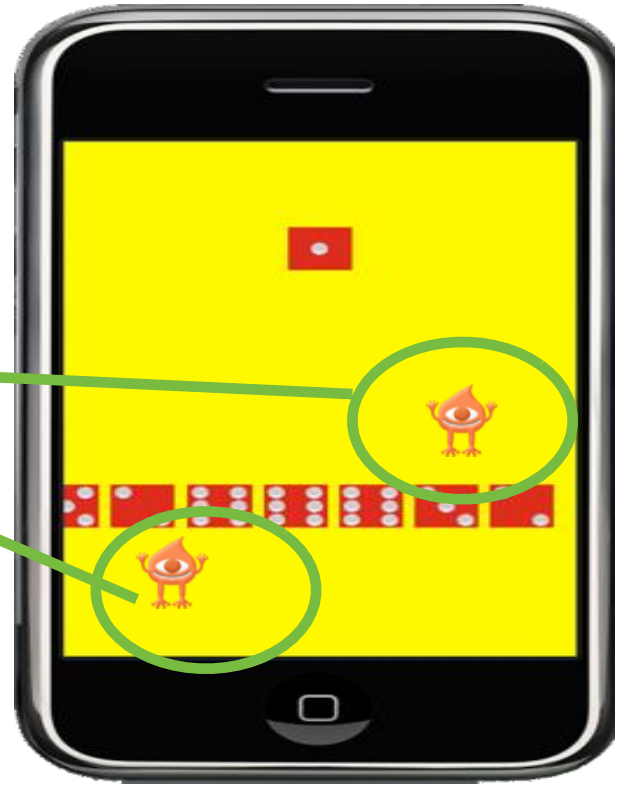
Piece user drags to remove...



...one of the boxes moving across the bottom of the screen...

Game Elements

...while avoiding the blockers who are protecting the boxes



Additional Notes

- Single-color background

I chose yellow, but you could pick another color

- You can use either color squares or dice images for UserSprite/boxes

Both are included in basic sample project Using Cocos2d version 0.99.5 beta 3

Game Items

- The piece the user moves around is a sprite (UserSprite)
- The boxes are sprites (BoxSprite)
- Blockers that move back and forth above and below the conveyor belt

GameActions

- Touch and move UserSprite
- Boxes just move right-to-left
- New boxes are periodically added on right
- Blockers move back and forth
- User Sprite can collide with boxes and blockers

Recommended Books

Text books

- Learn Cocos2d: Game Development for iOS. Steffen Itterheim, Apress, 2012. ISBN-10: 143024416X
- Cocos2d for iPhone Game Development Cookbook. Nathan Burba, Packt Publishing, 2011. ISBN-10: 1849514003
- Mike Mc Shaffrfy and David Graham, “Game Coding Complete”, Fourth Edition, Cengage Learning, PTR, 2012.
- Jason Gregory, “Game Engine Architecture”, CRC Press / A K Peters, 2009.
- David H. Eberly, “3D Game Engine Design, Second Edition: A Practical Approach to Real-Time Computer Graphics” 2 nd Editions, Morgan Kaufmann, 2006.

Reference Book

- 1. Ernest Adams and Andrew Rollings, “Fundamentals of Game Design”, 2 nd Edition Prentice Hall / New Riders, 2009.
- 2. Eric Lengyel, “Mathematics for 3D Game Programming and Computer Graphics”, 3 rd Edition, Course Technology PTR, 2011.
- 3. Jesse Schell, The Art of Game Design: A book of lenses, 1 st Edition, CRC Press, 2008.


Thank You