

# Building an Intelligent Flappy Bird

MADLab



Mobile Application  
Development Lab

# Agenda

- Intro to mobile development - iOS
- Building the Classic Flappy Bird Game
- Break
- Teaching the Bird How to Fly



# History of the Mobile Development





# What is mobile development?

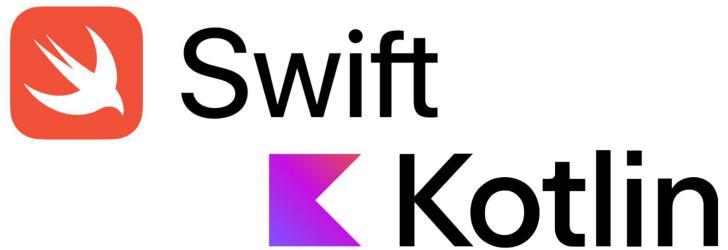
- Way to build applications “Apps” for mobile phones
- iOS: Apple phone devices
- Android: Everyone else (other players embargoed from North America i.e HarmonyOS)



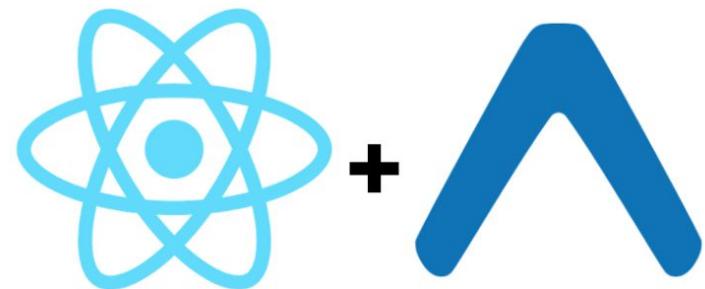
# iOS Game Dev

# Types of mobile development

Native development (Swift, Kotlin, Java)

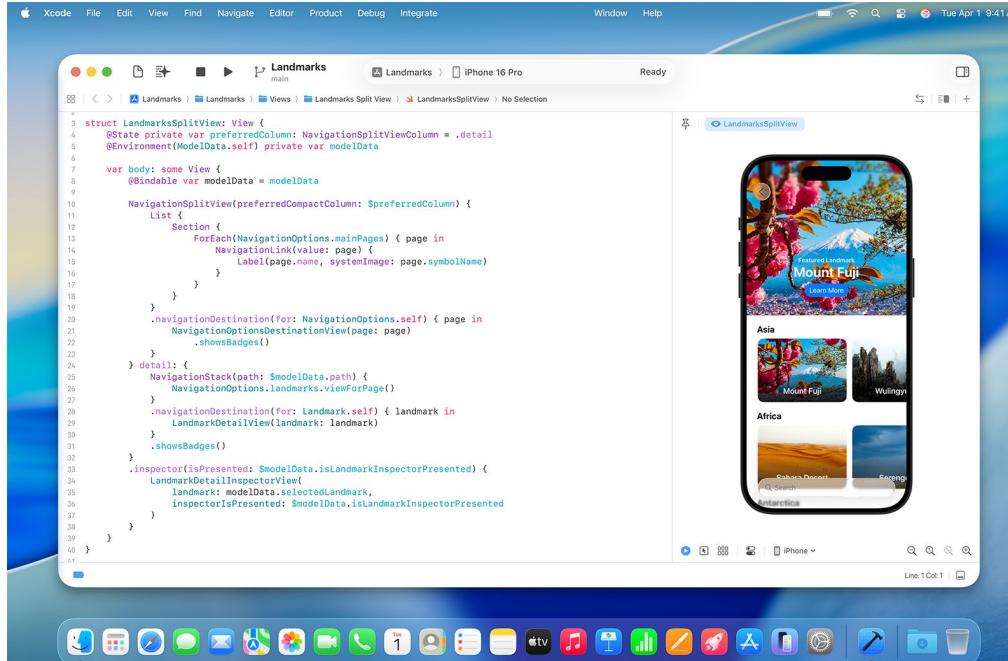


Cross platform development (Flutter, React Native)



# iOS Development

Uses Swift programming language since 2014 (prev. Objective-C)



A screenshot of the Xcode IDE. The main window shows a Swift file named "Landmarks.swift" with the following code:

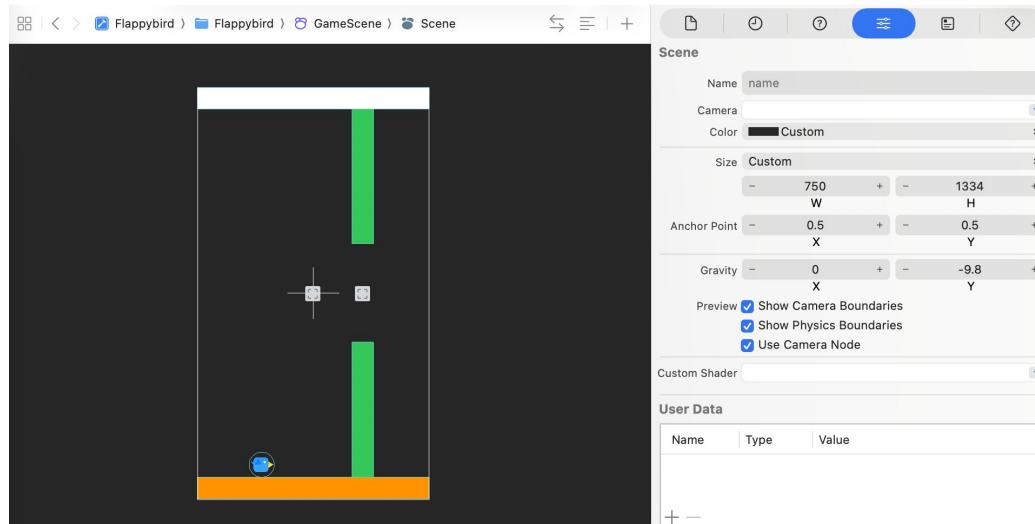
```
3 struct LandmarksSplitView: View {
4     @State private var preferredColumn: NavigationSplitViewColumn = .detail
5     @Environment(ModelData.self) private var modelData
6
7     var body: some View {
8         @Bindable var modelData = modelData
9
10        NavigationSplitView(preferredCompactColumn: $preferredColumn) {
11            List {
12                Section {
13                    ForEach(NavigationOptions.mainPages) { page in
14                        NavigationLink(value: page) {
15                            Label(page.name, systemImage: page.symbolName)
16                        }
17                    }
18                }
19            }.navigationDestination(for: NavigationOptions.self) { page in
20                NavigationOptionsDestinationView(page: page)
21            }.showsBadges()
22        }
23    } detail: {
24        NavigationStack(path: $modelData.path) {
25            NavigationOptions.landmarks.viewForPage()
26        }
27        .navigationDestination(for: Landmark.self) { landmark in
28            LandmarkDetailView(landmark: landmark)
29        }
30        .showsBadges()
31    }
32    .inspector(isPresented: $modelData.isLandmarkInspectorPresented) {
33        LandmarkDetailInspectorView(
34            landmark: modelData.selectedLandmark,
35            inspectorIsPresented: $modelData.isLandmarkInspectorPresented
36        )
37    }
38 }
39 }
40 }
```

The right side of the interface shows a preview of the application running on an iPhone 16 Pro. The app displays a travel-themed interface with sections for Asia, Africa, and Australia, featuring images of Mount Fuji and other landmarks.

# iOS Game Dev

# SpriteKit

- Apple's built-in framework for creating 2D games



# Flappy Bird



Dong Nguyen  
@dongatory



Following

I am sorry 'Flappy Bird' users, 22 hours from now, I will take 'Flappy Bird' down. I cannot take this anymore.



Reply



Retweet



Favorite



Buffer



More



HootSuite

RETWEET FAVORITE  
51,699 14,501



11:02 AM - 8 Feb 2014

1: Make Flappy Bird

# Clone the starter project

1. Open Xcode
2. Click “Clone a repository”
3. Paste the GitHub URL
4. Choose where to save the project
5. Select “Develop” branch
6. Wait for Xcode to open the project

<https://github.com/cyn900/FlappyBird>

# How SpriteKit works

- .sks (SpriteKit Scene) file defines **what exists**
- Swift code defines **what happens**



```
26 // MARK: - Game Scene
27 // This is the main game loop controller.
28 // It owns the world, spawns birds and pipes, handles input, collisions, and scoring.
29 final class GameScene: SKScene, SKPhysicsContactDelegate {
```

...

# Adjustable value

```
50 // MARK: Birds
51 private var birds = [SKSpriteNode]()
52 // ADJUSTABLE: Number of birds to spawn
53 private let birdCount = Int(100)
54
55 // MARK: Pipes
56 private var pipes = [Pipe]()
57
58 // Gameplay-controlled pipe gap.
59 private var pipeGap = CGFloat(130)    // Vertical opening between pipes, overwritten from SKS in didMove()
60 // ADJUSTABLE: Leftward (moving to the left) movement speed
61 private let pipeSpeed = CGFloat(-2.5)  // Larger absolute value = faster movement.
62 // ADJUSTABLE: Distance between pipe spawns
63 private let spawnDist = CGFloat(300)
64
65 private var pipeSpawnProgress = CGFloat(0)
66
67 // MARK: Physics tuning
68
69 // ADJUSTABLE: Downward gravity applied to birds.
70 private let gravityY = CGFloat(-12)
71
72 // ADJUSTABLE: Upward velocity applied when a bird flaps.
73 private var flapVelocityTarget = CGFloat(320)    // Computed again in didMove() to match world scale.
74
```

# Adjustable value

```
50 // MARK: Birds
51 private var birds = [SKSpriteNode]()
52 // ADJUSTABLE: Number of birds to spawn
53 private let birdCount = Int(100)
54
55 // MARK: Pipes
56 private var pipes = [Pipe]()
57
58 // Gameplay-controlled pipe gap.
59 private var pipeGap = CGFloat(130)    // Vertical opening between pipes, overwritten from SKS in didMove()
60 // ADJUSTABLE: Leftward (moving to the left) movement speed
61 private let pipeSpeed = CGFloat(-2.5)   // Larger absolute value = faster movement.
62 // ADJUSTABLE: Distance between pipe spawns
63 private let spawnDist = CGFloat(300)
64
65 private var pipeSpawnProgress = CGFloat(0)
66
67 // MARK: Physics tuning
68
69 // ADJUSTABLE: Downward gravity applied to birds.
70 private let gravityY = CGFloat(-12)
71
72 // ADJUSTABLE: Upward velocity applied when a bird flaps.
73 private var flapVelocityTarget = CGFloat(320)  // Computed again in didMove() to match world scale.
74
```

# Adjustable value

```
50 // MARK: Birds
51 private var birds = [SKSpriteNode]()
52 // ADJUSTABLE: Number of birds to spawn
53 private let birdCount = Int(100)
54
55 // MARK: Pipes
56 private var pipes = [Pipe]()
57
58 // Gameplay-controlled pipe gap.
59 private var pipeGap = CGFloat(130)    // Vertical opening between pipes, overwritten from SKS in didMove()
60 // ADJUSTABLE: Leftward (moving to the left) movement speed
61 private let pipeSpeed = CGFloat(-2.5)   // Larger absolute value = faster movement.
62 // ADJUSTABLE: Distance between pipe spawns
63 private let spawnDist = CGFloat(300) // ADJUSTABLE: Distance between pipe spawns
64
65 private var pipeSpawnProgress = CGFloat(0)
66
67 // MARK: Physics tuning
68
69 // ADJUSTABLE: Downward gravity applied to birds.
70 private let gravityY = CGFloat(-12)
71
72 // ADJUSTABLE: Upward velocity applied when a bird flaps.
73 private var flapVelocityTarget = CGFloat(320)  // Computed again in didMove() to match world scale.
74
```

# Adjustable value

```
50 // MARK: Birds
51 private var birds = [SKSpriteNode]()
52 // ADJUSTABLE: Number of birds to spawn
53 private let birdCount = Int(100)
54
55 // MARK: Pipes
56 private var pipes = [Pipe]()
57
58 // Gameplay-controlled pipe gap.
59 private var pipeGap = CGFloat(130)    // Vertical opening between pipes, overwritten from SKS in didMove()
60 // ADJUSTABLE: Leftward (moving to the left) movement speed
61 private let pipeSpeed = CGFloat(-2.5)   // Larger absolute value = faster movement.
62 // ADJUSTABLE: Distance between pipe spawns
63 private let spawnDist = CGFloat(300)
64
65 private var pipeSpawnProgress = CGFloat(0)
66
67 // MARK: Physics tuning
68
69 // ADJUSTABLE: Downward gravity applied to birds.
70 private let gravityY = CGFloat(-12)
71
72 // ADJUSTABLE: Upward velocity applied when a bird flaps.
73 private var flapVelocityTarget = CGFloat(320)    // Computed again in didMove() to match world scale.
74
```

# Adjustable value

```
50 // MARK: Birds
51 private var birds = [SKSpriteNode]()
52 // ADJUSTABLE: Number of birds to spawn
53 private let birdCount = Int(100)
54
55 // MARK: Pipes
56 private var pipes = [Pipe]()
57
58 // Gameplay-controlled pipe gap.
59 private var pipeGap = CGFloat(130)    // Vertical opening between pipes, overwritten from SKS in didMove()
60 // ADJUSTABLE: Leftward (moving to the left) movement speed
61 private let pipeSpeed = CGFloat(-2.5)  // Larger absolute value = faster movement.
62 // ADJUSTABLE: Distance between pipe spawns
63 private let spawnDist = CGFloat(300)
64
65 private var pipeSpawnProgress = CGFloat(0)
66
67 // MARK: Physics tuning
68
69 // ADJUSTABLE: Downward gravity applied to birds.
70 private let gravityY = CGFloat(-12)
71
72 // ADJUSTABLE: Upward velocity applied when a bird flaps.
73 private var flapVelocityTarget = CGFloat(320)    // Computed again in didMove() to match world scale.
74
```

# How SpriteKit works

```
89     override func didMove(to view: SKView) {
90         guard !didInit else { return }
91         didInit = true
92
93         physicsWorld.gravity = CGVector(dx: 0, dy: gravityY)
94         physicsWorld.contactDelegate = self
95
96         // Prefer // paths so SKS hierarchy changes don't break lookups
97         guard let w = childNode(withName: "World") else {
98             fatalError("Missing node named World")
99         }
100        world = w
101
102        guard let bp = w.childNode(withName: "birdPrototype") as? SKSpriteNode else {
103            fatalError("Missing SKSpriteNode named birdPrototype")
104        }
105        birdPrototype = bp
106        birdPrototype.removeFromParent()

    ...
```

when the scene is first presented, we set up the scene

# How SpriteKit works

```
89     override func didMove(to view: SKView) {
90         guard !didInit else { return }
91         didInit = true
92
93         physicsWorld.gravity = CGVector(dx: 0, dy: gravityY)
94         physicsWorld.contactDelegate = self
95
96         // Prefer // paths so SKS hierarchy changes don't break lookups
97         guard let w = childNode(withName: "World") else {
98             fatalError("Missing node named World")
99         }
100        world = w
101
102        guard let bp = w.childNode(withName: "birdPrototype") as? SKSpriteNode else {
103            fatalError("Missing SKSpriteNode named birdPrototype")
104        }
105        birdPrototype = bp
106        birdPrototype.removeFromParent()
107
108        ...
109    }
110
111    func setupGame() {
112        ...
113    }
114
115    func update(_ currentTime: TimeInterval) {
116        ...
117    }
118
119    func didBegin(_ contact: SKPhysicsContact) {
120        ...
121    }
122
123    func didEnd(_ contact: SKPhysicsContact) {
124        ...
125    }
126
127    func didMove(to view: SKView) {
128        ...
129    }
130
131    func didUpdate(to view: SKView) {
132        ...
133    }
134
135    func didSimulate(_ simulation: SKSimulation) {
136        ...
137    }
138
139    func didUpdate(_ simulation: SKSimulation) {
140        ...
141    }
142
143    func didMove(to view: SKView) {
144        ...
145    }
146
147    func didUpdate(_ simulation: SKSimulation) {
148        ...
149    }
150
151    func didSimulate(_ simulation: SKSimulation) {
152        ...
153    }
154
155    func didMove(to view: SKView) {
156        ...
157    }
158
159    func didUpdate(_ simulation: SKSimulation) {
160        ...
161    }
162
163    func didSimulate(_ simulation: SKSimulation) {
164        ...
165    }
166
167    func didMove(to view: SKView) {
168        ...
169    }
170
171    func didUpdate(_ simulation: SKSimulation) {
172        ...
173    }
174
175    func didSimulate(_ simulation: SKSimulation) {
176        ...
177    }
178
179    func didMove(to view: SKView) {
180        ...
181    }
182
183    func didUpdate(_ simulation: SKSimulation) {
184        ...
185    }
186
187    func didSimulate(_ simulation: SKSimulation) {
188        ...
189    }
190
191    func didMove(to view: SKView) {
192        ...
193    }
194
195    func didUpdate(_ simulation: SKSimulation) {
196        ...
197    }
198
199    func didSimulate(_ simulation: SKSimulation) {
200        ...
201    }
202
203    func didMove(to view: SKView) {
204        ...
205    }
206
207    func didUpdate(_ simulation: SKSimulation) {
208        ...
209    }
210
211    func didSimulate(_ simulation: SKSimulation) {
212        ...
213    }
214
215    func didMove(to view: SKView) {
216        ...
217    }
218
219    func didUpdate(_ simulation: SKSimulation) {
220        ...
221    }
222
223    func didSimulate(_ simulation: SKSimulation) {
224        ...
225    }
226
227    func didMove(to view: SKView) {
228        ...
229    }
230
231    func didUpdate(_ simulation: SKSimulation) {
232        ...
233    }
234
235    func didSimulate(_ simulation: SKSimulation) {
236        ...
237    }
238
239    func didMove(to view: SKView) {
240        ...
241    }
242
243    func didUpdate(_ simulation: SKSimulation) {
244        ...
245    }
246
247    func didSimulate(_ simulation: SKSimulation) {
248        ...
249    }
250
251    func didMove(to view: SKView) {
252        ...
253    }
254
255    func didUpdate(_ simulation: SKSimulation) {
256        ...
257    }
258
259    func didSimulate(_ simulation: SKSimulation) {
260        ...
261    }
262
263    func didMove(to view: SKView) {
264        ...
265    }
266
267    func didUpdate(_ simulation: SKSimulation) {
268        ...
269    }
270
271    func didSimulate(_ simulation: SKSimulation) {
272        ...
273    }
274
275    func didMove(to view: SKView) {
276        ...
277    }
278
279    func didUpdate(_ simulation: SKSimulation) {
280        ...
281    }
282
283    func didSimulate(_ simulation: SKSimulation) {
284        ...
285    }
286
287    func didMove(to view: SKView) {
288        ...
289    }
290
291    func didUpdate(_ simulation: SKSimulation) {
292        ...
293    }
294
295    func didSimulate(_ simulation: SKSimulation) {
296        ...
297    }
298
299    func didMove(to view: SKView) {
300        ...
301    }
302
303    func didUpdate(_ simulation: SKSimulation) {
304        ...
305    }
306
307    func didSimulate(_ simulation: SKSimulation) {
308        ...
309    }
310
311    func didMove(to view: SKView) {
312        ...
313    }
314
315    func didUpdate(_ simulation: SKSimulation) {
316        ...
317    }
318
319    func didSimulate(_ simulation: SKSimulation) {
320        ...
321    }
322
323    func didMove(to view: SKView) {
324        ...
325    }
326
327    func didUpdate(_ simulation: SKSimulation) {
328        ...
329    }
330
331    func didSimulate(_ simulation: SKSimulation) {
332        ...
333    }
334
335    func didMove(to view: SKView) {
336        ...
337    }
338
339    func didUpdate(_ simulation: SKSimulation) {
340        ...
341    }
342
343    func didSimulate(_ simulation: SKSimulation) {
344        ...
345    }
346
347    func didMove(to view: SKView) {
348        ...
349    }
350
351    func didUpdate(_ simulation: SKSimulation) {
352        ...
353    }
354
355    func didSimulate(_ simulation: SKSimulation) {
356        ...
357    }
358
359    func didMove(to view: SKView) {
360        ...
361    }
362
363    func didUpdate(_ simulation: SKSimulation) {
364        ...
365    }
366
367    func didSimulate(_ simulation: SKSimulation) {
368        ...
369    }
370
371    func didMove(to view: SKView) {
372        ...
373    }
374
375    func didUpdate(_ simulation: SKSimulation) {
376        ...
377    }
378
379    func didSimulate(_ simulation: SKSimulation) {
380        ...
381    }
382
383    func didMove(to view: SKView) {
384        ...
385    }
386
387    func didUpdate(_ simulation: SKSimulation) {
388        ...
389    }
390
391    func didSimulate(_ simulation: SKSimulation) {
392        ...
393    }
394
395    func didMove(to view: SKView) {
396        ...
397    }
398
399    func didUpdate(_ simulation: SKSimulation) {
400        ...
401    }
402
403    func didSimulate(_ simulation: SKSimulation) {
404        ...
405    }
406
407    func didMove(to view: SKView) {
408        ...
409    }
410
411    func didUpdate(_ simulation: SKSimulation) {
412        ...
413    }
414
415    func didSimulate(_ simulation: SKSimulation) {
416        ...
417    }
418
419    func didMove(to view: SKView) {
420        ...
421    }
422
423    func didUpdate(_ simulation: SKSimulation) {
424        ...
425    }
426
427    func didSimulate(_ simulation: SKSimulation) {
428        ...
429    }
430
431    func didMove(to view: SKView) {
432        ...
433    }
434
435    func didUpdate(_ simulation: SKSimulation) {
436        ...
437    }
438
439    func didSimulate(_ simulation: SKSimulation) {
440        ...
441    }
442
443    func didMove(to view: SKView) {
444        ...
445    }
446
447    func didUpdate(_ simulation: SKSimulation) {
448        ...
449    }
450
451    func didSimulate(_ simulation: SKSimulation) {
452        ...
453    }
454
455    func didMove(to view: SKView) {
456        ...
457    }
458
459    func didUpdate(_ simulation: SKSimulation) {
460        ...
461    }
462
463    func didSimulate(_ simulation: SKSimulation) {
464        ...
465    }
466
467    func didMove(to view: SKView) {
468        ...
469    }
470
471    func didUpdate(_ simulation: SKSimulation) {
472        ...
473    }
474
475    func didSimulate(_ simulation: SKSimulation) {
476        ...
477    }
478
479    func didMove(to view: SKView) {
480        ...
481    }
482
483    func didUpdate(_ simulation: SKSimulation) {
484        ...
485    }
486
487    func didSimulate(_ simulation: SKSimulation) {
488        ...
489    }
490
491    func didMove(to view: SKView) {
492        ...
493    }
494
495    func didUpdate(_ simulation: SKSimulation) {
496        ...
497    }
498
499    func didSimulate(_ simulation: SKSimulation) {
500        ...
501    }
502
503    func didMove(to view: SKView) {
504        ...
505    }
506
507    func didUpdate(_ simulation: SKSimulation) {
508        ...
509    }
510
511    func didSimulate(_ simulation: SKSimulation) {
512        ...
513    }
514
515    func didMove(to view: SKView) {
516        ...
517    }
518
519    func didUpdate(_ simulation: SKSimulation) {
520        ...
521    }
522
523    func didSimulate(_ simulation: SKSimulation) {
524        ...
525    }
526
527    func didMove(to view: SKView) {
528        ...
529    }
530
531    func didUpdate(_ simulation: SKSimulation) {
532        ...
533    }
534
535    func didSimulate(_ simulation: SKSimulation) {
536        ...
537    }
538
539    func didMove(to view: SKView) {
540        ...
541    }
542
543    func didUpdate(_ simulation: SKSimulation) {
544        ...
545    }
546
547    func didSimulate(_ simulation: SKSimulation) {
548        ...
549    }
550
551    func didMove(to view: SKView) {
552        ...
553    }
554
555    func didUpdate(_ simulation: SKSimulation) {
556        ...
557    }
558
559    func didSimulate(_ simulation: SKSimulation) {
560        ...
561    }
562
563    func didMove(to view: SKView) {
564        ...
565    }
566
567    func didUpdate(_ simulation: SKSimulation) {
568        ...
569    }
570
571    func didSimulate(_ simulation: SKSimulation) {
572        ...
573    }
574
575    func didMove(to view: SKView) {
576        ...
577    }
578
579    func didUpdate(_ simulation: SKSimulation) {
580        ...
581    }
582
583    func didSimulate(_ simulation: SKSimulation) {
584        ...
585    }
586
587    func didMove(to view: SKView) {
588        ...
589    }
590
591    func didUpdate(_ simulation: SKSimulation) {
592        ...
593    }
594
595    func didSimulate(_ simulation: SKSimulation) {
596        ...
597    }
598
599    func didMove(to view: SKView) {
600        ...
601    }
602
603    func didUpdate(_ simulation: SKSimulation) {
604        ...
605    }
606
607    func didSimulate(_ simulation: SKSimulation) {
608        ...
609    }
610
611    func didMove(to view: SKView) {
612        ...
613    }
614
615    func didUpdate(_ simulation: SKSimulation) {
616        ...
617    }
618
619    func didSimulate(_ simulation: SKSimulation) {
620        ...
621    }
622
623    func didMove(to view: SKView) {
624        ...
625    }
626
627    func didUpdate(_ simulation: SKSimulation) {
628        ...
629    }
630
631    func didSimulate(_ simulation: SKSimulation) {
632        ...
633    }
634
635    func didMove(to view: SKView) {
636        ...
637    }
638
639    func didUpdate(_ simulation: SKSimulation) {
640        ...
641    }
642
643    func didSimulate(_ simulation: SKSimulation) {
644        ...
645    }
646
647    func didMove(to view: SKView) {
648        ...
649    }
650
651    func didUpdate(_ simulation: SKSimulation) {
652        ...
653    }
654
655    func didSimulate(_ simulation: SKSimulation) {
656        ...
657    }
658
659    func didMove(to view: SKView) {
660        ...
661    }
662
663    func didUpdate(_ simulation: SKSimulation) {
664        ...
665    }
666
667    func didSimulate(_ simulation: SKSimulation) {
668        ...
669    }
670
671    func didMove(to view: SKView) {
672        ...
673    }
674
675    func didUpdate(_ simulation: SKSimulation) {
676        ...
677    }
678
679    func didSimulate(_ simulation: SKSimulation) {
680        ...
681    }
682
683    func didMove(to view: SKView) {
684        ...
685    }
686
687    func didUpdate(_ simulation: SKSimulation) {
688        ...
689    }
690
691    func didSimulate(_ simulation: SKSimulation) {
692        ...
693    }
694
695    func didMove(to view: SKView) {
696        ...
697    }
698
699    func didUpdate(_ simulation: SKSimulation) {
700        ...
701    }
702
703    func didSimulate(_ simulation: SKSimulation) {
704        ...
705    }
706
707    func didMove(to view: SKView) {
708        ...
709    }
710
711    func didUpdate(_ simulation: SKSimulation) {
712        ...
713    }
714
715    func didSimulate(_ simulation: SKSimulation) {
716        ...
717    }
718
719    func didMove(to view: SKView) {
720        ...
721    }
722
723    func didUpdate(_ simulation: SKSimulation) {
724        ...
725    }
726
727    func didSimulate(_ simulation: SKSimulation) {
728        ...
729    }
730
731    func didMove(to view: SKView) {
732        ...
733    }
734
735    func didUpdate(_ simulation: SKSimulation) {
736        ...
737    }
738
739    func didSimulate(_ simulation: SKSimulation) {
740        ...
741    }
742
743    func didMove(to view: SKView) {
744        ...
745    }
746
747    func didUpdate(_ simulation: SKSimulation) {
748        ...
749    }
750
751    func didSimulate(_ simulation: SKSimulation) {
752        ...
753    }
754
755    func didMove(to view: SKView) {
756        ...
757    }
758
759    func didUpdate(_ simulation: SKSimulation) {
760        ...
761    }
762
763    func didSimulate(_ simulation: SKSimulation) {
764        ...
765    }
766
767    func didMove(to view: SKView) {
768        ...
769    }
770
771    func didUpdate(_ simulation: SKSimulation) {
772        ...
773    }
774
775    func didSimulate(_ simulation: SKSimulation) {
776        ...
777    }
778
779    func didMove(to view: SKView) {
780        ...
781    }
782
783    func didUpdate(_ simulation: SKSimulation) {
784        ...
785    }
786
787    func didSimulate(_ simulation: SKSimulation) {
788        ...
789    }
790
791    func didMove(to view: SKView) {
792        ...
793    }
794
795    func didUpdate(_ simulation: SKSimulation) {
796        ...
797    }
798
799    func didSimulate(_ simulation: SKSimulation) {
800        ...
801    }
802
803    func didMove(to view: SKView) {
804        ...
805    }
806
807    func didUpdate(_ simulation: SKSimulation) {
808        ...
809    }
810
811    func didSimulate(_ simulation: SKSimulation) {
812        ...
813    }
814
815    func didMove(to view: SKView) {
816        ...
817    }
818
819    func didUpdate(_ simulation: SKSimulation) {
820        ...
821    }
822
823    func didSimulate(_ simulation: SKSimulation) {
824        ...
825    }
826
827    func didMove(to view: SKView) {
828        ...
829    }
830
831    func didUpdate(_ simulation: SKSimulation) {
832        ...
833    }
834
835    func didSimulate(_ simulation: SKSimulation) {
836        ...
837    }
838
839    func didMove(to view: SKView) {
840        ...
841    }
842
843    func didUpdate(_ simulation: SKSimulation) {
844        ...
845    }
846
847    func didSimulate(_ simulation: SKSimulation) {
848        ...
849    }
850
851    func didMove(to view: SKView) {
852        ...
853    }
854
855    func didUpdate(_ simulation: SKSimulation) {
856        ...
857    }
858
859    func didSimulate(_ simulation: SKSimulation) {
860        ...
861    }
862
863    func didMove(to view: SKView) {
864        ...
865    }
866
867    func didUpdate(_ simulation: SKSimulation) {
868        ...
869    }
870
871    func didSimulate(_ simulation: SKSimulation) {
872        ...
873    }
874
875    func didMove(to view: SKView) {
876        ...
877    }
878
879    func didUpdate(_ simulation: SKSimulation) {
880        ...
881    }
882
883    func didSimulate(_ simulation: SKSimulation) {
884        ...
885    }
886
887    func didMove(to view: SKView) {
888        ...
889    }
890
891    func didUpdate(_ simulation: SKSimulation) {
892        ...
893    }
894
895    func didSimulate(_ simulation: SKSimulation) {
896        ...
897    }
898
899    func didMove(to view: SKView) {
900        ...
901    }
902
903    func didUpdate(_ simulation: SKSimulation) {
904        ...
905    }
906
907    func didSimulate(_ simulation: SKSimulation) {
908        ...
909    }
910
911    func didMove(to view: SKView) {
912        ...
913    }
914
915    func didUpdate(_ simulation: SKSimulation) {
916        ...
917    }
918
919    func didSimulate(_ simulation: SKSimulation) {
920        ...
921    }
922
923    func didMove(to view: SKView) {
924        ...
925    }
926
927    func didUpdate(_ simulation: SKSimulation) {
928        ...
929    }
930
931    func didSimulate(_ simulation: SKSimulation) {
932        ...
933    }
934
935    func didMove(to view: SKView) {
936        ...
937    }
938
939    func didUpdate(_ simulation: SKSimulation) {
940        ...
941    }
942
943    func didSimulate(_ simulation: SKSimulation) {
944        ...
945    }
946
947    func didMove(to view: SKView) {
948        ...
949    }
950
951    func didUpdate(_ simulation: SKSimulation) {
952        ...
953    }
954
955    func didSimulate(_ simulation: SKSimulation) {
956        ...
957    }
958
959    func didMove(to view: SKView) {
960        ...
961    }
962
963    func didUpdate(_ simulation: SKSimulation) {
964        ...
965    }
966
967    func didSimulate(_ simulation: SKSimulation) {
968        ...
969    }
970
971    func didMove(to view: SKView) {
972        ...
973    }
974
975    func didUpdate(_ simulation: SKSimulation) {
976        ...
977    }
978
979    func didSimulate(_ simulation: SKSimulation) {
980        ...
981    }
982
983    func didMove(to view: SKView) {
984        ...
985    }
986
987    func didUpdate(_ simulation: SKSimulation) {
988        ...
989    }
990
991    func didSimulate(_ simulation: SKSimulation) {
992        ...
993    }
994
995    func didMove(to view: SKView) {
996        ...
997    }
998
999    func didUpdate(_ simulation: SKSimulation) {
1000       ...
1001   }
```

Create prototypes for things that are spawned multiple times or reset often.  
Don't create prototypes for things that exist once and persist.

# Update loop

## Called Every Frame

- Runs **once per rendered frame**
- Usually ~60 FPS

## What Happens Every Frame

- Spawn pipes
- Move pipes
- Remove off-screen pipes
- Update score
- And more

```
332 // MARK: Game loop
333 override func update(_ currentTime: TimeInterval) {
334
335     let dt: TimeInterval
336     if lastUpdateTime == 0 {
337         dt = TimeInterval(1.0 / 60.0)
338     } else {
339         dt = currentTime - lastUpdateTime
340     }
341     lastUpdateTime = currentTime
342
343     // Auto-restart after game over
344     if gameOver {
345         if let t = restartAt, currentTime >= t {
346             resetGame()
347         }
348         return
349     }
350
351     // Move pipes left
352     for p in pipes { p.move(pipeSpeed) }
```

# Pipes

## One Pipe = Two Sprites

- Top pipe
- Bottom pipe
- Gap in between

## What Happens When a Pipe Spawns

- Choose a **random gap position**
- Resize top & bottom pipes

```
254     private func spawnPipe() {
255         let worldMinY = groundNode.frame.maxY
256         let worldMaxY = ceilingNode.frame.minY
257
258         let minGapCenter = worldMinY + pipeGap * 0.5
259         let maxGapCenter = worldMaxY - pipeGap * 0.5
260
261         let gapYWorld: CGFloat
262         if minGapCenter < maxGapCenter {
263             gapYWorld = CGFloat.random(in: minGapCenter...maxGapCenter)
264         } else {
265             gapYWorld = (worldMinY + worldMaxY) * 0.5
266         }
267
268         let xWorld = sceneXToWorld(frame maxX + 60)
269
270         let pipe = Pipe(
271             template: pipePrototype,
272             xPosInWorld: xWorld,
273             gapYInWorld: gapYWorld,
274             gap: pipeGap,
275             worldMinY: worldMinY,
276             worldMaxY: worldMaxY
277         )
278
279         pipes.append(pipe)
280         world.addChild(pipe.node)
281     }
```

# Game Play

## The Bird Is Physics-Driven

- Gravity is **always on**
- No manual position updates
- Physics engine moves the bird

```
284     override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {  
285         flapAll()  
286     }  
287  
288     private func flapAll() {  
289         for b in birds {  
290             guard let body = b.physicsBody, body.isDynamic else { continue }  
291             body.velocity = CGVector(dx: body.velocity.dx, dy: flapVelocityTarget)  
292         }  
293     }  
294 }
```

## Flapping = Reset Vertical Velocity

- Tap does **not** push the bird
- Tap does **not** animate the bird
- Tap **sets Y velocity**

# Collision

- Every physics body has a **category**

```
19 enum PhysicsCategory {  
20     static let bird: UInt32    = 1 << 0 // 0001  
21     static let pipe: UInt32   = 1 << 1 // 0010  
22     static let ground: UInt32 = 1 << 2 // 0100  
23     static let ceiling: UInt32 = 1 << 3 // 1000  
24 }
```

## Three Important Masks

- categoryBitMask → *what am I?*
- collisionBitMask → *what stops me?* → physically stops movement
- contactTestBitMask → *what do I get notified about?* → trigger game over

```
body.categoryBitMask = PhysicsCategory.bird  
body.contactTestBitMask = PhysicsCategory.pipe | PhysicsCategory.ground |  
    PhysicsCategory.ceiling  
body.collisionBitMask = PhysicsCategory.pipe | PhysicsCategory.ground |  
    PhysicsCategory.ceiling
```

# Collision = Death

## When a Collision Happens

- SpriteKit notifies us automatically
- Called when **two physics bodies touch**

```
// MARK: Collision
func didBegin(_ contact: SKPhysicsContact) {
    let birdBody: SKPhysicsBody?
    if contact.bodyA.categoryBitMask == PhysicsCategory.bird { birdBody =
        contact.bodyA }
    else if contact.bodyB.categoryBitMask == PhysicsCategory.bird { birdBody =
        contact.bodyB }
    else { birdBody = nil }

    guard let bBody = birdBody, let node = bBody.node as? SKSpriteNode else {
        return }
    killBird(node)
}
```

# Game Restart

## What “Death” Means in This Game

- Physics is disabled
- Bird is hidden

## Game Over Condition

- Check if **all birds are inactive**
- Trigger game over once

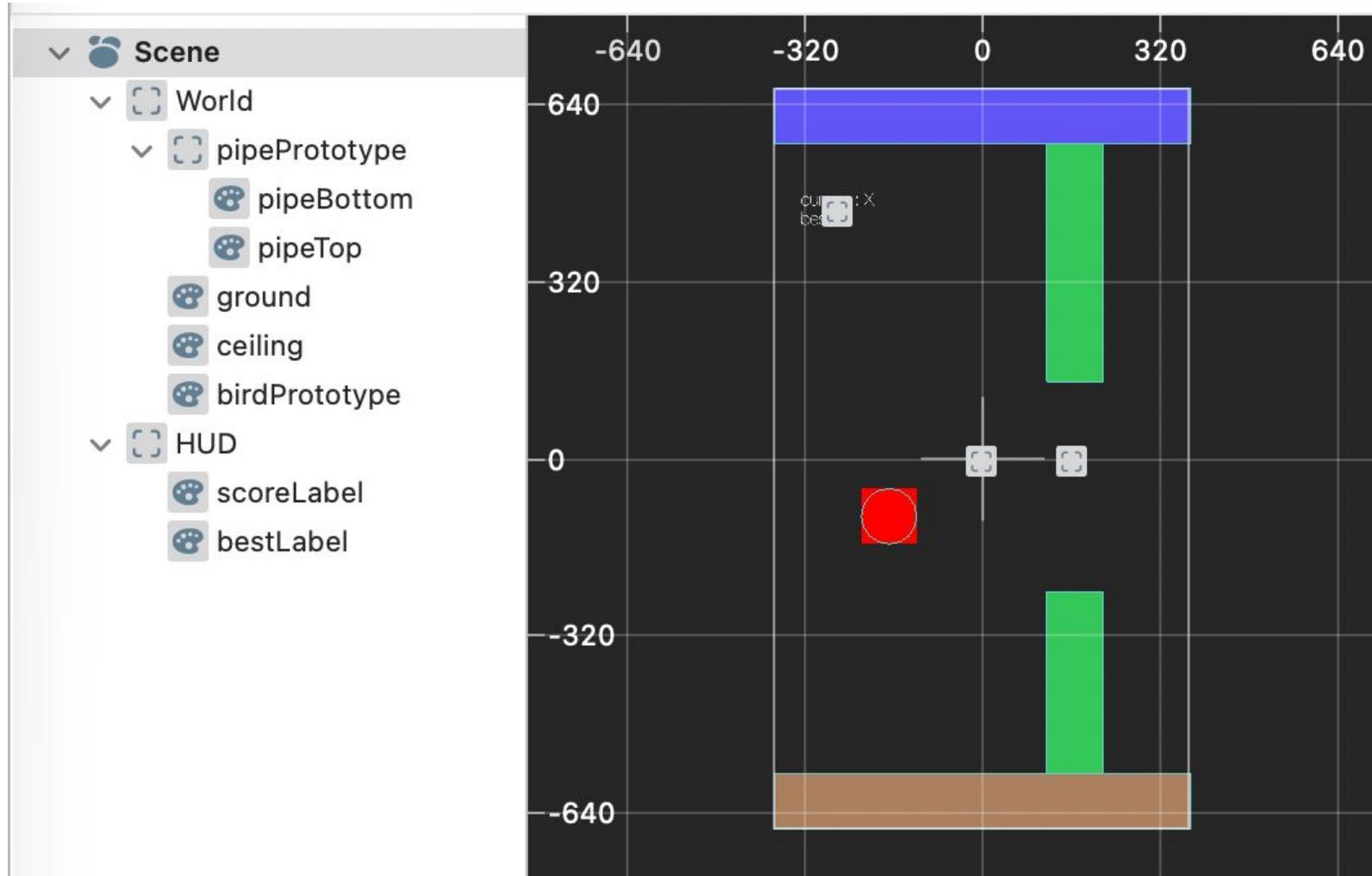
## Auto Restart

- Short delay
- Reset game state
- Start again

```
private func killBird(_ bird: SKSpriteNode) {
    guard let idx = birds.firstIndex(where: { $0 === bird }) else { return }
    let b = birds[idx]
    b.alpha = 0
    b.physicsBody?.isDynamic = false

    // If all birds are dead -> auto restart
    if birds.allSatisfy({ $0.physicsBody?.isDynamic == false }) {
        triggerGameOver()
    }
}

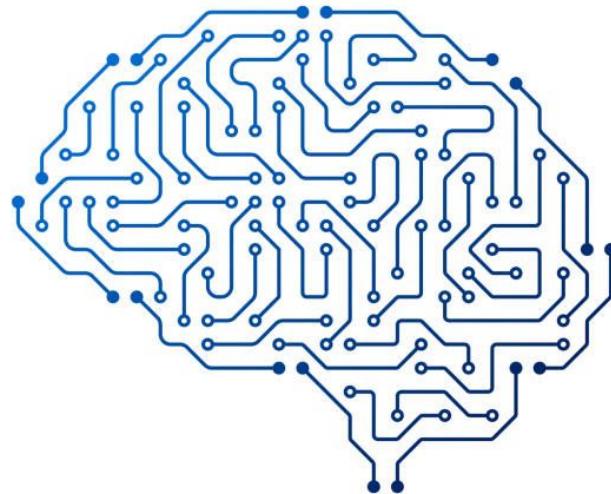
private func triggerGameOver() {
    guard !gameOver else { return }
    gameOver = true
    restartAt = lastUpdateTime + 0.6
}
```



## 2: Building the Bird's Brain

# What is AI?

- Process of collecting data
- Storing data
- Using data to make decisions



# What are we going to build today?



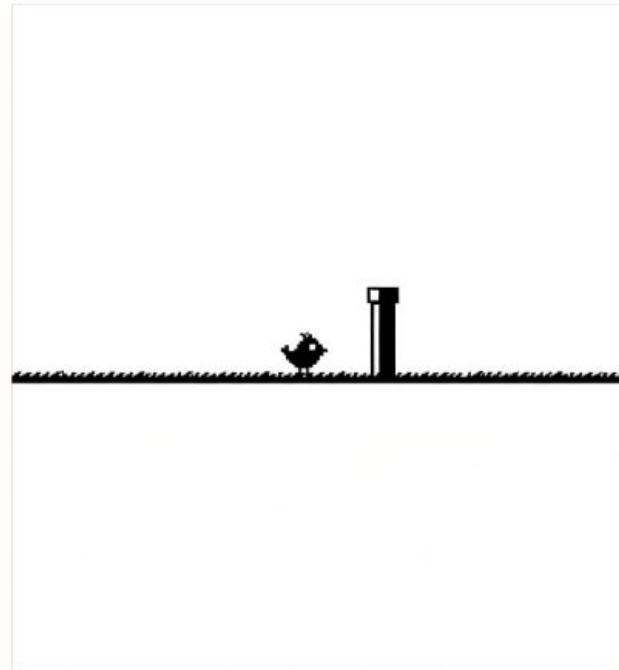
Here's our plan:

1. Make a bird that can decide when to flap
2. Give it a simple brain
3. Let it play the game
4. See why it fails
5. Figure out how to make it better

# What Are We Doing?

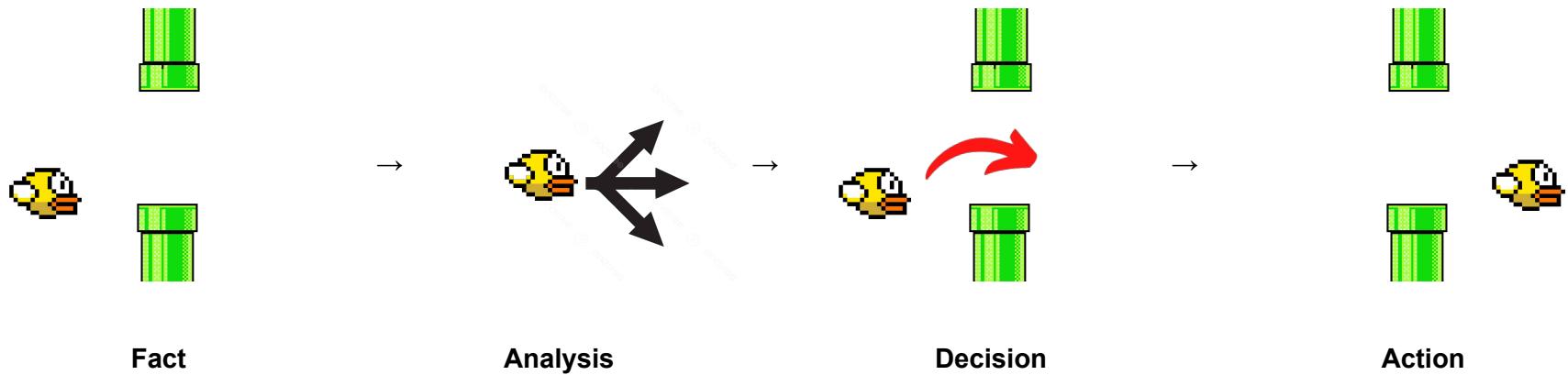
Flap or Not Flap

```
final class NeuralNetwork {
```



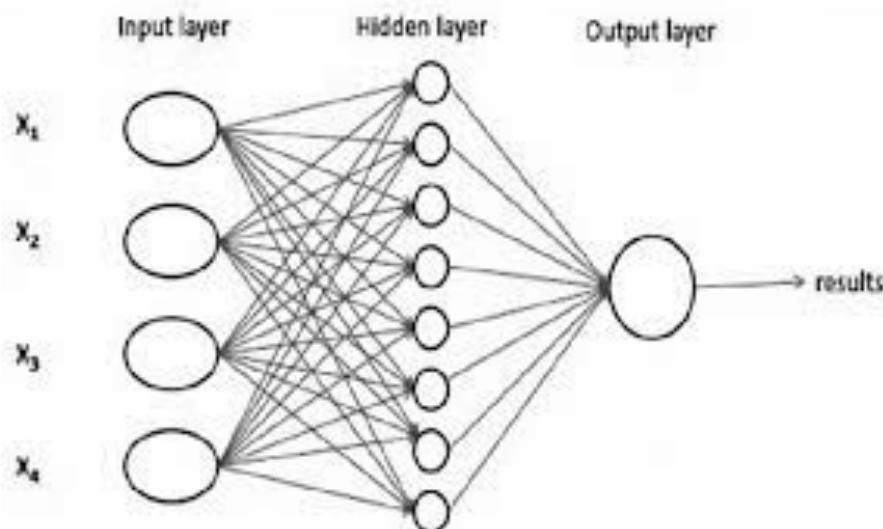
# From Facts to Action

The brain doesn't think in words — it processes information.



# Brain structure

- **4 inputs** → facts the bird know about the situation  
*(height, top pipe, bottom pipe, distance)*  
*(only raw information, nothing else)*
- **8 hidden neurons** → little helpers that mix and think about the facts  
*(they do NOT decide)*  
*(each sees all 4 inputs)*  
*(each focuses differently)*
- **1 output** → final decision



# Importance and instinct

## Trust level of input

- $w_1$  → tells hidden neurons **how important each fact is**  
*(pipe distance may matter more than height, etc.)*
- $w_2$  → tells output **how important each hidden neuron is**  
*(which helper we trust more)*

## Instinct

- $b_1$  → small adjustments for hidden neurons  
*(personal tendency of each helper)*
- $b_2$  → small adjustment for output  
*(overall instinct push before decision)*

# The Bird's Facts

That's **all** it gets

- Its current height
- Where the top of the pipe is
- Where the bottom of the pipe is
- How far the next pipe is



```
let inputs: [Double] = [
    birdY / height,           // bird height
    topY / height,            // top pipe
    botY / height,            // bottom pipe
    min(dist, 600) / 600.0 // distance to next pipe
]
```

# How the Brain Analyzes

- Takes facts
  - a. Each fact has importance (**weights**)
- Multiply each fact by its importance
- Adds all the values together
  - a. Plus a small extra number (**bias**) to help adjust decisions.
- Passed through a filter (**sigmoid**)
- Gets a number from 0 to 1

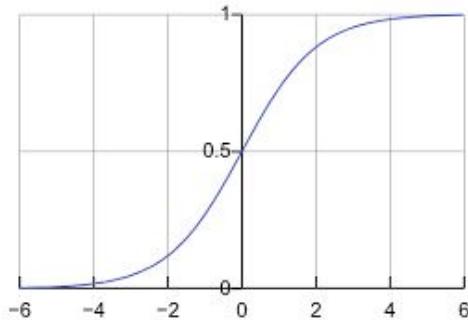
```
// calculate brain's decision based on input facts
func predict(_ inputs: [Double]) -> Double {
    var hidden = [Double](repeating: 0, count: 8) // store
    hidden neuron outputs

    // compute each hidden neuron
    for i in 0..<8 {
        var sum = b1[i] // start with bias (adjustment)
        for j in 0..<4 {
            sum += inputs[j] * w1[i][j] // each fact × importance
        }
        hidden[i] = sigmoid(sum) // filter result to 0-1
    }

    // compute output neuron
    var out = b2 // start with output bias
    for i in 0..<8 {
        out += hidden[i] * w2[i] // combine all hidden outputs
    }

    return sigmoid(out) // final probability to flap
}
```

# Sigmoid function



$$1.0 / (1.0 + \exp(-x))$$

This turns any number into a value between **0 and 1**, which is perfect for decisions.

# The Brain's Output



Analyzes → produces **one number**

- Big ( $> 0.5$ ) → flap
- Small → do nothing

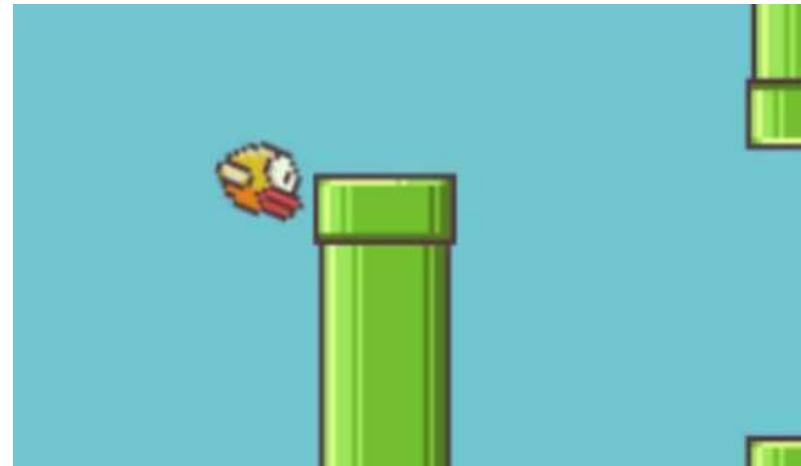


```
func shouldFlap(birdIndex: Int, birdY: Double, topY: Double,  
    botY: Double, dist: Double, height: Double) -> Bool {  
    let g = genomes[birdIndex]  
    guard g.alive else { return false } // dead birds don't flap  
  
    // convert game state into normalized facts  
    let inputs: [Double] = [  
        birdY / height,           // bird height  
        topY / height,           // top pipe  
        botY / height,           // bottom pipe  
        min(dist, 600) / 600.0 // distance to next pipe  
    ]  
    return g.brain.predict(inputs) > 0.5 // logic: flap if brain  
    says yes  
}
```

So... does this bird fly well?

# Why the First Brain Is Bad

- Starts random → importance chosen **by chance**
- Brain doesn't know what matters yet
- Flaps randomly → crashes



```
// create initial population with random brains
genomes = (0..<popSize>).map { _ in Genome(brain:
    NeuralNetwork()) }
}
```

# How do we make the brain better?

We have a problem:

- One bird
- One random brain
- No learning



How do we improve the brain without teaching it?

# Big Idea: use many birds

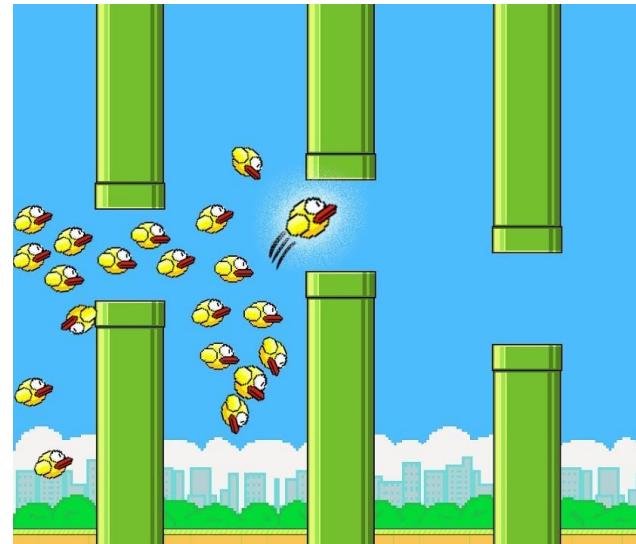
Instead of 1 bird...

we create **100 birds**.

All with:

- the same structure
- different random brains

We use **natural selection** — the best birds survive and become the next generation of 100 birds.



# Learning Through Evolution



```
func evolveToNextGen() {
```

The next generation is slightly better than the last.  
Over many generations, the brains improve automatically  
through **natural selection**.

AI:

## Nature:

- Many are born
- Some survive longer
- The best survive and pass traits

- 100 birds start
- some crash immediately, some fly longer
- keep and copy the best brains
- slightly change them, sometimes mix two

# About the Code

The full project is already pre-coded.

We will briefly explain the main classes  
and the purpose of each one.

# A bird = Genome

Each bird:

- Has a brain
- Can be alive or dead
- Gets points
- Travels distance

```
final class Genome {  
    let brain: NeuralNetwork // brain of this bird  
    var alive: Bool = true    // is it still flying?  
    var score: Int = 0        // pipes passed  
    var distance: Double = 0 // how far it traveled
```

# Fitness = how good the bird is

```
var fitness: Double { Double(score) * 1000 + distance }  
// logic: combines score and distance into one "how good" number
```

Fitness is:

- Mostly based on score
- Distance matters too
- Higher fitness = better bird

# Managing all birds — FlappyAI

```
final class FlappyAI {
```

This class:

- Creates birds
- Tracks them
- Kills them on collision
- Chooses the best
- Creates the next generation

# During the game

Each game tick:

- `shouldFlap(...)` → asks the brain if it should flap
- `addScore(i:)` → increases score when passing pipes
- `tickAlive(i:, distance:)` → updates distance
- `kill(i:)` → kills bird if it crashes

## Sort by best fitness

Best birds go first.

```
genomes.sort { $0.fitness > $1.fitness } // rank birds by  
fitness
```

# Select elites

Take the **top 10%** (at least 2 birds).

They become parents.

```
let eliteCount = max(2, popSize / 10) // keep top 10% as  
parents  
let elites = Array(genomes.prefix(eliteCount))
```

# Copy elites directly

Best birds go into the next generation **unchanged**.

```
// 1) keep elites exactly
for e in elites {
    newGen.append(Genome(brain: e.brain.copy())) // best
        birds survive unchanged
}
```

# Create children

While population is not full:

```
let p1 = elites.randomElement()! // parent 1
let p2 = elites.randomElement()! // parent 2

let child = NeuralNetwork.average(p1.brain, p2.brain) // combine traits
child.tinyMutate(chance: 0.03, amount: 0.08) // slight variation
```

## a) Crossover (averaging weights)

Child's brain = mix of **two parents' brains**.

Combines their strengths.

## b) Mutation

With a small chance (3%), some numbers change slightly.

This helps:

- Create new ideas
- Avoid getting stuck

# New generation is ready

Now all birds are new, and the cycle repeats.

```
genomes = newGen // replace old population  
generation += 1 // increment generation counter
```

# Why does this work?

Because:

- Bad birds die
- Good birds reproduce
- Children are slightly different
- Good differences survive

This is **evolution**, just like in nature.

# Making Learning Faster

## ***Simple code***

- Training time: ~1 hour
- Same result
- Slower learning

## ***Advanced code***

- Training time: ~5 minutes
- Same result
- Faster learning

Same goal. Same bird. Just faster training.

# IMPORTANT: This is NOT classic machine learning

There is:

✗ No correct answers

✗ No magic math that fixes mistakes automatically (no loss function)

✗ No complex brain training steps (no backpropagation)

✓ Birds play the game

✓ We watch who does well

✓ The best birds get to have children

✓ Children brains are copied and slightly changed

Back to the code

# Where Evolutionary AI Can Be Used

- Self-driving cars → AI decides how to drive safely
- Medical diagnosis → AI finds patterns in patient data
- Recommendations → AI picks the best options for users
- Robots → AI learns how to move or complete tasks
- Trading → AI adapts to market conditions



Anywhere you have **data** → **decision** → **improvement over time**

We didn't build a smart bird.

We built:

data → analysis → decision → improvement

That's the foundation of AI.