ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

ΤίτλοςΠτυχιακής Εργασίας	ΣΧΕΔΙΑΣΗ ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΠΑΙΓΝΙΟΥ ΜΝΗΜΗΣ ΓΙΑ ΦΟΡΗΤΕΣ ΣΥΣΚΕΥΕΣ ΜΕ ΛΕΙΤΟΥΡΓΙΚΟ iOS Memory Game for iOS Mobile Devices
Ονοματεπώνυμο Φοιτητή	ΕΥΑΓΓΕΛΟΣ ΑΓΓΕΛΟΣ ΡΕΠΟΥΣΗΣ
Πατρώνυμο	ΣΤΑΜΑΤΙΟΣ
Αριθμός Μητρώου	Π/ 20236
Επιβλέπων	ΕΥΑΓΓΕΛΟΣ ΣΑΚΚΟΠΟΥΛΟΣ, Καθηγητής

Ημερομηνία Παράδοσης Ιούλιος 2026

Επιτελική Σύνοψη

Η παρούσα εργασία πραγματεύεται τη δημιουργία μιας εφαρμογής iOS με τη γλώσσα προγραμματισμού Swift και το framework SwiftUI, που έχει ως στόχο να αποτελέσει ένα σύγχρονο ψηφιακό παιχνίδι μνήμης για χρήστες κινητών συσκευών Apple. Η ανάπτυξη της εφαρμογής κρίθηκε σκόπιμη λόγω της αυξανόμενης δημοτικότητας των γνωστικών παιχνιδιών σε κινητές πλατφόρμες, αλλά και της ανάγκης εφαρμογής σύγχρονων αρχιτεκτονικών προτύπων ανάπτυξης λογισμικού σε ένα πραγματικό, λειτουργικό project.

Η εφαρμογή MemorizeApp υλοποιεί το κλασικό παιχνίδι αντιστοίχισης καρτών (Concentration Game), εμπλουτισμένο με τρία θεματικά πακέτα emoji (Halloween, Ζώα, Αθλήματα), τρία επίπεδα δυσκολίας (Easy, Medium, Hard), σύστημα bonus βαθμολόγησης βασισμένο στον χρόνο αντιστοίχισης, αποθήκευση υψηλής βαθμολογίας ανά θέμα και δυσκολία μέσω UserDefaults, καθώς και πλούσια animations κατά τη διάρκεια του παιχνιδιού: 3D flip animation καρτών, sequential deal animation, bonus timer animation (Pie shape) και FlyingNumber animation βαθμολογίας.

Η υλοποίησή της πραγματοποιήθηκε με το Xcode IDE και τη βοήθεια του SwiftUI framework, που παρέχει δηλωτικό (declarative) τρόπο ανάπτυξης γραφικής διεπαφής. Για την κατανόηση και τη σαφή δομή του project υιοθετήθηκε το αρχιτεκτονικό πρότυπο MVVM (Model-View-ViewModel), το οποίο διαχωρίζει πλήρως τη λογική του παιχνιδιού από την παρουσίασή της. Για την ανάλυση και τη σχεδίαση χρησιμοποιήθηκαν διαγράμματα UML (Use case diagram και Sequence diagram), τα οποία συνέβαλαν στη σαφή απεικόνιση της λογικής και της αρχιτεκτονικής πριν την υλοποίηση.

Ευχαριστίες

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον καθηγητή κ. Ευάγγελο Σακκόπουλο για τη δυνατότητα που μου έδωσε να πραγματοποιήσω την πτυχιακή μου εργασία, καθώς και για την υποστήριξη που μου παρείχε όποτε τη χρειάστηκα κατά τη διάρκεια της υλοποίησής της.

Επίσης, θα ήθελα να ευχαριστήσω όλους τους καθηγητές του Τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς για τις πολύτιμες γνώσεις που αποκόμισα κατά τη διάρκεια των σπουδών μου, οι οποίες αποτέλεσαν το θεμέλιο για την ολοκλήρωση της παρούσας εργασίας.

Ιούλιος 2026
Ευάγγελος Αγγελος Ρεπούσης

ΚΕΦΑΛΑΙΟ 1ο

1 ΕΙΣΑΓΩΓΗ

1.1 Περιγραφή του υπό μελέτης προβλήματος

Τα παιχνίδια μνήμης ανήκουν στην κατηγορία των γνωστικών παιχνιδιών και έχουν μελετηθεί εκτενώς τόσο για τα ψυχαγωγικά όσο και για τα γνωστικά οφέλη που παρέχουν. Η αντιστοίχιση ζευγών καρτών απαιτεί συγκέντρωση, βραχυπρόθεσμη μνήμη (working memory) και παρατηρητικότητα, καθιστώντας το παιχνίδι κατάλληλο για ευρύ φάσμα ηλικιών. Ταυτόχρονα, η απλότητά του το καθιστά ιδανικό για υλοποίηση σε κινητή εφαρμογή, καθώς οι κανόνες μαθαίνονται γρήγορα και η παρτίδα ολοκληρώνεται σε σύντομο χρόνο.

Παρά τη μεγάλη δημοτικότητα του είδους, η πλειονότητα των διαθέσιμων εφαρμογών παιχνιδιού μνήμης στο App Store παρουσιάζει σημαντικές ελλείψεις. Οι περισσότερες εφαρμογές δεν αξιοποιούν το declarative UI framework SwiftUI, το οποίο παρέχει ισχυρά ενσωματωμένα εργαλεία animations και reactive διαχείριση κατάστασης. Παράλληλα, δεν ακολουθούν σύγχρονα αρχιτεκτονικά πρότυπα όπως το MVVM που διευκολύνει τη συντήρηση και επέκταση του κώδικα, δεν προσφέρουν ποικιλία θεμάτων και επιπέδων δυσκολίας που να κρατούν το ενδιαφέρον του χρήστη, και δεν διαθέτουν σύστημα βαθμολόγησης που να επιβραβεύει τη γρήγορη σκέψη αποθηκεύοντας την υψηλή βαθμολογία ανά θέμα και δυσκολία.

Επιπλέον, πολλές εφαρμογές χρησιμοποιούν παλαιότερες τεχνολογίες όπως το UIKit (imperative UI) ή ακόμα και Objective-C, γεγονός που τις καθιστά δύσκολες στη συντήρηση και λιγότερο επεκτάσιμες. Αυτές οι ελλείψεις, σε συνδυασμό με την επιθυμία εφαρμογής σύγχρονων τεχνολογιών iOS σε ένα πραγματικό project, οδήγησαν στην ανάπτυξη της εφαρμογής MemorizeApp.

1.2 Χρήση παρόμοιων εφαρμογών

Τα τελευταία χρόνια παρατηρείται αυξημένο ενδιαφέρον για εφαρμογές κινητής τηλεφωνίας που συνδυάζουν ψυχαγωγία με γνωστική εξάσκηση. Στο App Store υπάρχουν αρκετές

εφαρμογές παιχνιδιού μνήμης, αλλά καμία δεν συνδυάζει όλα τα επιθυμητά χαρακτηριστικά που τέθηκαν ως στόχοι.

Σε διεθνές επίπεδο, εφαρμογές όπως το 'Memory Match' και το 'Concentration Game' προσφέρουν παρόμοια παιχνίδια αλλά χρησιμοποιούν UIKit (imperative UI) αντί για SwiftUI (declarative UI), κάτι που κάνει τον κώδικα πιο επιρρεπή σε σφάλματα κατάστασης και πιο δύσκολο να επεκταθεί. Παρά την ύπαρξη των παραπάνω λύσεων, δεν υπάρχει ολοκληρωμένη εφαρμογή που να συνδυάζει: MVVM αρχιτεκτονική, SwiftUI animations, πολλαπλά θέματα, επίπεδα δυσκολίας, σύστημα high score ανά συνδυασμό θέματος και δυσκολίας, πλήρως offline λειτουργία και οθόνη οδηγιών παιχνιδιού.

1.3 Σκοπός και στόχοι της εργασίας

Σκοπός της εργασίας είναι η ανάπτυξη μιας ολοκληρωμένης iOS εφαρμογής παιχνιδιού μνήμης που θα εφαρμόζει πιστά το MVVM αρχιτεκτονικό πρότυπο, θα αξιοποιεί πλήρως τις δυνατότητες του SwiftUI framework και θα παρέχει εξαιρετική εμπειρία χρήστη που να δικαιολογεί δημοσίευση στο App Store.

Υπάρχουν κάποιοι βασικοί στόχοι που τέθηκαν για την ανάπτυξη της εφαρμογής. Ο πρώτος στόχος ήταν η υλοποίηση ενός generic Model (MemoryGame<CardContent>) που να διαχωρίζει πλήρως τη λογική του παιχνιδιού από το UI, επιτρέποντας δοκιμή της λογικής ανεξάρτητα από την παρουσίαση και εύκολη επέκταση με νέους τύπους περιεχομένου καρτών στο μέλλον. Ένας ακόμα στόχος ήταν η δημιουργία ελκυστικού και ομαλού συστήματος animations αξιοποιώντας το Animatable protocol για το 3D flip των καρτών, το matchedGeometryEffect για το deal animation, τα custom Shapes (Pie) για τον bonus timer και το FlyingNumber View για την εμφάνιση αλλαγής βαθμολογίας. Σημαντικός στόχος ήταν επίσης η ανάπτυξη δίκαιου και κινητοποιητικού συστήματος βαθμολόγησης που να επιβραβεύει τη γρήγορη σκέψη μέσω bonus timer (έως +6 βαθμοί ανά κάρτα) και να τιμωρεί τις επαναλήψεις γνωστών καρτών (-1 βαθμός ανά γνωστή κάρτα), με αποθήκευση της βέλτιστης βαθμολογίας ανά συνδυασμό θέματος και δυσκολίας. Τέλος, στόχος ήταν η δημιουργία εύχρηστης, ελκυστικής και πλήρως offline διεπαφής που να ακολουθεί τις

κατευθυντήριες γραμμές Apple Human Interface Guidelines (HIG), με dynamic color theming ανά θέμα παιχνιδιού.

1.4 Χρήση εργαλείων λογισμικού

Για την ανάπτυξη της εφαρμογής αξιοποιήθηκαν σύγχρονα εργαλεία λογισμικού, τα οποία επιλέχθηκαν με γνώμονα την ευκολία υλοποίησης, την επεκτασιμότητα και τις τελευταίες τεχνολογικές εξελίξεις στην ανάπτυξη εφαρμογών iOS.

Το περιβάλλον ανάπτυξης που χρησιμοποιήθηκε είναι το Xcode (έκδοση 16.x), το επίσημο Integrated Development Environment (IDE) της Apple για ανάπτυξη εφαρμογών iOS, macOS, watchOS και tvOS. Παρέχει editor κώδικα Swift με προηγμένη αυτόματη συμπλήρωση και intelligent refactoring, SwiftUI Canvas για real-time προεπισκόπηση Views χωρίς εκτέλεση της εφαρμογής, debugger με LLDB για εντοπισμό σφαλμάτων, Instruments για ανάλυση απόδοσης και κατανάλωσης μνήμης (CPU Profiler, Memory Graph Debugger), και iOS Simulator που επιτρέπει δοκιμή σε εικονικές iOS συσκευές — από iPhone SE έως iPhone 16 Pro Max και iPad Pro — χωρίς να απαιτείται φυσικό hardware.

Η ανάπτυξη της εφαρμογής έγινε αποκλειστικά με τη γλώσσα Swift (έκδοση 5.9). Η Swift έκανε την πρώτη εμφάνισή της το 2014 ως επίσημη γλώσσα της Apple αντικαθιστώντας σταδιακά την Objective-C, και από το 2015 είναι open-source αναπτυσσόμενη μέσω του swift.org. Παρέχει χαρακτηριστικά όπως ισχυρό στατικό σύστημα τύπων που εντοπίζει σφάλματα κατά τη μεταγλώττιση, Generics για επαναχρησιμοποιήσιμο κώδικα, Protocols για ορισμό συμβολαίων συμπεριφοράς, Closures για συναρτησιακό στυλ, Enumerations με associated values για εκφραστική αναπαράσταση καταστάσεων, και Automatic Reference Counting (ARC) για αυτόματη διαχείριση μνήμης.

Για το γραφικό περιβάλλον χρησιμοποιήθηκε το SwiftUI framework (έκδοση 5.0), το declarative UI framework της Apple που παρουσιάστηκε στο WWDC 2019. Σε αντίθεση με το UIKit (imperative UI) όπου ο προγραμματιστής δίνει ρητές εντολές για αλλαγές στο UI, το SwiftUI λειτουργεί declaratively: ο προγραμματιστής περιγράφει πώς πρέπει να φαίνεται το UI για κάθε δυνατή κατάσταση, και το framework αναλαμβάνει αυτόματα τις μεταβάσεις.

Αυτό οδηγεί σε πιο αναγνώσιμο κώδικα, ευκολότερη συντήρηση και λιγότερα bugs κατάστασης.

Για την αποθήκευση δεδομένων επιλέχθηκε το UserDefaults API, το ενσωματωμένο σύστημα key-value αποθήκευσης του iOS. Αποθηκεύει δεδομένα σε αρχείο property list (.plist) στον φάκελο Library/Preferences της εφαρμογής. Είναι ιδανικό για μικρές ποσότητες δεδομένων όπως υψηλές βαθμολογίες, παρέχει synchronous API και δεν απαιτεί κανένα setup. Για τη δημιουργία custom γραφικών στοιχείων αξιοποιήθηκε το Core Graphics framework μέσω του Shape protocol του SwiftUI. Τέλος, αξιοποιήθηκε το Git για έλεγχο έκδοσης σε συνδυασμό με το OneDrive για backup και διαμοιρασμό του project.

ΚΕΦΑΛΑΙΟ 2ο

2 Σχεδίαση της Εφαρμογής

Σε αυτό το κεφάλαιο θα εξετάσουμε την ανάλυση και τη σχεδίαση της iOS εφαρμογής MemorizeApp. Η ανάλυση απαιτήσεων μιας σύγχρονης εφαρμογής είναι πολύ κομβική προκειμένου να μειωθεί η πολυπλοκότητα κατά την υλοποίηση και να διασφαλιστεί μια πιο οργανωμένη και αποδοτική ανάπτυξη. Με έναν σωστά σχεδιασμένο χάρτη λειτουργιών, ο προγραμματιστής αποφεύγει σημαντικά λάθη αρχιτεκτονικής που θα ήταν κοστοβόρα να διορθωθούν αργότερα.

Συγκεκριμένα, χρησιμοποιήθηκαν διαγράμματα UML (Unified Modeling Language). Τα διαγράμματα αυτά επιτρέπουν στον προγραμματιστή να κατανοήσει πλήρως τις απαιτήσεις της εφαρμογής, τη συνολική λειτουργικότητά της και την οργάνωσή της. Η UML αποτελεί μια γλώσσα απεικόνισης ή μοντελοποίησης ενός πληροφοριακού συστήματος βασισμένου σε αντικείμενα. Η γλώσσα αυτή βοηθάει τόσο τον προγραμματιστή στη δημιουργία πολύπλοκων εφαρμογών όσο και τον αναγνώστη να τις κατανοήσει, ανεξάρτητα από την εξοικείωσή του με την πληροφορική και τον προγραμματισμό.

2.1 Χρήση διαγραμμάτων UML

Τα διαγράμματα UML είναι απαραίτητα στην ανάλυση και σχεδίαση μιας σύγχρονης πολύπλοκης εφαρμογής πληροφορικής. Υπάρχουν διάφοροι τύποι διαγραμμάτων που χρησιμοποιούνται για διαφορετικές πτυχές μοντελοποίησης. Οι διαφορετικοί αυτοί τύποι είναι οι εξής:

1. Διαγράμματα τάξεων ή κλάσεων
2. Διαγράμματα αντικειμένων
3. Διαγράμματα συνεργασίας
4. Διαγράμματα σειράς ή ακολουθίας
5. Διαγράμματα καταστάσεων
6. Διαγράμματα δραστηριοτήτων

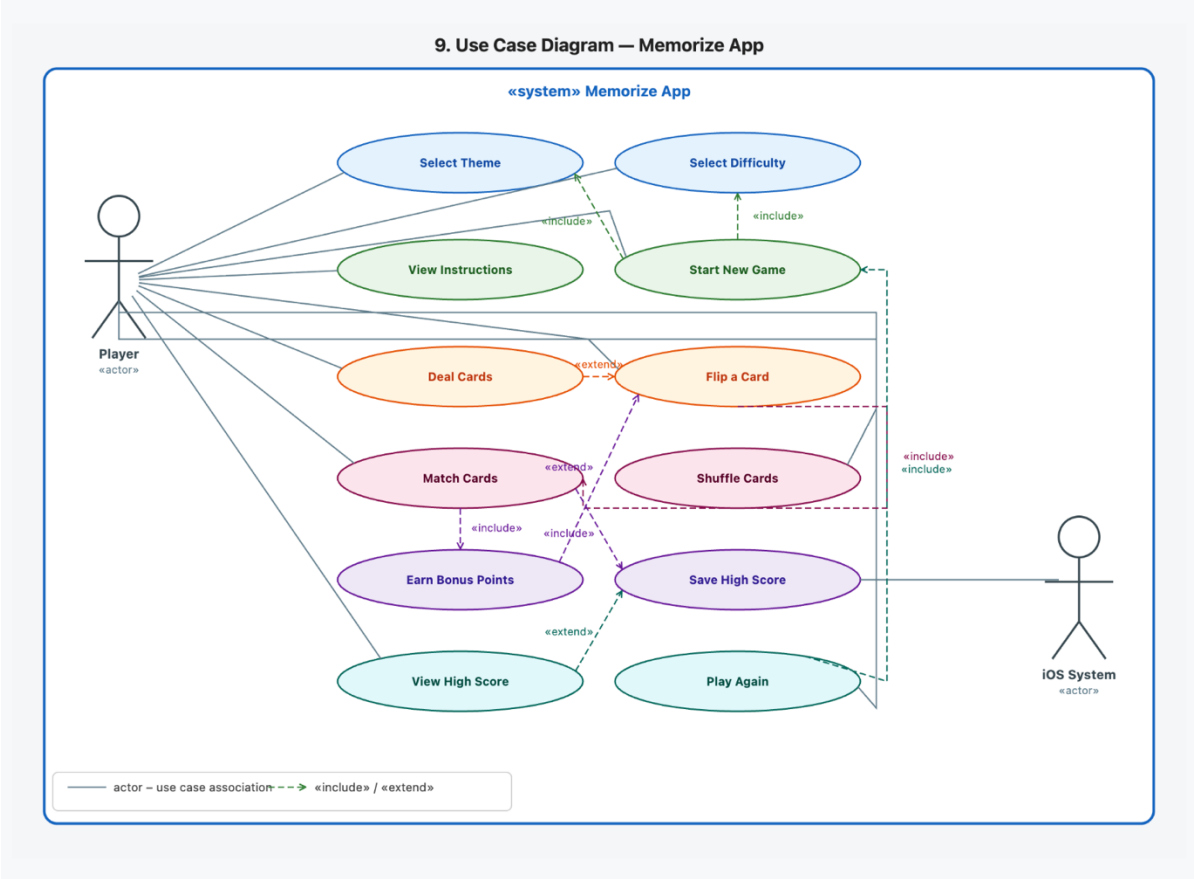
7. Διαγράμματα κατανομής
8. Διαγράμματα εξαρτημάτων
9. Διαγράμματα περιπτώσεων χρήσης

Στην ενότητα αυτή θα εμβαθύνουμε στα διαγράμματα κατηγοριών 4 και 9, ακολουθίας και περιπτώσεων χρήσης αντίστοιχα. Όλα τα διαγράμματα έχουν τη χρησιμότητά τους, αλλά συγκεκριμένα αυτά τα δύο είναι τα πιο απαραίτητα για την καλύτερη και αποδοτικότερη υλοποίηση μιας πολύπλοκης εφαρμογής. Για τη δημιουργία των παρακάτω διαγραμμάτων χρησιμοποιήθηκε η εφαρμογή Draw.io.

2.1.1 Διάγραμμα περίπτωσης χρήσης (Use case diagram)

Τα διαγράμματα περιπτώσεων χρήσης αποτελούν μια από τις πιο σημαντικές κατηγορίες διαγραμμάτων UML. Απεικονίζουν τη λειτουργικότητα ενός συστήματος από την οπτική γωνία ενός χρήστη, περιγράφοντας τις αλληλεπιδράσεις του ως «ενεργοποιό» (actor) με τις λειτουργίες που μπορεί να πραγματοποιήσει, οι οποίες ονομάζονται «περιπτώσεις χρήσης» (use cases). Κάθε περίπτωση χρήσης αναπαρίσταται ως ελλειπτικό σχήμα, ο ενεργοποιός ως ανθρωπάκι, και οι σχέσεις επέκτασης (extend) αναπαρίστανται ως διακεκομμένα βέλη.

Το παρακάτω διάγραμμα αποτελείται από έναν ενεργοποιό — τον Παίκτη — και τις ενέργειες που μπορεί να πραγματοποιήσει στην εφαρμογή MemorizeApp.



Εικόνα 1 — Διάγραμμα περιπτώσεων χρήσης παίκτη

Περιπτώσεις Χρήσης:

- Επιλογή Θέματος: Ο παίκτης επιλέγει ένα από τα τρία διαθέσιμα θέματα (Halloween, Ζώα, Αθλήματα) μέσω segmented Picker στο GameSetupView. Το χρώμα ολόκληρης της εφαρμογής αλλάζει δυναμικά ανάλογα με την επιλογή — πορτοκαλί για Halloween, πράσινο για Animals, μπλε για Sports.
- Επιλογή Δυσκολίας: Ο παίκτης επιλέγει επίπεδο δυσκολίας μέσω segmented Picker: Easy (έως 6 ζεύγη), Medium (έως 12 ζεύγη), Hard (όλα τα διαθέσιμα ζεύγη του θέματος). Ο αριθμός καρτών υπολογίζεται ως min(targetCount, theme.emojis.count).
- Προβολή High Score: Εμφάνιση της υψηλής βαθμολογίας για τον τρέχοντα συνδυασμό θέματος και δυσκολίας. Διαβάζεται από UserDefaults με σύνθετο κλειδί και ενημερώνεται αυτόματα κάθε φορά που αλλάζει η επιλογή.

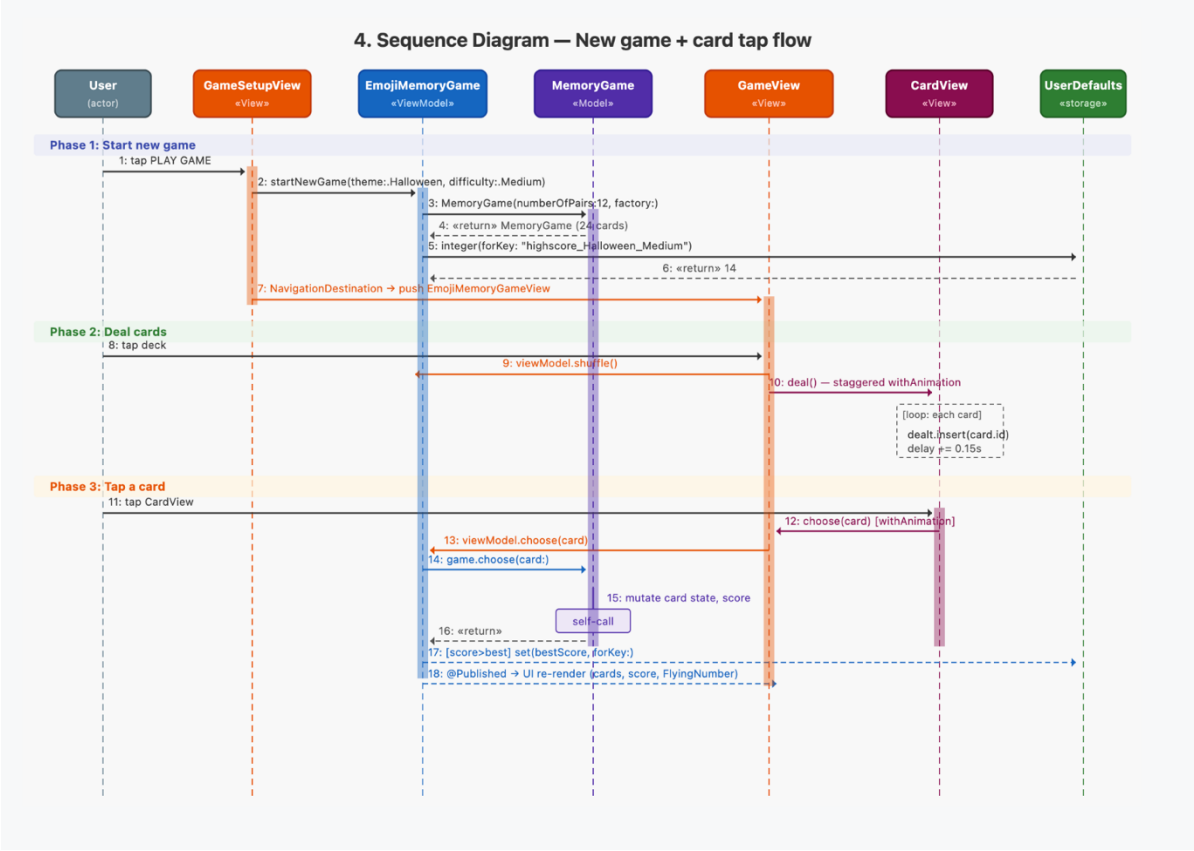
- Εμφάνιση Οδηγιών: Πάτημα κουμπιού ⓘ (SF Symbol: info.circle) στο toolbar για εμφάνιση της οθόνης InstructionsView ως sheet (modal presentation) με τέσσερις κανόνες παιχνιδιού.
- Εκκίνηση Παιχνιδιού: Πάτημα PLAY GAME → startNewGame(using:difficulty:) στο ViewModel → δημιουργία νέου Model instance → φόρτωση αντίστοιχου high score → πλοήγηση στο EmojiMemoryGameView μέσω NavigationStack.
- Διανομή Καρτών: Πάτημα στοίβας (deck) → shuffleCards() + deal() → οι κάρτες 'πετούν' από τη στοίβα στις θέσεις τους στο grid με sequential animation delay 0.15 δευτ. ανά κάρτα μέσω matchedGeometryEffect.
- Επιλογή Κάρτας: Πάτημα κάρτας → choose() → 3D flip animation (Cardify ViewModifier). Εμφανίζεται το emoji content και το bonus timer (Pie shape) αρχίζει να συρρικνώνεται σε πραγματικό χρόνο.
- Αντιστοίχιση Ζεύγους: Εάν δύο κάρτες ταιριάζουν → isMatched = true → βαθμολόγηση +2 + bonus1 + bonus2 → FlyingNumber animation πορτοκαλί → περιστροφή emoji 360° → κάρτες εξαφανίζονται.
- Αποτυχία Αντιστοίχισης (επέκταση): Εάν δύο κάρτες δεν ταιριάζουν → -1 βαθμός για κάθε γνωστή κάρτα → FlyingNumber animation κόκκινο → κάρτες γυρίζουν ανάποδα.
- Ανακάτεμα Καρτών: Πάτημα Shuffle → shuffle() στο Model → αναδιάταξη καρτών στο grid χωρίς απώλεια βαθμολογίας.
- Ολοκλήρωση Παιχνιδιού: Όταν cards.allSatisfy{$0.isMatched} → isGameOver == true → εμφάνιση Game Over screen με 'You Won! 🎉', τελική βαθμολογία, high score και κουμπί Play Again.
- Έξοδος: Πάτημα Play Again → dismiss() → επιστροφή στο GameSetupView. Το high score έχει ήδη αποθηκευτεί κατά τη διάρκεια του παιχνιδιού.

2.1.2 Διάγραμμα ακολουθίας (Sequence diagram)

Τα διαγράμματα ακολουθίας αποτελούν εξίσου πολύ σημαντική κατηγορία διαγραμμάτων UML. Χρησιμοποιούνται για να περιγράψουν τις αλληλεπιδράσεις μεταξύ των αντικειμένων ενός πληροφοριακού συστήματος με χρονική σειρά. Εστιάζουν στους τρόπους που

επικοινωνούν τα αντικείμενα μεταξύ τους μέσα από μηνύματα, κλήσεις και μεθόδους. Τα αντικείμενα αναπαρίστανται με ορθογώνιο σχήμα στην κορυφή και έχουν από μια κάθετη γραμμή (lifeline) προς τα κάτω που εκφράζει τη χρονική διάσταση. Οι αλληλεπιδράσεις αναπαρίστανται ως οριζόντια βέλη που συνδέουν τις κάθετες γραμμές.

Το παρακάτω διάγραμμα ακολουθίας παρουσιάζει όλες τις ενέργειες που μπορεί να κάνει ο παίκτης κατά χρονική σειρά χρησιμοποιώντας την εφαρμογή MemorizeApp. Λόγω του μεγέθους του, χωρίζεται σε τρία μέρη.



Εικόνα 2 — Διάγραμμα ακολουθίας

Παρατηρώντας τα παραπάνω διαγράμματα βλέπουμε ότι περιέχουν 6 αντικείμενα τα οποία επικοινωνούν μεταξύ τους με συγκεκριμένα μηνύματα και συγκεκριμένη χρονική σειρά. Αναλυτικά:

- Το αντικείμενο ΠΑΙΚΤΗΣ: Ο χρήστης της εφαρμογής, εκκινεί όλες τις ενέργειες.
- Το αντικείμενο GAMESETUPVIEW: Η αρχική οθόνη επιλογής θέματος, δυσκολίας και εκκίνησης.
- Το αντικείμενο EMOJIMEMORYGAME (ViewModel): Το ObservableObject που διαχειρίζεται κατάσταση και intent functions.
- Το αντικείμενο MEMORYGAME (Model): Η λογική παιχνιδιού, οι κάρτες και η βαθμολόγηση.
- Το αντικείμενο EMOJIMEMORYGAMEVIEW: Η κύρια οθόνη παιχνιδιού με grid, στοίβα και controls.
- Το αντικείμενο USERDEFAULTS: Ο μηχανισμός μόνιμης αποθήκευσης υψηλής βαθμολογίας.

Μελετώντας το διάγραμμα, μπορούμε να κατανοήσουμε όλες τις ενέργειες με τη χρονική σειρά που συμβαίνουν. Τα βήματα 1-5 αποτελούν την υποχρεωτική ακολουθία εκκίνησης:

10. Ο παίκτης επιλέγει θέμα και δυσκολία στο GameSetupView.
11. Πατά PLAY GAME → καλείται startNewGame(using:difficulty:) στο ViewModel.
12. Το ViewModel καλεί createMemoryGame(theme:difficulty:) και δημιουργεί νέο Model.
13. Το ViewModel καλεί loadHighScore() που διαβάζει από UserDefaults.
14. Το EmojiMemoryGameView εμφανίζεται με κενό dealt set και το μήνυμα 'Tap the deck to deal!'.

Μετά υπάρχουν εναλλακτικές ροές ανάλογα με τι επιθυμεί ο παίκτης:

Εναλλακτική ροή 1 (alt1) — Διανομή Καρτών: Πάτημα στοίβας → shuffleCards() (shuffle + reset dealt) → deal() με sequential delay 0.15 δευτ. ανά κάρτα → κάρτες μεταφέρονται από στοίβα σε grid μέσω matchedGeometryEffect.

Εναλλακτική ροή 2 (alt2) — Επιτυχής Αντιστοίχιση: Πάτημα κάρτας → choose() → αν content ταιριάζει → isMatched = true → score += 2 + bonus → ViewModel ελέγχει score > bestScore → αν ναι: bestScore = score + UserDefaults.set().

Εναλλακτική ροή 3 (alt3) — Αποτυχία Αντιστοίχισης: Πάτημα κάρτας → choose() → αν content δεν ταιριάζει → score -= 1 για κάθε γνωστή κάρτα → κάρτες γυρίζουν ανάποδα.

Εναλλακτική ροή 4 (alt4) — Ολοκλήρωση: Όταν isGameOver == true → Game Over screen → Play Again → dismiss() → επιστροφή στο GameSetupView με ενημερωμένο high score.

ΚΕΦΑΛΑΙΟ 3ο

3 Αναλυτική Περιγραφή της Υλοποίησης

Σε αυτό το κεφάλαιο θα γίνει αναλυτική περιγραφή της υλοποίησης της iOS εφαρμογής MemorizeApp και πώς αυτή επιτεύχθηκε. Θα γίνει η σύνδεση των διαγραμμάτων UML και της αρχιτεκτονικής MVVM με την τελική μορφή της εφαρμογής. Για κάθε βασικό component παρουσιάζεται ο αντίστοιχος κώδικας με επεξήγηση της λειτουργίας του, καθώς και στιγμιότυπο οθόνης από τον Xcode ή τον Simulator.

Για το στάδιο σχεδιασμού των θεμάτων emoji αξιοποιήθηκε ο κατάλογος unicode emoji της Apple. Επιλέχθηκαν σύμβολα άμεσα αναγνωρίσιμα, θεματικά συνεκτικά και σαφώς διαφορετικά μεταξύ τους ώστε να αποφεύγεται σύγχυση. Η επιλογή emoji αντί για custom graphics μείωσε σημαντικά τον χρόνο ανάπτυξης και εξασφάλισε συνέπεια εμφάνισης σε όλες τις iOS συσκευές.

3.1 Αρχιτεκτονική MVVM

Η εφαρμογή MemorizeApp εφαρμόζει πιστά το αρχιτεκτονικό πρότυπο MVVM (Model-View-ViewModel). Η ροή δεδομένων ακολουθεί μονοδρομική (unidirectional) πορεία: Παίκτης → View (intent call) → ViewModel → Model → ViewModel (@Published update) → View (αυτόματη ανακατασκευή). Κάθε στρώμα έχει σαφώς ορισμένες ευθύνες.

Επίπεδο	Αρχείο	Τύπος	Κύρια Ευθύνη
Model	MemoryGame.swift	struct generic	Λογική παιχνιδιού, αντιστοίχιση, βαθμολόγηση, bonus timer
Model	Theme.swift	struct + enum	Ορισμός θεμάτων emoji, χρωμάτων, enum Difficulty
ViewModel	EmojiMemoryGame.swift	class ObservableObject	Κατάσταση εφαρμογής, intent functions, UserDefaults, high score

View	GameSetupView.swift	struct View	Επιλογή θέματος/δυσκολίας, navigation, εμφάνιση high score
View	EmojiMemoryGameView.swift	struct View	Κύρια οθόνη, grid καρτών, deal animation, score display
View	CardView.swift	struct View	Μεμονωμένη κάρτα, Pie overlay, emoji, Cardify modifier
ViewModifier	Cardify.swift	struct ViewModifier+Animatable	3D flip animation μέσω rotation3DEffect
Shape	Pie.swift	struct Shape	Custom clockwise arc για bonus timer visualization
View Helper	AspectVGrid.swift	struct View generic	Adaptive grid layout — βέλτιστο card size ανά οθόνη
View Helper	FlyingNumber.swift	struct View	Animated +/- αλλαγή βαθμολογίας πάνω από κάρτα

3.2 Η εφαρμογή από τη πλευρά του χρήστη

Το MemorizeApp λειτουργεί πλήρως offline χωρίς εγγραφή ή σύνδεση. Κατά την εκκίνηση εμφανίζεται αυτόματα η αρχική οθόνη GameSetupView. Το entry point της εφαρμογής (MemorizeAppApp.swift) δημιουργεί ένα @StateObject (game: EmojiMemoryGame) που ζει για όλη τη διάρκεια της εφαρμογής και το παρέχει στο GameSetupView.

3.2.1 Αρχική Οθόνη — GameSetupView

Η πρώτη οθόνη που βλέπει ο χρήστης είναι το GameSetupView. Εδώ επιλέγει θέμα και δυσκολία μέσω segmented Pickers, βλέπει την υψηλή βαθμολογία για τον τρέχοντα συνδυασμό, και ξεκινά νέο παιχνίδι. Ο τίτλος 'Memorize!' εμφανίζεται σε 60pt με δυναμικό χρωματισμό. Κουμπί ⓘ στο toolbar ανοίγει τις οδηγίες.

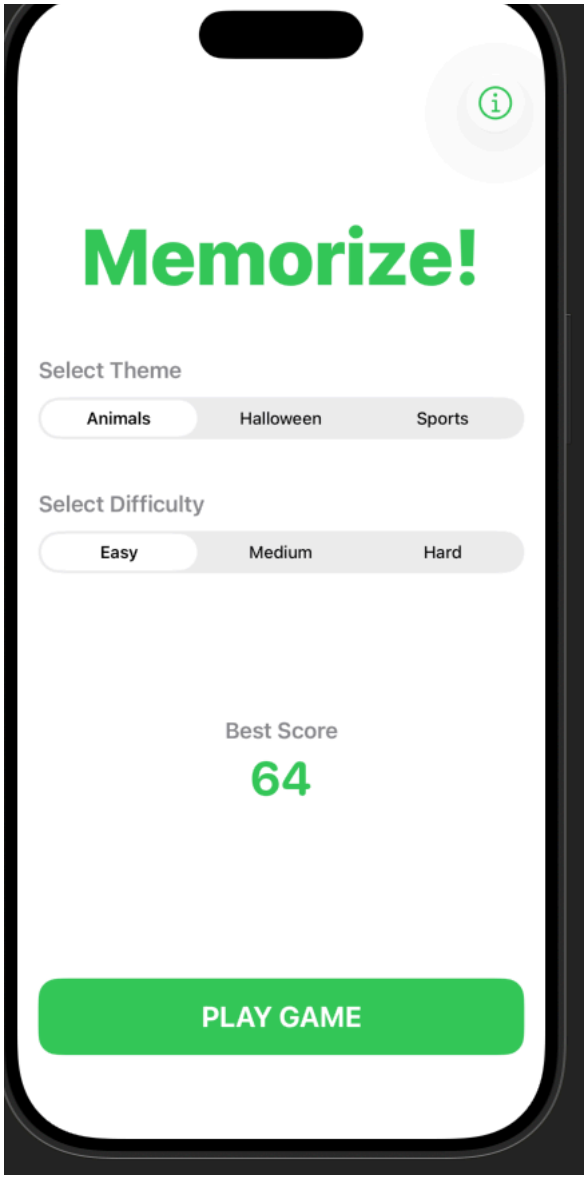
Κομμάτι κώδικα από το GameSetupView — Picker επιλογής θέματος και εμφάνιση high score:

```
struct GameSetupView: View {
   @ObservedObject var viewModel: EmojiMemoryGame

   @State private var selectedTheme: Theme = Theme.allThemes[0]
   @State private var selectedDifficulty: Difficulty = .Easy
   @State private var navigateToGame = false
   @State private var showInstructions = false

   var body: some View {

      NavigationStack {
         VStack(spacing: 40) {
            Text("Memorize!")
               .font(.system(size: 60, weight: .heavy))
               .foregroundColor(themeColor)
               .padding(.top, 50)
            VStack(alignment: .leading) {
               Text("Select Theme").font(.headline).foregroundColor(.secondary)
               Picker("Theme", selection: $selectedTheme) {
                  ForEach(Theme.allThemes) { theme in
                     Text(theme.name).tag(theme)
                  }
               }
               .pickerStyle(.segmented)
            }
            .padding(.horizontal)
            VStack(alignment: .leading) {
               Text("Select Difficulty").font(.headline).foregroundColor(.secondary)
               Picker("Difficulty", selection: $selectedDifficulty) {
                  ForEach(Difficulty.allCases) { diff in
                     Text(diff.rawValue).tag(diff)
                  }
               }
               .pickerStyle(.segmented)
            }
            .padding(.horizontal)
            Spacer()
            VStack(spacing: 5) {
                        Text("Best Score")
                           .font(.headline)
                           .foregroundColor(.secondary)

                        Text("\(currentSelectionHighScore)")
                           .font(.system(size: 36, weight: .bold, design: .rounded))
                           .foregroundColor(themeColor)                        }
                     .padding(.bottom, 20)
```

```
            Spacer()
            Button {
                viewModel.startNewGame(using: selectedTheme, difficulty:
selectedDifficulty)

                DispatchQueue.main.async {
                    navigateToGame = true
                }
            } label: {
                Text("PLAY GAME")
                    .font(.title2)
                    .bold()
                    .frame(maxWidth: .infinity)
                    .padding()
                    .background(themeColor)
                    .foregroundColor(.white)
                    .cornerRadius(15)
                    .padding(.horizontal)
                    .padding(.bottom, 30)
            }
            .navigationDestination(isPresented: $navigateToGame) {
                EmojiMemoryGameView(viewModel: viewModel)
            }
        }.toolbar {
            ToolbarItem(placement: .topBarTrailing) {
                Button {
                    showInstructions = true
                } label: {
                    Image(systemName: "info.circle")
                        .font(.title3)
                }
            }
        }.foregroundColor(themeColor)
        // 2. Tells the app to slide the InstructionsView up when the button is tapped!
        .sheet(isPresented: $showInstructions) {
            InstructionsView()
        }
    }
}


private var themeColor: Color {
    switch selectedTheme.color {
    case "orange": return .orange
    case "green": return .green
    case "blue": return .blue
    default: return .purple
    }
}
```

```
    private var currentSelectionHighScore: Int {
        let key = "highscore_\(selectedTheme.name)_\(selectedDifficulty.rawValue)"
        return UserDefaults.standard.integer(forKey: key)
    }
}
```

Ο παραπάνω κώδικας αντιστοιχεί στην αρχική οθόνη επιλογής. Τα selectedTheme και selectedDifficulty είναι @State properties: κάθε φορά που αλλάζουν, το SwiftUI ανακατασκευάζει αυτόματα το View και υπολογίζεται εκ νέου το currentSelectionHighScore. Το Picker με .segmented style δημιουργεί το γνωστό segmented control του iOS. Το computed property currentSelectionHighScore διαβάζει από UserDefaults χρησιμοποιώντας σύνθετο κλειδί που συνδυάζει θέμα και δυσκολία.



Εικόνα 5 — Αρχική οθόνη GameSetupView

3.2.2 Κύρια Οθόνη — EmojiMemoryGameView

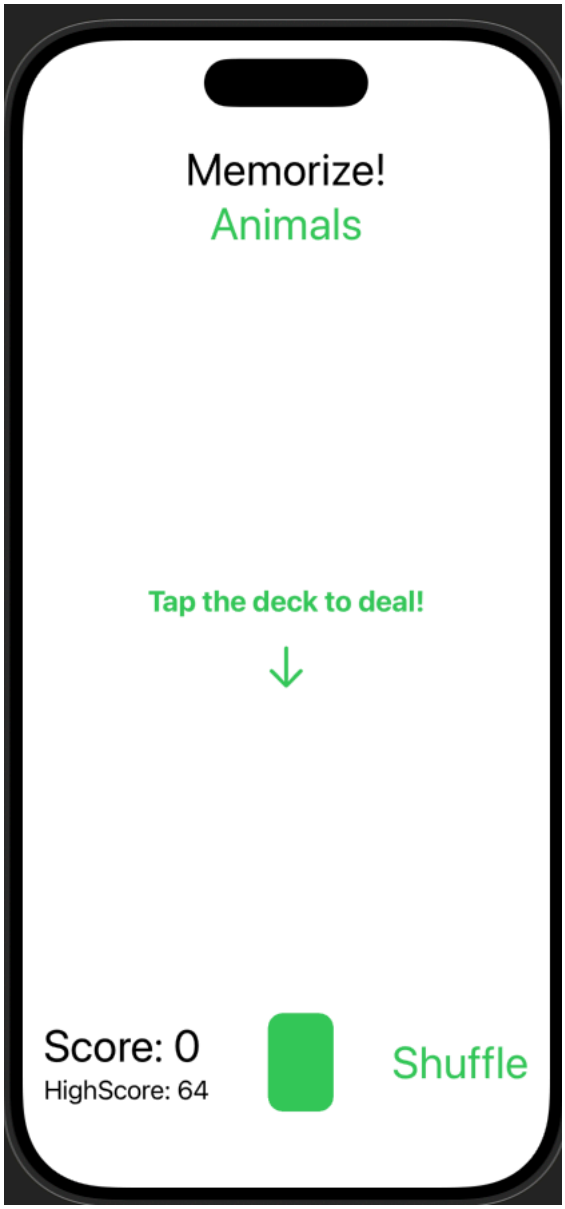
Μετά από εκκίνηση παιχνιδιού ο παίκτης μεταφέρεται στο EmojiMemoryGameView. Αυτό εμφανίζει τον τίτλο, το θέμα, το grid καρτών ή τη στοίβα, τη βαθμολογία, τον high score και τα κουμπιά Shuffle/Deck. Διατηρεί @State dealt Set<Card.ID> και @Namespace dealingNamespace.

Κομμάτι κώδικα — body και score display:

```
 var body: some View {
       VStack{
          Text("Memorize!").font(.largeTitle)
          HStack{
             Text(viewModel.currentTheme.name).font(.largeTitle).foregroundColor(viewModel
.themeColor)

          }
          if viewModel.isGameOver{
             Spacer()
             gameOverScreen
             Spacer()
             Spacer()
          }else{
             ZStack {
                    cards
                if dealt.isEmpty {
                   VStack {
                      Text("Tap the deck to deal!")
                         .font(.title2)
                         .bold()
                         .foregroundColor(viewModel.themeColor)


                      Image(systemName: "arrow.down")
                         .font(.largeTitle)
                         .foregroundColor(viewModel.themeColor)
                         .padding(.top, 5)
                   }
                   .animation(nil,value: dealt.isEmpty)
                }
             }

              HStack{
                score
                Spacer()
                deck.foregroundColor(viewModel.themeColor)
                Spacer()
                 shuffle.foregroundColor(viewModel.themeColor)
             }.font(.largeTitle)
             HStack{
                Spacer()
             }.font(.headline)
          }
       }
       .padding()
       }

    private var score: some View {
       VStack(alignment: .leading){
```

```
            Text("Score: \(viewModel.score)").animation(nil)
            Text("HighScore: \(viewModel.bestScore)").font(.title3)
        }
    }
```

Ο παραπάνω κώδικας αντιστοιχεί στην κύρια οθόνη παιχνιδιού. Το viewModel είναι @ObservedObject, οπότε κάθε φορά που αλλάζει κάποιο @Published property (score, bestScore, cards, isGameOver), το View ανακατασκευάζεται αυτόματα. Το .animation(nil) στο score αποτρέπει ανεπιθύμητο animation κατά την αλλαγή αριθμού. Το dealt Set ελέγχει ποιες κάρτες είναι ορατές στο grid.



Εικόνα 6 — Κύρια οθόνη πριν το dealing

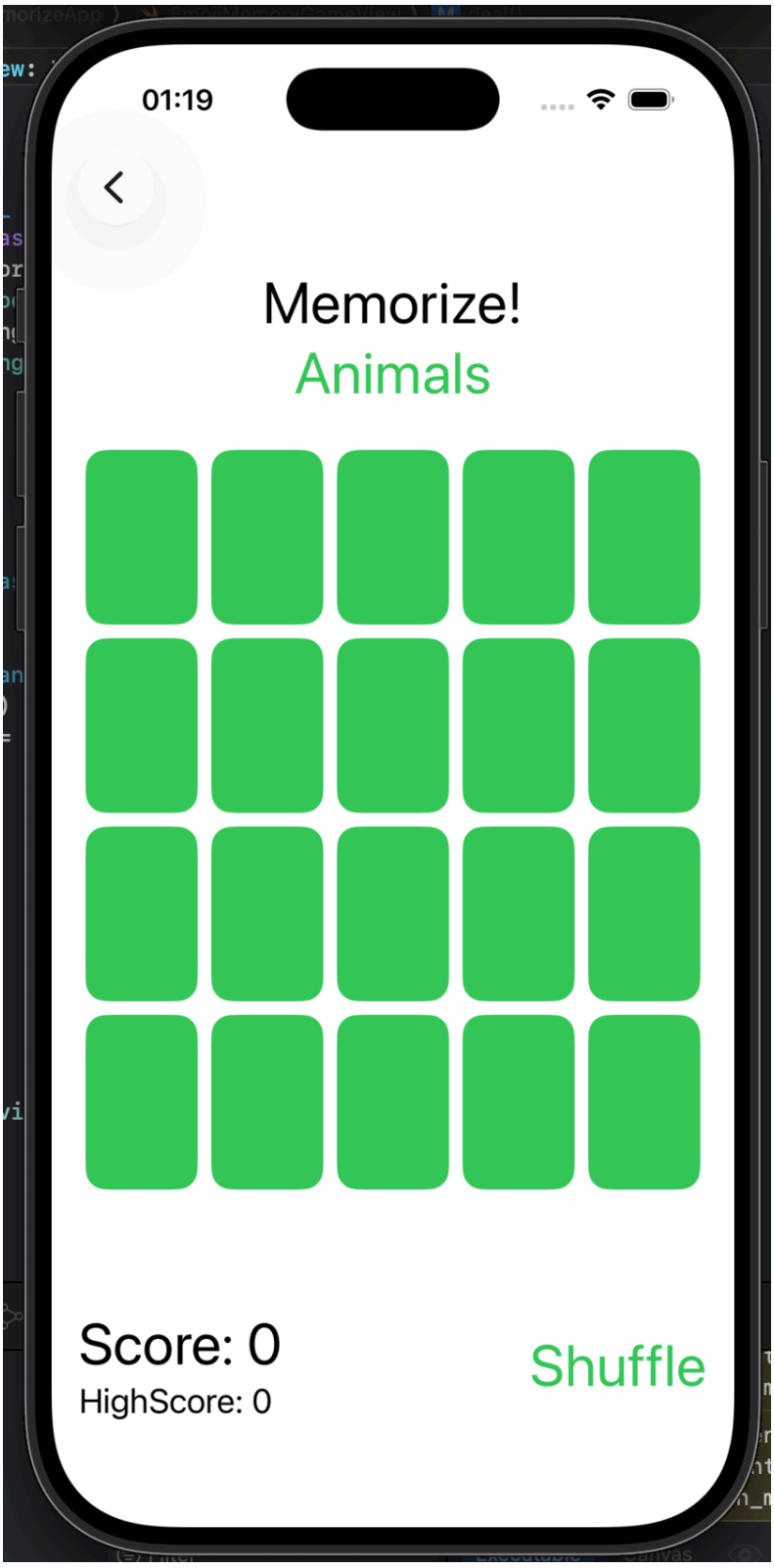
3.2.3 Deal Animation — matchedGeometryEffect

Η διανομή καρτών από τη στοίβα στο grid υλοποιείται μέσω matchedGeometryEffect, μια ισχυρή δυνατότητα του SwiftUI που δημιουργεί ομαλές μεταβάσεις μεταξύ Views με το ίδιο id σε ένα @Namespace. Το effect υπολογίζει αυτόματα τη διαφορά θέσης και μεγέθους μεταξύ της στοίβας και της θέσης στο grid και δημιουργεί animation μεταφοράς.

Κομμάτι κώδικα — deck View και deal function:

```
private var deck: some View {
    ZStack {
        ForEach(undealtCards) { card in
            CardView(card)
                .matchedGeometryEffect(id: card.id, in: dealingNamespace)
                .transition(.asymmetric(insertion: .identity, removal: .identity))


        }
    }
    .frame(width: deckWidth, height: deckWidth / aspectRatio)
    .onTapGesture{
        shuffleCards()
        deal()
    }
}

private func deal(){
    var delay: TimeInterval = 0
    for card in viewModel.cards {
        withAnimation(.easeInOut(duration: 1).delay((delay))) {
            _ = dealt.insert(card.id)
        }
        delay += 0.15

    }

}

private func shuffleCards() {
    viewModel.shuffle()
    dealt.removeAll()
}
```

Ο παραπάνω κώδικας υλοποιεί το deal animation. Η στοίβα (deck) και το grid (cards) χρησιμοποιούν το ίδιο @Namespace dealingNamespace, οπότε το SwiftUI 'ξέρει' ότι το CardView με id X στη στοίβα και το CardView με id X στο grid αναπαριστούν το ίδιο στοιχείο. Έτσι δημιουργείται αυτόματα animation μεταφοράς. Το sequential delay 0.15 δευτ. ανά κάρτα δημιουργεί το εφέ 'πεταλούδας'. Κρίσιμη λεπτομέρεια: η shuffleCards() πρέπει να καλεί shuffle() ΠΡΩΤΑ και μετά removeAll(), αλλιώς το matchedGeometryEffect δημιουργεί glitches.


01:19
Memorize!
Animals
Score: 0
HighScore: 0
Shuffle

Εικόνα 7 — Κύρια οθόνη μετά το dealing

3.2.4 Εμφάνιση Κάρτας — CardView

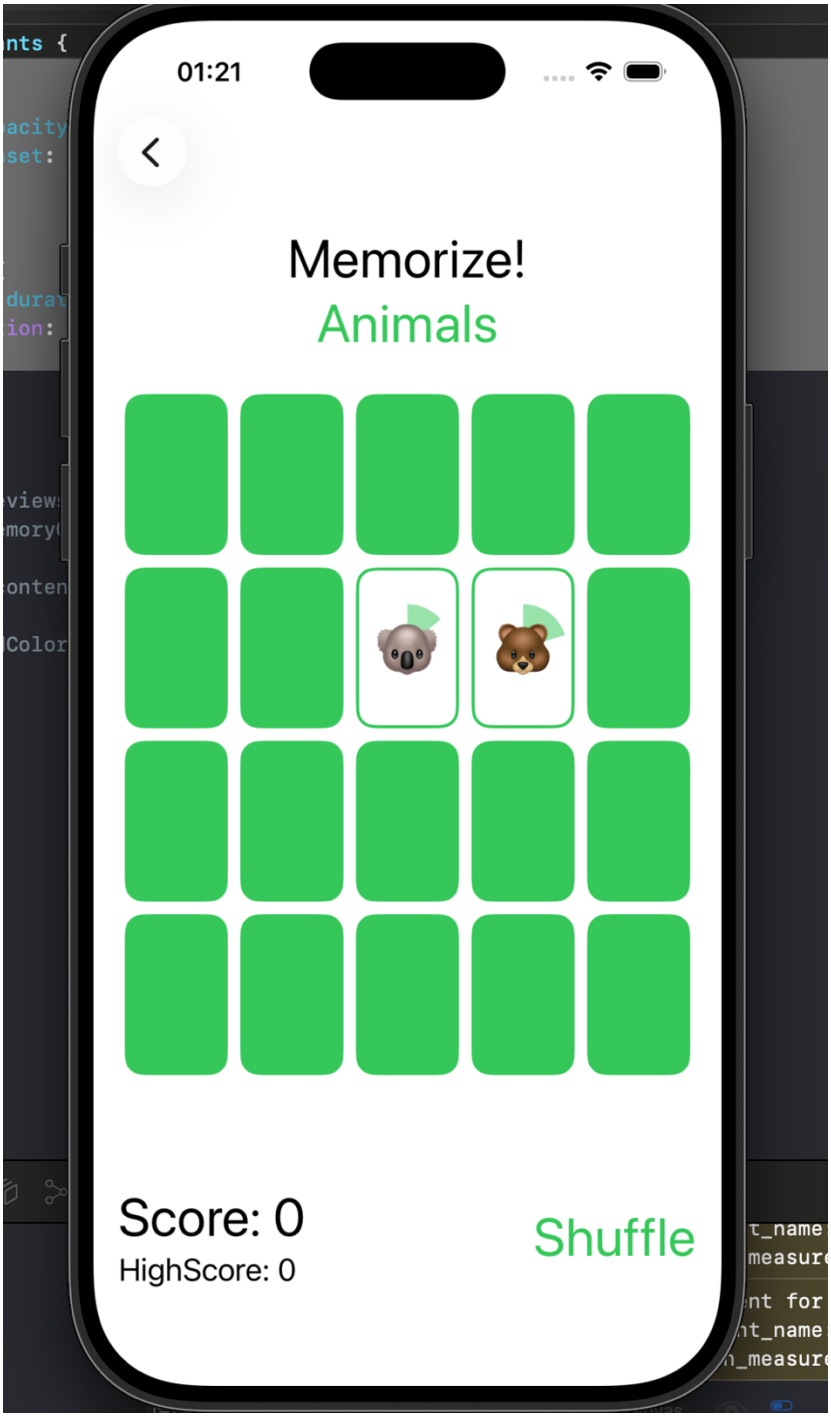
Κάθε κάρτα εμφανίζεται μέσω του CardView που χρησιμοποιεί TimelineView(.animation()) για real-time ενημέρωση του bonus timer (Pie shape) σε κάθε frame animation. Το CardView ελέγχει αν η κάρτα πρέπει να εμφανιστεί (face up ή unmatched) και εφαρμόζει το Cardify modifier για το 3D flip.

Κομμάτι κώδικα από το CardView:

```
struct CardView: View {
   let card: MemoryGame<String>.Card

   init(_ card: MemoryGame<String>.Card) {
      self.card = card
   }

    var body: some View {
       TimelineView(.animation()){ timeline in
          if card.isFaceUp || !card.isMatched {
             Pie(endAngle: .degrees(card.bonusPercentRemaining * 360))
                .opacity(Constants.Pie.opacity)
                .overlay(cardContents.padding(Constants.Pie.inset))
                .padding(Constants.inset)
                .cardify(isFaceUp: card.isFaceUp)
                .transition(.scale)
          } else {
             Color.clear
          }
       }
   }

   var cardContents: some View {
      Text(card.content)
         .font(.system(size: Constants.FontSize.large))
         .minimumScaleFactor(Constants.FontSize.scaleFactor)
         .aspectRatio(1,contentMode: .fit)
         .rotationEffect(.degrees(card.isMatched ? 360 : 0))
         .animation(.spin(duration: 0.8), value: card.isMatched)
   }
}
private struct Constants {
   static let cornerRadius: CGFloat = 12
   static let lineWidth: CGFloat = 2
   static let inset: CGFloat = 5
```

```
    struct FontSize {
        static let large: CGFloat = 200
        static let small: CGFloat = 10
        static let scaleFactor = small / large
    }
    struct Pie {
        static let opacity: CGFloat = 0.5
        static let inset: CGFloat = 5
    }
}
extension Animation {
    static func spin(duration: TimeInterval) -> Animation {
        .linear(duration: 0.8).repeatForever(autoreverses: false)
    }
}
```

Ο παραπάνω κώδικας αντιστοιχεί στην εμφάνιση κάθε κάρτας. Το TimelineView(.animation()) εξασφαλίζει ότι το Pie shape ενημερώνεται σε κάθε frame animation, οπτικοποιώντας τη συρρίκνωση του bonus timer. Το rotationEffect με .spin animation εκτελεί συνεχή περιστροφή 360° όταν η κάρτα αντιστοιχιστεί. Το minimumScaleFactor επιτρέπει συρρίκνωση του emoji ώστε να χωρά στην κάρτα ανεξάρτητα από το μέγεθος.



Εικόνα 8 — Ανοιχτές κάρτες με bonus timer (Pie shape)

3.2.5 3D Flip Animation — Cardify ViewModifier

Το Cardify struct υλοποιεί ταυτόχρονα ViewModifier (εφαρμογή ως modifier) και Animatable (παρεμβολή τιμών κατά animation). Ο συνδυασμός αυτός επιτρέπει στο SwiftUI

να παρεμβάλει τη γωνία περιστροφής αυτόματα κατά τη διάρκεια του flip, δημιουργώντας ομαλό 3D animation.

Κομμάτι κώδικα από το Cardify:

```
struct Cardify: ViewModifier, Animatable{
   init (isFaceUp: Bool = false) {
      rotation = isFaceUp ? 0 : 180
   }

   var isFaceUp: Bool {
      rotation < 90
   }

   var rotation: Double

   var animatableData: Double {
      get { return rotation }
      set { rotation = newValue }

   }

   func body(content: Content) -> some View {
      ZStack {
         let base = RoundedRectangle(cornerRadius: Constants.cornerRadius)
            base.strokeBorder(lineWidth: Constants.lineWidth).background(base.fill(.white))
               .overlay(content)
               .opacity(isFaceUp ? 1 : 0)
            base.fill()
               .opacity(isFaceUp ? 0 : 1)
      }
      .rotation3DEffect(.degrees(rotation), axis: (x: 0, y: 1, z: 0))
   }
   private struct Constants {
      static let cornerRadius: CGFloat = 12
      static let lineWidth: CGFloat = 2
   }
}
extension View {
    func cardify(isFaceUp: Bool) -> some View {
       return self.modifier(Cardify(isFaceUp: isFaceUp))
   }
}
```

Ο παραπάνω κώδικας αντιστοιχεί στο 3D flip animation. Το animatableData property είναι η γέφυρα μεταξύ SwiftUI animation system και της ιδιότητας rotation: όταν η rotation αλλάζει από 180 σε 0 (face up), το SwiftUI παρεμβάλει αυτόματα ενδιάμεσες τιμές (180, 162, 144... 0) δημιουργώντας ομαλό animation. Η εναλλαγή opacity (isFaceUp ? 1:0) για τις δύο όψεις εξασφαλίζει εμφάνιση της σωστής πλευράς ανά πάσα στιγμή. Η extension cardify(isFaceUp:) επιτρέπει απλή χρήση: .cardify(isFaceUp: card.isFaceUp).

3.2.6 Custom Shape — Pie

Το Pie struct υλοποιεί το Shape protocol χρησιμοποιώντας Core Graphics (CGPath) μέσω του Path API του SwiftUI. Σχεδιάζει ένα γεμιστό κυκλικό τόξο που ξεκινά από τη 12 ώρα (μετατόπιση -90°) και καταλαμβάνει γωνία ανάλογη του bonusPercentRemaining, συρρικνούμενο καθώς περνά ο χρόνος.

Κομμάτι κώδικα από το Pie Shape:

```
struct Pie: Shape {
   var startAngle: Angle = .zero
   let endAngle: Angle
   let clockwise: Bool = true


   func path(in rect: CGRect) -> Path {
      let startAngle = startAngle - .degrees(90)
      let endAngle = endAngle - .degrees(90)

      let center = CGPoint(x: rect.midX, y: rect.midY)
      let radius = min(rect.width, rect.height) / 2
      let start = CGPoint(
         x:center.x + radius * cos(startAngle.radians),
         y:center.y + radius * sin(startAngle.radians)
      )
      var p = Path()
      p.move(to: center)
      p.addLine(to: start)
      p.addArc(center: center, radius: radius, startAngle: startAngle, endAngle:
endAngle, clockwise: !clockwise)
      p.addLine(to: center)
      return p
   }
```

```
}
```

Ο παραπάνω κώδικας δημιουργεί το custom Pie shape. Η μετατόπιση -90° ευθύνεται για την εκκίνηση από τη 12 ώρα: στο σύστημα συντεταγμένων iOS, γωνία 0° αντιστοιχεί στα δεξιά (3 ώρα), οπότε -90° = 12 ώρα. Η endAngle υπολογίζεται ως card.bonusPercentRemaining × 360°: bonusPercentRemaining = 1.0 σημαίνει πλήρης κύκλος (νέα κάρτα), 0.0 σημαίνει καμία γωνία (παρέλυσε ο χρόνος). Η TimelineView στο CardView εξασφαλίζει ότι το Pie επανυπολογίζεται σε κάθε frame.

3.2.7 Adaptive Layout — AspectVGrid

Το AspectVGrid είναι ένα generic View που υπολογίζει αυτόματα το βέλτιστο μέγεθος κελιών για LazyVGrid χρησιμοποιώντας GeometryReader. Εξασφαλίζει ότι ανεξάρτητα από τον αριθμό καρτών (6-32) και τη διαγώνιο της οθόνης, όλες οι κάρτες χωράνε πάντα ορατές χωρίς scroll.

Κομμάτι κώδικα από το AspectVGrid:

```
struct AspectVGrid<Items: Identifiable, ItemView : View>: View{
   var items: [Items]
   var aspectRatio: CGFloat = 1
   @ViewBuilder var content: (Items) -> ItemView

   var body: some View {
      GeometryReader { geometry in
         let gridItemSize = max(10, gridItemWidthThatFits(
            count: items.count,
            size: geometry.size,
            atAspectRatio: aspectRatio
         ))

         LazyVGrid(columns: [GridItem(.adaptive(minimum: gridItemSize),spacing:
0)],spacing: 0) {
            ForEach(items) { item in
               content(item)
                  .aspectRatio(aspectRatio, contentMode:.fit)
            }
         }
```

```
        }
    }
    func gridItemWidthThatFits(
        count: Int,
        size: CGSize,
        atAspectRatio aspectRatio: CGFloat
    ) -> CGFloat {
        let count = CGFloat(count)
        var columnCoutn = 1.0
        repeat {
            let width = size.width / columnCoutn
            let height = width / aspectRatio

            let rowCount = (count / columnCoutn).rounded(.up)
            if rowCount * height < size.height {
                return (size.width / columnCoutn).rounded(.down)
            }
            columnCoutn += 1
        }while columnCoutn < count

        return min(size.width / count, size.height * aspectRatio).rounded(.down)

    }
}
```

Ο παραπάνω αλγόριθμος ξεκινά με 1 στήλη και αυξάνει διαδοχικά: για κάθε αριθμό στηλών υπολογίζει το πλάτος κελιού, στη συνέχεια το ύψος βάσει aspect ratio, και ελέγχει αν το συνολικό ύψος όλων των γραμμών χωράει στο διαθέσιμο ύψος. Μόλις χωράει, επιστρέφει αυτό το πλάτος. Το max(10, ...) εξασφαλίζει ελάχιστο πλάτος 10pt. Το .adaptive(minimum: gridItemSize) στο GridItem επιτρέπει στο LazyVGrid να τοποθετεί όσες στήλες χωράνε.

3.2.8 FlyingNumber — Animation Βαθμολογίας

Το FlyingNumber View εμφανίζει animated αριθμητική αλλαγή βαθμολογίας πάνω από κάθε κάρτα. Χρησιμοποιεί @State offset: CGFloat για animation κίνησης και ταυτόχρονη μείωση opacity, δημιουργώντας φυσικό εφέ εξαφάνισης.

Κομμάτι κώδικα από το FlyingNumber:

```
struct FlyingNumber: View {
   let number: Int

   // - is up in y axis
   @State private var offset: CGFloat = 0

   var body: some View {
      if number != 0 {
         Text(number, format: .number.sign(strategy: .always()))
            .font(.largeTitle)
            .foregroundColor(number < 0 ? .red : .orange)
            .shadow(color:.black, radius: 1.5, x: 1,y: 1)
            .offset(x: 0 , y: offset)
            .opacity(offset != 0 ? 0 : 1)
            .onAppear {
               withAnimation(.easeIn(duration: 1.2)) {
                  offset = number < 0 ? 200 : -200


               }
            }
            .onDisappear {
               offset = 0
            }
      }
   }
}
```

Ο παραπάνω κώδικας εμφανίζει animated αριθμό βαθμολογίας. Η μορφοποίηση .number.sign(strategy: .always()) εμφανίζει πάντα το πρόσημο (+2, -1 κ.λπ.). Το opacity=0 κατά την κίνηση (offset != 0) δημιουργεί ομαλό εφέ εξαφάνισης. Το .onDisappear μηδενίζει το offset για επαναχρησιμοποίηση. Στο EmojiMemoryGameView, το scoreChange(causeBy: card) επιστρέφει την αλλαγή βαθμολογίας μόνο για την κάρτα που προκάλεσε την αλλαγή.

3.2.9 Αλγόριθμος Επιλογής Κάρτας — MemoryGame Model

Ο αλγόριθμος choose() αποτελεί τον πυρήνα της λογικής του παιχνιδιού. Υλοποιείται εξ ολοκλήρου στο Model, χωρίς γνώση του UI. Η computed property indexOfTheOneAndOnlyFaceUpCard διαχειρίζεται τη μοναδική ανοιχτή κάρτα μέσω get/set.

Κομμάτι κώδικα — αλγόριθμος choose() και indexOfTheOneAndOnlyFaceUpCard:

```
var indexOfTheOneAndOnlyFaceUpCard: Int?{
      get{
         return cards.indices.filter { index in cards[index].isFaceUp}.only
      }
      set{
         cards.indices.forEach {cards[$0].isFaceUp = (newValue == $0)}
      }
   }

   mutating func choose(card: Card){
      if let chosenIndex = index(of: card){
         if !cards[chosenIndex].isFaceUp && !cards[chosenIndex].isMatched{
            if let potentialMatchIndex = indexOfTheOneAndOnlyFaceUpCard{
               if cards[chosenIndex].content == cards[potentialMatchIndex].content{
                  cards[chosenIndex].isMatched = true
                  cards[potentialMatchIndex].isMatched = true
                  score += 2 + cards[chosenIndex].bonus +
                  cards[potentialMatchIndex].bonus
               }else {
                  if cards[chosenIndex].hasBeenSeen{
                     score -= 1
                  }
                  if cards[potentialMatchIndex].hasBeenSeen{
                     score -= 1
                  }
               }
            }else {
               indexOfTheOneAndOnlyFaceUpCard = chosenIndex
            }
            cards[chosenIndex].isFaceUp = true
         }
      }
   }
```

Ο παραπάνω κώδικας αντιστοιχεί στον κεντρικό αλγόριθμο. Η indexOfTheOneAndOnlyFaceUpCard στο set γυρίζει ανάποδα όλες τις κάρτες εκτός από αυτή του νέου index, υλοποιώντας τον κανόνα 'μόνο μία ανοιχτή κάρτα τη φορά'. Η χρήση mutating func υπενθυμίζει ότι το MemoryGame είναι struct (value type): κάθε αλλαγή

δημιουργεί νέο αντίγραφο, εξασφαλίζοντας immutability. Το hasBeenSeen γίνεται true μόλις μια κάρτα γυρίσει ανάποδα μετά από πρώτη εμφάνιση, μέσω του didSet του isFaceUp.

3.2.10 Διαχείριση High Score — EmojiMemoryGame ViewModel

Η αποθήκευση και ανάκτηση υψηλής βαθμολογίας διαχειρίζεται αποκλειστικά από το ViewModel (EmojiMemoryGame.swift). Χρησιμοποιούνται σύνθετα κλειδιά UserDefaults για ξεχωριστή αποθήκευση ανά θέμα και δυσκολία. Η ενημέρωση γίνεται σε πραγματικό χρόνο σε κάθε choose() call.

Κομμάτι κώδικα — high score management στο ViewModel:

```
 @Published private var game: MemoryGame<String>

   init() {
      let startingTheme = Theme.allThemes.randomElement() ?? Theme.allThemes[0]
      self.currentTheme = startingTheme
      self.currentDifficulty = .Easy
      self.game = EmojiMemoryGame.createMemoryGame(theme: startingTheme, difficulty:
.Easy)
      self.loadHighScore()
   }

   var cards: Array<MemoryGame<String>.Card> {
       game.cards
   }

   var score: Int {
      game.score
   }
   var isGameOver: Bool {
      !cards.isEmpty && cards.allSatisfy{$0.isMatched}
   }


   private var currentHighScoreKey: String {
      "highscore_\(currentTheme.name)_\(currentDifficulty.rawValue)"
   }
   private func loadHighScore() {
      bestScore = UserDefaults.standard.integer(forKey: currentHighScoreKey)
   }
```

Ο παραπάνω κώδικας διαχειρίζεται το high score. Το currentHighScoreKey είναι computed property που δημιουργεί μοναδικό κλειδί ανά συνδυασμό (π.χ. 'highscore_Animals_Easy',

'highscore_Sports_Hard'). Η ενημέρωση γίνεται μέσα στη choose() κάθε φορά που επιλέγεται κάρτα: αν score > bestScore, αποθηκεύεται άμεσα. Αυτό εξασφαλίζει ότι ακόμα και αν ο χρήστης τερματίσει την εφαρμογή πριν τελειώσει το παιχνίδι, το καλύτερο επίτευγμά του σώζεται. Η loadHighScore() καλείται κατά την εκκίνηση νέου παιχνιδιού για να φορτωθεί το αντίστοιχο high score.

ΚΕΦΑΛΑΙΟ 4ο

4 Εγχειρίδιο Χρήσης της Εφαρμογής

Στο κεφάλαιο αυτό παρουσιάζεται το εγχειρίδιο χρήσης της iOS εφαρμογής MemorizeApp. Το εγχειρίδιο χρήσης είναι απαραίτητο στην ανάλυση της εφαρμογής, καθώς περιγράφει αναλυτικά με βήματα και στιγμιότυπα οθόνης τον τρόπο που μπορεί να χρησιμοποιηθεί από τους χρήστες, διευκολύνοντάς τους ακόμα και αν δεν έχουν προηγούμενη εμπειρία με αντίστοιχες εφαρμογές. Επιπλέον, αναλύονται τα βήματα εγκατάστασης.

4.1 Εγκατάσταση της Εφαρμογής

Η εφαρμογή αναπτύχθηκε με Xcode (έκδοση 16.x) και απαιτεί συσκευή iOS 17.0+. Διατίθεται μέσω του project φακέλου στο OneDrive. Στο μέλλον η διάθεση θα μπορούσε να πραγματοποιηθεί μέσω του Apple App Store, ώστε να διασφαλίζεται ευκολότερη πρόσβαση και αυτόματη ενημέρωση.

Ο χρήστης έχει δύο επιλογές εγκατάστασης:

15. Εκτέλεση μέσω Xcode σε iOS Simulator: Άνοιγμα MemorizeApp.xcodeproj και πάτημα ▶️ (Run). Δεν απαιτείται Apple Developer account.
16. Εγκατάσταση σε φυσική συσκευή: Απαιτεί Apple Developer account για ρύθμιση Code Signing.

Για εγκατάσταση σε φυσική συσκευή ακολουθήστε τα εξής βήματα:

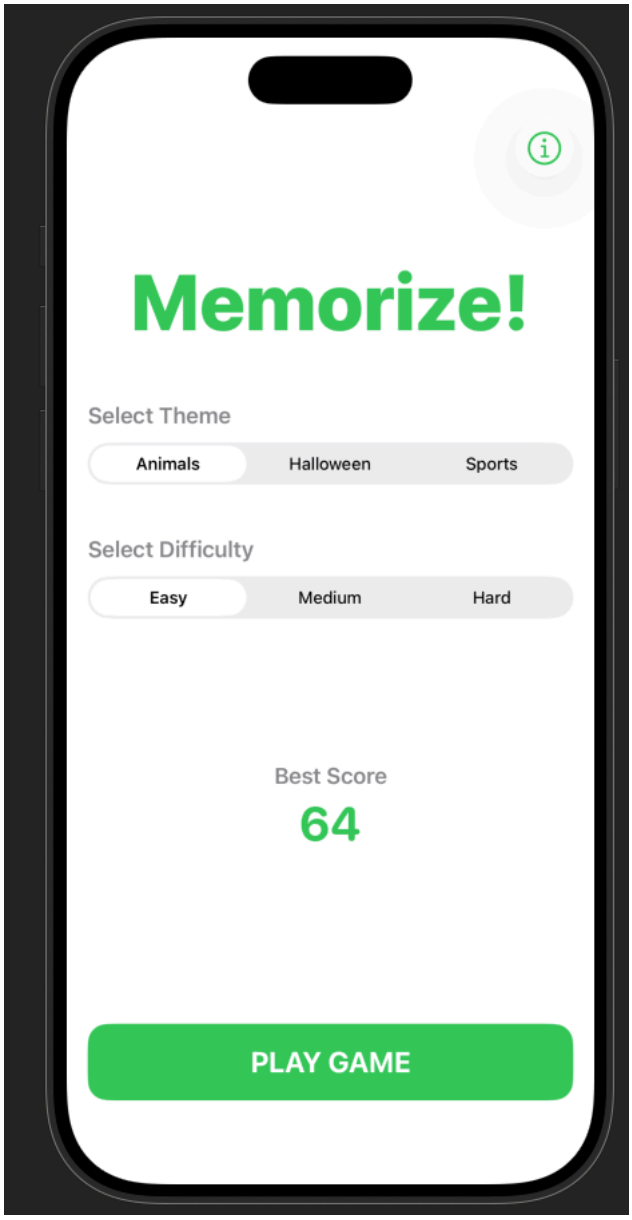
17. Κατεβάστε τον φάκελο project από τον σύνδεσμο OneDrive.
18. Αποσυμπιέστε το αρχείο zip και κάντε διπλό κλικ στο MemorizeApp.xcodeproj.
19. Στο Xcode: κλικ στο project → target MemorizeApp → Signing & Capabilities.
20. Επιλέξτε Team (Apple ID) και αλλάξτε Bundle Identifier σε μοναδικό (π.χ. com.yourname.memorizeapp).
21. Συνδέστε τη συσκευή iOS με USB και πατήστε ▶️ (Run).
22. Στη συσκευή: Ρυθμίσεις → Γενικά → VPN & Διαχείριση Συσκευής → Trust.

23. Η εφαρμογή εμφανίζεται στην αρχική οθόνη και είναι έτοιμη για χρήση.

4.2 Χρήση της Εφαρμογής

Κατά την εκκίνηση της εφαρμογής, η πρώτη οθόνη που βλέπει ο χρήστης είναι η αρχική οθόνη επιλογής GameSetupView.

Αρχικά, κατά την πρώτη χρήση της εφαρμογής η πρώτη οθόνη που θα δει ο χρήστης είναι αυτή:



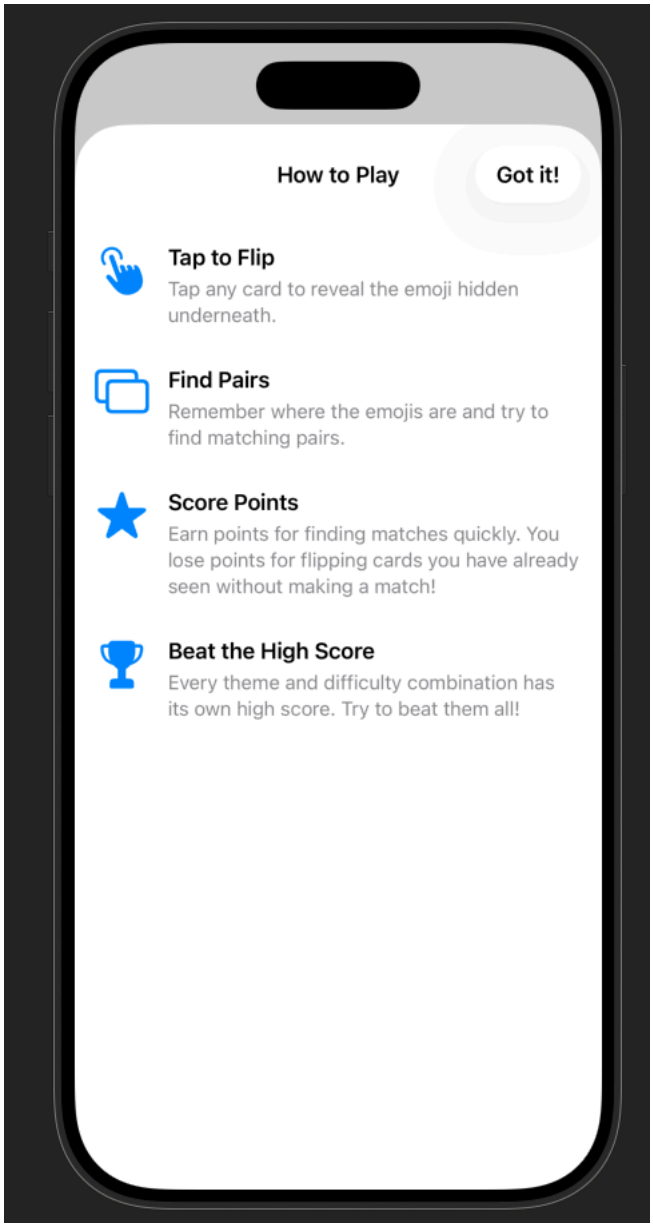
Εικόνα 15 — Αρχική οθόνη κατά την πρώτη εκκίνηση

Στην εικόνα 15, την πρώτη οθόνη της εφαρμογής:

24. Επιλέγει θέμα από τον segmented Picker (Animals / Halloween / Sports). Το χρώμα της οθόνης αλλάζει αμέσως.
25. Επιλέγει δυσκολία (Easy / Medium / Hard). Το Best Score ενημερώνεται αντίστοιχα.

26. Βλέπει το Best Score για τον τρέχοντα συνδυασμό. Κατά την πρώτη χρήση εμφανίζεται 0.
27. Πατά PLAY GAME για να ξεκινήσει νέο παιχνίδι.
28. Εναλλακτικά, πατά το κουμπί ⓘ (info.circle) επάνω δεξιά για να δει τις οδηγίες.

Πατώντας το κουμπί ⓘ εμφανίζεται η οθόνη οδηγιών ως sheet:



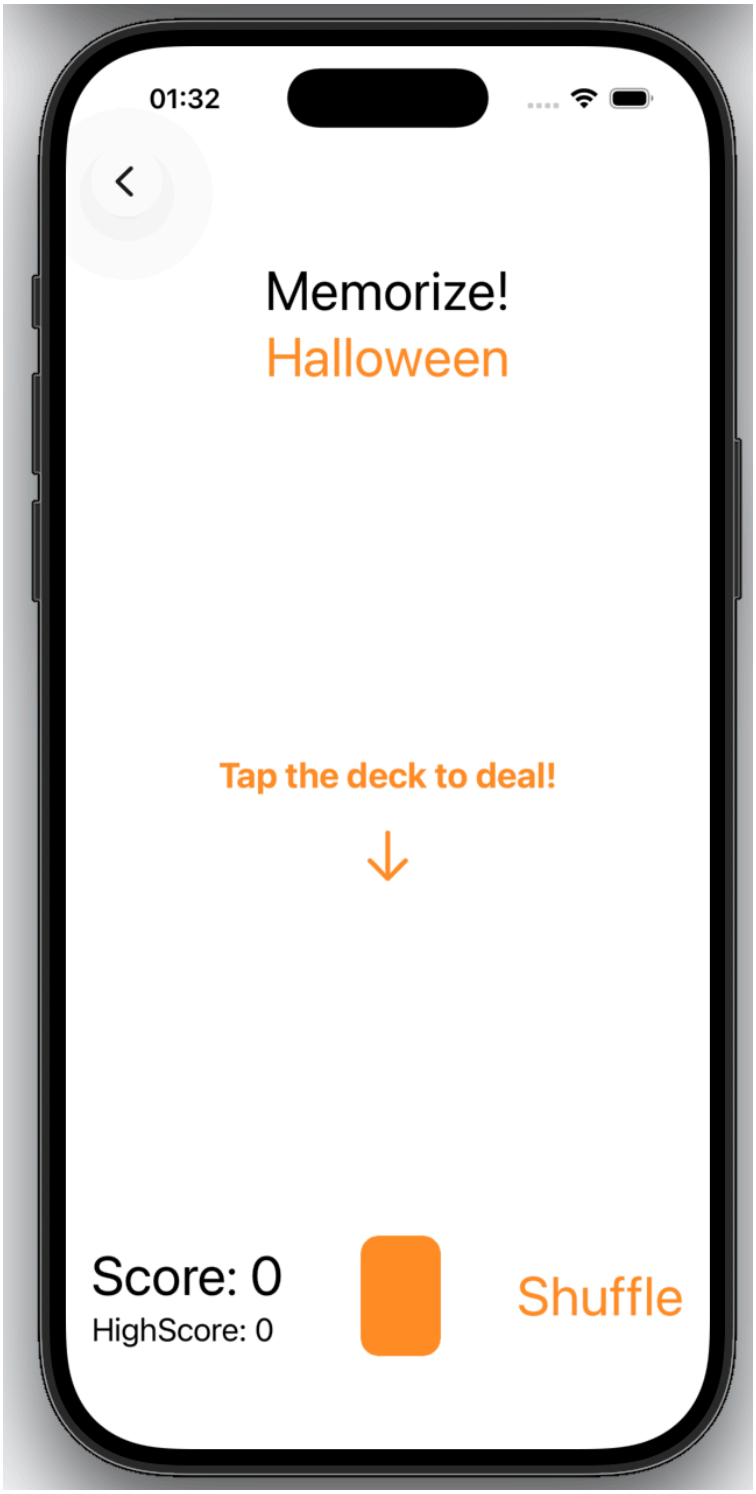
Εικόνα 16 — Οθόνη οδηγιών παιχνιδιού (InstructionsView)

Στην εικόνα 16 φαίνεται η οθόνη οδηγιών παιχνιδιού:

29. Tap to Flip (χέρι με βέλος): Πατάς κάρτα για να αποκαλυφθεί το emoji.
30. Find Pairs (δύο κάρτες): Θυμάσαι τις θέσεις και βρίσκεις ίδια ζεύγη.
31. Score Points (αστέρι): Κερδίζεις βαθμούς για γρήγορες αντιστοιχίσεις. Χάνεις για επαναλήψεις.

32. Beat the High Score (τρόπαιο): Κάθε θέμα και δυσκολία έχουν δικό τους high score.
33. Πατά 'Got it!' για κλείσιμο και επιστροφή στην αρχική οθόνη.

Μετά από επιτυχημένη εκκίνηση παιχνιδιού μεταφέρεται στην κύρια οθόνη:

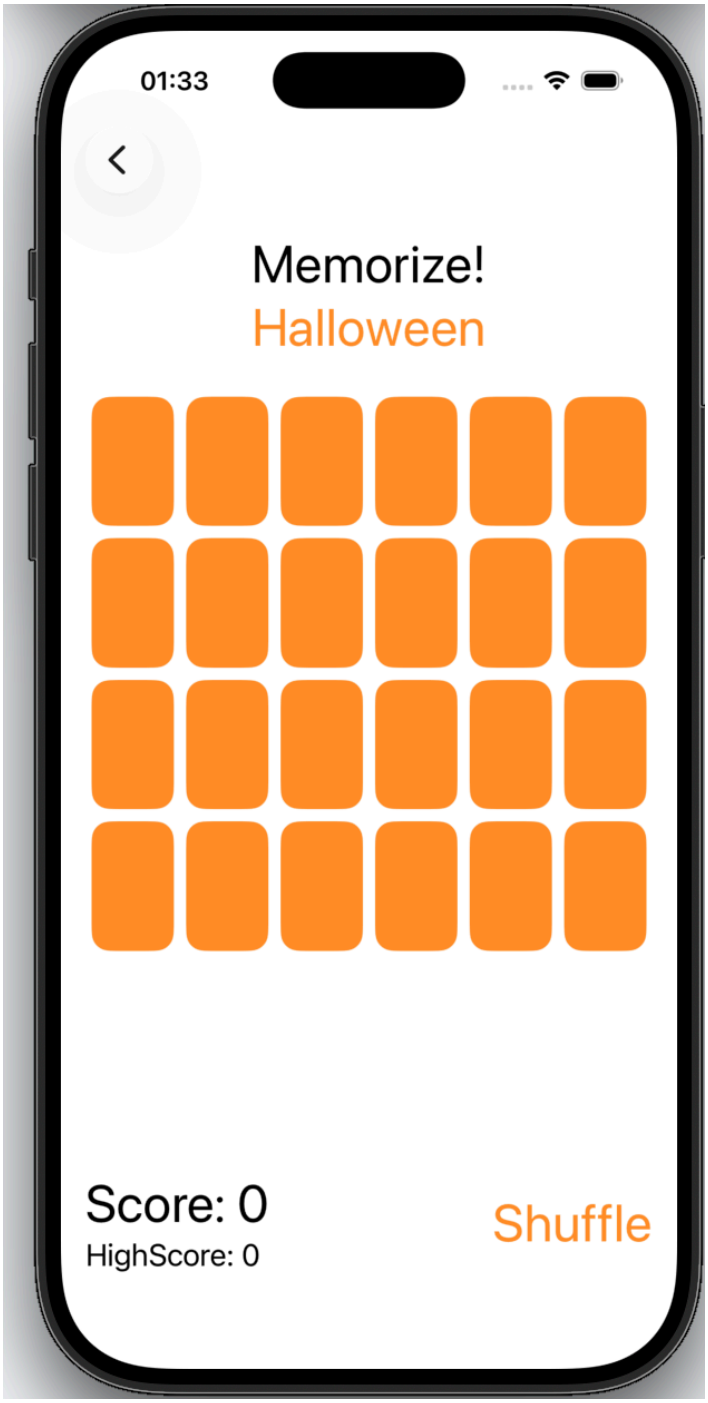


Εικόνα 17 — Κύρια οθόνη πριν τη διανομή καρτών

Στην εικόνα 17 που αποτελεί την κύρια οθόνη πριν τη διανομή:

34. Εμφανίζεται το μήνυμα 'Tap the deck to deal!' με animation βέλους προς τα κάτω.
35. Στα αριστερά εμφανίζεται Score: 0 και HighScore: X.
36. Στη μέση εμφανίζεται η στοίβα καρτών (deck).
37. Στα δεξιά εμφανίζεται το κουμπί Shuffle.
38. Πατώντας τη στοίβα, οι κάρτες διανέμονται με sequential animation.

Μετά από πάτημα της στοίβας, οι κάρτες διανέμονται στο grid:

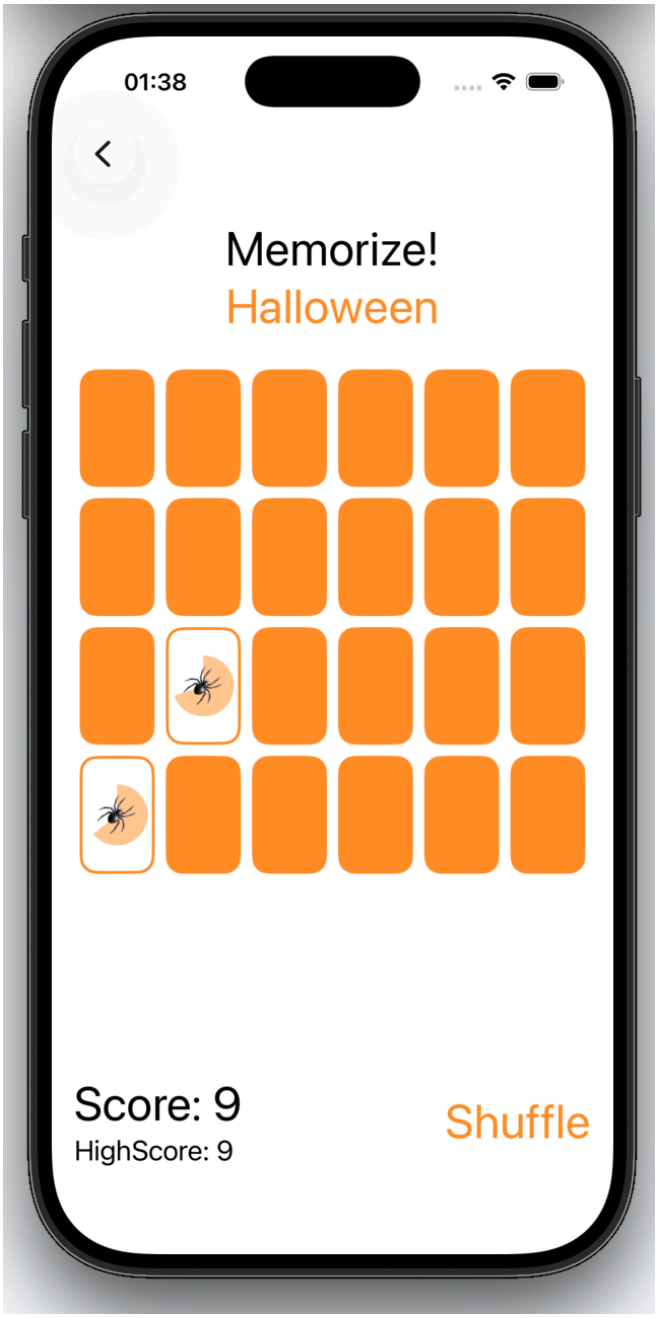


Εικόνα 18 — Κύρια οθόνη μετά τη διανομή καρτών

Στην εικόνα 18 βλέπουμε το grid των καρτών αφού έγινε η διανομή:

39. Οι κάρτες εμφανίζονται ανάποδα στο grid. Το AspectVGrid υπολόγισε αυτόματα βέλτιστο μέγεθος.
40. Πατώντας κάρτα εκτελείται το 3D flip animation και αποκαλύπτεται το emoji.
41. Ο bonus timer (Pie shape) εμφανίζεται ημιδιαφανής πάνω από κάθε ανοιχτή κάρτα και μικραίνει.
42. Η βαθμολογία ενημερώνεται σε πραγματικό χρόνο.

Όταν ο παίκτης βρει επιτυχώς ένα ζεύγος:



Εικόνα 19 — Επιτυχής αντιστοίχιση ζεύγους

Στην εικόνα 19 βλέπουμε την επιτυχή αντιστοίχιση:

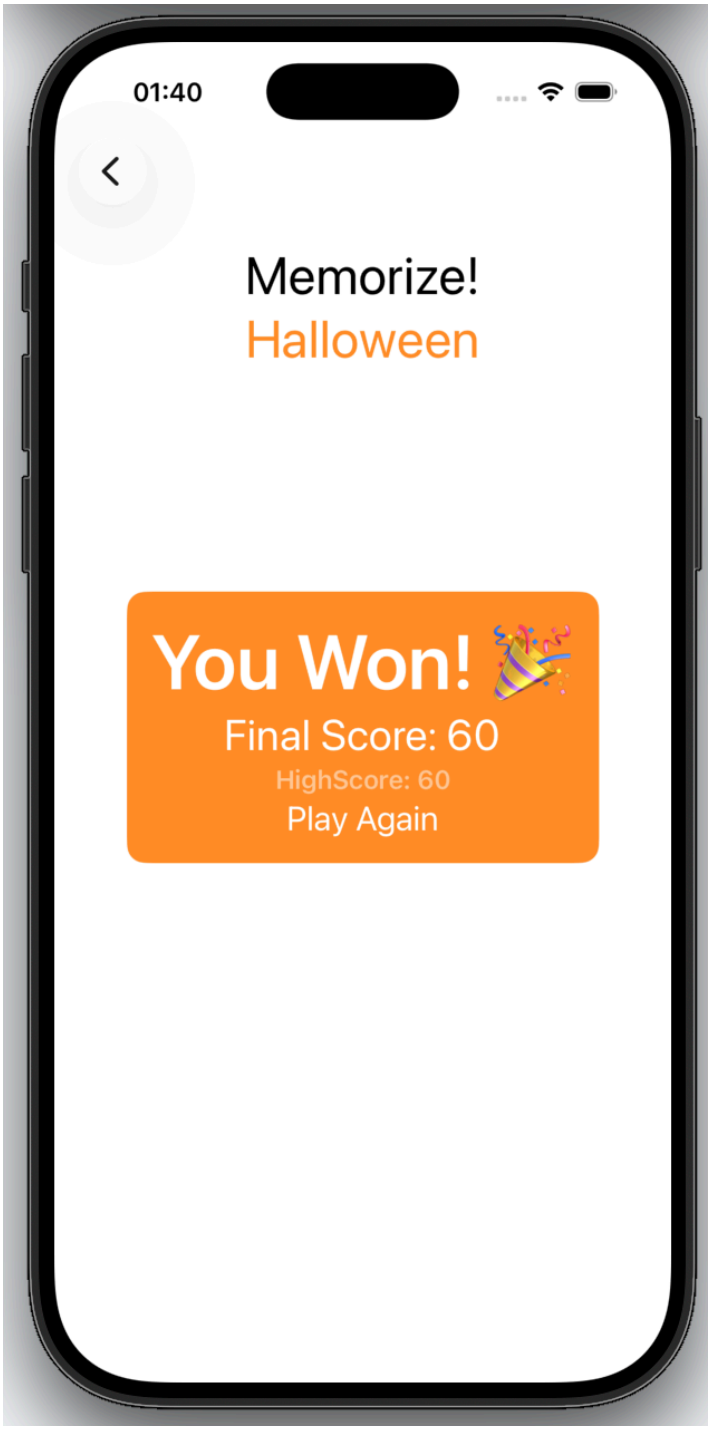
43. FlyingNumber animation πορτοκαλί κινείται προς τα πάνω.

44. Οι αντιστοιχισμένες κάρτες εκτελούν animation περιστροφής 360° και εξαφανίζονται.
45. Η βαθμολογία αυξάνεται αντίστοιχα. Αν ξεπεραστεί ο high score, αποθηκεύεται αμέσως.

Σε περίπτωση αποτυχίας αντιστοίχισης σε γνωστή κάρτα:

46. FlyingNumber animation κόκκινο (-1) κινείται προς τα κάτω για κάθε γνωστή κάρτα.
47. Η βαθμολογία μειώνεται κατά 1 για κάθε γνωστή κάρτα.
48. Και οι δύο κάρτες γυρίζουν ανάποδα.

Όταν όλες οι κάρτες αντιστοιχιστούν, εμφανίζεται η οθόνη ολοκλήρωσης:



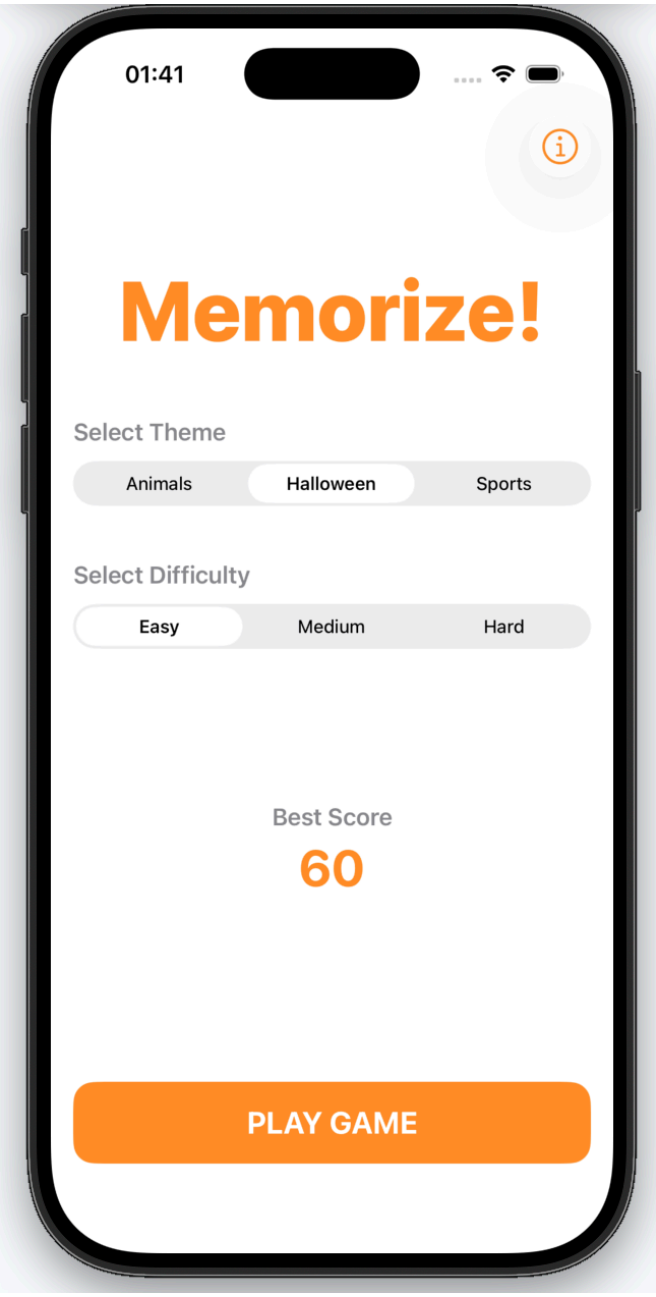
Εικόνα 21 — Οθόνη Game Over

Στην οθόνη αυτή (εικόνα 21) εμφανίζεται το game over screen:

49. Εμφανίζεται το μήνυμα 'You Won! 🎉' σε μεγάλο μέγεθος.

50. Εμφανίζεται η Final Score (τελική βαθμολογία της παρτίδας).
51. Εμφανίζεται ο HighScore (υψηλή βαθμολογία αποθηκευμένη στο UserDefaults).
52. Πατώντας 'Play Again' επιστρέφει στην αρχική οθόνη.

Αφού ολοκληρωθεί μια παρτίδα, ο παίκτης παρατηρεί ενημερωμένο high score στην αρχική οθόνη:



Εικόνα 22 — Αρχική οθόνη με ενημερωμένο Best Score

Στην εικόνα 22 φαίνεται η ενημερωμένη αρχική οθόνη μετά παρτίδα:

53. Το Best Score εμφανίζει τη νέα υψηλή βαθμολογία που αποθηκεύτηκε.
54. Επιλέγοντας διαφορετικό θέμα ή δυσκολία, το Best Score αλλάζει αντίστοιχα.

55. Κάθε συνδυασμός θέματος και δυσκολίας διατηρεί ξεχωριστό high score.

ΚΕΦΑΛΑΙΟ 5ο

5 Συμπεράσματα

Από τα στοιχεία που παρουσιάστηκαν στο παρόν έγγραφο γίνεται κατανοητό ότι η εφαρμογή MemorizeApp αποτελεί μια ολοκληρωμένη λύση για την υλοποίηση παιχνιδιού μνήμης στην πλατφόρμα iOS, αξιοποιώντας πλήρως τις δυνατότητες του SwiftUI framework και εφαρμόζοντας πιστά το MVVM αρχιτεκτονικό πρότυπο. Η εφαρμογή αντιμετωπίζει επιτυχώς τα προβλήματα που εντοπίστηκαν στις υπάρχουσες λύσεις: παρέχει πλούσια animations, ποικιλία θεμάτων και επιπέδων δυσκολίας, δίκαιο σύστημα βαθμολόγησης και πλήρως offline λειτουργία.

Η ιδέα για τη δημιουργία της εφαρμογής προήλθε από την ανάγκη εφαρμογής σύγχρονων τεχνολογιών iOS σε ένα λειτουργικό και ελκυστικό project. Όπως φαίνεται από το εγχειρίδιο χρήσης, η εφαρμογή είναι απλή, εύχρηστη και υλοποιεί όλες τις απαραίτητες λειτουργίες για μια ολοκληρωμένη εμπειρία παιχνιδιού μνήμης. Ο χρήστης με ελάχιστα, κατανοητά βήματα μπορεί να επιλέξει θέμα και δυσκολία, να ξεκινήσει παιχνίδι, να αντιστοιχίσει κάρτες και να ανταγωνιστεί με τον εαυτό του για υψηλότερη βαθμολογία.

Τα βασικά τεχνικά συμπεράσματα που εξάγονται από την ανάπτυξη: Πρώτον, το MVVM pattern ταιριάζει φυσικά με το SwiftUI και επιτρέπει σαφή διαχωρισμό λογικής από παρουσίαση — αλλαγές στο Model δεν επηρεάζουν το UI και αντίστροφα. Δεύτερον, το SwiftUI παρέχει ισχυρά εργαλεία animations (Animatable, matchedGeometryEffect, custom Shapes, TimelineView) που επιτρέπουν οπτικά εντυπωσιακή εμπειρία με σχετικά μικρή ποσότητα κώδικα. Τρίτον, η generic παραμετροποίηση του Model (MemoryGame<CardContent>) εξασφαλίζει επεκτασιμότητα — ο ίδιος κώδικας μπορεί να χρησιμοποιηθεί με String, Int, εικόνες ή οποιοδήποτε Equatable τύπο. Τέταρτον, η χρήση Date intervals αντί για Timer αντικείμενα για τον bonus timer ήταν η σωστή επιλογή: αποφεύγει πολυπλοκότητα διαχείρισης κύκλου ζωής και λειτουργεί σωστά ακόμα και αν το View ανακατασκευαστεί.

Παράλληλα, η ανάπτυξη ανέδειξε σημαντικές τεχνικές προκλήσεις: το matchedGeometryEffect απαιτεί συγκεκριμένη σειρά εκτέλεσης (shuffle πριν reset dealt) για αποφυγή visual glitches, η ταυτόχρονη εκτέλεση πολλών TimelineView instances στο Hard επίπεδο επιβαρύνει τον επεξεργαστή χωρίς όμως αισθητά προβλήματα σε φυσική συσκευή, και η συνέπεια κλειδιών UserDefaults μεταξύ ViewModel και GameSetupView απαιτεί ιδιαίτερη προσοχή ώστε να εμφανίζεται πάντα το σωστό high score.

Εν κατακλείδι, η εφαρμογή αποτελεί μια καινοτόμα, εύχρηστη και τεχνικά ολοκληρωμένη λύση. Μελλοντικά υπάρχει δυνατότητα περαιτέρω επεκτάσεων, όπως Game Center integration για παγκόσμιους πίνακες κατάταξης, iCloud sync μέσω SwiftData με CloudKit, custom θέματα από τον χρήστη επιλέγοντας emoji από το πληκτρολόγιο, multiplayer mode μέσω GameKit, haptic feedback μέσω CoreHaptics, ήχους μέσω AVFoundation, iOS widget για το home screen, και μετεγκατάσταση από UserDefaults σε SwiftData για καλύτερη δομή δεδομένων.

ΚΕΦΑΛΑΙΟ 6ο

6 Βιβλιογραφικές Πηγές

56. Apple Developer Documentation. Xcode Overview. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/xcode/
57. Apple Developer Documentation. SwiftUI Documentation. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/documentation/swiftui
58. Apple Developer Documentation. The Swift Programming Language (Swift 5.9). Διαθέσιμο στον διαδικτυακό τόπο: https://docs.swift.org/swift-book
59. Apple Developer Documentation. UserDefaults. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/documentation/foundation/userdefaults
60. Apple Developer Documentation. Animatable Protocol. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/documentation/swiftui/animatable
61. Apple Developer Documentation. matchedGeometryEffect. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/documentation/swiftui/view/matchedgeometryeffect
62. Apple Developer Documentation. ViewModifier Protocol. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/documentation/swiftui/viewmodifier
63. Apple Developer Documentation. Core Graphics Framework. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/documentation/coregraphics
64. Apple Developer Documentation. TimelineView. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/documentation/swiftui/timelineview
65. Apple Developer Documentation. LazyVGrid. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/documentation/swiftui/lazyvgrid
66. Apple Developer Documentation. NavigationStack. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/documentation/swiftui/navigationstack
67. Apple Developer Documentation. Human Interface Guidelines for iOS. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/design/human-interface-guidelines

68. Apple Developer Documentation. ObservableObject Protocol. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/documentation/combine/observableobject

69. Stanford University CS193p. Developing Apps for iOS (Spring 2023). Διαθέσιμο στον διαδικτυακό τόπο: https://cs193p.sites.stanford.edu

70. Hacking with Swift. 100 Days of SwiftUI. Διαθέσιμο στον διαδικτυακό τόπο: https://www.hackingwithswift.com/100/swiftui (Ημερομηνία εισόδου: 13/05/2026)

71. Gossman, J. (2005). Introduction: Model/View/ViewModel. Microsoft MSDN Blogs. Διαθέσιμο στον διαδικτυακό τόπο: https://blogs.msdn.microsoft.com/johngossman

72. Apple Developer. WWDC 2019 — Introducing SwiftUI. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/videos/play/wwdc2019/204 (Ημερομηνία εισόδου:

73. Apple Developer. WWDC 2021 — Add rich graphics to your SwiftUI app. Διαθέσιμο στον διαδικτυακό τόπο: https://developer.apple.com/videos/play/wwdc2021/10021

74. owlcation.com. How to Include Code Snippets in a Research Paper. Διαθέσιμο στον διαδικτυακό τόπο: https://owlcation.com/stem/code-snippets

75. Stewart, P. & Strawn, S. (2023). SwiftUI by Tutorials, 6th Edition. New York: Kodeco Press.

76. Hudson, P. (2023). Hacking with Swift — SwiftUI. London: Hacking with Swift Ltd.

77. Draw.io (diagrams.net). Free online diagram software. Διαθέσιμο στον διαδικτυακό τόπο: https://www.diagrams.net