Treball Final de Carrera: Videojocs educatius

Kuku Memory, un videojoc per exercitar la memòria a dispositius iOS

Universitat Oberta de Catalunya, Juny 2012

Consultors: Jordi Duch Gavaldà i Heliodoro Tejedor Navarro
Alumne: David Pol Ahonen

A Daniel Fernández i Jaime Herrera
de Mutant Games, perquè aquest joc
també és vostre.

A la meva família, pel vostre suport
al llarg de tots aquests anys.

Índex

1. Introducció....4

2. Motivació....5

3. Objectius....5

4. Planificació....6

5. Disseny....8

6. Procés de desenvolupament....13

6.1. Estructura general del projecte....13

6.2. Jerarquia de classes....15

6.3. Integració amb Cocos2d....16

6.4. Atles de sprites....18

6.5. ARC (Automatic Reference Counting)....23

6.6 Blocs....24

6.7. Persistència dels millors resultats del jugador....25

6.8. Integració amb la plataforma social K-Pax....26

7. Conclusió i possibles millores....28

8. Bibliografia i referències....29

1. Introducció

Als darrers anys la indústria dels videojocs per a dispositius mòbils ha experimentat un enorme creixement. El principal motiu d'aquest el podem trobar en l'increment en el nombre de terminals (*smartphones* i *tablets*) capaços d'executar tota classe d'aplicacions exigents i a la popularització de mecanismes de distribució digital per a aquest tipus de dispositius. Els dos exemples més notables són *l'App Store* de *Apple* i el *Play* (anteriorment denominat *Android Market*) de *Google*.

Actualment, una gran part de les vendes totals de videojocs per a dispositius mòbils prové de maquinari *iOS* i *Android*, superant inclús les vendes de consoles dedicades. Per exemple, aquestes són les dades corresponents als Estats Units:

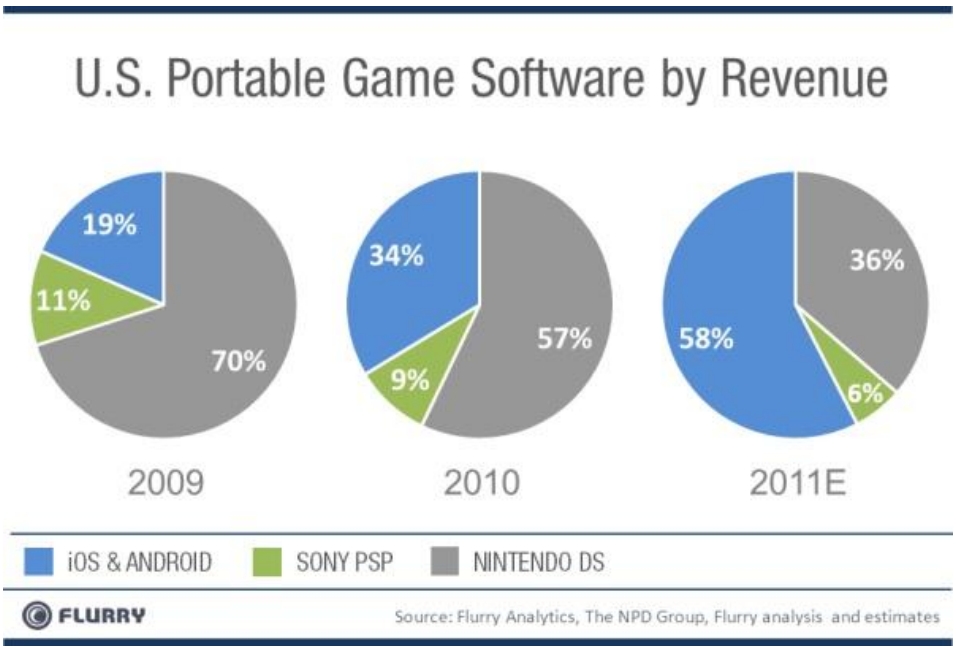


Figura 1. Evolució en la venda de videojocs per a dispositius mòbils als Estats Units

A més del fet que hi ha hagut un augment constant en la potència gràfica dels processadors mòbils, que permeten obtenir una qualitat comparable a la d'una consola dedicada en un *smartphone* o *tablet*, també ha tingut una influència important en aquesta evolució el fet que els videojocs *freemium* continuen revolucionant la manera en què els desenvolupadors rendibilitzen el seu treball, que el preu dels títols generalment és molt inferior als preus en consoles dedicades i que el desenvolupament en aquestes plataformes té una barrera d'entrada molt inferior.

En particular, una gran majoria de desenvolupadors ha apostat per *iOS* com a plataforma mòbil en la què concentrar els seus esforços. Com es pot apreciar en la figura 2, set de cada deu noves aplicacions es desenvolupen per *iOS*. El motiu principal d'aquest xifra es troba en la posició dominant que ocupen l'*iPhone*, l'*iPod Touch* i l'*iPad* en el mercat, i també en el fet que *Apple* proporciona mecanismes per executar una mateixa aplicació tant en *smartphones* com en *tablets* de manera particularment senzilla pels desenvolupadors.

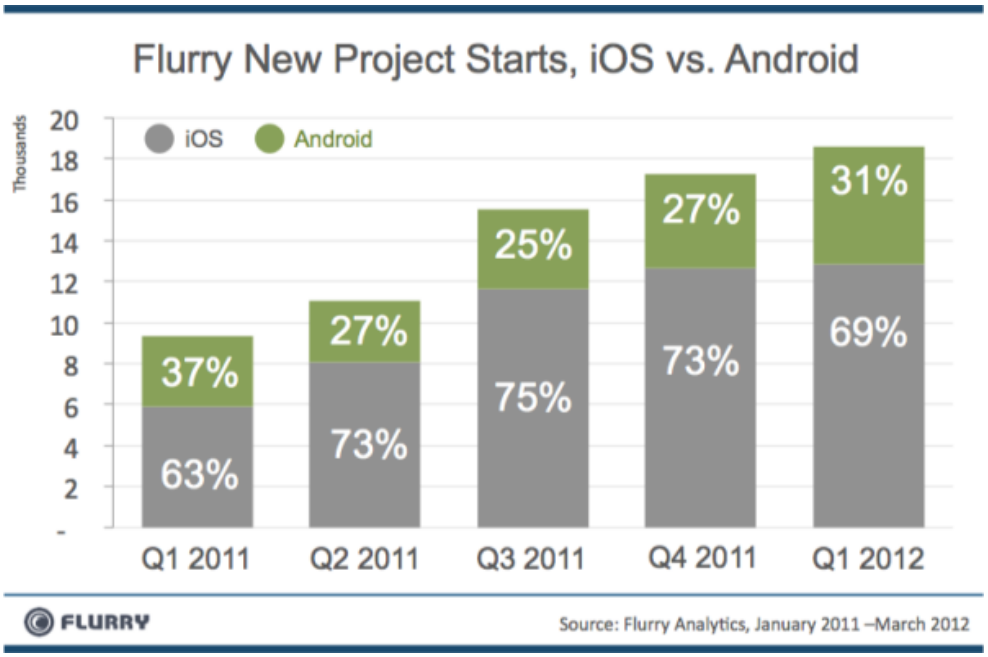


Figura 2. Estadístiques de noves aplicacions desenvolupades per dispositius iOS i Android

2. Motivació

A la meva trajectòria professional he participat en varis projectes de desenvolupament de videojocs en diferents plataformes (*Nintendo DS*, *iPhone*/*iPod Touch*, *PC*, *Xbox 360* i *PlayStation 3*). La motivació principal per realitzar aquest treball de fi de carrera consisteix en aprofundir en el coneixement de la plataforma *iOS* i la llibreria *Cocos2d* mitjançant la realització d'un videojoc complet que pugui ser publicat a la *App Store* de manera oficial en un futur. A més, és la primera vegada que tinc la oportunitat de realitzar un videojoc dins l'àmbit educatiu.

3. Objectius

L'objectiu d'aquest treball final de carrera és el desenvolupament d'un videojoc educatiu per dispositius *iOS* (en particular, la família formada per l'*iPhone* i l'*iPod Touch* amb versions de *iOS* iguals o superiors a la 4). La temàtica del videojoc es centra en el reforçament de la memòria immediata i el seu disseny contempla tres minijocs diferents amb aquesta intenció.

Amb aquest desenvolupament es pretén assolir un resultat final professional i que pugui donar pas a un futur llançament oficial a la *App Store* de *Apple*. Per aquest motiu, s'ha comptat amb la col·laboració de *Mutant Games*, un estudi de videojocs ubicat a Palma de Mallorca i especialitzat en el desenvolupament de videojocs per a dispositius mòbils, que s'ha encarregat d'elaborar tots els gràfics del joc. El disseny visual gira entorn els *kuku beans*, uns personatges

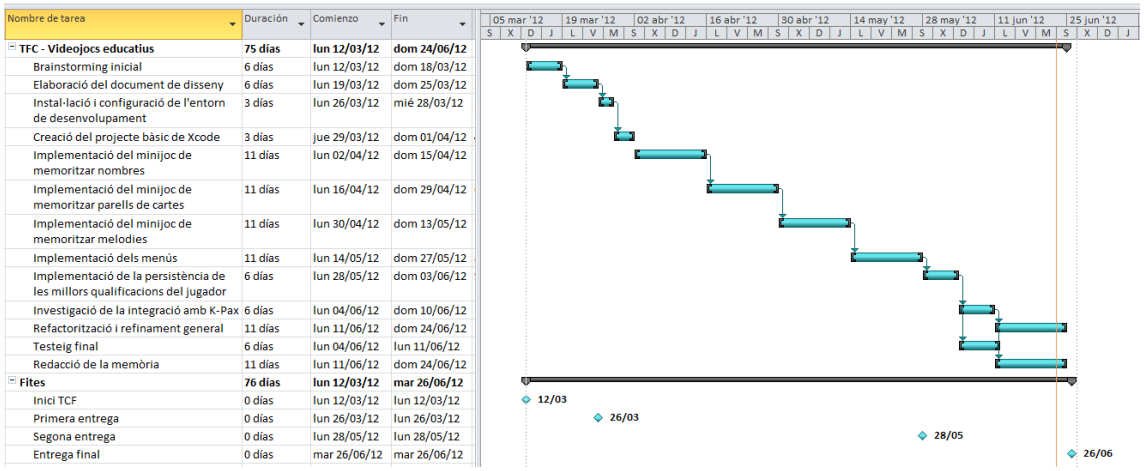
que contribueixen a donar-li un caire familiar al producte i que són els protagonistes de tot un conjunt de videojocs de temàtiques molt diverses de *Mutant Games*.

Figura 3. Els kukus, protagonistes a Kuku Memory

És important destacar que els drets sobre l'art de *Kuku Memory* pertanyen a *Mutant Games* mentre que els drets sobre el codi del projecte pertanyen a aquest autor.

4. Planificació

A continuació es mostra el diagrama de Gantt del projecte, que desglossa la planificació detallada del mateix al llarg del semestre:



Nombre de tarea	Duración	Comienzo	Fin
TFC - Videojocs educatius	**75 días**	**lun 12/03/12**	**dom 24/06/12**
Brainstorming inicial	6 días	lun 12/03/12	dom 18/03/12
Elaboració del document de disseny	6 días	lun 19/03/12	dom 25/03/12
Instal·lació i configuració de l'entorn de desenvolupament	3 días	lun 26/03/12	mié 28/03/12
Creació del projecte bàsic de Xcode	3 días	jue 29/03/12	dom 01/04/12
Implementació del minijoc de memoritzar nombres	11 días	lun 02/04/12	dom 15/04/12
Implementació del minijoc de memoritzar parells de cartes	11 días	lun 16/04/12	dom 29/04/12
Implementació del minijoc de memoritzar melodies	11 días	lun 30/04/12	dom 13/05/12
Implementació dels menús	11 días	lun 14/05/12	dom 27/05/12
Implementació de la persistència de les millors qualificacions del jugador	6 días	lun 28/05/12	dom 03/06/12
Investigació de la integració amb K-Pax	6 días	lun 04/06/12	dom 10/06/12
Refactorització i refinament general	11 días	lun 11/06/12	dom 24/06/12
Testeig final	6 días	lun 04/06/12	lun 11/06/12
Redacció de la memòria	11 días	lun 11/06/12	dom 24/06/12
Fites	**76 días**	**lun 12/03/12**	**mar 26/06/12**
Inici TCF	0 días	lun 12/03/12	lun 12/03/12
Primera entrega	0 días	lun 26/03/12	lun 26/03/12
Segona entrega	0 días	lun 28/05/12	lun 28/05/12
Entrega final	0 días	mar 26/06/12	mar 26/06/12

Figura 4. Planificació de les tasques del projecte

El desenvolupament de *Kuku Memory* s'ha completat sense cap desviació notable sobre la planificació prevista inicialment. El major factor de risc al llarg de tot el projecte ha sigut la dependència amb un element extern, els recursos gràfics provinents de *Mutant Games*, però aquests sempre han estat disponibles a temps i en sincronia amb l'estat del codi.

La mitjana de d'hores dedicades al projecte ha sigut de dues diàries de dilluns a diumenges. Com que el projecte ha tingut una durada de 16 setmanes, la dedicació total ha sigut de 224 hores.

5. Disseny

Kuku Memory disposa de tres minijocs destinats al reforçament de la memòria immediata del jugador. A continuació, es descriu l'objectiu específic de cadascun d'aquests tres minijocs:

- Recordar nombres: a la pantalla apareix una sèrie de nombres durant un període de temps prefixat. Al finalitzar aquest temps, alguns dels dígits que conformen aquests nombres desapareixen i el jugador ha de tornar a introduir-los en l'ordre original mitjançant un teclat numèric.
- Recordar parelles de cartes: a la pantalla apareix una sèrie de parelles de cartes en un ordre arbitrari durant un període de temps prefixat. Al finalitzar aquest temps, totes les cartes es posen de cap per avall i el jugador ha de tornar a girar-les seleccionant-les per parelles.
- Recordar melodies: a la pantalla apareix un conjunt de *kukus* que reprodueixen una melodia. Al finalitzar aquesta, l'objectiu del jugador es reproduir-la en la seva totalitat, de manera similar al clàssic *Simó*.

Els tres minijocs tenen una sèrie d'elements en comú:

- Estan estructurats en rondes (deu) que augmenten la seva dificultat de manera progressiva.
- El jugador disposa d'un temps màxim per jugar totes les rondes.
- El jugador no té la possibilitat de repetir cap ronda.
- Al final del minijoc, el sistema qualifica l'actuació del jugador en funció del nombre de rondes superades correctament amb un sistema de puntuació que contempla els valors S, A+, A, B, C, D i E (de millor qualificació a menor, respectivament) i que es persisteix a la memòria del dispositiu.

A continuació, es mostren els conceptes inicials realitzats amb *Photoshop* per Jaime Herrera, artista de *Mutant Games*, per a cadascuna de les pantalles del videojoc.

Figura 5. Pantalla de càrrega de l'aplicació

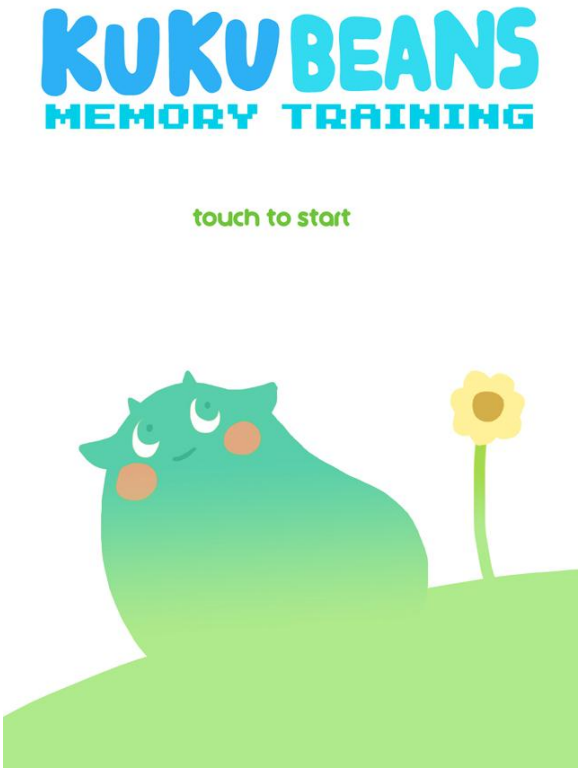


Figura 6. Menú principal

Figura 7. Menú de selecció de minijoc

Figura 8. Minijoc de memoritzar nombres

Figura 9. Minijoc de memoritzar parelles de cartes

Figura 10. Minijoc de memoritzar melodies

El resultat final és fonamentalment equivalent a aquests dissenys conceptuals inicials. Les diferències més notables són que el *kuku* que apareix a la barra inferior a les pantalles de minijocs està animat en lloc de ser un gràfic estàtic i que aquesta barra ja no presenta la informació de cada ronda individualment sinó només la qualificació final.

6. Procés de desenvolupament

El desenvolupament de *Kuku Memory* s'ha dut a terme amb la darrera versió de *Xcode 4* (en el moment de redactar aquesta memòria), l'entorn de desenvolupament oficial d'aplicacions per *Mac OS* i *iOS*, utilitzant un Mac Mini configurat amb una llicència individual oficial de desenvolupament de l'*iOS Dev Center*. El llenguatge de programació escollit ha sigut *Objective-C*, que és el llenguatge natiu de desenvolupament per *Mac OS* i *iOS*. A més, s'ha fet ús de les llibreries *Cocos2d*, que té com a objectiu facilitar la creació de videojocs 2d sobre plataformes *Mac OS* i *iOS* (essent, essencialment, una abstracció sobre *OpenGL*), així com també de *MKNetworkKit*, que té com a objectiu facilitar el desenvolupament de codi de xarxa de propòsit general. Com a eines addicionals, per a la creació dels atles de sprites s'ha fet ús de *Zwoptex* mentre que per a la creació dels atles de fonts s'ha utilitzat *bmGlyph*.

El videojoc s'ha provat tant en el simulador que incorpora *Xcode* com en tota una sèrie de dispositius reals (*iPhone*/*iPod Touch*).

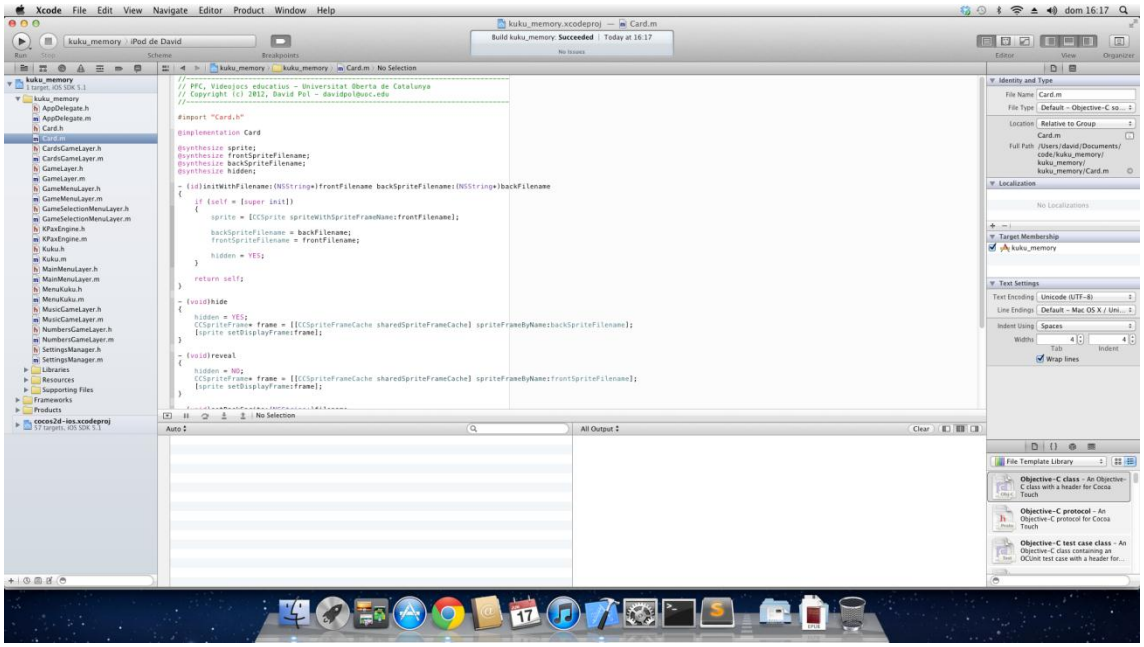


Figura 11. Captura de l'entorn de desenvolupament integrat Xcode 4, amb el projecte de Kuku Memory

A continuació, es descriu amb més detall el procés de desenvolupament de *Kuku Memory* i la interacció de tot el programari esmentat.

6.1. Estructura general del projecte

L'estructura del projecte de *Xcode* pel *Kuku Memory* és la següent:

- Fitxers de codi font (**.h*/**.m*): escrits en *Objective-C*, contenen tota la lògica del videojoc.

- Directori */Dependencies*: conté el codi de les llibreries externes utilitzades pel projecte; en aquest cas *Cocos2d* i *MKNetworkKit*.
- Directori */Resources*: conté tots els recursos de contingut del projecte, en particular tot l'art (icones, atles de sprites i atles de fonts).

El projecte és una modificació de la plantilla de projecte que incorpora de sèrie la llibreria *Cocos2d*. Aquesta modificació és necessària per tal de poder fer ús de ARC al codi del joc i també per reduir la mida total d'aquest. Como tot projecte de *Xcode*, proporciona per defecte dues configuracions de construcció de l'aplicació (*build*): *Debug* i *Release*, essent l'objectiu de la primera el disposar d'una versió de desenvolupament amb possibilitat d'esser depurada i el de la segona disposar d'una versió final no depurable amb el major rendiment possible.

Un aspecte molt important en la configuració del projecte consisteix en què aquest està preparat per generar versions distribuïbles de l'aplicació. Això permet realitzar proves a diferents dispositius reals des de l'inici del desenvolupament i també verificar que tot el procés de creació d'una versió distribuïble funciona correctament en tot moment, aspecte essencial un cop arriba el moment de publicar l'aplicació a la *App Store* (que s'ha de generar amb un procés fonamentalment equivalent a aquest).

El procés per generar versions distribuïbles de testeig del videojoc consisteix en crear un nou tipus de configuració de *build* que sigui una còpia de la configuració *Release* (la més òptima), però amb els certificats apropiats de distribució obtinguts del *iOS Dev Center*.

6.2. Jerarquia de classes

Aquest diagrama UML detalla la jerarquia de classes de què es composa el *Kuku Memory*:

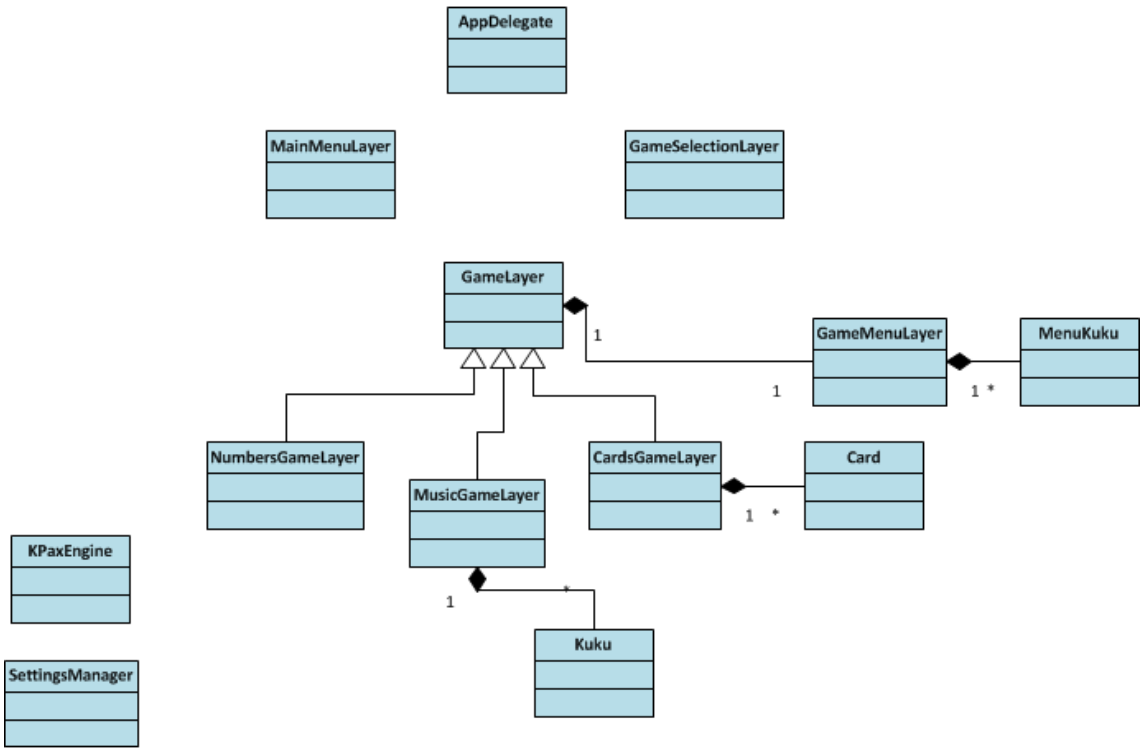


Figura 12. Jerarquia de classes a Kuku Memory

Un dels objectius principals en el disseny d'aquesta jerarquia ha sigut mantenir les responsabilitats de cada classe ben definides, perquè això contribueix a facilitar el desenvolupament i el manteniment del codi produït.

A continuació es descriu el propòsit de cadascuna de les classes desenvolupades:

- *MainMenuLayer*: representa la primera pantalla del joc, que és únicament de presentació.
- *GameSelectionMenuLayer*: representa la pantalla de selecció de minijoc, que també mostra les millors qualificacions obtingudes pel jugador en cadascun d'aquests.
- *GameLayer*: representa una pantalla de minijoc. Abstreu els elements comuns a tots els minijocs, que són principalment les propietats internes per controlar l'evolució de les rondes i també el menú del joc.
- *GameMenuLayer*: representa el menú que apareix a totes les pantalles de minijoc, i que mostra el botó de pausa, el temps de partida restant i el *kuku* que informa al jugador de la seva evolució durant la partida.
- *NumbersGameLayer*: representa el minijoc de memoritzar nombres.
- *MusicGameLayer*: representar el minijoc de memoritzar melodies.
- *CardsGameLayer*: representar el minijoc de memoritzar parells de cartes.
- *Card*: representa una carta del minijoc de cartes.
- *Kuku*: representa un kuku del minijoc de música.
- *MenuKuku*: representa el kuku que apareix al menú de joc.

- *SettingsManager*: proporciona funcionalitat per persistir els millors resultats del jugador a la memòria del dispositiu.
- *KPaxEngine*: proporciona funcionalitat per interactuar amb la plataforma social K-Pax.

6.3. Integració amb Cocos2d

Kuku Memory fa un ús extensiu de les facilitats proporcionades per la llibreria *Cocos2d* pel tractament de gràfics bidimensionals. El concepte més fonamental és el de capa, que representa una escena individual del joc i que actua com a contenidor d'elements gràfics. Aquest concepte es correspon amb la classe *CCLayer*, de la que deriven, directament o indirecta, totes i cadascuna de les classes que representen escenes en *Kuku Memory* (menús, pantalles de minijocs, etc.).

La implementació de les escenes a *Kuku Memory* segueix sempre el mateix patró:

- Al mètode *init* es creen tots elements que pertanyen a la escena i s'inicialitza la lògica d'aquesta.
- Al mètode *update* es duu a terme l'actualització de la lògica de l'escena a intervals regulars.
- Al mètode *ccTouchBegan* es gestiona l'entrada de l'usuari a la pantalla tàctil del dispositiu (si escau).
- Mai es sobrecarrega el comportament per defecte de *CCLayer* del mètode *draw*, perquè no és fa cap tipus de renderitzat especial.

És important destacar que totes les classes que representen escenes de minijocs deriven de *GameLayer*, un tipus propi que al seu torn deriva de *CCLayerColor* (una variant de *CCLayer* que permet definir el color de la capa), i que abstreu els elements comuns a tots els minijocs (principalment, el menú de joc i la progressió de les rondes).

```
/*
 * Base type for all the minigame layers.
 */
@interface GameLayer : CCLayerColor
{
    BOOL isGameFinished;

    float accTime;

    int numRoundsToPlay;
    int numRoundsLost;
    int numRoundsWon;

    NSString* settingKey;

    GameMenuLayer* gameMenu;

    CCLabelBMFont* memoCaption;
    CCLabelBMFont* guessCaption;
}

- (void)finishGame;
```

```
- (void)awardRank;

- (void)changeCaption;

@end
```

La classe *GameLayer* es composa de propietats que permeten controlar l'evolució de les rondes del minijoc, del menú de joc, dels títols de la ronda de memorització i de joc respectivament, i del nom de la clau a utilitzar per emmagatzemar la millor qualificació. Com que el menú dintre de les pantalles de minijoc és també el mateix sempre, s'ha abstret en un tipus independent (referenciat per *GameLayer*):

```
/*
 * This type is used across all game layers. It represents the in-game GUI:
 * pause button, countdown timer, bottom bar...
 */
@interface GameMenuLayer : CCLayer
{
    BOOL isPaused;
    BOOL isFinished;

    float accTime;
    float gameTime;

    CCLabelTTF* timer;
    CCSprite* pauseBg;
    CCLabelTTF* pauseCaption;

    MenuKuku* kuku;
}

@property(nonatomic) float gameTime;
@property(nonatomic) BOOL isPaused;
@property(nonatomic) BOOL isFinished;

- (void)displayResult:(BOOL)win position:(CGPoint)pos;

- (void)displayRank:(enum RankType)rank;

- (void)pause;

- (void)updateCountdownTimer;

@end
```

A més, *Cocos2d* gestiona les transicions entre capes (objectes *CCLayer*) de manera automàtica amb la classe *CCDirector*. El programador pot escollir diferents efectes gràfics per aquestes transicions; en el cas de *Kuku Memory* sempre es realitza un fos a negre d'un segon de duració. Per exemple, aquest és el codi que transiciona del menú principal al menú de selecció de minijoc:

```
[[CCDirector sharedDirector] replaceScene:[CCTransitionFade
transitionWithDuration:1.0f scene:[GameSelectionMenuLayer scene]]];
```

6.4. Atles de sprites

Un sprite és una imatge bidimensional que s'integra en una escena més gran. És un dels conceptes més fonamentals en el món dels gràfics per computador i, particularment, el dels videojocs (especialment videojocs 2D). El motiu d'això és que són el principal mecanisme de comunicació d'informació visual al jugador. Aquests són alguns exemples de sprites utilitzats a *Kuku Memory*:

Figura 13. Sprites utilitzats al minijoc de memoritzar parelles de cartes

Els atles de sprites són textures que agrupen tot un conjunt d'imatges individuals relacionades per motius d'eficiència. El canvi de textura és una operació costosa per la GPU, així que una pràctica habitual quan es dibuixen elements a pantalla en un videojoc consisteix en agrupar les operacions a realitzar en grups (*batches*) que comparteixen la configuració de l'estat de renderitzat (fonamentalment, el tipus de *blending* i la textura a utilitzar).

Així, en lloc de tenir un flux com el següent:

```
Canviar a textura de sprite 1
Dibuixar sprite 1
Canviar a textura de sprite 2
Dibuixar sprite 2
...
Canviar a textura de sprite n
Dibuixar sprite n
```

és preferible tenir:

```
Canviar a atles 1
Dibuixar tots els sprites que utilitzen l'atles 1
Canviar a atles 2
Dibuixar tots els sprites que utilitzen l'atles 2
...
Canviar a atles n
Dibuixar tots els sprites que utilitzen l'atles n
```

La segona opció és molt més eficient pel que respecta a la utilització de la GPU, ja que implica un nombre menor (potencialment molt menor) d'operacions de canvi de textura. Els elements es dibuixen a pantalla per blocs que comparteixen la configuració de renderitzat.

La utilització de atles de sprites esdevé, doncs, una necessitat imprescindible per poder fer ús d'aquesta optimització. Cocos2d suporta l'ús d'atles de sprites i de *batches* de sprites en totes les seves classes, i *Kuku Memory* en fa ús d'aquests de manera extensiva.

Un exemple molt clar d'aquestes dues tècniques es pot trobar al codi d'inicialització de la pantalla principal del joc, on es dibuixen els elements de la següent manera:

```
[[CCSpriteFrameCache sharedSpriteFrameCache]
addSpriteFramesWithFile:@"menu.plist"];

CCSpriteBatchNode* spriteSheet = [CCSpriteBatchNode
batchNodeWithFile:@"menu.png"];

CCSprite* logo = [CCSprite spriteWithSpriteFrameName:@"menu_logo.png"];

logo.position = ccp(160, 380);

[spriteSheet addChild:logo];

CCSprite* floor = [CCSprite spriteWithSpriteFrameName:@"menu_floor.png"];
```

```
floor.position = ccp(160, floor.textureRect.size.height/2);

[spriteSheet addChild:floor];

// ...

[self addChild:spriteSheet];
```

En primer lloc, es pot observar com s'afegeix l'atles de sprites identificat pel fitxer *'menu.plist'* a l'objecte *sharedSpriteFrameCache*. Aquest objecte és una instància global que manté *Cocos2d* per emmagatzemar referències a sprites que estan carregats a memòria. El fitxer .plist defineix les metadades de l'atles (nombre, mida i posicions a la textura de cada sprite) i té el mateix nom que el fitxer d'imatge corresponent a la textura que representa l'atles (aquesta és la convenció que segueix internament *Cocos2d* i que el programador ha de respectar). Una vegada afegit l'atles a la cau global, és possible crear nous sprites a partir d'aquest, per exemple amb *[CCSprite spriteWithSpriteFrameName:@"nom_sprite"]*.

Un punt a destacar és que en *Kuku Memory* s'ha fet ús de la capacitat de mostrar gràfics en alta resolució (també anomenats *Retina Display* o *HD*) de què disposen les darreres generacions de dispositius *iOS*. Això permet utilitzar imatges del doble de resolució en el mateix espai de pantalla (640x960 enfront de 320x480 píxels), per un resultat final molt més nítid pel jugador. Des del punt de vista de la programació, *Cocos2d* detecta automàticament la presència d'aquestes textures i les utilitza si existeixen, així que només és necessari respectar la convenció de *Cocos2d* a l'hora de nombrar els recursos en alta resolució (tant el .plist com el .png afegeixen l'extensió –hd).

Per a crear els atles de textures s'ha utilitzat el software *Zwoptex*, que, donat un conjunt d'sprites individuals i unes especificacions de mida i format de textura, genera automàticament un atles que conté tots els sprites agrupats de manera òptima i les metadades associades a aquest (en un format compatible amb *Cocos2d*).

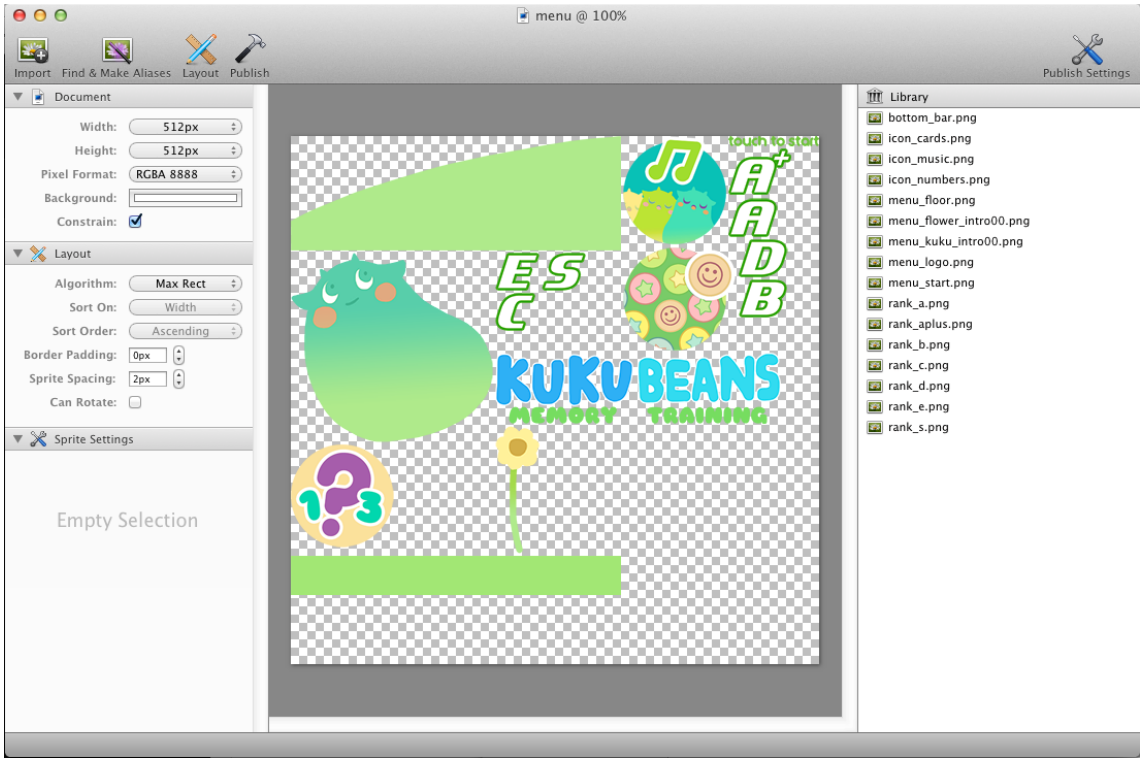


Figura 14. Atles de sprites corresponent als menús de Kuku Memory

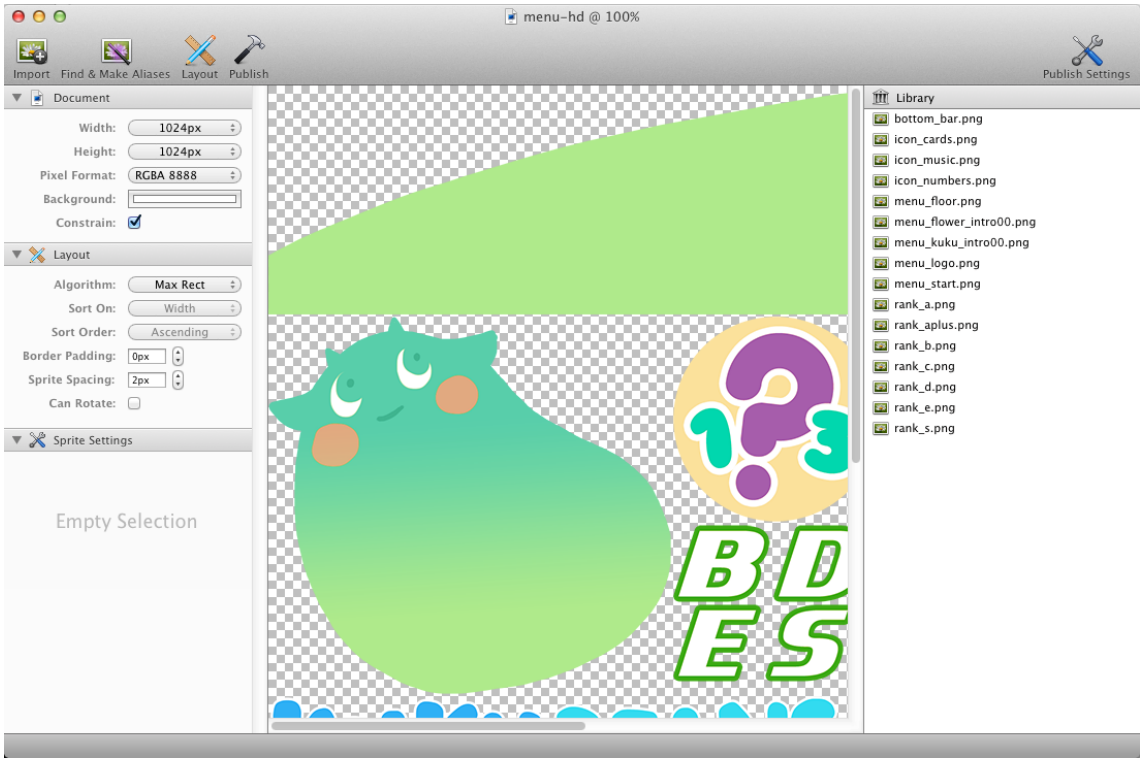


Figura 15. Atles de sprites corresponent als menús de Kuku Memory (versió HD)

Es por observar com tots els sprites que es creen per aquesta pantalla s'afegeixen a un objecte de tipus *CCSpriteBatchNode*. Aquest objecte internament s'encarrega de verificar que tots els sprites que té afegits com a fills efectivament pertanyen a la mateixa textura i de evitar fer canvis d'estat de GPU redundants. En aquest sentit, pels usuaris de Cocos2d aquesta optimització és gairebé "gratuïta" en el sentit que només cal fer utilitzar aquest tipus de node intermedi per aprofitar-se dels seus avantatges de manera transparent al programador.

Pel tractament de les fonts s'ha dut a terme un procés fonamentalment equivalent, però utilitzant un editor específic, *bmGlyph*. De manera similar a *Zwoptex*, aquest editor genera un atles de font i un fitxer de metadades a partir d'una sèrie d'especificacions sobre la font desitjada (fitxer de font origen, tamany, color, ús de *kerning*, efectes, etc.). En aquest cas, la textura de l'atles generat conté tots els caràcters desitjats.

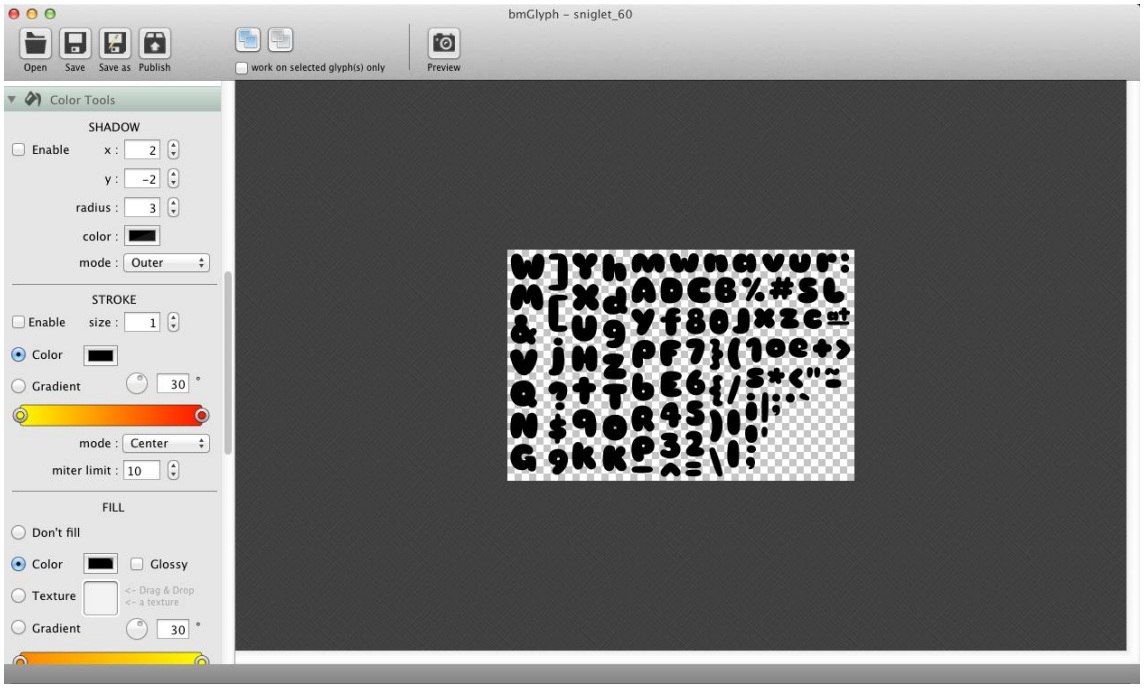


Figura 16. Atles de la font utilitzada a Kuku Memory

El suport a *Cocos2d* del fitxer de font generat per *bmGlyph* és directe mitjançant la classe *CCLabelBMFont*:

```
CCLabelBMFont* titleCaption = [CCLabelBMFont labelWithString:@"Select a game:"
fntFile:@"sniglet_40.fnt"];
titleCaption.position = ccp(160, 400);
titleCaption.color = ccc3(160, 117, 23);
[self addChild:titleCaption];
```

6.5. ARC (Automatic Reference Counting)

La darrera versió del sistema operatiu *iOS* introdueix un mecanisme anomenat *Automatic Reference Counting* que simplifica el procés de gestió del temps de vida dels objectes d'una aplicació, això és, la gestió de la seva memòria. En la majoria de situacions, l'ARC converteix aquesta tasca en un procés trivial, encara que el programador encara manté la responsabilitat sobre com gestionen les seves classes les referències a altres objectes.

La memòria gestionada manualment en *iOS* funciona de la següent manera: quan es crea un objecte utilitzant *alloc* o *init* (o qualque mètode similar), l'objecte és retornat amb un *retainCount* (comptador de referències) de 1, el que vol dir que s'obté la propietat de l'objecte.

```
NSObject* obj = [[NSObject alloc] init];
// ...
[obj release];
```

Entre el moment d'obtenir la propietat de l'objecte i eliminar aquest amb un crida al mètode *release*, el programador pot utilitzar l'objecte amb la seguretat de que la seva memòria no serà alliberada mentre aquest estigui en ús.

De manera similar, si s'afegeix un objecte a una *autorelease pool*, l'objecte es manté en memòria mentre es faci ús d'ell i s'allibera automàticament pel sistema en un moment posterior quan ja no es necessita.

```
- (NSObject*)method
{
        NSObject* obj = [[[NSObject alloc] init] autorelease];
        return obj; // serà alliberat de manera automàtica posteriorment
}
```

L'ARC és un pas previ a la compilació que afegeix sentències *retain*/*release*/*autorelease* al codi automàticament. És a dir, no és una forma de *garbage collection*, simplement automatitza un procés que abans era manual i, com a tal, estava subjecte a errors per part del programador. Codi escrit de la següent manera:

```
NSObject* obj = [[NSObject alloc] init];
// utilitzar obj...
```

, amb ARC es transforma automàticament en:

```
NSObject* obj = [[NSObject alloc] init];
// utilitzar obj...
[obj release]; // inserit automàticament pel compilador
```

En *Kuku Memory* s'ha utilitzat l'ARC i l'increment en la productivitat i correcció del codi final han sigut notables. L'ús d'ARC pot tenir, a més, un efecte beneficiós en el rendiment final de l'aplicació, pel fet d'utilitzar menys les *autorelease pools*, que tenen un cost de manteniment en temps d'execució associat.

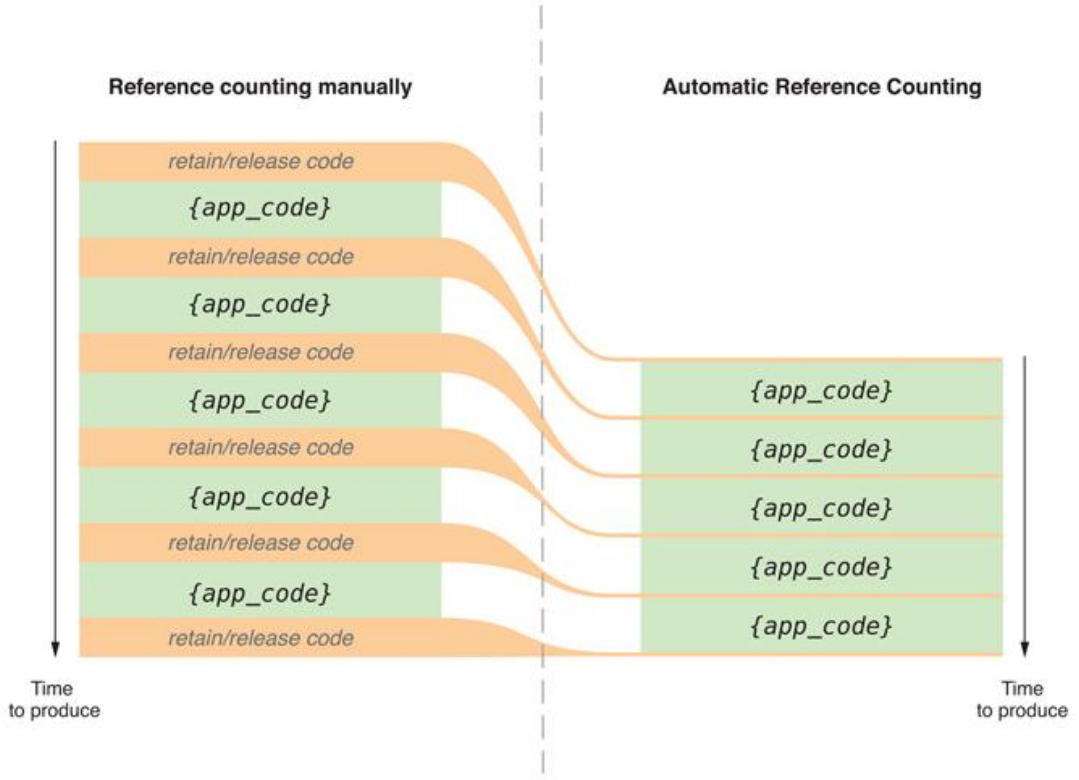


Figura 17. Millora en el temps de desenvolupament amb l'ús d'ARC

6.6. Blocs

Els blocs són col·leccions anònimes en línia de codi que:

- Tenen una llista d'arguments amb tipus, al igual que una funció.
- Tenen un tipus de retorn (inferit o declarat).
- Poden capturar estat de l'àmbit lèxic dins el qual estan definits.
- Poden, de manera opcional, modificar l'estat d'aquest àmbit lèxic.

Els blocs representen petites peces de codi auto-contingudes. Com a tals, son particularment útils com una manera d'encapsular unitats de treball que poden ser executades concurrentment o com a *callback* quan una operació ha terminat.

Els blocs són una alternativa útil a les tradicionals funcions de *callback* per dos motius: que permeten escriure codi directament al punt d'invocació que serà executat més tard en el context de la implementació del mètode i que tenen accés a les variables locals del seu àmbit lèxic.

A *Kuku Memory*, els blocs s'han utilitzat de manera extensiva en els mètodes de les llibreries *Cocos2d* i *MKNetworkKit* que els utilitzen com a paràmetres. Per exemple, al minijoc de memoritzar nombres s'utilitzen blocs a la declaració dels botons que constitueixen la calculadora que utilitza el jugador per introduir els dígits de la seva resposta, per tal d'anar emmagatzemant aquesta internament de manera apropiada:

```
CCMenuItemSprite* number0 = [CCMenuItemSprite itemWithNormalSprite:[CCSprite
spriteWithSpriteFrameName:@"GUI_num_button_0a.png"]
            selectedSprite:[CCSprite
spriteWithSpriteFrameName:@"GUI_num_button_0b.png"]
            disabledSprite:[CCSprite
spriteWithSpriteFrameName:@"GUI_num_button_0c.png"]
            block:^ (id sender) { [self inputNumber:0]; }];

CCMenuItemSprite* number1 = [CCMenuItemSprite itemWithNormalSprite:[CCSprite
spriteWithSpriteFrameName:@"GUI_num_button_1a.png"]
            selectedSprite:[CCSprite
spriteWithSpriteFrameName:@"GUI_num_button_1b.png"]
            disabledSprite:[CCSprite
spriteWithSpriteFrameName:@"GUI_num_button_1c.png"]
            block:^ (id sender) { [self inputNumber:1]; }];

// ...
```

Un altre exemple d'ús de blocs es troba al codi de la classe *KPaxEngine*, quan es defineixen els *callbacks* de "connexió satisfactòria" i "connexió errònia" a les operacions de connexió al servidor de la plataforma K-Pax:

```
MKNetworkOperation* op = [self
operationWithPath:@"webapps/svrKpax/user/auth/elgg" params:params
httpMethod:@"POST" ssl:NO];
[op onCompletion:^(MKNetworkOperation* operation)
{
    NSLog(@"%@", [operation responseString]);
}
onError:^(NSError* error)
{
    NSLog(@"%@", error);
}
];
```

6.7. Persistència dels millors resultats del jugador

Kuku Memory emmagatzema les millors qualificacions del jugador en cadascun dels minijocs. Això permet que aquest sempre tingui fàcilment accessible la seva millor referència per tal d'intentar superar-la.

Per tal d'aconseguir mantenir un conjunt de dades de manera persistent entre sessions de joc, al codi es fa ús de la classe *NSUserDefaults* de la llibreria estàndard del llenguatge. Aquesta classe permet accedir a la base de dades de preferències de l'usuari del dispositiu. Per facilitar la gestió de la persistència des de diferents punts del videojoc (concretament, des dels diferents minijocs), s'ha desenvolupat un tipus propi que abstreu la lògica de les operacions específiques amb *NSUserDefaults* anomenat *SettingsManager*. Aquest tipus proporciona una propietat estàtica amb accessibilitat global (els dispositius tenen una única base de dades de preferències centralitzada) i té la següent interfície:

```
@interface SettingsManager : NSObject
{
    NSMutableDictionary* settings;
}

- (int)getIntValue:(NSString*)key;
- (void)setIntValue:(NSString*)key value:(int)value;

- (BOOL)keyExists:(NSString*)key;

- (void)load;
- (void)save;

// Set this up as a singleton for easy access.
+ (SettingsManager*)sharedSettingsManager;

@end
```

La classe consisteix en un diccionari intern que emmagatzema parells identificador de minijoc/millor qualificació. Puntualment, aquest diccionari es carrega al joc a partir de la base de dades de preferències i es persisteix des del joc cap a aquesta.

6.8. Integració amb la plataforma social K-Pax

Una de les línies d'investigació d'aquest projecte ha consistit en determinar la factibilitat d'integració del videojoc desenvolupat amb la plataforma social de jocs de la UOC, K-Pax.

Tot i que es va poder registrar el videojoc a la plataforma sense cap tipus de problema, l'absència de llibreries funcionals per l'autorització de connexions amb OAuth i criptografia per *iOS* (requisits indispensables per connectar-se a K-Pax) ha fet impossible una integració completa amb la plataforma dins l'àmbit d'aquest projecte. Llibreries com *RestKit* no són compatibles amb projectes que utilitzen característiques més noves del SDK de *iOS* com són l'ARC (*que Kuku Memory* utilitza), i llibreries com *MKNetworkKit* no suporten OAuth.

La classe *KPaxEngine* conté el codi de base utilitzant la llibreria *MKNetworkKit* per realitzar connexions a K-Pax utilitzant la seva API (però sense autorització):

```
/*
 * This type enables easy access to the game's network engine, which abstracts
 * away a connection with the K-Pax platform server. As it stands, this class
 * is a placeholder that is not currently used in Kuku Memory because there is
 * no direct library support for OAuth authentication and RSA-SHA1 encryption
 * in iOS, both of which are needed to communicate with the K-Pax platform.
 */
@interface KPaxEngine : MKNetworkEngine

// Authenticates the user with the given credentials into K-Pax.
// Returns: the secretSession.
- (void)authenticateUser:(NSString*)username password:(NSString*)password;

// Inits the game's instance.
// Returns: INIT GAME.
- (void)initGame:(NSString*)secretSession secretGame:(NSString*)secretGame;
```

```
// Ends the game's instance. The given score is encrypted with AES.
// Returns: OK if everything went well.
- (void)endGame:(NSString*)secretSession secretGame:(NSString*)secretGame
points:(NSString*)points;

// Set this up as a singleton for easy access.
+ (KPaxEngine*)sharedKPaxEngine;

@end
```

Una possible millora de cara al futur consistiria en implementar aquest suport per autorització OAuth dins la pròpia llibreria *MKNetworkKit* tot mantenint les seves interfícies fonamentals d'operacions amb connexions de xarxa.

7. Conclusions i possibles millores

La realització d'aquest treball final de carrera ha sigut enormement satisfactòria i didàctica, principalment pel fet d'haver dut a terme un projecte complet de videojoc per dispositius mòbils amb un conjunt de llenguatge de programació, eines i tècniques que no havia utilitzat anteriorment de manera extensiva i d'haver treballat amb una temàtica diferent a l'habitual en la meva experiència professional, com és l'educativa.

A l'enllaç http://www.youtube.com/watch?v=PuHoBPyEayc s'ha pujat un vídeo que mostra el resultat final del videojoc desenvolupat en aquest treball de fi de carrera executant-se sobre el simulador d'*iPhone* que incorpora l'entorn de desenvolupament integrat *Xcode 4*. Com s'ha mencionat en aquesta memòria, també és possible generar versions especials de testeig que poden ser executades en dispositius reals sempre que s'autoritzi específicament l'identificador únic d'aquests dispositius dins l'*iOS Dev Center*, perquè *Apple* no permet la lliure distribució d'aplicacions fora de la seva *App Store*.

De cara a un futur llançament de *Kuku Memory* a la *App Store* de *Apple*, es podrien implementar les següents característiques per millorar l'experiència dels jugadors:

- Afegir nous minijocs, perquè la xifra actual de tres es pot quedar una mica curta en poc temps.
- Afegir nivells de dificultat globals (fàcil, mitjà i difícil, per exemple, agrupats per edats) en lloc d'utilitzar una dificultat progressiva dintre de cada minijoc, perquè això pot facilitar que diferents tipus de jugadors trobin el nivell més adequat a les seves capacitats. Ara mateix la dificultat pot ser massa alta per alguns jugadors.
- Afegir un conjunt d'estadístiques més complet, i no només una qualificació general.
- Afegir música i efectes de so. Això pot resultar especialment interessant en el minijoc de memoritzar melodies.
- Integrar el videojoc amb plataformes socials como el K-Pax de la UOC o el *Game Center* d'*Apple* per tal què els jugadors puguin compartir els seus resultats en línia.
- Desenvolupar una versió compatible amb el *tablet* d'*Apple*, l'*iPad*.

8. Bibliografia i referències

- *Programming in Objective-C (4th Edition)*: Stephen G. Kochan, Addison-Wesley Professional 2011

- *Learning Cocos2d: A Hands-On Guide to Building iOS Games with Cocos2d, Box2D, and Chipmunk:* Rod Strougo i Ray Wenderlich, Addison-Wesley Professional 2011

- Mutant Games: http://mutant-games.com/

- iOS Dev Center: https://developer.apple.com/devcenter/ios/index.action

- Flurry: http://www.flurry.com/

- Cocos2d: http://www.cocos2d-iphone.org/

- MKNetworkKit: http://blog.mugunthkumar.com/ios-components/mknetworkkit/

- RestKit: http://restkit.org/

- Zwoptex: http://zwopple.com/zwoptex/

- bmGlyph: http://www.bmglyph.com/