Advanced Mobile: iOS Development – Software Report

Student Number: 15026592

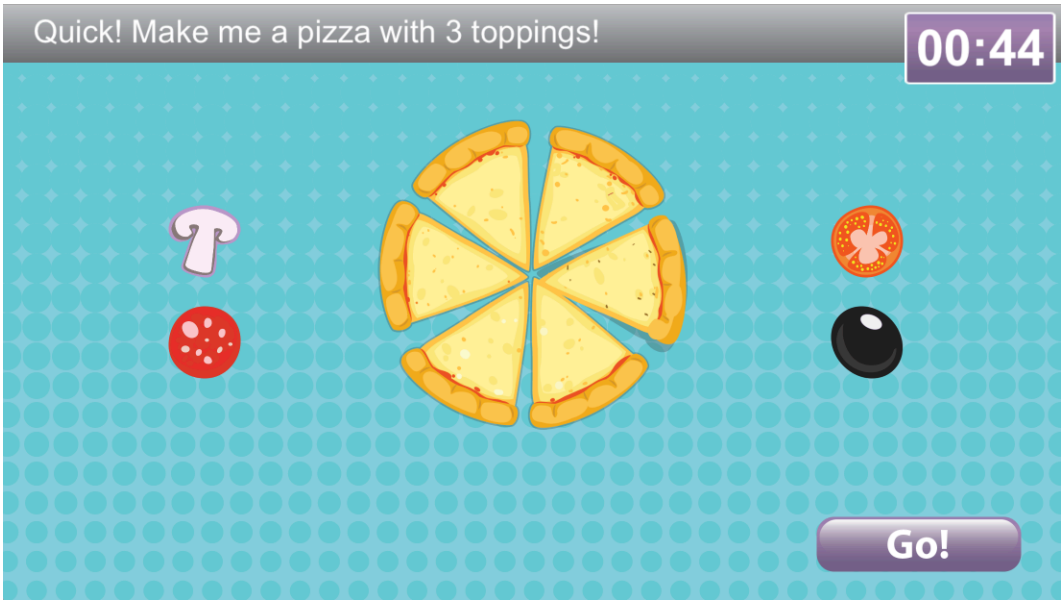
1. User Guide and Usability Evaluation

The Diner Deduction app was developed using Xcode 8.3.3 on macOS 10.12.6, and written in Swift 3 for iOS 10.3. The game is playable in landscape mode only. The app was originally intended for iPad too, and had proposed Facebook integration, though due to time restraints these features were not possible for this iteration. The following screenshots are taken from the iPhone 7 emulator, though the app was also tested on a physical device. The introductory menu scene (Figure 1) displays the game logo with a flashing "Start game" button.

Figure 1: Menu scene

After tapping the button, the user is then presented with the main game play scene (Figure 2) and the timer begins counting down.

Figure 2: Main game play scene



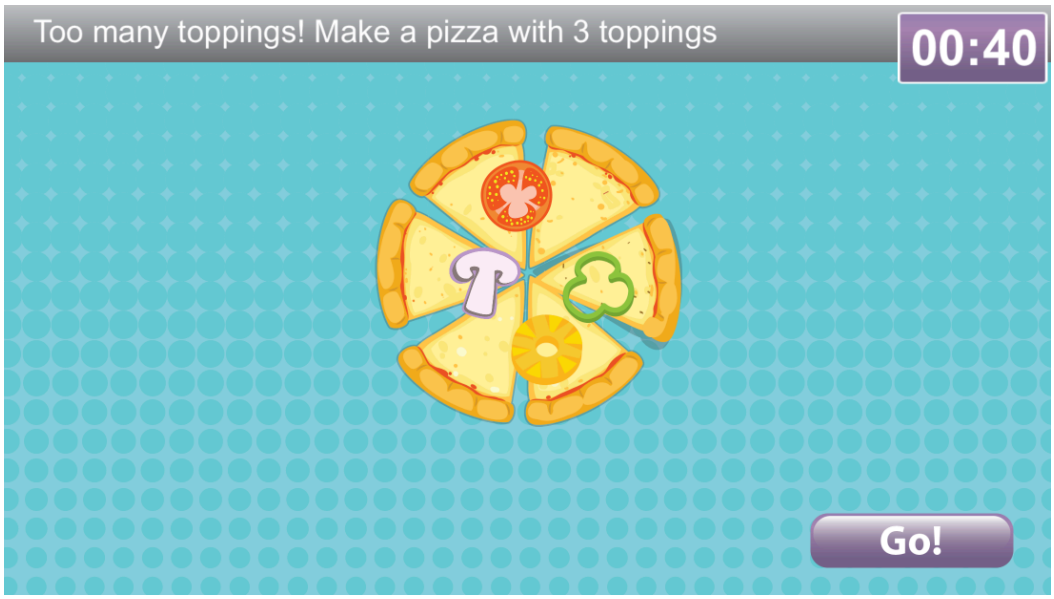
The user can now drag and drop ingredients onto the pizza. A random selection of ingredients is generated at the start of each game to add variation. Drag and drop was chosen over simply tapping the ingredients in order to achieve a better "game feel" (a term first coined by Swink[1]). This makes the user more actively involved in the game. As the app was developed using short iterative cycles of development, user testing occurred early on, and

revealed that some users were unsure about the placement of their items. In the first version of the game, users could drag and drop an item anywhere on the screen, but the game would only register items placed on the pizza.

This revealed that some user feedback was needed about correct and incorrect placement of ingredients. The Cocoapods dependency manager[2] was used to install the SpriteKitEasingSwift library[3], which was used to provide easing animations to the pizza ingredients, which are SprikeKit objects. When the ingredients are incorrectly placed, the easing animations will spring them back to the original positions, providing a visual clue. When an ingredient has been correctly placed, a brief "CloudPuff" particle effect is generated. Ideally sound feedback would also be added to the game before publication to enhance the user experience further.

If the user places too many or too few items on the pizza, the notification bar at the top will communicate the error when the user taps the 'Go!' button (Figure 3).

Figure 3: Too many ingredients placed in game



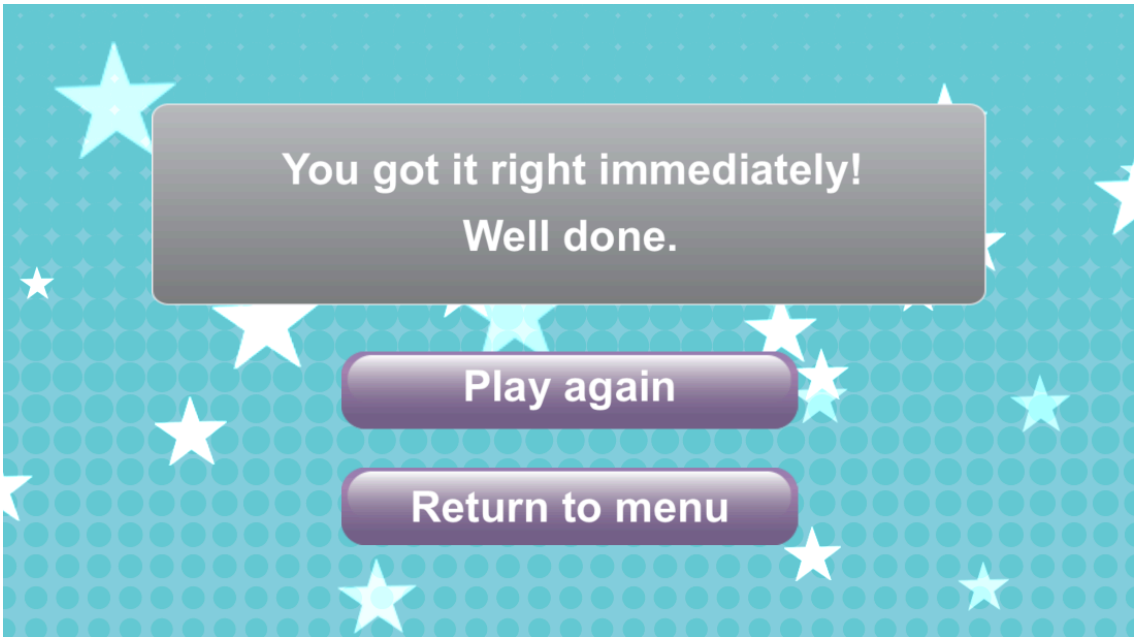
User testing revealed that this information was sometimes not obvious enough, so flashing text was added to draw the user's eye. In a future iteration of the game it might be better to have modal pop-up windows with the information presented over the pizza to ensure that is completely apparent. This would also allow more room to make the pizza and ingredient sprites bigger, which was a usability concern. Some users found the ingredients in particular hard to drag quickly due to their size. Space was also an issue when the previous guess boxes appeared (Figure 4).

Figure 4: Game scene with previous guess boxes

The previous guess boxes appear when the user placed the correct number of ingredients on the pizza, but the ingredients list is wrong. This acts as visual reminder to help the user deduct the correct ingredients. User testing also revealed the need to add text stating how many ingredients were right on each pizza, so this was added to the top of each box. Usability and accessibility is a concern here as they are a little small to see easily, especially if harder levels with more ingredients to guess were added. This may be less of an issue if the game is optimised for iPad. By removing the top navigation bar, it would allow them to be a little larger.

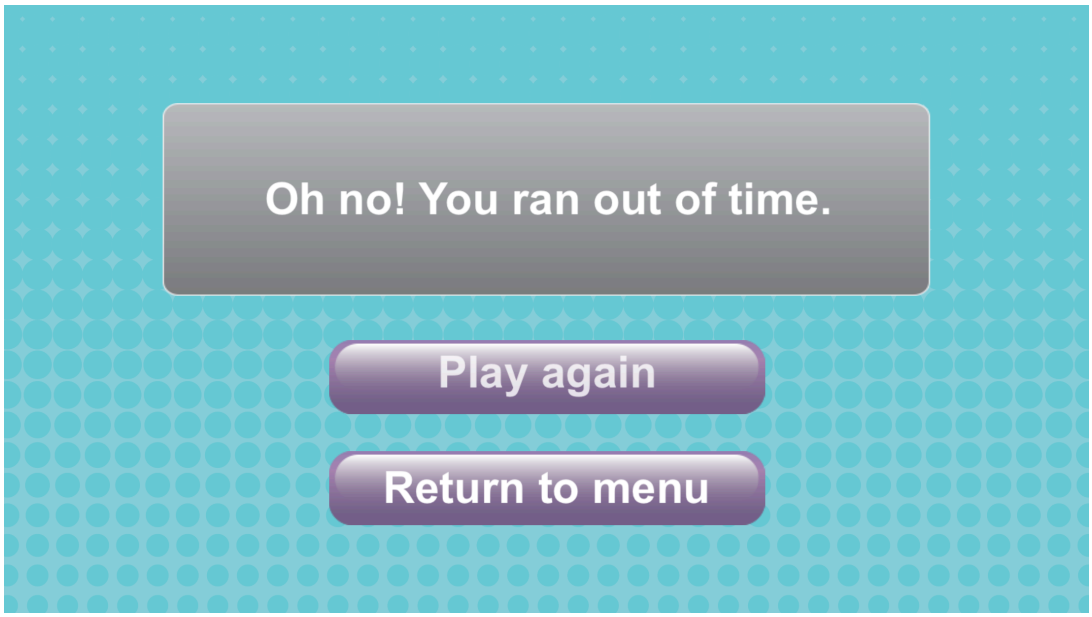
When the user correctly guesses the ingredients, they are taken to the Game Won scene (Figure 5).

Figure 5: Game Won Scene



The 'Sparkles' particle effect has been added to distinguish it from the Game Lost scene (Figure 6), and add a small surprise as a reward to the player. The text also changes depending on the number of guesses the player had to make before winning. If the player takes more than three times (four, as an example), the text will say, "Hmm. You completed the pizza in four tries. Maybe you need more practice?" The user is then given the option of playing again or returning to the menu scene.

Figure 6: Game Lost Scene



The game lost scene appears when the timer has expired, and the player has not guessed the correct combination of ingredients. Again, the player is given the options of starting a new game or returning. The text on the "Play again" button flashes to act as a call to action.

2. App Design and Development

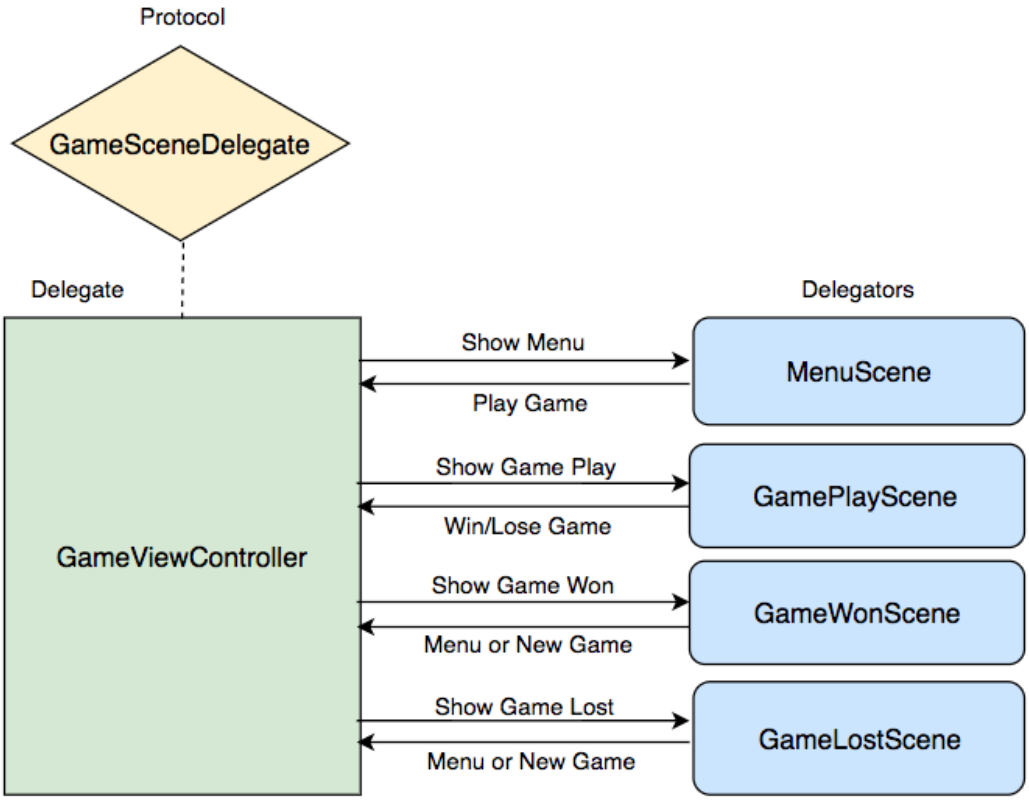
2.1 Design Patterns

The app was designed using the MVC framework, which separates code into Model, View and Controller. In this framework, the Model contains the data, the View contains the user interface, and the Controller acts as an intermediary between the two. It responds to user input from the View and data changes from the Model whilst keeping them separate.[4] The main concept behind this pattern is to separate code concerns in order to keep the code maintainable and easily extensible. It means that new views or data sources may be switched in at a later data with minor code interference.

To achieve this decoupling, a messaging system must be utilised to communicate between the Model, View and Controller. Delegation is a common method employed by Swift applications, and was utilised in Diner Deduction to control the game scenes. Delegation is a pattern where an object in a program acts on behalf of another, decoupling the logic between the two[5]. The delegator object keeps a reference to the other object (the delegate), and then at the appropriate time, the delegator sends a message to the delegate. This message will instruct the delegate about an event that has just been, or is about to be, handled by the delegator.

A protocol called GameSceneDelegate was introduced as an abstraction layer to define the methods handled in the GameViewController. This meant that the logic to switch between views in the game was kept in a single place – the GameViewController – rather than being spread across the various game scenes. This makes it easy to see immediately see an overview of the game's main flow, and keeps it separate from the scenes. Each individual scene uses the delegate to communicate with the GameViewController to trigger a scene change when needed (Figure 7).

Figure 7: Delegation for game scenes



Within the scenes, the Observer pattern[4] was used to manage game state changes. Each scene is comprised of selection of SpriteKit objects (the ingredients, the pizza, the buttons, etc.), which are decoupled from each other by using Apple's NotificationCenter[6] – an inbuilt implementation of the Observer pattern. This pattern posts a message on a central bus rather than calling another object directly. Objects can then 'listen' for relevant notifications and act accordingly. This ensures that the objects are completely decoupled and independent of each other, and means that multiple objects can listen for a single notification. It also makes for cleaner code as if listeners are later removed, no errors will be caused by the events being posted (and removing them is a simple process).

At the proposal stage of the project, a State Machine was suggested to maintain game state in the form of a Singleton, but during development it became apparent that this was unnecessary. Using a combination of delegation (to manage the views) and notifications (to manage events within the game) is a more lightweight solution that utilises Apple's own inbuilt tools.

As certain Sprites can be interacted with, it made sense to encapsulate their touch behaviour inside the Sprites themselves, rather than create large chains of logic in the scenes' touch methods. The Strategy pattern[4] was used to create GameSprite instances. This pattern uses a protocol to define a family of methods. Every object that applies the protocol can then encapsulate them and implement different behaviour at runtime. To accomplish this, a GameSprite protocol with touch, drag, and drop methods was created. GameSprites could then implement this protocol to declare their own touch behaviour. For example, a button needs to react differently to touches than a draggable ingredient. The touch behaviour of the GameSprite instances was thereby fully decoupled from the scenes.

A Button class was also created for all the buttons used throughout the game. This also implements GameSprite, making it easy to create a new button instance with touch behaviour.

Documentation for the project components was autogenerated from code markup into HTML using a command-line utility called Jazzy[7], and can be found inside the Diner Deduction project inside the 'docs' folder. It was generated using the following command:

```
jazzy --min-acl private --no-skip-undocumented --author "Leonie Kenyon".
```

2.2. Issues and Improvements

The timeline for development in the proposal proved difficult to abide by, and several features had to be left out of the app prototype, including extra levels, surprise "flash events", and Facebook integration.

There are a few outstanding issues with the code. The MVC pattern could be adhered to more closely by properly extracting out a Model. Currently there are a couple of classes that could be considered to be the Model – namely the Customer, the IngredientsListGenerator and the ArrayShuffler. These could be extracted away into a combined Model, which would keep the GameViewController logic cleaner. Similarly, there are a lot of Sprites that have behavioural logic such as physics and touch response attached to them – it would have been better to separate this logic away from the Sprites themselves into smaller controllers. For example, having a PizzaView for the visual presentation only and a PizzaController for the collision detection. It would also be beneficial to separate the different classes into separate folders in the project depending on whether they relate to the Model, View, or Controller.

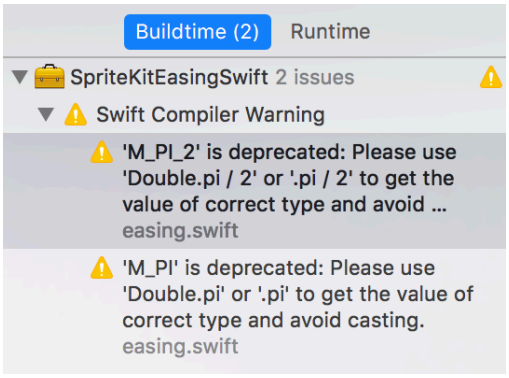
A key issue with the gameplay is that the game only ever requests a combination of 3 ingredients from a possible 4. This means that there is only ever 4 possible combinations, so if the player approaches the game logically, they will always beat the game within 4 guesses. The intention in the proposal was to feature progressively harder levels with more and various ingredients, so as such the code produced may be considered to represent the first level in a series. Adding extra levels would be relatively straightforward as more delegate methods could be added to the GameViewController to display more scenes. Additionally, if the Model logic was further extracted from the scenes as outlined above, it would be easy to generate data for each level separately from the Controllers.

The code could also be improved by going through the classes and making the majority of properties and methods private - by default a Swift class will make its variables public. By ensuring the properties and methods are private, the logic inside the classes can be properly encapsulated.

Some behavioural logic could also be added to make the app respond more gracefully to interruptions such as notifications from other apps. A way to pause the game from the main gameplay screen might also be valuable.

The third party library, SpriteKitEasingSwift, also generates a couple of warnings related to the deprecation of M_PI_2 , though it would be inadvisable to modify the library logic directly (Figure 8). There is also a small bug where the app icon zooms to fullscreen as the app is loading, which would need addressing before publishing. The code quality could also be greatly improved with the addition of unit tests, which would also improve code confidence when refactoring.

Figure 8: SpriteKitEasing warnings



3. Suitability for the App Store

To be considered a complete game, music and more levels would need to be added. Before submitting the app, a full review of the assets would have to be undertaken. Due to time restrictions, only one large asset was included for each game Sprite. Apple's Assets Catalogue provides the ability to add different assets for display on different devices for optimisation[8], so each game asset should be replicated at the correct sizes and added to the catalogue. This has already been done for Diner Deduction's launch screens and app icons, but all the other assets used would need thorough checking. Additionally, not all of the artwork in the prototype app was original - the pizza image and a few of the ingredients were provided by Freepik[9] - and fresh artwork should be created to avoid potential copyright issues.

Although the game is functional on iPad, many of the game assets will need scaling correctly before the app can be published, so some code adjustments would need to be made alongside the asset review. It is likely that that game will be more enjoyable on iPad, as the larger screen size will make it easier to see the images (especially as the levels progress in difficulty and more ingredients are required), therefore optimising the game for iPad would be a priority. A code review should also be completed, ensuring that there is no unused logic, the comments are removed, and that the code is optimised. All features should be documented. The total game size should be as small as possible, and the app should be tested on as many physical devices as possible, including lower-end devices.

Data for iTunes Connect would also need to be submitted, including screenshots of the game at various sizes, and an app preview poster frame[10]. The metadata would need to be correct; a version number would need to be set, a pricing structure chosen, and an appropriate category selected. In the proposal it was suggested that the app could be submitted to the Kid's Category. However, as Facebook has a minimum age of 13,[11] this would be inappropriate if Facebook integration was eventually added. It was also found in user testing that younger children found the core game concept of deduction hard to grasp, making the game more suited to teenagers and adults. The game would therefore be listed under the 'Games' category.

4. Marketing Plan and Future Development

There are multifarious features suggested in the proposal that could be added to future updates of the game. More ingredients, strange ingredients, different meals to create (such as burritos, sushi, hamburgers) and random sudden events such as "Christmas" with abrupt changes in music and ingredients could all be implemented. The element of surprise is a key component of solid game design[12], and it is this aspect that makes players return, and feel inspired to share it with their friends.

As the game does not include any in-app purchases, social media integration would be a key strategy for app promotion. Facebook or other social media presence for the game would also be essential to support this. The development of a simple and clean website to accompany the app would also be beneficial, featuring clear and fun screenshots of the game, videos of gameplay, and a prominent button to the App Store. Continuous promotion would be a key part of maintaining user interest, by having a mailing list and announcing on Facebook when new features are added, such as new dishes to create.

The game would be pay-to-download, with a reasonably low price (say £1.50 or so), rather than be more costly with a free version that has limited features. This is because Hsu et al. (2015) have found that offering a free alternative to a paid app can have a negative effect on purchase intention, and that paid-for apps are perceived as having greater value[13]. The marketing strategy is that the game would be low priced enough to entice new users to try it, and they would be encouraged to share it with their friends due to the strong social media component of the game.

Reviews are extremely important for launching a successful app[14], especially initially, so it will be crucial to deliver a polished and complete build of the game at launch that looks good on all supported devices. Joorabchi et al. (2013)[15] noted that one of the principle challenges of app creation for small companies is that developers are

often also testers, which can result in bugs being overlooked. Therefore it may be beneficial to invest in some external manual testers before publishing the game, and take time to introduce automated testing where possible. The game should also prompt users that have been playing for a while to leave a review of the game, thereby increasing the total reviews. Any negative reviews left should be analysed, and issues should be remedied.

5. References

[1] Swink, S. (2008) *Game Feel: A Game Designer's Guide to Virtual Sensation*. CRC Press.

[2] Cocoapods (2015) https://cocoapods.org Accessed 3rd October 2017.

[3] Grummitt, C.(2017) SpriteKitEasingSwift Library https://github.com/craiggrummitt/SpriteKitEasingSwift Accessed 10th September 2017.

[4] Gamma, E. Helm, R. Johnson, R. Vlissides, J. (1995) *Design patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[5] Apple Developer (2015) *Delegation*. https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html Accessed 6th October 2017.

[6] Apple Developer (2015) *Class: NotificationCenter*. https://developer.apple.com/documentation/foundation/notificationcenter Accessed 6th October 2017.

[7] Jazzy: Soulful docs for Swift & Objective C (2017) https://github.com/realm/jazzy Accessed 8th October 2017.

[8] Apple Developer (2015) *Asset Catalog Contents*. https://developer.apple.com/library/content/documentation/Xcode/Reference/xcode_ref-Asset_Catalog_Format Accessed 7th October 2017.

[9] Freepik (2015) *Slices of Pizza Free Vector.* http://www.freepik.com/free-vector/slices-of-pizza_800504.htm Accessed 28th July 2017.

[10] Apple Developer (2017) *iTunes Connect Developer Help.* https://help.apple.com/itunes-connect/developer Accessed 7th October 2017.

[11] Facebook Help Center (2017). *How do I report a child under the age of 13?* https://www.facebook.com/help/157793540954833 Accessed 7th October 2017.

[12] Schell, J. (2015) *The Art of Game Design: A Book of Lenses*. 2nd edition. CRC Press.

[13] Hsu, C.-L. and Lin, J.C.-C. (2015) What drives purchase intention for paid mobile apps? – An expectation confirmation model with perceived value. *Electronic Commerce Research and Applications* 14, 46-47.

[14] Hughes, J. (2012) *iPhone & iPad Apps Marketing*. 2nd edition. Que Publishing.

[15] Joorabchi, M.E. Mesbah, A. Kruchten, P. (2013) Real Challenges in Mobile App Development. *ACM/IEEE International Symposium of Emprirical Software Engineering and Management*, 15-24.